Simplifying a Polygonal Subdivision While Keeping it Simple

Regina Estkowski*
HRL Laboratories
Malibu, CA 90265, USA
regina@hrl.com

Joseph S. B. Mitchell
Applied Math & Statistics
State University of New York
Stony Brook, NY 11794, USA
jsbm@ams.sunysb.edu

ABSTRACT

We study the problem of simplifying a polygonal subdivision, subject to a given error bound, ϵ, and subject to maintaining the topology of the input, while not introducing new (Steiner) vertices. In particular, we require that the simplified chains may not cross themselves or cross other chains. In GIS applications, for example, we are interested in simplifying the banks of a river without the left and right banks getting "tangled" and without "islands" becoming part of the land mass. Maintaining topology during subdivision simplification is an important constraint in many real GIS applications.

We give both theoretical and experimental results.

(a). We prove that the general problem we are trying to solve is in fact difficult to solve, even approximately: we show that it is MIN PB-complete and that, in particular, assuming P $\neq$ NP, in the general case we cannot obtain in polynomial time an approximation within a factor $n^{1/5-\delta}$ of an optimal solution.

(b). We propose some heuristic methods for solving the problem, which we have implemented. Our experimental results show that, in practice, we get quite good simplifications in a reasonable amount of time.

Keywords

polygonal subdivisions, simplification, map generalization, geographic information systems, approximation algorithms

1. INTRODUCTION

In many applications, such as geographic information systems (GIS), very large polygonal subdivisions (*maps*) must be handled and displayed. It is often necessary to compress the original data, simplifying the subdivision in order to reduce the total number of vertices that define it. The *map simplification* problem (or the *map generalization* problem) is to compute a simplification of a given subdivision, subject to various constraints that affect the retention of important features and the aesthetics of the simplified map.

A polygonal subdivision $\mathcal{S}$ is a straight-edge embedding of a planar graph without crossing edges. The non-crossing property is often referred to as the "simplicity" of the input data. The main focus of this paper is on the problem of map simplification with topological constraints, including primarily the constraints of *simplicity* and of maintaining *sidedness* of other point features (we do not want a feature point to change which face contains it after simplification). In Figure 1 we show an example of what can happen when a polygonal map is simplified: the approximations of the two banks of the river intersect each other, and the smallest face of the map gets simplified to a point, which now lies on the other side of a simplified boundary.

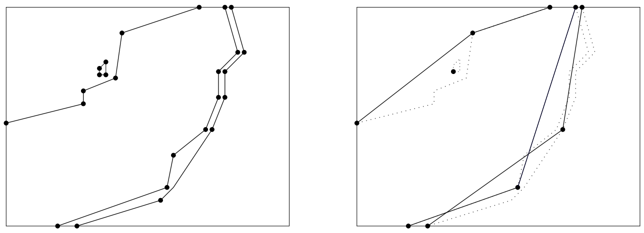


Figure 1: Example of topological errors arising in simplification of maps.

In GIS practice, most often the polygonal chains are considered in isolation, each being simplified using a favorite "line simplification" algorithm, such as that of Douglas and Peucker [8, 12, 14]; see also [4, 5, 16, 17, 19]. Not only does this local approach lead to possible crossings between pairs of simplified chains, it can also lead to self-intersection (nonsimplicity) of a single chain. In this paper, we investigate the problem of map simplification with topological constraints.

The input to our problem is a set $\mathcal{S}$ of polygonal chains (polylines) and feature points, and an error tolerance $\epsilon > 0$. It is assumed that, while two or more chains can share an endpoint, no two chains of $\mathcal{S}$ cross. We let n denote the total number of vertices in $\mathcal{S}$. The output is a set, $\mathcal{S}^*$, of noncrossing chains and feature points. Any isolated feature point of

*This work was conducted while R. Estkowski was a PhD student at the University at Stony Brook.

SCG'01, June 3-5, 2001, Medford, Massachusetts, USA.

$\mathcal{S}$, as well as any vertex of degree one or degree greater than two is preserved in $\mathcal{S}^*$. Each chain of degree-two vertices in $\mathcal{S}^*$ is required to consist of *ϵ-feasible edges*, which are line segments $v_i v_j$ that approximate to within ϵ the corresponding subchain of the input $\mathcal{S}$: each v_k lies within distance ϵ of the line segment $v_i v_j$, for $i < k < j$. We do not permit Steiner points to be introduced; thus, the vertex set of $\mathcal{S}^*$ is a subset of the vertex set of $\mathcal{S}$, omitting (hopefully) a large number of vertices of degree two. We let NSP-SBSIMP ("No Steiner Point Subdivision Simplification") denote the optimization problem of computing an $\mathcal{S}^*$ with a minimum number of vertices.

Summary of Results.

(a) We prove that the problem (NSP-SBSIMP) of minimizing the complexity of $\mathcal{S}^*$ is MIN PB-complete and that it is NP-hard to obtain an approximation whose size is within a factor $n^{1/5-\delta}$ of optimal. Previously, Estkowski [9] had shown the problem to be NP-complete; ours is the first hardness of approximability result for the problem.

(b) We propose a heuristic method, termed the "Simple Detours" (SD) heuristic, for obtaining simplified subdivisions that are *simple*. The method is based on first applying a standard chain simplification method and then "untangling" it to remove intersections that were generated in the simplification process. We also present an extension of this method that allows us to simplify a planar subdivision while preserving homotopy type.

(c) We perform an experimental investigation of the performance of implemented versions of both of our heuristic algorithms, comparing them to the common technique that avoids intersections explicitly during simplification. We use both real and simulated data in our comparisons. The real data includes USGS data (hypsography, hydrography, transportation, and boundary data) and census map data. While our theoretical results are "negative", our experimental results show that, in practice, we obtain quite good reduction in a reasonable amount of time.

