A New Algorithm for Computing Shortest Paths in Weighted Planar Subdivisions

(Extended Abstract)

Cristian S. Mata * Joseph S. B. Mitchell †

Abstract

We present a practical new algorithm for the problem of computing low-cost paths in a weighted planar subdivision or on a weighted polyhedral surface. The algorithm is based on constructing a relatively sparse graph, a "pathnet", that links selected pairs of subdivision vertices (and "critical points of entry") with locally optimal paths. The pathnet can be searched for paths that are provably close to optimal and approach optimal, as one varies the parameter that controls the sparsity of the pathnet.

We analyze our algorithm both analytically and experimentally. We report on the results of a set of experiments comparing the new algorithm with other standard methods.

1 Introduction

For a given weight function, $F : \Re^2 \rightarrow \Re$, the *weighted length* of an s-t path π in the plane is the path integral, $\int_\pi F(x,y)d\sigma$, of the weight function along the path π, linking the *start* s to the *goal* t. The *weighted region metric* associated with F defines the distance $d_F(s,t)$ to be the infimum over all s-t paths π of the weighted length of π. The *weighted region problem* (WRP) is to find an optimal s-t path according to the weighted region metric, d_F, induced by a given piecewise-constant weight function, F. This problem is a natural generalization of the shortest-path problem in a polygonal domain: Consider a weight function that assigns weight 1 to P and weight ∞ (or a sufficiently large constant) to the obstacles (the complement of P).

The weighted region problem models the minimum-time path problem for a point robot moving in a terrain of varied types (e.g., grassland, brushland, blacktop, bodies of water, etc), where each type of terrain has an assigned weight equal to the reciprocal of the maximum speed of traversal for the robot. It also arises in numerous other applications involving route planning in geographic data ([5, 15, 23]), military mission planning and decision support ([1, 10, 11, 13, 14, 15, 21, 22]), and fluid flow in injection molding [8].

We assume that f is specified by a triangulation having n vertices, with each face assigned an integer weight $\alpha \in \{0, 1, \ldots, W, +\infty\}$. We can allow edges of the triangulation to have a weight that is possibly distinct from that of the triangular facets on either side of it; in this way, "linear features" such as "roads" can be modeled.

This paper describes an algorithm to compute nearly optimal paths between two points in weighted polygonal subdivisions. We call the data structure built by the algorithm a *pathnet*. The pathnet can be used to answer path queries between pairs of points in the subdivision.

The pathnet algorithm is strongly motivated by results on approximating Euclidean shortest paths among obstacles (as in Clarkson [3] and Mitchell [16]), t-spanners [9], and approximation methods devised for minimum spanning trees [24]. In essence, the method is simply a generalization of the "fixed orientation" metric approximation to Euclidean distances, as employed in [3, 16], with appropriate modifications and complications that arise in the weighted region metric; e.g., a "straight" ray eminating from a point is now a locally optimal path, which refracts according to Snell's Law.

The new algorithm has been fully implemented (in C++) and tested against various other approaches on a large set of data. We report here on our experimental results.

Summary

(1) We present a new algorithm for the weighted region problem, and analyze its effectiveness in approximating an optimal path. The algorithm is based on constructing a graph, a *pathnet*, that is guaranteed to contain an approximately optimal path between two query points. By varying the parameter, k, that controls the density of the graph, we are able to get arbitrarily close to optimal.

(2) We report on an implementation in C++ of our pathnet algorithm. The implementation is part of a more extensive system, called *WRP-Solve*, that has been built, with a convenient graphical user interface, to compare various approaches to route planning. In ad-

*Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400, email: cristian@cs.sunysb.edu; Supported by NSF (CCR-9204585 and CCR-9504192) and by a grant from Hughes Aircraft.

†Department of Applied Mathematics and Statistics, State University of New York, Stony Brook, NY 11794-3600, email: jsbm@ams.sunysb.edu; Partially supported by NSF grants CCR-9204585 and CCR-9504192, and by grants from Boeing Computer Services and Hughes Aircraft.

Computational Geometry 97 Nice France

dition to the pathnet algorithm, the system contains implementations of at least four other approaches to solving the WRP, and the user can freely select among them.

The software has been integrated within an extensive military simulation system, ModSAF, by our industrial collaborators at Hughes Aircraft. We expect to release the *WRP-Solve* code to the public domain.

(3) We provide experimental results comparing the pathnet approach to other approaches to the WRP. We present results obtained in both real and simulated datasets.

Related Work

There are numerous papers on optimal route planning in weighted terrain maps. We refer the reader to the survey [15] for pointers to the literature.

The main theoretical results known for the weighted region problem are presented in Mitchell and Papadimitriou [17]. They give an algorithm, based on the "continuous Dijkstra method", to find a path whose weighted length is guaranteed to be within a factor $(1+\epsilon)$ of optimal, where $\epsilon > 0$ is any user-specified degree of precision. The time complexity of the algorithm is $O(E \cdot S)$, where E is the number of "events" in the continuous Dijkstra algorithm, and S is the complexity of performing a numerical search to solve the following subproblem: Find a $(1+\epsilon)$-shortest path from s to t that goes through a given sequence of k edges of the triangulation. They show that $E = O(n^4)$ and that there are examples where E can actually achieve this upper bound. The numerical search can be done using a form of binary search that exploits the local optimality condition: An optimal path bends according to "Snell's Law of Refraction" when crossing a region boundary. (The earliest reference we have found to the use of Snell's Law in optimal route planning applications is to the work of Warntz [23].) This leads to a bound of $S = O(k^2 \log(nNW/\epsilon))$ on the time needed to perform a search on a k-edge sequence, where N is the largest integer coordinate of any vertex of the triangulation. Since one can show that $k = O(n^2)$, this yields an overall time bound of $O(n^8 L)$, where $L = \log(nNW/\epsilon)$ can be thought of as the bit complexity of the problem instance.

The algorithm of [17] assumes that the start (source) point is fixed, and computes a representation of optimal paths from the start to all other points (a form of "shortest path map"). If the start point is moved, the algorithm would need to be run all over again from the new start point. In contrast, our new algorithm computes a data structure that can be used for path queries between *pairs* of points. (The query time, though, is not logarithmic, as is the case if one builds a shortest path map and queries with a new goal point.) Further, the algorithm of [17] has worst-case running time that is *logarithmic* in $1/\epsilon$, while our new algorithm is linear in $1/\epsilon$; however, the worst-case dependence on n is much better for the new algorithm: $O(n^3)$ versus $O(n^8)$. Finally, in contrast with the algorithm of [17], our new algorithm has been fully implemented and its practicality has been shown.

