The min-# problem, a hybrid error criterion for near-linear time performance

Lilian Buzer[*†]

Abstract

For a given polygonal chain and a family of tolerance regions, we study the **min-# problem**, which consists in finding an approximate and ordered subchain with a minimum number of vertices. Our algorithm computes an approximation that retains the shape of the original chain. Moreover, our method reaches a near-linear time complexity and its implementation is based on classical functions. To our knowledge, this is the first algorithm providing all these advantages.

1 Introduction

A polygonal chain, or a path P in the plane, with n vertices, is defined as an ordered list of vertices $(p_1, p_2, \ldots, p_n)$ such that any two consecutive vertices p_i, p_{i+1} are connected by a line segment. The *polygonal chain approximation problem* consists in approximating a polygonal chain P by another one whose vertices must be an ordered subset of the vertices of P. This problem arises in many applications like including geographic information systems (GIS), cartography, computer graphics and data compression. Two main problems have emerged. The *min-ϵ* problem consists in finding an approximation with at most k vertices and with the smallest approximation error. In this paper, we focus on the second problem, namely the **min-# problem**: given a polygonal chain and a family of tolerance regions, find an approximate chain with a minimum number of vertices. We use the notation S_i^j to denote the subchain $(p_i, \ldots, p_j)$ and the notation $p.x$ to denote the abscissa of the vertex p.

2 Approximation criteria

The algorithm we have designed retains the shape of the initial chain, builds an approximation with a minimum number of segments and reaches a near-linear time complexity. We present all the existing approximation criteria and we show that none of them, as far as we know, provides these three advantages at the same time.

*A2SI Laboratory, ESIEE, 2 bd Blaise Pascal, Cité Descartes, BP 99, 93162 Noisy-Le-Grand Cedex, France, buzerl@esiee.fr

†Unité Mixte CNRS-UMLV-ESIEE, UMR 8049

2.1 History

The min-# problem has been studied extensively during the last two decades. Imai and Iri in [12] introduced a unified approach that has since been employed by most of the algorithms devoted to this problem. They formulated the problem in terms of graph theory. Let $\epsilon > 0$ denote a given error bound and let $\mathrm{TR}_\epsilon(uv)$ denote the tolerance region associated with the segment uv. They defined an unweighted graph $G_\epsilon(P) = (V, E_\epsilon)$, where $V = \{p_1, \ldots, p_n\}$ and $E_\epsilon = \{\ p_ip_j \mid (p_i, p_{i+1}, \ldots, p_j) \subset \mathrm{TR}_\epsilon(p_ip_j)\}$. By computing the shortest path in G_ϵ (an unweighted directed acyclic graph), the min-# problem is solved in time linear in the number of G_ϵ edges. We now describe the different tolerance regions associated with two vertices p_i and p_j: • *Segment distance*, the maximum distance from the segment p_ip_j • *Infinite beam* or *parallel-strip*, the maximum distance from the line p_ip_j • *Minimum height*, a rectangle such that p_i and p_j lie on two opposite sides which are orthogonal to the segment p_ip_j and whose length is less than ϵ • *Minimum width*, a rectangle whose width is less than ϵ and which contains the segment p_ip_j. The segment distance criterion under the L_∞ metric is called the *uniform measure criterion* and that under the L_2 metric the *tolerance zone criterion.*

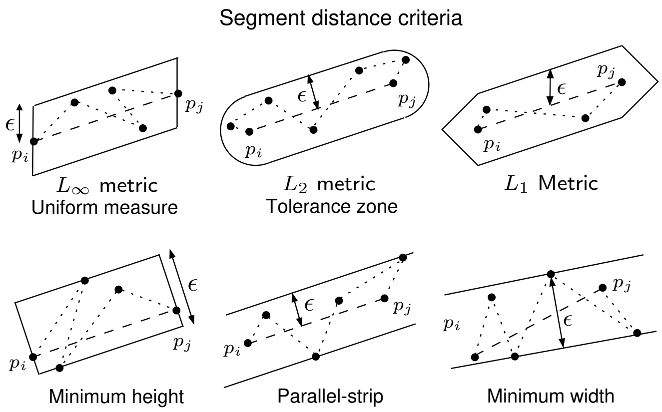

Figure 1: The different error criteria.