Related Work. The problem of simplifying geometric objects while preserving simplicity has had some prior study in both the GIS literature and the computational geometry community. Zhan and Mark [21] have done a cognitive (non-algorithmic) study of how "conflicts" from topological errors in GIS map simplification can be corrected after they occur. Guibas et al [11] prove that computing a minimum-link simple polygon of a given homotopy type is NP-complete, as is computing a minimum-link simple polygonal subdivision that is homeomorphic to an input subdivision, within a polygonal domain. de Berg et al [6, 7] have shown how the methods of [4, 16, 19] can be applied, in conjunction with constraints to guarantee topological consistency, to obtain in $O(n(n+m)\log n)$ time a minimum-size simplification of an x-monotone chain (having n vertices) that is within an approximation error ϵ and homotopically consistent with the input, with respect to a set of m points. They generalize their results to handle simple polygonal subdivisions, but they have no guarantee of being close to optimal. No implementation or experimental results are reported in [6]. Many other methods for polygonal simplification do not address the issue of undesirable topological changes that may occur in the simplification process, such as the loss of simplicity or a change in homotopy type. Currently, in most practical applications, these topological changes are not handled, or are done so in an ad hoc manner; there has been a need for methods which preserve simplicity and homotopy type while producing a usable solution in a reasonable amount of time.

In the time since our work was done, there has been some recent work of [18, 20], which also addresses practical considerations in topologically correct map simplification. In particular, [18] use a natural method of preventing topological changes from occurring by defining "safe sets" (using a Voronoi diagram) to guarantee that standard simplification algorithms within safe sets do not cause changes in topology. While this approach has the advantage of better worst-case time bounds (linear), it is potentially less aggressive in its ability to simplify compared to our own. The related approach of [20] also uses Voronoi diagrams (computed rapidly and approximately using graphics z-buffer support) to constrain standard polygonal chain approximation methods so that only "compliant" shortcut segments are used. These approaches may be more limited in their ability to compress some data; see Figure 2.

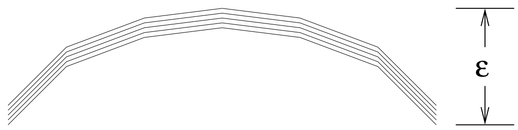


Figure 2: For a set of nested curved chains, constraining the approximation to stay within Voronoi regions may prevent simplification. Here, the optimal simplification replaces each chain with a single segment.

2. HARDNESS OF APPROXIMATION

Our main theorem places our optimization problem in the class of problems that are among the hardest to approximate:

THEOREM 1. *NSP-SBSIMP is MIN PB-complete and is not polynomial-time approximable within a factor $n^{1/5-\delta}$ of optimal, for any $\delta > 0$, unless P=NP.*

PROOF. (Sketch: for the full proof, see [10].) We recall the definition of an *S-reduction* (see Hochbaum [15]). MIN PB denotes the set of minimization problems that are polynomially bounded and are in NP. Suppose that $X \in$ MIN PB, S is a solution for the instance I of X, and Opt is an optimal solution for I; then we define the *performance ratio* of S with respect to Opt to be $R_X(I,S) = |S|/(|Opt|+1)$. We let $\mathcal{I}_X$ denote the set of problem instances of $X \in$MIN PB; we let $\mathcal{S}(I)$ denote the set of all solutions for instance I. Suppose that $X_1, X_2 \in$MIN PB. An *S-reduction from X_1 to X_2 with amplification factor $a(n)$* consists of the following:

(i) A polynomial time computable function $r_1 : \mathcal{I}_{X_1} \to \mathcal{I}_{X_2}$.

(ii) A monotonically increasing positive function $a(n)$ such that for any $I \in \mathcal{I}(X_1)$, $|r_1(I)| \le |a(|I|)|$.

(iii) A polynomial-time computable function r_2 such that for any $I \in \mathcal{I}(X_1)$, r_2 sends $(r_1(I), S)$ to a solution of I, where S is a solution of $r_1(I)$.

(iv) A constant c such that for any $I \in \mathcal{I}_{X_1}$ and any solution S of $r_1(I)$, $R_{X_1}(I, r_2(r_1(I), S)) \leq cR_{X_2}(r_1(I), S)$.

If there is an S-reduction from X_1 to X_2 with amplification factor $a(n) = n^k$ and n^c is a lower bound on an approximation factor for X_1, then $n^{\frac{c}{k}}$ is a lower bound on an approximation factor for X_2.

We prove our theorem by giving an S-reduction, with amplification factor $a(n) = O(n^5)$ from Minimum Independent Dominating Set (MIDS), which is known not to have a polynomial-time algorithm with approximation factor $n^{1-\delta}$, for any $\delta > 0$. MIDS takes as input a graph $G = (V, E)$ and a constant $N > 0$ and asks if there is a "dominating set" $V' \subset V$ of size at most N (i.e., if there is a set of at most N vertices, no two of which are adjacent, such that every vertex $v \in V$ is adjacent to some member of V').

For an instance I of MIDS, with associated graph G, we construct a corresponding instance $I' = r_1(I)$ of NSP-SBSIMP. We define a new graph, G', as follows. There is a *vertex node* of G' corresponding to each node of G; there is an *edge node* of G' corresponding to each edge e of G, and we connect it with an edge (of G') to each of the two vertex nodes at the endpoints of e. Finally, G' has a special *control node*, D, which is joined by an edge to each vertex node and each edge node. We now embed G' in the plane so that no three edges intersect at one point and place a *crossing node* at each edge crossing (splitting in two each of the edges that cross there). The resulting planar graph, G'', is embedded in the plane so that the nodes are "far enough" apart. We place a rectangular *frame* (annulus) at each node and construct a narrow *tube* to surround each edge.

We then construct *vertex gadgets*, *edge gadgets*, *crossing gadgets*, and a *control gadget* within each frame centered at the corresponding node. These gadgets consist of a number of polygonal chains, which partially constitute the planar subdivision I'. Traveling through each tube is a *hinge-link sequence* as defined in [10], which also consists of a number of polygonal chains and connects the gadgets placed at the tube endpoints. Any hinge-link sequence has only one possible simplification so that in any simplification of I' each hinge-link sequence is either simplified or not simplified. Crossing gadgets are constructed so that the hinge-link sequences in the tubes corresponding to a crossing edge maintain the same parity.

Each vertex gadget contains an *enforcing vertex chain* of size $O(n^4)$, where n is the number of nodes in the instance I. An enforcing vertex chain simplifies to one segment and has no other possible simplification. Vertex gadgets having an enforcing vertex chain that is not simplified correspond to nodes that are in the independent dominating set.