Various special cases of the weighted region problem admit faster and simpler algorithms. For example, if the weighted subdivision is rectilinear, and path length is measured according to weighted L_1 length, then efficient algorithms for single-source and two-point queries can be based upon searching a path-preserving graph [2]. Similarly, if the region weights are restricted to $\{0, 1, \infty\}$ (while edges may have arbitrary (nonnegative) weights), then an $O(n^2)$ algorithm can be based on constructing a path-preserving graph similar to a visibility graph [6]. This also leads to an efficient method for performing *lexicographic* optimization, in which one prioritizes various types of regions according to which is most important for path length minimization.

2 Preliminaries

We let S denote a planar polygonal subdivision having convex faces. (In the event that the input subdivision has nonconvex faces, we first convexify (e.g., by triangulation).) We will assume that S is given to us in a convenient data structure that allows the usual basic operations to be done in constant time; e.g., the quad-edge data structure [7] is one possibility (a variant of which is used by our code).

Let n denote the total number of vertices of S. Then S also has $O(n)$ faces and edges. Each face f has an associated integer weight $\alpha_f \in \{0, 1, \ldots, W, +\infty\}$. The weight of an edge e is also an integer $\alpha_e \in \{0, 1, \ldots, W, +\infty\}$ and is assumed to be at most $\min\{\alpha_f, \alpha_{f'}\}$, where f and f' are the faces incident on e. (In the event that the input does not specify a weight for e, we define $\alpha_e = \min\{\alpha_f, \alpha_{f'}\}$.)

A path is said to be *locally optimal* (an *LO-path*) if an infinitesimal perturbation of it does not result in its weighted length decreasing. The following properties of LO-paths in weighted regions are important to our algorithm; see [17] for proofs and further details.

(1) An LO-path is piecewise-linear, with bend points only at vertices or on edges of the subdivision.

(2) If an LO-path bends at a point on an edge, while crossing that edge, then it does so in a manner that obeys Snell's Law of Refraction: $\alpha_f \sin\theta = \alpha_{f'} \sin\theta'$ where θ and θ' are the angles of incidence and refraction respectively, and α_f and $\alpha_{f'}$ denote the corresponding face weights. (The angle of incidence (refraction) is defined to be the angle between the incoming (outgoing) segment and the normal to the edge.)

(3) If an LO-path bends at a point on an edge, while *entering* the edge at that point (turning to follow the edge for some distance), then it enters the edge at the critical angle of refraction, $\theta_c = \sin^{-1}(\alpha_e/\alpha_f)$.

(4) An LO-path cannot be incident on an edge at an angle greater than the critical angle, θ_c.

3 The Pathnet Algorithm

The pathnet algorithm constructs a graph, a *pathnet* $\mathcal{G} = (V, E)$, on the vertices V of the subdivision. The basic idea is simple: We discretize the continuum of orientations about each vertex $v \in V$, using k evenly-spaced directions to approximate the full range $[0, 2\pi]$. Within each of the k cones determined at v, we determine a possible edge (a *link*, which is a chain of segments, crossing edges of the subdivision) joining v to another vertex, u, of S, or to a critical point of entry on some edge of S. It is possible that no link is created for a given cone; but there is *at most one* link per cone.

The (possible) link is determined by tracing out a locally-optimal path (*refraction ray*) from v at each of the two orientations bounding the cone, advancing both paths (rays) in lock-step across faces of S, obeying Snell's Law of refraction

with each crossing, until the two refraction rays first encounter a "topological fork" – the edge sequence of one ray first starts to differ from the edge sequence of the other ray. (Actually, for the sake of efficiency, we advance all k rays out of v in lock step, so that each ray is traced only once, not twice.) In other words, we determine the longest common edge sequence prefix of the refraction rays bounding each cone. This fork event must be of one of the following types:

(a) The two refraction rays are incident on distinct edges of S – then there is at least one *splitting vertex*, u, on the face f where they first fork, such that u "splits" the two paths. We now create a link (an edge in E) between v and (any) one such vertex u. Associated with this link is a pointer to the polygonal chain consisting of the line segments (along either of the two rays) that brought us to face f, and the segment joining the entry point onto f with the vertex u. This chain represents an approximation to a refraction path from v to u. We store its length as the weight of the edge $(v, u) \in E$. Refer to Figure 3.

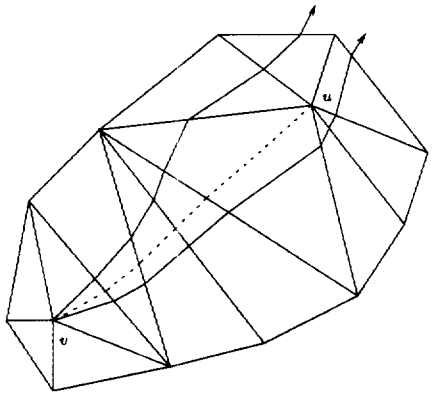

Figure 1: A cone determined by two refraction rays from vertex v gets split by vertex u.

Lemma 1 *There exists a unique locally-optimal path from v to u that stays within the refraction cone that is split by u.*

Proof. The proof follows from the monotonicity lemmas proven in [17]. □

Remark. Optionally, we can search for a chain linking v and u that is arbitrarily close to being a refraction (locally optimal) path; our code does this by a simple binary search, or a coordinate descent method. In practice, this may lead to somewhat shorter paths, and better "looking" paths in some cases. However, we see no way to make the worst-case error analysis take advantage of this extra optimization. We omitted it from the subset of experiments reported in this abstract.

(b) The two rays are incident on the same edge, e, but one or both rays are incident at an angle whose magnitude is greater than that of the critical angle at e.

