Path Simplification for Metro Map Layout

Damian Merrick[1,2] and Joachim Gudmundsson[2]

[1] School of Information Technologies, University of Sydney, Australia
dmerrick@it.usyd.edu.au
[2] National ICT Australia*, Sydney, Australia
joachim.gudmundsson@nicta.com.au

Abstract. We investigate the problem of creating simplified representations of polygonal paths. Specifically, we look at a path simplification problem in which line segments of a simplification are required to conform with a restricted set of directions $\mathcal{C}$. An algorithm is given to compute such simplified paths in $\mathcal{O}(|\mathcal{C}|^3 n^2)$ time, where n is the number of vertices in the original path. This result is extended to produce an algorithm for graphs induced by multiple intersecting paths. The algorithm is applied to construct schematised representations of real world railway networks, in the style of metro maps.

1 Introduction

Metro maps have been used to effectively illustrate transportation networks for many decades. They incorporate a carefully balanced trade-off between geographical accuracy and readability of the diagram. Traditionally, these diagrams are drawn manually, and it is a significant challenge to create computer algorithms that produce high-quality metro maps. A common feature of metro maps is the logical division of the network into a set of intersecting paths, or train lines. One may consider the simpler problem of drawing these paths individually, instead of the entire network at once.

In the field of cartography, and particularly in geographical information systems (GIS), it is an important problem to represent detailed geographical features on a map in a simple, easily understandable form. Consequently, efficient algorithms for simplifying lines or paths in a geographical data set are desirable.

The problem of computing a simplification of a given polygonal path where the vertices of the simplification are required to be a subset of the input points has been studied extensively. Imai and Iri [19–21] formulated the problem as a graph problem. They constructed an unweighted directed acyclic graph and then used breadth-first search to compute a shortest path in this graph. The same approach has been used by many algorithms devoted to this problem [3, 4, 6, 7]. A widely used heuristic for path-simplification is the Douglas-Peucker algorithm [10]. If the path is given in the plane then it can be implemented to run in $\mathcal{O}(n \log^* n)$ time [17], but it does not guarantee an optimal solution.

* National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

M. Kaufmann and D. Wagner (Eds.): GD 2006, LNCS 4372, pp. 258–269, 2007.

Numerous criteria have been proposed for simplifying polygonal paths. In [4, 8, 19, 21, 23] the so-called *tolerance zone* criterion was used. Other measurements are the *infinite beam* criterion [7, 11, 29], the *uniform measure* criterion [2, 13], *distance preserving* criterion [15] and the *area preserving* criterion [3].

In this paper we address a related problem. As input, a polygonal *path* P is given, consisting of a sequence of points $\langle p_1, \ldots, p_n \rangle$, with closed line segments called *links* joining each pair of consecutive points. Also given is a finite set of directions $\mathcal{C}$. The problem we address is to produce a simplification of P, subject to some constraints, where every link l_i in the simplification is parallel to some orientation $c \in \mathcal{C}$. A path conforming to this restriction is called a $\mathcal{C}$-directed path. Requiring a $\mathcal{C}$-directed path as output results in a "schematised" approximation of the original path. Such an approximation can greatly improve the readability of diagrams in which several paths must be drawn. The vertices of the simplification are not restricted to be a subset of the input points.

In the case when the links are restricted to a given set of orientations Neyer [26] proposed an algorithm that minimises the number of links in the output path. This algorithm runs in time $\mathcal{O}(nk^2 \log n)$, where k is the number of links in the output. The output is a path which remains within a given distance of the original path, according to the Fréchet distance metric. Utilising Fréchet distance in this respect, however, can produce undesirable "zig-zags" in some paths when the set of allowed orientations is small.

The problem considered in this paper uses a more relaxed restriction on distance. For each point p in the input path P, consider its *ε-circle* $E(p)$; that is, the closed disc of radius ε centred at p. Our requirement is that a simplification must intersect $E(p)$ for every $p \in P$, subject to a restriction on the order of intersections. This is equivalent to enforcing a maximal Haussdorf distance of the path simplification from the set of input points. Compared to the Fréchet metric, more freedom is given to the simplification at bends.

Guibas et al. [16] considered a version of this problem where the links are not restricted to a certain set of directions. They presented a dynamic programming algorithm that generates an optimal solution in $\mathcal{O}(n^2 \log^2 n)$ time, and a 2-approximation with running time $\mathcal{O}(n \log n)$. In this paper we show that the restricted direction path simplification problem can be solved in $\mathcal{O}(|\mathcal{C}|^3 n^2)$ time.