Each vertex gadget, v, and each edge gadget, e, has a distinguished simplification, $S_D(v)$ or $S_D(e)$. A vertex gadget is simplified by S_D if and only if at least one enforcing vertex chain in the neighborhood of corresponding nodes, including itself, is simplified. Thus, simplification of a vertex gadget, v, by $S_D(v)$ corresponds to domination. Also an edge gadget, e, is simplified by $S_D(e)$ if and only if at most one of the enforcing vertex chains at the nodes incident on the corresponding edge are simplified. Thus, simplification by $S_D(e)$ corresponds to independence.

The control gadget has one possible simplification and is simplified if and only if all vertex and edge gadgets are simplified by S_D. In this way we obtain that if any enforcing vertex chain is simplified then all non-simplified enforcing vertex chains correspond to an independent dominating set and that if no vertex chain is simplified then either there is no independent dominating set or G is a trivial graph consisting of one node. The formidable details of all of these gadgets and the proofs are contained in [10]. $\square$

3. THE SIMPLE DETOURS HEURISTIC

We now describe our Simple Detours (SD) heuristic. The main idea is to use intersections that occur in an unconstrained simplification to "suggest" problem areas where further work will be done to prevent crossings. At each intersection, we define a graph (the "detour graph") of ϵ-feasible segments that can be used in a path that replaces one of the segments that is involved in the intersection. A search in this graph (hopefully) yields a simplest detour – a minimum-link detour around the intersection. (If it fails, then we have an additional heuristic to modify the chains locally, described below.) Of course, by taking detours to untangle one intersection, we may end up creating some new intersections; thus, the algorithm must continue checking for intersections after each iteration of computing detours, until finally no intersections are present. Thus, our algorithm yields a sequence of subdivision simplifications, $\mathcal{S}' = \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$, in k main iterations, with $\mathcal{S}_{i+1}$ being obtained from $\mathcal{S}_i$ by the ith iteration of the simple detours step, designed to do local elimination of intersections in $\mathcal{S}_i$. The final subdivision $\mathcal{S}_k$ is guaranteed to consist of ϵ-feasible segments and have no intersections.

In more detail, the main steps of the method are as follows:

(1) We compute an ϵ-feasible approximation to the input subdivision, $\mathcal{S}$, using any standard line simplification technique that does not introduce new vertices and that uses our notion of ϵ-feasibility. (Our experiments use the standard implementation of the Douglas-Peucker algorithm.)

(2) We compute all pairs of intersecting segments in the simplification, placing them in a list $\mathcal{I}$. In our implementation, we use a regular grid heuristic to speed up intersection computations. (We found this to be superior to a prior implementation based on a plane sweep algorithm; see [1, 2].)

(3) For each intersection in $\mathcal{I}$, we consider the two segments s and s' that give rise to the intersection.

   (a). One of them (call it s) is declared to be the *detour segment*; it is the one that we attempt to replace with a minimum-link detour that avoids the other segment. Specifically, we select s to be the segment that has an odd number of crossings with the spanning (original) chain $C(s')$ for which s' is an ϵ-feasible shortcut. (Thus, s is not an original segment of $\mathcal{S}$, since $s \cap C(s') \neq \emptyset$.) One can prove that exactly one of $\{s, s'\}$ has an odd number number of crossings with the other's spanning chain. We have found that our selection of detour segment s works best in practice.

(b). We construct the *detour graph*, $G(s)$, corresponding to s. The vertices of $G(s)$ are the vertices of the spanning chain $C(s)$, and two vertices are joined by an edge in $G(s)$ if and only if the corresponding line segment is ϵ-feasible and does not intersect s'. This is done in time $O(|C(s)|^2)$, using the algorithm of Chan and Chin as described in [6] ("*Compute-Allowed-Shortcuts*(C, ϵ)"). See Figure 3.

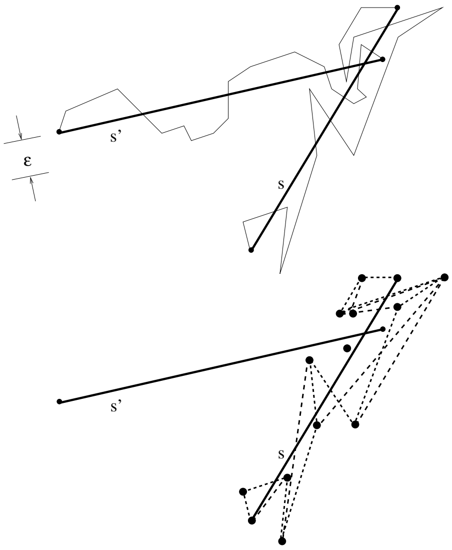


Figure 3: Left: Shortcut segments s and s' intersect; the original chains $C(s)$ and $C(s')$ are shown as well (thin solid). Right: The detour graph $G(s)$ is shown with thick dashed edges and fat black dots as nodes.

(c). We search the detour graph $G(s)$ for a shortest (minimum-link) path connecting the endpoints of s. If we succeed in finding such a detour path, we add the corresponding edges to the subdivision.

It is possible that there is no detour path (in fact, Figure 4 shows a case in which neither $G(s)$ nor $G(s')$ has a detour path). If we fail to find a detour path, then we perform the following splitting process to search for a pair of noncrossing ϵ-feasible paths to replace the pair $(C(s), C(s'))$: We select a random vertex v on $C(s)$ and v' on $C(s')$. We replace $s = u_1u_2$ (resp., $s' = u_1'u_2'$) with the two segments u_1v and vu_2 (resp., $u_1'v'$ and $v'u_2'$), if they are both ϵ-feasible. If either or both segments are not ϵ-feasible, we continue the splitting process recursively on the corresponding subchains. (The technique is similar to the Douglas-Peucker algorithm except in our choice of splitting point.) In this way, we are guaranteed to find ϵ-feasible replacement paths for s and s', although we do not explicitly require that they be disjoint. (If there are intersections, they will be discovered and resolved in the next iteration.)

(d). Go to (2) to begin the next major iteration of the algorithm.