Lemma 2 *There exist at most two locally-optimal paths (corresponding to incidence angles $+\theta_c$ and $-\theta_c$) from v to a critical entry point on edge e that stays within the refraction cone.*

Proof. The proof follows from the monotonicity lemmas proven in [17]. □

We can trace a critical refraction path from a critical entry point $u \in e$ back to v, through the cone, since the edge sequence is known, and the critical angle at e will therefore determine all of the angles of incidence along the edge sequence (even though the point u is not discovered until we trace the path forward again). (Refer to Figure 3.) A pointer to this chain is stored with the link $(v, u) \in E$, and its length is recorded as the weight of the pathnet edge. Furthermore, the critical entry point u is instantiated as a new vertex of the pathnet (it is added to V), and it is linked to neighboring such critical points along e (or the endpoints of e). Note that, in total, no more than $O(kn)$ critical entry points will ever be added to the pathnet.

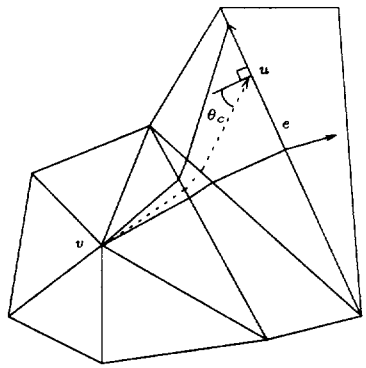

Figure 2: A cone determined by two refraction rays from vertex v gets split at edge e, due to a critical entry point $u \in e$.

Remark. If one ray is incident on e at an angle greater than θ_c and the other ray is incident at an angle less than $-\theta_c$, then it is possible that within the refraction cone there are *two* critical refraction paths incident on e. However, if both rays are incident at angles greater than θ_c (or both $< -\theta_c$), then there will be *no* critical refraction path within the cone.

(c) Both rays encounter the outer boundary of the subdivision. In this case, we simply cease the tracing of the cone, as no link of the pathnet will be made within it.

Once the pathnet is constructed, we search for a path between two vertices simply by running Dijkstra's shortest-path algorithm within the pathnet. (Alternatively, we can apply standard heuristic search variants of Dijkstra's shortest path algorithm, such as the A^* algorithm [15]; however, our experiments reported here are all based on using Dijkstra's algorithm directly, not with the A^* algorithm.) For arbitrary start/goal points, we first do point location queries to locate them in the subdivision, and then temporarily link them into the pathnet, by applying the same refraction ray tracing that was done from each vertex of S, in constructing $\mathcal{G}$. Then, a search of the augmented pathnet yields the desired path.

Alternatively, as we mention again later, our implementation also has an "on-demand" feature, which allows one to build the relevant portion of the pathnet "on the fly", during path search for a given query, without first constructing the full pathnet.

4 Analysis

The pathnet is a graph with $O(kn)$ nodes, since each of the k cones at each of the n vertices can produce at most one link. The propagation of the two rays defining a cone can be truncated after at most $O(n^2)$ steps (and typically it does not require more than about 5-10, in practice). This follows from Lemma 7.1 of [17], which proves that a locally optimal path that does not go through vertices or critical points can cross each edge at most $O(n)$ times, implying an edge sequence of at most $O(n^2)$ crossed edges. Thus, we get an overall worst-case time complexity of $O(kn^3)$, to build a data structure of size $O(kn)$.

It now remains to prove that a shortest path in the pathnet approximates the shortest weighted path between two points. The full details of the proof are deferred to the full paper, as they are somewhat tedious and are not central to the main ideas and applicability of the pathnet algorithm.

We begin by observing that if a large enough number k of cones is used, then any vertex to vertex path (or vertex to critical entry point path) is contained in a single cone and the pathnet algorithm provides exact answers to shortest path queries.

Next, we observe that if we use k cones per vertex, then the angle between two rays that define a cone begins at angle $2\pi/k$, and, after multiple refractions of the rays along a common edge sequence, never gets larger than $O(W/kw)$, where W is the maximum finite weight and w is the minimum nonzero weight. (This follows from the simple geometry of Snell's Law.)

Consider an optimal path, π^*, from a vertex s to a vertex t. We will prove the existence of a path, π, that lies in the pathnet whose (weighted) length is at most $(1+\epsilon)$ times that of π^*, with an appropriate choice of ϵ, which depends on k. Now, π^* is partitioned into subpaths by the vertices and critical entry points along π^*. Consider one such subpath of π^*, call it $\pi^*_{u,v}$, joining vertex u with vertex v. (The case of a subpath joining two critical entry points, or a vertex and a critical entry point, is handled similarly.) In the proposition below, we prove that there exists a path, $\pi_{u,v}$, from u to v, within the pathnet, whose weighted length is at most $(1+\epsilon)$ times that of $\pi^*_{u,v}$, for an appropriate choice of ϵ, as a function of k. It is then easy to conclude that any optimal path π^* can be approximated by paths within the pathnet, within the same approximation factor, since each of the subpaths can be approximated.

Proposition 1 *There exists a path, $\pi_{u,v}$, from u to v, within the pathnet, whose weighted length is at most $(1+\epsilon)$ times that of $\pi^*_{u,v}$, where $\epsilon = O(\frac{W/w}{k\theta_{min}})$, θ_{min} is the minimum among the internal face angles of the subdivision, and W/w is the ratio of the maximum non-infinite weight to the minimum non-zero weight.*

Proof. (Sketch) Consider the subpath $\pi^*_{u,v}$. Consider the refraction cone with apex u that locally contains $\pi^*_{u,v}$. If this cone is split by vertex v, then we appeal to the claim below, and we are done. Otherwise, if the cone is split by some $r \neq v$, then we show that we can make a detour to r (at a "small" cost), and then we can appeal to induction. (Specifically, we do induction on the index i of the sorted lengths λ_i, which are the lengths of the links on the pathnet: $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$, where $N \leq \min\{nk, \binom{n}{2}\}$.)

Let P be the polygon determined by the two refraction rays bounding the cone, stopping the rays at the points where they exit the face f where the rays are first split, and closing the polygon P using the boundary of f (which contains the splitting vertex r). When the cone is split, the bounding refraction rays are at angle $O(W/kw)$. This fact, together with the lower bound on the face angles, allows one to show

Claim: The detour necessary to make $\pi^*_{u,v}$ go visit r (by a detour from the point of exit from face f) is at most $O(\frac{W/w}{k\theta_{min}})$ times the length of $\pi^*_{u,v}$ within P. □

Theorem 3 *In time $O(kn^3)$ a pathnet of size $O(kn)$ can be constructed such that, when searched, it yields paths that are within factor $(1+\epsilon)$ of optimal, where $\epsilon = O(\frac{W/w}{k\theta_{min}})$.*

5 Other Algorithms

In order to determine the practicality and relative effectiveness of our new algorithm, we implemented several other methods, for experimental comparison.