In addition to the simplification of individual lines, we enter into the problem of simplifying multiple intersecting paths in a network. This is of particular interest in metro map layout. Hong et al. [18] present some force-directed graph drawing approaches to metro map layout, which are also used to produce metro map layouts of non-geographical networks. Slower but more geographically accurate optimisation-based methods are detailed by Stott and Rodgers [28], and more recently by Nöllenburg and Wolff [27]. One of the main benefits of the approach we present in this paper is its fast running time. This gives the potential to handle much larger instances than those solvable by slower methods. In addition, faster methods bring about the possibility of real-time interactivity. For example, in a network design application or diagramming tool, a designer may want to modify a network and immediately visualise the effect on the diagram.

A second problem related to metro map layout is the schematisation of networks. Cabello et al. [5] give an $\mathcal{O}(n \log n)$ time algorithm which creates a schematised map of a given network, where intersection points remain in fixed locations and paths between intersection points are drawn using two or three links. This approach ensures that the topology of the network is preserved. Other recent work investigates problems in drawing schematised layouts of trees [14].

We restrict our attention to networks induced by multiple intersecting paths, as in metro maps. We propose an efficient method for simplifying such networks given a restricted set of directions and an error threshold. Preservation of topology is not guaranteed in this approach.

Section 2 formalises the path simplification problem we address, and proposes an algorithm to solve it. The extension of the path simplification algorithm to metro map layout is presented in Section 3, and implementation results are given in Section 3.1. Section 4 offers some concluding remarks and acknowledgements.

Due to space constraints most details are omitted and can be found in [25].

2 $\mathcal{C}$-Directed Path Simplification

The path simplification problem we address in this paper is defined as follows:

Problem 1. The $\mathcal{C}$-Directed Path Simplification Problem

Input: A path P, a set of directions $\mathcal{C}$ and a distance threshold ε.

Output: A $(\mathcal{C}, \varepsilon)$-simplification P' of P with the minimum possible number of links, such that the ε-circles of all points in P are stabbed in order by P'.

To make the problem definition clear we also need to define "order".

Definition 1. *A directed line segment ℓ that intersects the ε-circles of a sequence of points $\mathcal{S} = \langle p_1, \ldots, p_n \rangle$ is said to follow the* order *of $\mathcal{S}$ if and only if for every pair of points $p_i, p_j \in \mathcal{S}, i < j$ there exist points q_i, q_j on ℓ within the ε-circles of p_i and p_j respectively, for which it holds that q_i is encountered before or at the same position as q_j along ℓ.*

To guarantee that a solution exists, we assume that for any direction $c \in \mathcal{C}$, the opposite direction $\bar{c}$ is also in $\mathcal{C}$.

The algorithm we propose to solve the $\mathcal{C}$-directed path simplification problem is composed of two parts. First, given a path to simplify, it constructs a boundary path (to be defined) which determines a set of minimum-link $(\mathcal{C}, \varepsilon)$-simplifications of the given path, see Fig. 1(a). In the second part, a single minimum-link $(\mathcal{C}, \varepsilon)$-simplification of the path is extracted from the boundary path, see Fig. 1(b). We start with some necessary definitions.

Given a pair of directions (c_1, c_2) and a set of points $\mathcal{S} = \{p_1, \ldots, p_n\}$ let τ_i be the c_1-most tangent of the ε-circle of p_i with direction c_2, and let α_i be the point (if any) on τ_i with the smallest c_2-coordinate for which it holds that a line through α_i with direction c_1 stabs $\{p_1, \ldots, p_i\}$ in order as shown in Fig. 2(a–b). Let ι be the smallest value for which there is no such point α_ι, and let $\mathcal{S}' = \{p_1, \ldots, p_{\iota-1}\}$. For every $1 \leq j < \iota$ let ℓ_j be the ray with direction c_1 emanating from α_j.

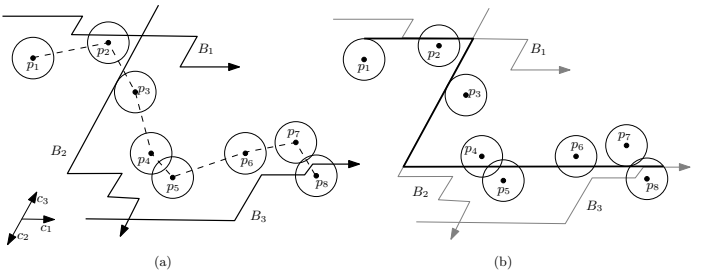

Fig. 1. (a) An example showing a sequence of eight ε-circles and three link boundaries B_1, B_2 and B_3 forming a boundary path. (b) Extracting a ($\mathcal{C}$, ε)-simplification from the boundary path.

Definition 2. *The* link boundary L *of* $\mathcal{S}$ *with direction pair* (c_1, c_2) *is the polyline described by the upper envelope in direction* c_2 *of the set* $\ell_1, \ldots, \ell_j$. *The link boundary* L *has* size j *and is said to* stab $\mathcal{S}'$ *(in order).*

If the stabbing order is ignored, a single maximum link boundary B can easily be computed in linear time with respect to the number of points stabbed by B. In Section 2.2 we discuss in more detail how to do it in the ordered case.