2.2 Previous work

The minimum height, the segment distance and the parallel strip criteria [12, 17, 15, 10, 5, 6] under various metrics lead at least to quadratic algorithms where the bottleneck is the construction of G_ϵ. Using a graph compression technique, Agarwal and Varadarajan [2] presented an $f(\gamma).O(n^{\frac{4}{3}+\gamma})$ time method for the uniform measure criterion and the segment distance criterion under the L_1 metric. One of the oldest and most popular algorithms is the Douglas-Peucker heuristic [9]. Hershberger and Snoeyink [11] showed that this algorithm can be implemented in $O(n \log^* n)$ time. This method achieves efficiency but fails to build an approximation with a minimum number of vertices. Other recent approaches, a breadth-first traversal of the graph [7], a query-based technique [8] (for the infinite beam criterion) and an error measure based on the Fréchet distance [1] achieve near-linear time performance but only under certain assumptions. The minimum width criterion allows a greedy method to be used and thus Imai and Iri in [12] proposed an $O(n \log n)$ time algorithm. Nevertheless, they describe the approximations under this criterion as peculiar (under the minimum height criterion as well). The intrinsic properties of an error criterion influence the behavior of the algorithm. Certain properties may be excessive and may increase the number of points unnecessarily. Conversely, a relaxed criterion may produce too coarse a simplification. In the following section, we tackle this question in order to set up a criterion which is an appropriate trade-off.

2.3 Problems inherent to the error criteria

The main problem is to control the approximation in the neighborhood of p_i and p_j (see Fig. 1). By allowing the subchain to go beyond the two endpoints, we may lose pieces of the original chain (see Fig. 2.a). Thus, neither the parallel strip nor the minimum width criteria can be used to retain the shape of the original chain. Moreover, even though the uniform measure and the minimum height criteria prevent the subchain from locally passing beyond one endpoint, they sometimes fail to simplify in certain configurations (a zig-zag in a bend, see Fig. 2.b) and they produce a useless segment. Consequently, only the segment distance criterion with L_1 and L_2 metrics can be used to avoid these inconsistencies.

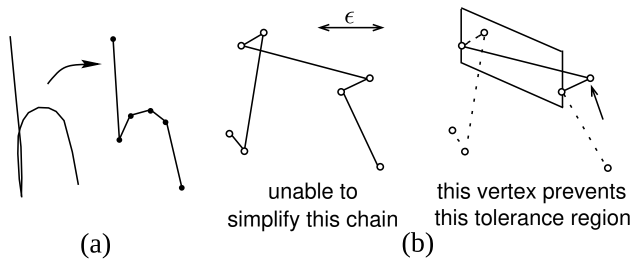

Figure 2: The different inconsistencies.

2.4 Introducing a hybrid criterion

Our hybrid criterion is a development of the previous ones. The tolerance areas we use correspond to particular parallelograms divided into two families. The first and second families correspond to parallelograms which have respectively two vertical and horizontal sides, called the borders, of length ϵ and such that the slope of the two other sides lies between $[-\frac{\pi}{2}, +\frac{\pi}{2}]$, respectively, relative to the horizontal and vertical axes (Fig. 3). Moreover, each of the two endpoints p_i and p_j must lie within distance $\epsilon/2$ of a different border.

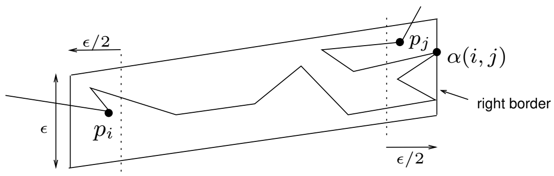

Figure 3: The hybrid criterion.