5.1 Grid-Based Algorithm

The most basic and most popular practical approach to solving path planning problems in varied terrain is to us a discretization into a uniform grid, and then to search the (weighted) *grid graph* for shortest paths. This method has many advantages over other more complex algorithms: (1) it is simple to implement; (2) it is particularly well suited for problems in which the input data structure comes as a grid; and (3) it is highly versatile, allowing for many other heuristic cost criteria (e.g., charging for turns) to be added quite easily.

We assume as input an array of sampled values of the underlying weight function F, giving values, $F_{i,j}$, at regular grid points indexed by (i,j). Since our pathnet algorithm assumes as input a polygonal subdivision, not a weight array, our grid algorithm initializes a weight array by sampling the polygonal subdivision at grid points (whose spacing, γ, is a user-specified parameter).

The grid data points determine a *grid graph*, whose vertices are the grid points and whose edges link "nearby" grid points. Most grid-based algorithms assume a 4- or 8-connectivity of the grid, joining point (i,j) to its four immediate neighbors $((i,j+1), (i,j-1), (i-1,j), (i+1,j))$, plus possibly the diagonals.

One of the main drawbacks to a grid graph solution is that, even for a trivial (constant) weight function F, there is an inherent error in distance estimate, since we measure grid distance instead of Euclidean distance. For 4-connectivity this "digitization bias" (aliasing) results in an error factor of $\sqrt{2}$; for 8-connectivity, it results in a factor of $(\sqrt{2}+1)/\sqrt{5} = 1.08$. See [18, 20] for discussions on the digitization bias problem, and on different distance transforms that can be used to address it.

One approach to reducing grid bias is to increase the connectivity of the grid. Thus, the grid-based algorithm that we implemented includes an enhancement to the usual methods, in that we permit higher degrees of connectivity: We connect (i,j) to each of the grid points $(i+I, j+J)$,

for $I, J \in \{-K, -K+1, \ldots, K-1, K\}$, except those that can be reached directly (without error) by two shorter edges (e.g., we do not connect (i,j) to $(i+2, j+2)$, since this is effected by two diagonal connections already). Thus, for $K = 1, 2, 3, 4$, we get connectivity of 8, 16, 32, 48, respectively. Allowing this K as a parameter implies that, as K goes to infinity, and the resolution γ goes to zero, the grid-based solution will in fact go to the true optimum. (The dependence on K is $O(1/K^2)$.) This will allow a fair comparison with our pathnet algorithm, for which we can also get arbitrarily close to optimum by increasing a parameter (k, the number of cones).

Since our underlying input function F is given in polygonal subdivision form, we assign weights to the edges in the grid graph by computing the actual weighted cost of the direct (straight) line segment joining the endpoints. This is done by a simple ray tracing (walk) through S, integrating F along the segment.

5.2 Edge Subdivision Algorithm

We also devised and implemented another natural method for searching a weighted subdivision for paths, based on discretizing the *edges* of the subdivision (rather than discreting the *faces*, as is the case with a grid-based method). This method has been implemented and used by several others, including Mitchell (who implemented it as part of a system built at Hughes, for the DARPA ALV project in 1985), Johansson [8] (who implemented it for use in fluid flow computations for injection molding), and Lanthier, Maheshwari, and Sack [12] (who, in independent work closely related work our own, have conducted experiments on this, and related, practical methods for the WRP).

The basic *edge subdivision algorithm* is as follows. For each edge e of the subdivision, we place $\lfloor \alpha_e/\delta \rfloor$ new (Steiner) nodes, evenly spaced along e. Here, δ is a parameter of the algorithm, and α_e denotes the weight of edges e. We construct (weighted) edges joining each such Steiner node with all other nodes (both original vertices and Steiner nodes) that appear on the boundary of each of the two faces incident on e. In this way, each face of the subdivision corresponds to a complete graph on the set of vertices and Steiner nodes that appear on its boundary. Since faces are assumed to be convex, the line segment joining any pair, u and v, of boundary points lies entirely within the face, so we assign the weight of (u,v) to be $|uv| \cdot \alpha$, where α is the weight of the face. (Note that the shortest path joining u and v need not be the single edge, (u,v), especially if the weight α of the face is very high.)

The worst-case error in the approximation is easy to estimate: For an optimal subpath that crosses m edges, we can perturb it onto the graph induced by the subdivision nodes, adding weighted length $O(m\delta)$ to the path. This shows that the shortest path in the network constructed will be at most an additive term $O(m\delta)$ longer than optimal; as shown in [17], $m = O(n^2)$.

The edge subdivision algorithm has the following design choices: (1) the parameter δ (or one can specify a maximum number of Steiner points on any one edge); and (2) how to space the points on an edge, e.g., evenly (as we do), in geometric progression (as Papadimitriou [19] does for shortest paths in $\Re^3$), or in some other way. We have not yet seen how to use uneven spacing to achieve provably better approximation bounds; this may be a subject of future experimental analysis.

5.3 Other Approaches

Our *WRP-Solve* system also has a couple of other heuristic algorithms implemented:

(1) Search the (weighted) edge graph of S for a shortest path between two vertices. (If the start/goal are not vertices of S, then we simply augment the graph by connecting them to the vertices of the convex containing cell.) This method has the advantage of being very simple and fast, but of course it can produce paths that are arbitrarily bad with respect to optimal.

(2) Search the dual graph of S, using edge weights that are the average of the two adjacent face weights, times the distance between the centroids of the faces. Again, this method is fast, but can yield arbitrarily bad paths with respect to optimum. (Note that if the faces are all unit squares – i.e., S is a regular grid – then we are simply doing a 4-connectivity grid graph method, and the approximation error factor is then bounded by $\sqrt{2}$.)

Other approaches being currently designed for the *WRP-Solve* system include a simulated annealing approach (using, e.g., a starting seed path determined by either of the fast heuristics (1) or (2) above, together with a post-processing local optimality stage), and more sophisticated grid-based or quadtree-based algorithms. It will also be interesting to implement and test *hybrid* methods of computing optimal paths, using the best aspects of pathnets, grid graphs, edge subdivisions, etc.