Since at least one vertex is added to the subdivision during each iteration, and no vertex is ever removed, it is clear that the algorithm converges. Our algorithm can be implemented to run in time proportional to

$$\sum_{i=1}^{k} \left[|\mathcal{I}_i| + n \log n + \sum_{(s,s') \in \mathcal{I}_i} |C(s)|^2 \right],$$

where $\mathcal{I}_i$ is the set of intersections found at iteration i. The total number of iterations, k, can be $\Omega(n)$ in the worst case (see Figure 5); however, in practice, we find that k tends to be a small constant (it never exceeded 40 in our experiments). The term $|\mathcal{I}_i| + n \log n$ assumes the use of an optimal segment intersection algorithm; in practice, we use our simple grid-based bucketing approach, which can be expected to perform close to optimally. Also, in order to perform the intersection computation more efficiently from iteration to iteration, we keep track of those subchains that are modified in the previous iteration and, in the current iteration, check for intersection only with those subchains that are marked as modified. While the worst-case running time of the algorithm is $O(n^3)$, our experimental results show that it runs reasonably fast in practice.

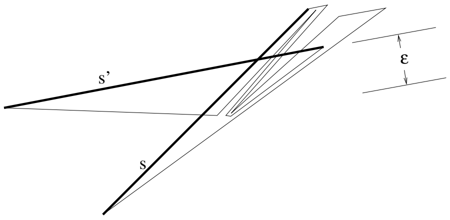


Figure 4: Neither $G(s)$ nor $G(s')$ contains a (ϵ-feasible) detour path.

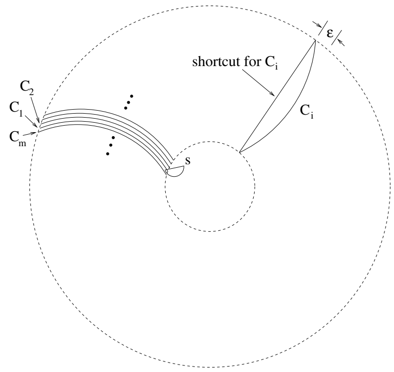


Figure 5: A family of m "nested" chains,, forming a "fan blade", for which the number of iterations is linear in the input size n. The initial simplifications are all straight segments. The intersection with the simplified short chain causes a detour of C_1 to be computed that intersects the next chain (C_2, going clockwise), whose detour intersects the next one, etc, until the first one is intersected again by the mth one, and a new detour is required for C_1. Finally, the SD method would end up spending $\Omega(n)$ iterations to add back all vertices on the original nested chains, while an optimal solution would be to take an $O(1)$-size detour only for the short segment.

4. EXTENDED SD HEURISTIC

The Simple Detours heuristic is a method for obtaining a simplified subdivision that is simple – it has no crossing segments. However, this method does not preserve the homotopy type of the input subdivision and does not attempt to handle point features. For example, in Figure 1, the SD heuristic would permit the simplification shown, in which the smallest face is contained in a different face after simplification. In order to address the more general problem of preserving homotopy type, we have developed an extension to the basic SD heuristic, which we call the Extended Simple Detours heuristic (ESD heuristic).

We now assume that we are given a set of input points, $\mathcal{P}$, along with the input subdivision $\mathcal{S}$. The initial simplification is done so that the simplification obtained is homotopic to the original subdivision with respect to the set of points $\mathcal{P}$ and the endpoints of chains of $\mathcal{S}$. Thus, if C' represents the simplified version of an input chain C, then C can be continuously deformed into C' without passing through any points of $\mathcal{P}$ and without passing through any endpoints of chains (other than C). We let $\mathcal{Q}$ denote the union of $\mathcal{P}$ and the endpoints of chains of $\mathcal{S}$.

The ESD method is essentially the same as the SD method except that we impose an additional constraint on the edges that make up the detour graph that we construct when we want to resolve an intersection $s \cap s'$. In particular, we consider an ϵ-feasible shortcut edge e to be an edge of $G(s)$ only if e does not intersect s' (as before) *and* e is homotopically equivalent, with respect to $\mathcal{Q}$, to the chain $C(e)$ that it replaces. In order to test if $C(e)$ is homotopically equivalent to e, we could apply the techniques of [13] to compute a shortest path homotopically equivalent to $C(e)$ within a triangulation of the points $\mathcal{Q}$; if the shortest path is a straight segment (e), then they are homotopically equivalent, and otherwise they are not. In order to avoid computing and storing a triangulation, and for simplicity of implementation, we have opted to code a simpler algorithm that makes a stronger restriction on e when we test homotopy feasibility. Our condition is "stronger" in the sense that it is a *sufficient* (but not necessary) condition for homotopy feasibility. In particular, we utilize a regular grid partition (as we do for computing intersections among segments) and we demand that e be homotopically equivalent to $C(e)$ "with respect to the grid", in the following sense: for each (rectangular) grid element B, (a). e and $C(e)$ must intersect the sides of B in the same order; (b). $e \cap B$ must be homotopically equivalent to each connected component of $C(e) \cap B$. Condition (b) is checked in a straightforward manner, as we store explicitly with each portion of a chain within a grid box B the set of points that lie within distance ϵ on one side of it; these points can be checked directly against e.

5. EXPERIMENTS

We have conducted implemented the heuristics (SD and ESD) we have proposed and have conducted an extensive set of experiments to test their efficiency and effectiveness. Our tests were done on a Silicon Graphics SGI with one 150 Mhz IP22 processor and 64MB of memory.

Data Sets. Our data sets included USGS digital line graph (DLG) data sets, containing from 12,317 to 147,609 vertices, a census data setWe thank Dr. Robert Freimer of Caliper Corporation for providing the census data., containing 279,989 vertices, and randomly generated simple polygons (using the RPG system of Auer and Held [3]), containing from 2001 to 100,001 vertices. The USGS data consists of hypsography, hydrography, transportation, and boundary data. We have run both the SD and ESD methods on all types and have found that the hypsography data is by far the most difficult to handle. In general the hypsography data sets are the largest data sets of the four types and contain relatively few chains. Also, many more intersections are created in the initial simplification on hypsography data than on the other three data set types. Point features used in the ESD method were randomly generated using a uniform distribution. Between 100 and 100,000 point features were used in individual runs. Data sets were all scaled to an 800×800 area. We considered values of ϵ ranging from $\epsilon = 2$ to $\epsilon > 800\sqrt{2}$.