3 Algorithm design

3.1 The guide

To obtain a subquadratic complexity, we must bypass the construction of G_ϵ. In [2], a graph compression technique is used to achieve an $O(n^{\frac{4}{3}+\gamma})$ time complexity. Nevertheless, to obtain a near-linear time complexity, we must combine the shortest path computation with the graph construction at the same time. Therefore, we process only useful edges. Thus in [7, 8], a breadth first traversal (BFT) achieves better performance under certain assumptions. Nevertheless, using a BFT approach we may visit the same node in G_ϵ several times. To avoid this problem, we set up a *Guide* which can determine the next unvisited node from a given node. Thus, the following assertion holds:

Theorem 1 *During the BFT, if a* Guide *detects the next unvisited node in G_ϵ in $\theta(n)$ time, then the min-# problem can be solved in $O(n.\theta(n))$ time.*

3.2 The monotonicity tree

We now focus on a subproblem where we develop a convenient data structure that we reuse in our algorithm. Consider a subchain S_i^j and the configuration displayed in Figure 3. The rightmost vertex with maximum index is denoted $\alpha(i,j)$. Finding this special point for each couple (i,j) is an expensive operation because the number of couples can be quadratic. Let us consider the reverse problem. For a given index i, we prefer to detect the *candidate points* $\alpha(i,.)$ and then find the corresponding vertices p_j. We define the *candidate points* of a vertex p_i to be the points p_k such that $k > i$ and that p_k is the rightmost point of the subchain S_i^j. A natural order of these candidate points emerges if we sort them by increasing indices or similarly by increasing abscissae. Thus, we can create a rooted tree where each node is associated with a vertex p_i. The parent-child relationship in this tree is as follows: the node linked to the vertex p_k is the parent of the node linked to the vertex p_i when p_k is the candidate point of p_i with the smallest abscissa. Therefore, when we successively traverse the ascendants of the node linked to the vertex p_i, we obtain all the existing points $\alpha(i, .)$ by increasing abscissa (see Fig. 4). We call such a tree the *increasing abscissae tree* (IAT). It can be preprocessed in linear time after a sorting step. Similarly, we define the *decreasing abscissae tree* (DAT).

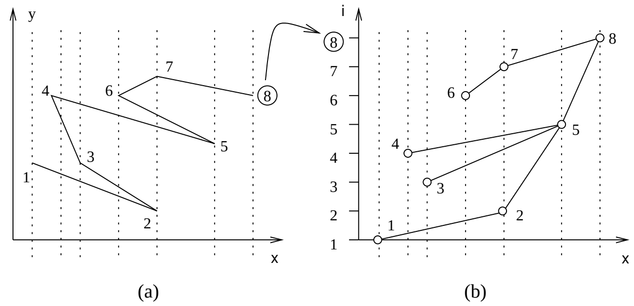

Figure 4: Construction of the increasing abscissae tree.

3.3 Sketching the Guide

With respect to the associated tolerance areas, four different cases appear depending on the family of the parallelograms and on the direction of traversal, for example from right to left or left to right. During the BFT in G_ϵ, we process each case from the current vertex separately and then combine the results. This approach is equivalent to working with four different Guides. For convenience, only the configuration presented in Fig. 3 is considered. From a vertex p_i we want the Guide to find the next vertices p_j such that:

1- p_j is unvisited

2- S_i^j is included in a strip with vertical section less than ϵ and with a valid slope

3- $\forall p \in S_i^j,\ p.x \geq p_i.x - \epsilon/2$

4- $\forall p \in S_i^j,\ p.x \leq p_j.x + \epsilon/2$

Checking Condition 3. Let p_{β_i} denote the first vertex in the list p_i that verifies $p_{\beta_i}.x < p_i.x - \epsilon/2$. For any $k,\ i < k < \beta_i$, p_k satisfies the condition and for any $k \geq \beta_i$, p_k is not valid. So we preprocess the β_i values in $O(n \log n)$ time by a depth first traversal of the DAT coupled to binary searches in the stack. Therefore, the condition is checked in constant time.