6 Implementation and Experimental Setup

The *WRP-Solve* system is implemented in C++. Experiments were run on a set of SGI Indigos with R5000 processors at 150MHz running IRIX6.2. Internal memory size was 64MB for all, 512KB level 2 and 32KB separate data and instruction level 1 caches. Times quoted in the experiments were measured only on the SGI. For memory intensive experiments we used an Intel Pentium Pro computer at 180MHz with 128MB of memory.

Th system has several options for the user to set parameters, read/write data, time experiments, and select different algorithms. A screen shot showing one example is given in Figure 3.

6.1 Design choices

All of the algorithms model weighted subdivisions and grid sampled data with an undirected network where the nodes of the network correspond to points in the plane and the edge weights represent the weighted distance between nodes. The nodes are associated to a feature of the subdivision – vertex, edge, face – or to a Steiner point added by sampling an edge or a face of the subdivision.

Network representation is dependent on the search algorithm. For example, the edge subdivision algorithm does not require an explicit representation of the links between nodes on the same face. Determining the neighbors of a node during the search algorithm and the weighted length of the links can be done while the search algorithm is executed. This network representation would require no more storage than storage for the nodes but it would be inefficient in terms of execution time because dynamically determining the neighbors of a node has an impact on performance. On

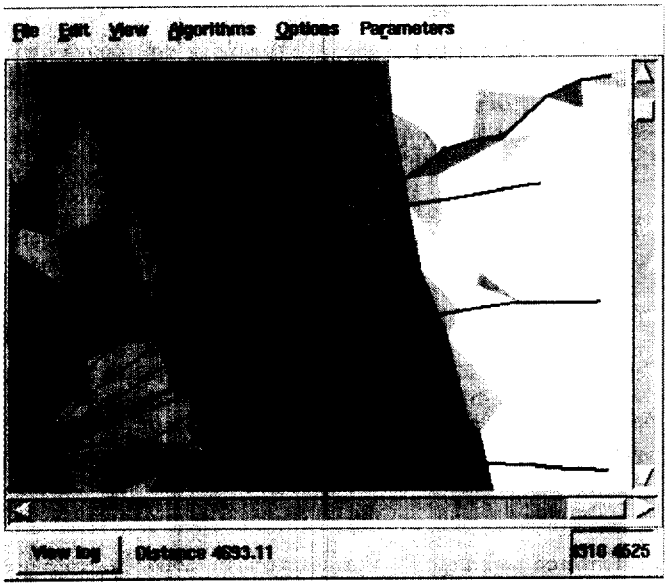

Figure 3: A screen shot from *WRP-Solve*. The starting point is in the lower left corner. Pathnet links to the start are shown with thin segments. Thicker segments (shown in "red" when viewed on a color workstation) are used to highlight the results of various shortest path queries to other points in the subdivision. Shading is used to indicate weight, with whiter regions having higher weights.

the other hand, building a complete representation – nodes and adjacency lists – for a large sampled grid is not practical given the large number of nodes and links.

The requirements that each algorithm imposes upon the representation of the search network had to be balanced with the fact that the experimental setup must compare the performance of algorithms. The choice was made to use the the same data structures for all algorithms.

The advantage of this solution is that the code used to implement the search algorithm for all the approximation algorithms is the same. This means that the number of elementary operations to compute a shortest path can be used to compare the performance of the algorithms. On the other hand, a specialized implementation for any one of the search methods would result in better performance – in terms of space and time used.

6.2 On-demand behavior

For all of the algorithms implemented, we have the option in *WRP-Solve* to do the preprocessing (e.g., pathnet construction) off-line or "on demand". In the "on-demand" mode, which is the default, the search network is constructed only as needed during the processing of a shortest path query. This implies that a search network may not contain all the links and nodes for the dataset. This on-demand feature is for some practical applications in which it is not necessary to preprocess the entire dataset. Once a portion of the search network is built, it is used by all later queries. This efffectively amortizes the cost of building the search network over multiple shortest path queries without requiring any preprocessing time.

For the experiments described in this paper, however, we chose to separate the preprocessing times from the query times. The search network is therefore created off-line – by selecting a start point and by building the shortest path tree for that point. This provides, on one hand, a complete data structure and, on the other hand, the results can be used to compare preprocessing time. (The same data is collected during preprocessing as for the shortest path queries.)

6.3 Datasets

The Weighted Regions Problem has practical applications in multiple domains, including decision support systems for the military [13, 14], manufacturing [8] and Geographic Information Systems. In practice, datasets – weighted subdivisions – have up to 10000 vertices. Most applications use smaller subdivisions – for example in the manufacturing application the size of the dataset is less than a 1000 vertices.

The experiments were run with a mix of real terrain and artificially generated datasets. The drawback of using real datasets is the presence of features that are degenerate or topologically inconsistent. In the implementation these problems are accounted for by letting the algorithms abandon computing a link – locally optimal path – if the underlying data structure is inconsistent.

Real World Datasets. The subdivisions used for the experiments are extracted from the "microterrain" elements in the Hunter-Ligett Compact Terrain Database (CTDB). The terrain database is used in the ModSAF[13, 14] training simulator for visibility and shortest path computation. The weighted subdivision corresponding to microterrain elements is obtained by assigning weights to the faces of the subdivision using a heuristic that assigns to each face integer weights between 3 and 12. The number of different weights used is higher than that which is typically used in practice by ModSAF; it was chosen to make shortest path planning even more challenging for our experiments.

The sizes of the weighted subdivisions vary from less than 100 vertices to around 1000 vertices. The faces of the subdivision are mostly triangles, although buildings and other man-made features have more vertices. The complicating problem with this data is the relatively large number of topologically incorrect features (e.g. overlaping faces and dangling edges).

Simulated Datasets. Artificial datasets are generated using two methods:

1. One method starts with a regular rectilinear grid and randomly perturbs the positions of the vertices, while constraining the perturbation so that all faces of the subdivision remain convex.

2. A second method uses Delaunay triangulations of point sets (based on the algorithm and code of Devillers [4]). In one set of datasets the vertices of the subdivision are generated uniformly at random within a square. Another set of triangulations is designed to yield nonuniform point distributions, by concentrating a majority of the points in a few small disks; the goal is to simulate the irregular distribution of points encountered in applications.