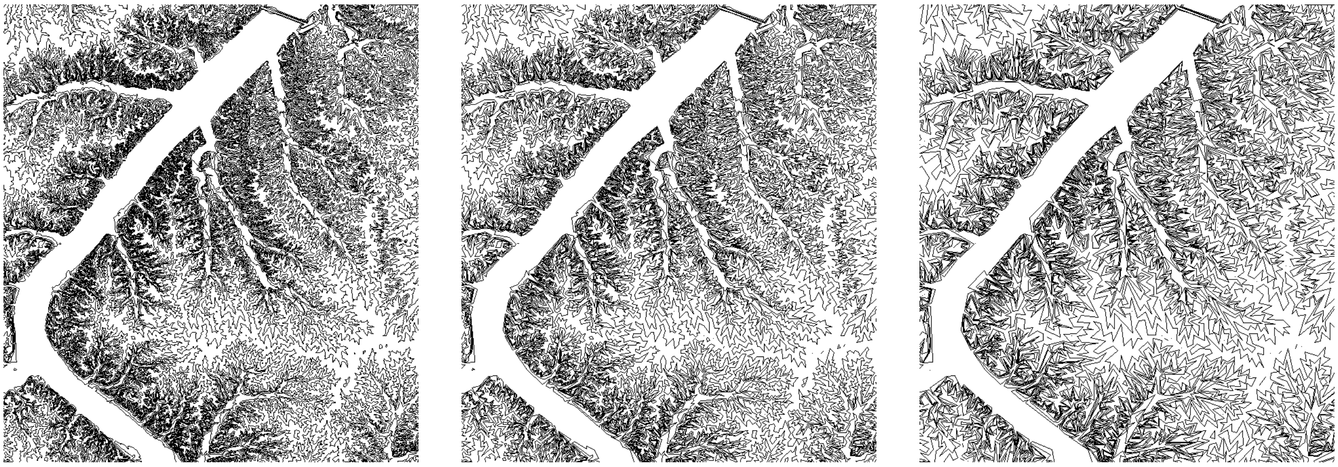
Details of the Experimental Setup. For the rectangular grid to accelerate intersection computation, we divided the bounding box (of dimensions Δx-by-Δy) of the data into I_x equal-sized intervals, where $I_x = (1.25)\beta_x\sqrt{n+3n'}$, n is the original number of vertices, n' is the number of vertices after Douglas-Peucker simplification, and $\beta_x = \Delta x/\Delta y$ (rounded to 0.1 or to 10 if this ratio falls outside the interval (0.1,10)). The number, I_y, of y-intervals is defined similarly. Before settling on this choice of grid parameters, we experimented with various choices of grid size.

We have found that it is often useful to conduct simplifications in *stages*: In the ith stage, we compute an approximation with error bound ϵ_i, starting with input given by the simplified subdivision computed in the $(i-1)$st stage. Since the errors can compound from stage to stage, the total error at the end of the ith stage is given by $\epsilon_1 + \cdots + \epsilon_i$. After some initial experimentation, we determined that it is rarely beneficial to use more than two stages, but it is often advantageous (in terms of running time) to use two stages. In many of the results reported here, two stages were used (in which case we give not only ϵ ($= \epsilon_1 + \epsilon_2$), but also ϵ_1). Running times in our tables are total times for all stages.

Method for Comparison: Constrained Decimation. We have also implemented a simple constrained decimation algorithm in order to have a basis for comparison with our SD heuristics. As opposed to the SD heuristic, the decimation method avoids intersections during the initial simplification. In this method, we start with an input subdivision in which it is assumed that chains are ordered and the vertices in each chain are ordered. We iteratively attempt to remove vertices from the chains while obeying the constraint that the shortcut segment resulting from the removal of a vertex is ϵ-feasible *and* that it does not create any intersection with other segments of the subdivision.

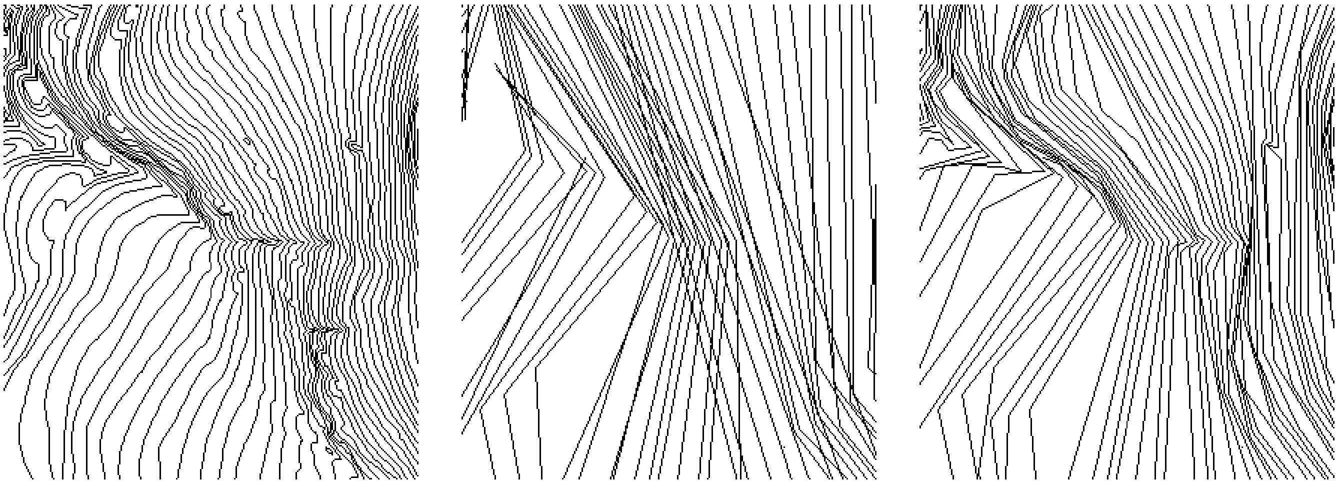
Experimental Results. Pictorial examples of results obtained with our software are shown in the images of Figure 6 and Figure 7. In our tables of results, we use "DP" to stand for the Douglas Peucker algorithm and "SD" to stand for the Simple Detours algorithm. The *reduction ratio* ("Red. Ratio") is the ratio of the number of vertices in the simplification to the number of vertices in the original subdivision. We show reduction ratios both for the DP algorithm alone and for the SD heuristic algorithm. We also show the number of intersections that occur in the DP simplification (before the SD heuristic is run).