Checking Condition 2. Dynamic width and dynamic convex hull are active fields in computational geometry [3, 4]. Rather than incorporating a complicated data structure which enables width computation, we prefer to use the vertical section that leads to a simple and efficient implementation. Nevertheless, if we reuse results from [3] and adapt our method, we obtain an algorithm for the width that works in $O(n^{1+\gamma})$ time, with γ an arbitrary small constant. The sectional length, SL, of a convex hull can be computed in logarithmic time [13]. First, we check this query for a given convex hull: we see if the SL is less than ϵ and we verify the angle of the antipodal pair that defines the SL. If this angle is outside our interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$, we look for the first edge whose slope is in this interval in $O(\log n)$ time. The remaining computation of the sectional length of the covering strip associated with this edge can be done in $O(\log n)$ time. Thus, for a given convex hull, we check the second condition in logarithmic time.

Notice that when S_i^j is invalid regarding the second condition, S_i^k is also invalid for any $k > j$. Thus we can define Ω_i to be the last index j for which S_i^j implies a positive answer. With a simple polygonal chain as input, we can preprocess this array in $O(n \log n)$ time using online convex hull algorithm [16]. Notice that Ω_i is a non-decreasing function and that for any k and j, $k \leq j \leq \Omega_k$, we have $\Omega_k \leq \Omega_j \leq \Omega_{\Omega_k}$. We first compute the $\Omega^i(1) = \Omega(\Omega^{i-1}(1))$ values using incremental hulls only. Then for each point $p_k = p_{\Omega^i(1)}$, we manage one decremental hull from p_{Ω_k} to p_k and one incremental hull from p_{Ω_k} to $p_{\Omega_{\Omega_k}}$. One property of a simple polygonal chain is that the convex hulls of two consecutive subpaths can only have two intersection points. Thus, the full hull can be processed in logarithmic time by merging these two hulls. To compute $\Omega_{j>k}$, we successively add points one by one to the incremental hull until the full hull is no longer valid. We then remove p_j from the decremental hull and proceed with the $\Omega_{j+1<\Omega_k}$ computation.

Checking Condition 1. We look for the first unvisited ascendant of p_i in the IAT. For this, the parent-child relationship is stored separately in an array. We use a union-find data structure so that the algorithm remembers the visited nodes and finds the first unvisited ancestor of a given node. To mark a node n as visited, we unite it with its parent: Union$(n, \text{Parent}(n))$. The representative of the set containing Parent(n) is kept for the new set. So, in order to find the first unvisited ascendant of p_i, we simply perform a Find(Parent(p_i)) operation. A union by rank is sufficient to achieve an $O(m \log n)$ time complexity, where m denotes the number of queries.

Checking Condition 4. As stated, each ascendant p_l of p_i in the IAT represents the rightmost vertex of the subchain S_i^l, by definition, such points always verify the fourth query. Therefore, until we reach the limit given by $p_{\min(\beta_i,\Omega_i)}$, each unvisited ascendant of p_i represents an answer of the Guide. Nevertheless (see Fig. 5), the valid points, called *return points*, that do not lie on the right border remain unprocessed. For a return point p_j, consider the rightmost point p_l of the subchain S_i^j. It is actually a candidate point of p_i. Thus, instead of checking if all the vertices of the subchain S_i^j verify the fourth condition, we look for the points p_j within distance $\epsilon/2$ of the candidate points p_l. Given the definition of the candidate points, no vertex with an index less than the index of Parent(p_l) can lie on the right side of p_l. So all the valid vertices p_j verify the next condition: $(j, p_j.x) \in [l, \text{index of parent}_{IDL}(p_l)[\times[p_l.x - \epsilon/2, +\infty[$. Using a priority search tree [14], we obtain our answers in $O(\log n + q)$ where q denotes the number of retrieved vertices. Then, the marked vertices are withdrawn from the tree. Each deletion has a logarithmic cost.