Each of the artificial datasets is generated with a variable number n of vertices, where n = 100, 200, 500, 1000, 2000, 5000, 10000 and 20000. Larger subdivisions are possible, but these datasets incur run-time penalties – for some of the algorithms we compare, these subdivisions lead to

data structures that exceed the amount of internal memory available. It is an open question how the shortest path algorithms fare experimentally in external memory settings.

6.4 Generation of Start/Goal Pairs

The *WRP-Solve* system allows start and goal points to be chosen arbitrarily. A simple (worst-case linear-time) algorithm is used for point location, with special code to handle start/goal points that fall on edges/vertices of the subdivision.

Start/goal point pairs are picked at random with a minimum Euclidean distance between start and goal of 20% of the diameter of the dataset.

Start and goal points act as special vertices of the search network. These vertices are inserted when the search begins and deleted when the search ends. Time measurements include the time to insert the special vertices because this time (exclusive of point location) is partially dependent on the search algorithm. For example, inserting a vertex into a pathnet takes more time than the time to insert a vertex in a grid. In contrast, deleting vertices is a standard operation that involves the same operations for all algorithms implemented.

6.5 Data structures

The choice of representation for the subdivision and network data structures influences directly the performance of the algorithms. This effect is lessened by measuring the time performance for the each algorithms measured on the CPU clock and by counting the number of elementary operations performed by each algorithm during a shortest path search.

Subdivision. The natural choice for representing the input subdivisions is the *quad-edge* [7] data structure. Because of the limited operations needed, and the fact that the datasets are terrains, the implementation is simplified: *Rot* (edge-dual) and *Flip* (orientation) are not implemented.

Faces and vertices of the planar subdivision are objects with attributes. The vertices have (x, y)-coordinates and faces have weights. In all experiments the edges have the default weight which is the minimum weight of the incident faces.

It is possible to define obstacles in the subdivision when the weight of a face is over a user-defined threshold. In this case, no shortest path is allowed to cross an obstacle face. Free (no cost) regions are implemented by using low unit weight faces. Most applications use only a small number of weights. For example, in applications of *WRP-Solve* to air traffic control, Seagull Technologies, Inc. uses heuristics to assign five different weights to regions. The experiments use $\{1, \ldots, 15\}$ for weight values.

Search Network. The search network is a data structure designed to answer efficiently membership queries like "Is there a node at vertex v of the planar subdivision?" Hash tables are used to store the nodes because they are efficient, simple and a robust implementation is available from the Tcl library. The average size of the buckets used in the hash tables was always less than two.

To speed things up separate hash tables are maintained for each of the objects making up a weighted subdivision: vertices, edges and faces.

6.6 Robustness issues

The pathnet algorithm assumes a topologically correct and consistent weighted subdivision. Subdivisions can be input interactively with click and drag. After an input operation, *WRP-solve* performs a topological check to verify consistency before a new edge is added to the data structure.

The procedure that solves Snell's Law of Refraction for each crossing is particularly sensitive to numerical errors. An exact solution requires solving a 4^{th} degree polynomial. If the refraction path is sufficiently close to a subdivision vertex and an incorrect decision is made – because of floating point rounding – the program will go into an infinite loop. To address this issue, each refraction case subproblem is solved with the following constraint: if the crossing point is within a small distance – a constant fraction of edge length, divided by the number of cones – of an endpoint, then the path "snaps" to the edge endpoint.

7 Results

Measured parameters We measured the following parameters for all shortest path queries:

- The number of operations performed during each shortest path search; here, "operations" counts the number of nodes visited by the search algorithm and the number of insertion and update operations to the heap.
- The time spent during each search. We measured two times: (1) the elapsed time which measures the interval of time elapsed between the start of the search and the end of the search - as returned by the function *gettimeofday* and (2) the user and system times as measured by *getrusage*. The reason was that the experiments were run on hardware in use by other users and the elapsed time might prove an incorrect measure.
- The length of the shortest path approximation and the actual Euclidean length between the start and goal parameters.
- The number of nodes in the network.

In addition to the above values, which were measured for all methods, we also measured for the pathnet algorithm some additional parameters, including the number of critical entry points, the number of *simple* and the number of *chain* links, the average degree of the nodes of the search network, and the average number of edge crossings in a chain.

Average query time vs. approximation factor The pathnet achieves consistently approximation factors of less than 5 percent, for all choices of parameters that we tested, and for all datasets. Figure 4 presents a plot of the average time required to process a query versus the approximation factor. The edge subdivision algorithm achieves the same approximation factors using more processing time. The dataset being used for the experiment is a Delaunay triangulation with 1000 vertices. The grid used in the experiment is 8-connected. Results show that the contenders for accurate results are the edge and the pathnet algorithms. The approximation factor obtained with the grid algorithm shows the digitization bias, although by using a higher degree of grid connectivity, results for the grid algorithm can be improved (see the full paper).

Preprocessing time and memory use for pathnet algorithm Preprocessing times required to build pathnets for different parameter values are measured. Results show that building the search network takes time proportional to the number of cones. Build times depend on the number of nodes in the search network, see Figure 6. Triangulations – terrain and Delaunay datasets – have larger size pathnets than grid subdivisions.

The pathnet algorithm achieves good approximation results for shortest paths even when the number of cones used is relatively small – less than 20. The data shows that the time to preprocess a subdivision with 1,000 nodes is less than 10 seconds. Note that in practical use the pathnet is built on demand and the preprocessing penalty is correspondingly lower. For the same subdivision the edge algorithm is on average 2.5 times slower for the same approximation factor. The grid – 8-connected – for a weaker approximation factor uses 10 times more.

The amount of memory used by the pathnet algorithm is proportional to the number of cones, see Figure 5. Artificial datasets have less variation than terrain data.

Number of operations The edge and pathnet algorithms perform fewer operations than the grid algorithm to achieve the same approximation factor. The pathnet has the advantage of links spanning across multiple faces. Figure 7 presents the average number of nodes visited per unit distance for different approximation factors. Scaling to unit distance allows an estimate of the time required to complete a shortest path query – a useful feature in an application where time resources are limited.

Complementary to the average number of operations is the average time required to plan a shortest path for a unit distance, see Figure 8.

Approximation factors for pathnets The experiment looks at the correlation between approximation factors and the number of cones used to build a pathnet. Figure 9 links the parameter – number of cones – necessary to achieve a given approximation factor, with respect to the optimal path.