Table 1 shows running times for 2-stage simplification on



(a). Original: 123,871 vertices. (b). Simplification: 1 stage, $\epsilon = 2$. (c). Simplification: 2 stages, $\epsilon = 5$.

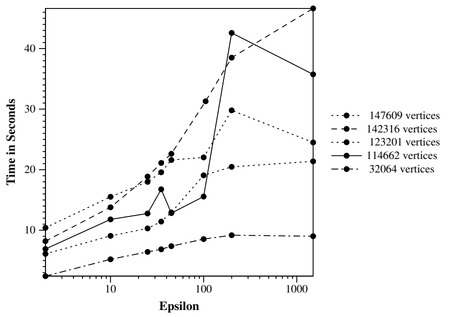
Figure 6: Example of a hypsography data set and two simplifications using the SD heuristic. For (b), there were 2664 intersections after DP, 25,051 vertices in the final simplification, and the total cpu time was 9.71 seconds. For (c), there were 4603 intersections after DP, 14,902 vertices in the final simplification, and the total cpu time was 16.94 seconds (11.97 seconds of which was for the first stage, most of which was running DP). The first stage took 4 iterations, and the second stage took 7 iterations of the SD algorithm.



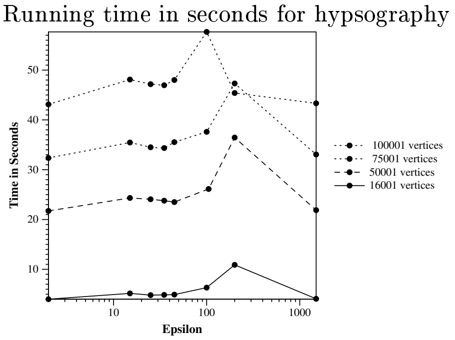
(a). Original: 32,064 vertices. (b). Douglas-Peucker simplification. (c). SD heuristic simplification.

Figure 7: Zoomed in example of a USGS hypsography data set and its simplification using only the Douglas-Peucker, as well as the SD heuristic. The total cpu time is 7.37 seconds, obtained in two stages with $\epsilon_1 = 2$, $\epsilon_2 = 43$. The second stage of the simplification took $k = 17$ iterations. The final subdivision has 3796 vertices.

USGS hypsography data sets, for various values of $\epsilon = \epsilon_1 + \epsilon_2$ (always with $\epsilon_1 = 2$). This data is also plotted in Figure 8(a). A similar plot (Figure 8(b)) shows how the running times vary with ϵ for the random polygons data sets. Figure 9 shows how the total running time breaks down among the various phases of the algorithm (computing the original DP simplification, finding intersections, computing detour graphs and searching them, and other operations), for the 147,609-vertex hypsography data set. Breakdowns of running time and the reduction ratios are reported in Tables 3 and 4 for the census data and random polygon data, respectively. We note that more variation occurs in real world data than in the artificially generated random polygon data. However, the running times are generally higher on the random polygon data. This is due largely to the time spent in computing intersections, which, recall, is done using a simple grid-based bucketing technique. The random polygon data tends to have many long edges that extend across the data set, making the bucketing technique much less effective in pruning the intersection search.



(a). Running time in seconds for hypsography data.



(b). Running time in seconds for random polygons.

Figure 8: SD Method : 2 stages, $\epsilon_1 = 2.0$.

A comparison of the SD heuristic and the constrained decimation is shown in Table 2 for random polygon data. It is seen that the SD heuristic gives dramatically better reduction ratios in a fraction of the time required by the constrained decimation algorithm.

A comparison of 1-stage and 2-stage simplification with the SD heuristic is shown in Tables 5 and 6. We see that the 2-stage process gives a reduction ratio that is somewhat worse, while giving running times that are definitely better. This particular data set gives one of the worst comparisons of reduction ratios between 1-stage and 2-stage; in general, the reduction ratios are comparable between the 1- and 2-stage processes, while the 2-stage process is substantially faster.

For the Extended Simple Detours (ESD) heuristic, we

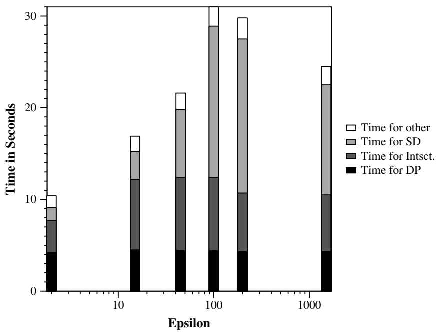


Figure 9: SD Method : A breakdown of times for 2-stage simplification with $\epsilon_1 = 2.0$ for hypsography data set with 147,609 original vertices.

show running times, as a function of the number of point features (on a log scale), in Figures 11(a) and 11(b) for hypsography data and random polygons. We show a breakdown of the running time in Figure 10 for the 147,609-vertex hypsography data set.

Note that since there is an additional constraint when applying the ESD heuristic, we do not obtain as aggressive a reduction in the vertex count as we do with the SD heuristic. Figure 12 shows the ratio of the number or vertices obtained via ESD for no point features to the number of vertices obtained via SD for two HPF data sets. We see that there is a substantial difference in behavior between the two data sets. When ϵ is small this ratio is near one in both cases. This is generally true when ϵ is small relative to grid size. However, as ϵ increases, this factor can grow quite large if individual chains in the subdivision are highly non-monotonic. In the case that individual chains can be decomposed into a small number of monotone chains, the grid constraint imposed by our method of testing homotopy feasibility has less effect.

Table 7 shows a breakdown of times for a 2-stage ESD method with $\epsilon_1 = 2$ on a hypsography data set containing 32,064 vertices with 100 point features.