Missed markings When reaching the last candidate point, the bound $p_{\min(\beta_i,\Omega_i)}$ may prevent some return vertices from being selected (see Fig. 5). We cannot mark such a candidate point as visited, as otherwise we would have no means to find its unprocessed return points later. Fortunately, for each node traversed during the BFT, it is only possible to miss one marking. So, the number of times we process this configuration is bounded by the number of points, and so this produces a linear number of useless operations.

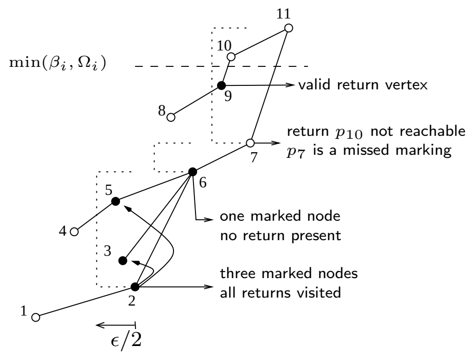

Figure 5: Nodes visited by the Guide.

4 Conclusion

Our algorithm computes an approximation with a minimum number of vertices and our hybrid criterion guarantees that the approximate subchain retains the shape of the original chain. This is the first near-linear time algorithm that ensures this optimality and this property. The overall time complexity of our method is $O(n \log n)$. Its implementation only requires classical data structures: priority search trees, arrays and union by rank. It is based on academic algorithms like the online Melkman convex hull algorithm, tangent finding and sectional length computation.

References

[1] P. K. Agarwal, S. Har-Peled, N. Mustafa, Y. Wang. Near-linear time approximation algorithms for curve simplification In *Algorithmica*, to appear.

[2] P. K. Agarwal, K. R. Varadarajan. Efficient Algorithms for Approximating Polygonal Chains. In *Discrete Comput. Geom.*, pp. 273-291, 23, 2000.

[3] G. S. Brodal, R. Jacob. Dynamic planar convex hull. In *Proc. 43rd Annual Symp. on Foundations of Computer Science* pp.617-626, 25, 2002.

[4] T. M. Chan. A Fully Dynamic Algorithm for Planar Width. In *Discrete & Computational Geometry*, pp:17-24, 30(1), 2003.

[5] W. S. Chan, F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. In *Internat. J. Comput. Geom. Appl.*, pp. 59-77, 6(1), 1996.

[6] D.Z. Chen, O. Daescu. Space-efficient algorithms for approximating polygonal curves in two dimensional space. In *Int. J. Comput. Geom. Appl.* pp. 95-112, 13(2), 2003.

[7] O. Daescu. New results on path approximation. In *Algorithmica* pp. 131-143, 38(2), 2003.

[8] O. Daescu, N. Mi. Polygonal chain approximation: a query based approach. In *Computational Geometry, theory and Appllications*, pp. 41-58, 30(1), 2005.

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

[10] D. Eu, G.T. Toussaint. On approximation polygonal curves in two and three dimensions. In *Graphical Models and Image Processing*, pp.231-246, 56(3), 1994.

[11] J. Hersberger, J. Snoeyink. Cartographic line simplification and polygon csg formulae in O(n log* n) time. In *Proc. 5th International Workshop on Algorithms and Data Structures* pp. 93-103, 1997.

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

[13] D. Kirpatrick and J. Snoeyink. Tentative prune-and-search for computing fixed-points with applications to geometric computation. In *Fundamenta Informaticae*, pp:353-370, 22(4), 1995.

[14] E. M. McCreight. Priority search tress. In *SIAM J. Comput.*, pp:257-276, 14(1), 1985.

[15] A. Melkman, J. O'Rourke. On polygonal chain approximation. In *Computational Morphology, North-Holland*, pp. 87-95, Amsterdam, 1988.

[16] A.A. Melkman. On-line construction of the convex hull of a simple polyline. In *Information Processing Letters*, pp:11-12, 25, 1987.

[17] G.T. Toussaint. On the complexity of approximating polygonal curves in the plane. In *Proc. IASTED, Inter. Symp. on Robotics and Automation*, pp. 59-62, 1985.