The x-axis represents the number of vertices. For artificial datasets the number of cones remains relatively constant and decreases as the number of vertices increases.

For a given approximation factor the number of cones is the smallest number such that *all* shortest path queries tested are within the approximation factor. For example, a pathnet with 15 cones will in most cases achieve an approximation factor of 2 percent. A pathnet built with 20 cones achieves on average a 1 percent approximation factor.

The peaks in the plot indicate that some datasets require a large number of cones for a good approximation factor. A useful observation is that the "behavior" of the dataset remains the same over several values of the parameter. In this case, the approximation factor achieved with a small number of cones is a predictor on the approximation factor once the number of cones increases.

8 Conclusion

In this paper, we have taken a practical look at the problem of computing low-cost paths in weighted terrains. Our main contribution is a new method to compute paths in weighted terrains. The method has been fully implemented and tested in planar weighted regions, and comparisons have been made with several other simple and natural heuristic algorithms, such as regular grids and edge subdivision. Our experiments show that the pathnet method performs very well in practice and is highly competitive with other methods, generally yielding a shorter path in less query time. One other advantage of the pathnet algorithm is that its form of discretization is independent of the scale of the dataset. The grid and the edge subdivision algorithms depend in their choice of parameters on the (Euclidean) sizes of the features (edges and faces) in the map.

While the worst-case dependence on ϵ is worse than that of Mitchell and Papadimitriou [17], the worst-case dependence on n is substantially lower, both in terms of time and space complexity: $O(n^3)$ versus $O(n^8)$ time and $O(n)$ versus $O(n^4)$ space.

Several extensions to our work are natural:

1. Our method applies directly to polyhedral surfaces with weighted facets. The only change to the implementation necessary is to modify the ray tracing procedure to compute the refraction path in the "unfolded" version of the local geometry. This code exists already, and will soon be fully tested.

2. The same method also applies to cases of weight functions that are not necessarily constant within faces of the subdivision (or facets of the polyhedron). For example, we could consider piecewise-linear weight functions, or other simple-to-describe weight functions for which we can compute the local optimality condition when a path crosses a region boundary. We also plan to consider weight functions that vary with time.

3. At the other extreme, we can apply our method to the unweighted (constant weight) case too, obtaining an approximation for shortest paths on polyhedral surfaces. For this problem, we know that a locally optimal path can cross only a linear number of edges (since each edge can be crossed at most once), so the complexity becomes $O(kn^2)$. Further, the actual performance in practice may be close to linear, since the average running time is $O(knm)$, where m is the average number of edges crossed by a cone before its defining rays get split; in practice, we expect m to be much less than n.

In conclusion, we should add that there are many other interesting experiments still to be conducted. In particular, we plan to conduct direct comparisons between the methods presented here and the methods recently developed in a parallel effort by Lanthier, Maheshwari, and Sack [12]. Further, it would be interesting to compare our methods' performance to the simulated annealing approaches implemented by Kindl, Shing, and Rowe [10, 11]. By combining different heuristic methods, we expect that the ultimate winner in our quest for better heuristics will be a "hybrid" method, which can take advantage of data given both in terms of regular grids and in terms of polygonal subdivisions.

References

[1] R. Alexander and N. Rowe. Path planning by optimal-path-map construction for homogeneous-cost two-dimensional regions. In *Proc. IEEE Internat. Conf. Robot. Autom.*, 1990.

[2] D. Z. Chen, K. S. Klenk, and H-Y. T. Tu. Shortest path queries among weighted obstacles in the rectilinear plane. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 370–379, 1995.

[3] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 56–65, 1987.

[4] O. Devillers. Robust and efficient implementation of the Delaunay tree. Report 1619, INRIA Sophia-Antipolis, Valbonne, France, 1992.

[5] D. H. Douglas. Least cost path in geographic information systems. Research note No. 61, Department of Geography, University of Ottawa, Ottawa, Ontario, August 1993.

[6] L. Gewali, A. Meng, J. S. B. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. *ORSA J. Comput.*, 2(3):253–272, Summer 1990.

[7] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4:74–123, 1985.

[8] P. Johansson. On a weighted distance model for injection moulding. Linköping Studies in Science and Technology, Thesis No. 604 LiU-TEK-LIC-1997:05, Division of Applied Mathematics, Linköping University, Linköping, Sweden, February 1997.

[9] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete Comput. Geom.*, 7:13–28, 1992.

[10] M. Kindl, M. Shing, and N. Rowe. A stochastic approach to the weighted-region problem, I: The design of the path annealing algorithm. Technical report, Computer Science, U.S. Naval Postgraduate School, Monterey, CA, 1991.

[11] M. Kindl, M. Shing, and N. Rowe. A stochastic approach to the weighted-region problem, II: Performance enhancement techniques and experimental results. Technical report, Computer Science, U.S. Naval Postgraduate School, Monterey, CA, 1991.

[12] M. Lanthier, A. Maheshwari, and J. Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, page To appear, 1997.

[13] M. J. Longtin. Cover and concealment in ModSAF. In *Proc. Fourth Conference on Computer Generated Forces and Behavioral Representation*, pages 239–247. STRICOM-DMSO, 1994.

[14] M. J. Longtin and D. Megherbi. Concealed routes in ModSAF. In *Proc. Fifth Conference on Computer Generated Forces and Behavioral Representation*, pages 305–313. STRICOM-DMSO, 1995.

[15] J. S. B. Mitchell. An algorithmic approach to some problems in terrain navigation. In S. Sitharama Iyengar and Alberto Elfes, editors, *Autonomous Mobile Robots: Perception, Mapping, and Navigation*, pages 408–427. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[16] J. S. B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8:55–88, 1992.

[17] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *J. ACM*, 38:18–73, 1991.

[18] J. S. B. Mitchell, D. W. Payton, and D. M. Keirsey. Planning and reasoning for autonomous vehicle control. *Internat. J. Intell. Syst.*, II:129–198, 1987.

[19] C. H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Inform. Process. Lett.*, 20:259–263, 1985.

[20] I. Ragnemalm. The Euclidean distance transform. Linköping Studies in Science and Technology, Ph.D. Dissertation 304, Department of Electrical Engineering, Linköping University, Sweden, 1993.