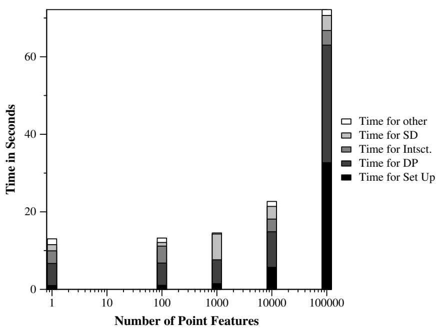


Figure 10: ESD Method : Breakdown for 1-stage simplification with $\epsilon = 2$ on a hypsography data set containing 147609 vertices and 1063 chains.

# of Verts.	ϵ 2	15	25	35	45	100	200	∞
12317	0.610	0.920	0.920	1.140	0.800	0.880	0.750	1.840
32064	2.420	5.68	6.420	6.850	7.370	8.540	9.180	9.010
41852	1.920	2.36	2.47	2.64	2.84	6.56	5.31	4.05
48125	2.200	3.680	4.880	5.940	4.840	8.430	8.930	3.130
69191	2.950	4.15	4.6	5.940	5.75	7.01	5.78	4.91
76145	3.910	6.18	6.90	7.63	7.69	6.53	6.50	6.48
86803	5.000	7.10	7.48	8.800	9.120	9.830	8.130	9.390
88787	4.060	5.44	5.76	5.56	5.84	5.74	6.59	6.55
113142	5.890	9.33	10.54	10.67	11.39	20.93	31.18	27.72
114662	6.900	12.775	12.77	16.760	12.830	15.560	42.590	35.740
123201	6.10	9.490	10.300	11.430	12.940	19.070	20.470	21.380
123871	11.970	19.830	22.920	26.010	29.910	60.430	62.45	61.92
142316	8.200	16.290	18.860	21.110	22.620	31.300	38.500	46.630
147609	10.410	16.910	16.290	19.570	21.600	22.030	29.800	24.500

Table 1: SD Method : Running time in seconds for hypsography data sets : 2-stage simplification with $\epsilon_1 = 2$.

# of Vertices	SD Time in Seconds	SD Reduction Ratio	Dec. Time in Seconds	Dec. Reduction Ratio
2000	0.150	0.0290	61.480	0.3103
3000	0.140	0.0290	139.710	0.3116
7000	0.630	0.0116	749.460	0.2950
16000	1.970	0.1312	3524.460	0.4546

Table 2: 1-stage Simple Detours vs. In Order Decimation on Random Polygons with $\epsilon = 100$.

ϵ	Time Seconds	Time DP	Time Intsct	Time SD	Red. (DP) Ratio	Red. Ratio	# of Intscts.
2	18.910	2.160	3.730	10.710	0.0758	0.0765	297
12	20.42	2.31	4.87	10.75	0.0677	0.0679	107
22	20.39	2.30	4.85	10.74	0.0673	0.0675	92
32	20.39	2.30	4.84	10.75	0.0673	0.0675	93
42	20.39	2.30	4.84	10.75	0.0673	0.0675	93
102	20.39	2.30	4.85	10.75	0.0673	0.0675	90
∞	20.39	2.30	4.85	10.75	0.0673	0.0675	90

Table 3: SD Method : 2-stage simplification ($\epsilon_1 = 2$) on a census data set with 279,989 vertices, 9419 chains.

ϵ	Time Seconds	Time DP	Time Intsct	Time SD	Red. (DP) Ratio	Red. Ratio	# of Intscts.
2	43.120	6.110	34.990	0.960	0.2060	0.2119	951
15	48.100	7.16	37.87	1.74	0.0217	0.0235	253
25	47.17	7.04	36.35	2.49	0.0097	0.0104	162
35	46.96	6.96	35.92	2.74	0.0053	0.0056	77
45	48.01	6.92	35.95	3.72	0.0036	0.0040	71
100	57.67	6.75	35.38	14.31	0.0010	0.0011	13
200	45.4	6.52	35.22	2.47	0.0003	0.0003	4
∞	43.33	6.2	34.990	0.960	0.00002	0.00002	0

Table 4: SD Method : 2-stage simplification ($\epsilon_1 = 2$) on a random polygon containing 100,001 vertices.

ϵ	Time Seconds	Time DP	Time Intsct	Time SD	Red. (DP) Ratio	Red. Ratio	# of Intscts.
2	2.420	0.550	1.410	0.190	0.2027	0.2075	250
15	12.400	0.290	4.350	7.150	0.0808	0.1207	1364
25	19.250	0.260	6.140	11.910	0.0737	0.1223	1549
35	25.220	0.250	6.800	17.100	0.0695	0.1264	1749
45	28.52	3.230	10.32	12.52	0.0204	0.0475	2238
110	30.91	3.19	10.6	14.66	0.0165	0.0485	1626
210	34.8	3.17	11.24	17.76	0.0157	0.0460	901
∞	36.54	3.150	9.62	21.17	0.0154	0.0403	1067

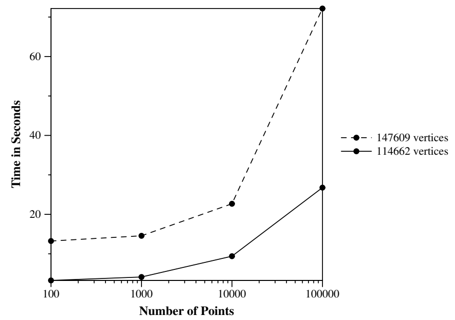
Table 5: SD Method : 1-stage simplification on a hypsography data set (32,064 vertices, 941 chains).