[21] R. F. Richbourg, N. C. Rowe, M. J. Zyda, and R. McGhee. Solving global two-dimensional routing problems using Snell's law. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1631–1636, Raleigh, NC, 1987.

[22] N. C. Rowe and R. F. Richbourg. An efficient Snell's law method for optimal-path planning across multiple two-dimensional, irregular, homogeneous-cost regions. *Internat. J. Robot. Res.*, 9:48–66, 1990.

[23] W. Warntz. Transportation, social physics, and the law of refraction. *The Professional Geographer*, 9(4):2–7, 1957.

[24] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11:721–736, 1982.

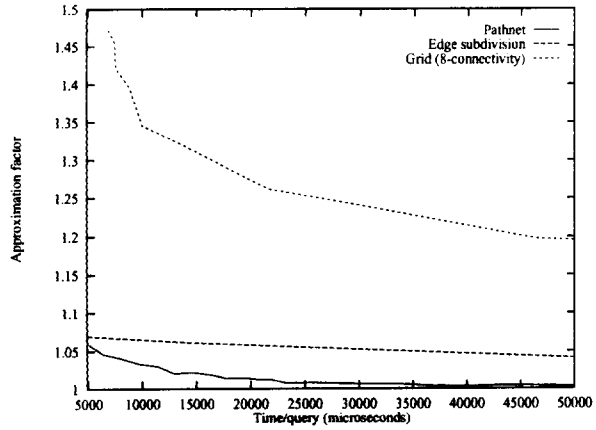

Figure 4: Approximation factor for the pathnet, grid and edge subdivision algorithms with respect to shortest path query time

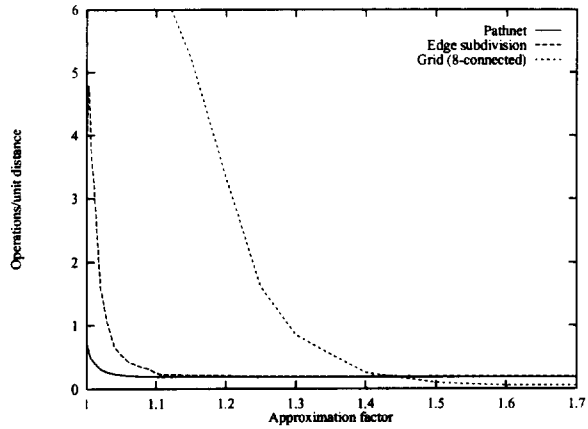

Figure 7: Number of nodes visited with respect to the approximation factor

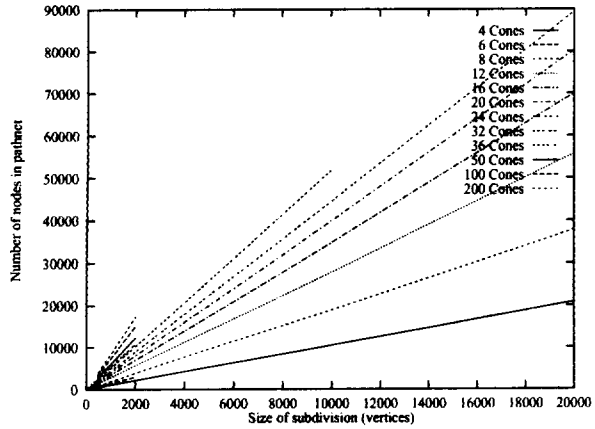

Figure 5: Number of nodes in pathnets with respect to the size of the weighted subdivision

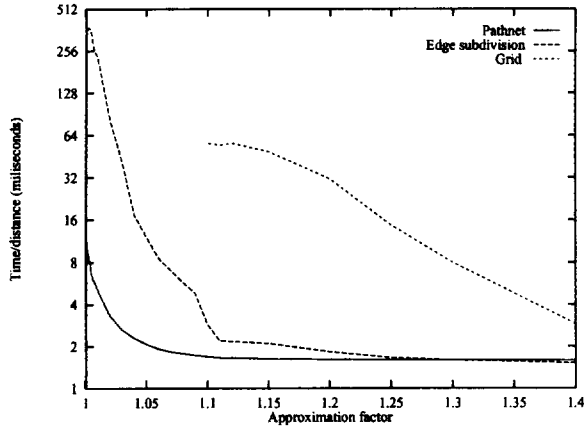

Figure 8: Average search time versus approximation factor, normalized to unit euclidean distance.

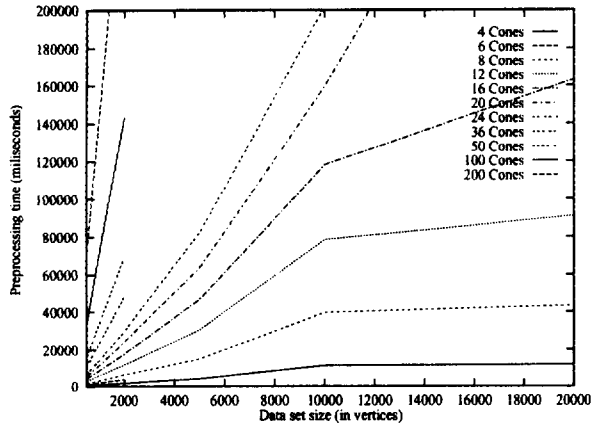

Figure 6: Time required to build a pathnet search graph versus number of cones for each vertex of the weighted subdivision

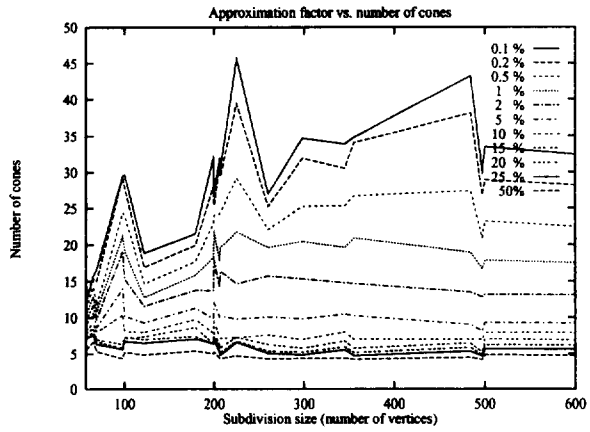

Figure 9: Maximum approximation factors for the pathnet algorithm versus number of cones.