ϵ	Time Seconds	Time DP	Time Intsct	Time SD	Red. (DP) Ratio	Red. Ratio	# of Intscts.
2	2.420	0.550	1.410	0.190	0.2027	0.2075	250
15	5.200	6.20	3.69	0.53	0.0972	0.1249	1039
25	6.420	6.10	4.54	0.88	0.0750	0.1148	1456
35	6.850	0.600	4.62	1.18	0.0701	0.1160	1758
45	7.370	0.610	4.93	1.42	0.0675	0.1183	1899
100	8.540	0.590	5.17	2.28	0.0610	0.1229	2067
200	9.180	0.580	5.55	2.62	0.0591	0.1213	1670
∞	9.010	0.580	5.37	6.23	0.0587	0.1217	1523

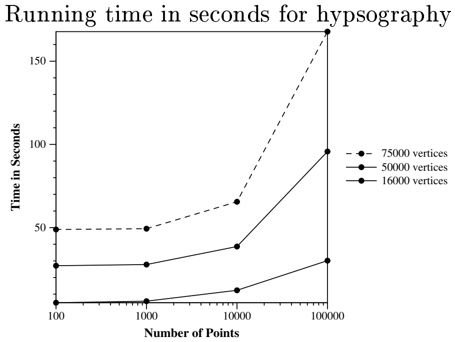
Table 6: SD Method : 2-stage simplification ($\epsilon_1 = 2$) on a hypsography data set (32,064 vertices, 941 chains).

ϵ	Time Seconds	Time SetUp	Time DP	Time Intsct	Time SD	Red. (DP) Ratio	Red. Ratio	# of Intscts.
2	3.270	0.350	1.170	1.320	0.180	0.2640	0.2687	240
10	6.93	0.540	1.72	3.62	0.63	0.1804	0.1777	1081
25	7.72	0.54	1.84	3.94	1.00	0.1848	0.1499	1526
35	7.71	0.53	1.92	3.87	1.61	0.1857	0.1509	1582
∞	8.05	1.25	2.01	3.96	1.15	0.1898	0.1491	1864

Table 7: ESD Method : 2-stage simplification ($\epsilon_1 = 2$) with 100 point features for hypsography data (32,064 vertices).



(a). Running time in seconds for hypsography data.



(b). Running time in seconds for random polygons.

Figure 11: ESD Method : One stage, $\epsilon = 2.0$.

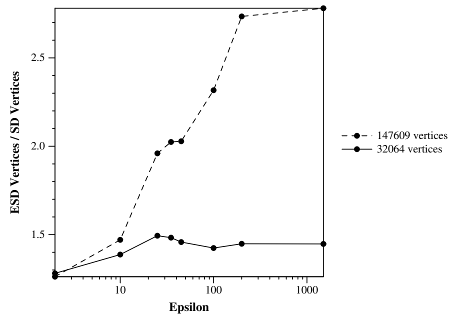


Figure 12: ESD Method : Ratio of the number of vertices obtained via the ESD method for no point features to the number of vertices obtained via the SD method, for hypsography data.

6. ACKNOWLEDGMENTS

This work was conducted while R. Estkowski was a PhD student at the University at Stony Brook, supported by grants from the HRL Laboratories, ISX Corporation, and the National Science Foundation (CCR-9732221). J. Mitchell is supported in part by HRL Laboratories, the National Science Foundation (CCR-9732221), NASA Ames Research Center, Northrop-Grumman Corporation, Sandia National Labs, Seagull Technology, and Sun Microsystems.

7. REFERENCES

[1] D. S. Andrews and J. Snoeyink. Geometry in GIS is not combinatorial: Segment intersection for polygon overlay. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C24–C25, 1995.

[2] D. S. Andrews, J. Snoeyink, J. Boritz, T. Chan, G. Denham, J. Harrison, and C. Zhu. Further comparison of algorithms for geometric intersection problems. In *6th International Symposium on Spatial Data Handling*, pages 709–724, 1994.

[3] T. Auer and M. Held. Heuristics for the generation of random polygons. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 38–43, 1996.

[4] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *Internat. J. Comput. Geom. Appl.*, 6:59–77, 1996.

[5] R. G. Cromley. A vertex substitution approach to numerical line simplification. In *Proc. 3rd Internat. Sympos. Spatial Data Handling*, pages 57–64, 1988.

[6] M. de Berg, M. van Kreveld, and S. Schirra. A new approach to subdivision simplification. In *Proc. of Auto-Carto 12*, pages 79–88, 1995.

[7] M. de Berg, M. van Kreveld, and S. Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and GIS*, 25:243–257, 1998.

[8] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, Dec. 1973.

[9] R. Estkowski. No steiner point subdivision simplification is NP-complete. In *Proc. 10th Canad. Conf. Comput. Geom.*, 1998.

[10] R. Estkowski. *Algorithms and Complexity Results for Three Problems in Applied Computational Geometry: Subdivision Simplification, Stripification of Triangulations, and Unions of Jordan Regions.* PhD thesis, Dept. Applied Mathematics and Statistics, University at Stony Brook, 2000.

[11] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Internat. J. Comput. Geom. Appl.*, 3(4):383–415, Dec. 1993.

[12] J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line simplification algorithm. In *Proc. 5th Internat. Sympos. Spatial Data Handling*, pages 134–143, 1992.

[13] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.*, 4:63–98, 1994.

[14] J. Hershberger and J. Snoeyink. An $O(n \log n)$ implementation of the Douglas-Peucker algorithm for line simplification. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 383–384, 1994.

[15] D. Hochbaum, editor. *Approximation Problems for NP-Complete Problems.* PWS Publishing Company, Boston, MA, 1997.

[16] H. Imai and M. Iri. Polygonal approximations of a curve-formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology*, pages 71–86. North-Holland, Amsterdam, Netherlands, 1988.

[17] C. Jones, G. Bundy, and J. Ware. Map generalization with a triangulated data structure. *Cartography and Geographic Information Systems*, 22:317–331, 1995.

[18] A. Mantler and J. Snoeyink. Safe sets for line simplification. In *Abstracts of the Tenth Annual Fall Workshop on Computational Geometry*, October 2000.

[19] A. Melkman and J. O'Rourke. On polygonal chain approximation. In G. T. Toussaint, editor, *Computational Morphology*, pages 87–95. North-Holland, Amsterdam, Netherlands, 1988.

[20] N. Mustafa, E. Koutsofias, S. Krishnan, and S. Venkatasubramanian. Hardware assisted view-dependent map simplification. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, page xxx, June 2001.

[21] F. Zhan and D. M. Mark. Conflict resolution in map generalization: a cognitive study. In *Proc. Auto-Carto*, volume 13, pages 406–413, 1993.