Definition 3. *A* boundary path *is a sequence of link boundaries that stabs a sequence of points* $\mathcal{S}$. *A size* k *maximal boundary path of* $\mathcal{S}$ *is a boundary path containing* k *link boundaries that stabs the maximal start sequence of* $\mathcal{S}$.

2.1 A High-Level Description

In this section we give an overview of the algorithm. The idea is to construct a boundary path B where each link boundary, denoted $B_1, \ldots, B_k$, along B corresponds to a segment in the final path simplification P' of P, and then extract a path from B. The algorithm is given as pseudocode below.

Building the boundary path. The boundary path is built by incrementally adding a link boundary. Let $\beta_k(c_i, c_j)$ denote a maximal boundary path of size k whose last link boundary has direction pair (c_i, c_j). In each iteration of the algorithm, $\mathcal{O}(|\mathcal{C}|^2)$ active candidate boundary paths are maintained together with a counter k of the number of iterations, i.e., the size of all the boundary paths. Initially, $\beta_k(c_i, c_j)$ is the maximal single link boundary path of $\mathcal{S}$ with direction pair (c_i, c_j), and $k = 1$. If any of the $\mathcal{O}(|\mathcal{C}|^2)$ boundary paths stabs the entire sequence $\mathcal{S}$ then we are done, otherwise continue iteratively as follows.

For every triple of directions (c_i, c_j, c_ℓ) in $\mathcal{C}$ extend $\beta_k(c_i, c_j)$ with a link boundary of direction pair (c_j, c_ℓ), excluding cases where c_i is parallel to c_j or c_j is parallel to c_ℓ. Consider all the maximal boundary paths with $k + 1$ links whose last link boundary has direction pair (c_j, c_ℓ), and save the boundary path that stabs the longest sequence of $\mathcal{S}$ in $\beta_{k+1}(c_j, c_\ell)$. Finally, increment the value of k. The process ends when a boundary path stabs $\mathcal{S}$. The first boundary path that stabs $\mathcal{S}$ is denoted $\Delta(\mathcal{S})$ and has size k.

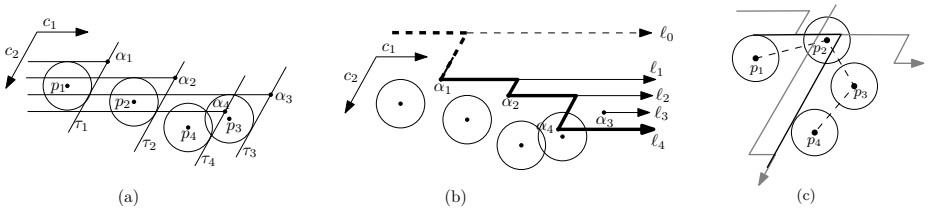

Fig. 2. (a) τ_i is the c_1-most tangent of the ε-circle of p_i with direction c_2. (b) The link boundary with direction pair (c_1, c_2) of $\langle p_1, \ldots, p_4 \rangle$. (c) A potential problem when two link boundaries are joined; the truncation of one of the links at the bends results in p_3 not being stabbed.

Constructing a path from the boundary path. Given the boundary path $\Delta(\mathcal{S})$ containing a sequence of link boundaries $\langle B_1, \ldots, B_k \rangle$, construct the $(\mathcal{C}, \varepsilon)$-simplification $P' = \langle p'_1, p'_2, \ldots, p'_{k+1} \rangle$ as follows. Let p'_{k+1} be the point within p_n's ε-circle that is furthest along B_k in B_k's primary direction. Let ℓ_k be the line through the last ray of B_k. Set p'_k to the intersection of ℓ_k and B_{k-1}, and let ℓ_{k-1} be the line through p'_k in the primary direction of B_{k-1}. Continue this process for every point p'_i back to p'_2, setting p'_i to the intersection of ℓ_i and B_{i-1}. Finally, let p'_1 be the point within p_1's ε-circle lying on B_1 that is backmost in B_1's primary direction.

Algorithm StabbingPath
1. done:= false
2. **for** $i = 1$ **to** $|\mathcal{C}|$ **do**
3. **for** $j = 1$ **to** $|\mathcal{C}|$ **do**
4. **for** $k = 1$ **to** n **do**
5. $\beta_k(c_i, c_j)$:=null
6. $k := 0$
7. **while** not done **do**
8. **for** $i = 1$ **to** $|\mathcal{C}|$ **do**
9. **for** $j = 1$ **to** $|\mathcal{C}|$ **do**
10. **for** $\ell = 1$ **to** $|\mathcal{C}|$ **do**
11. tmp:=ExpandBoundaryPath($\beta_k(c_i, c_j), c_\ell, \mathcal{S}$)
12. $\beta_{k+1}(c_j, c_\ell)$:=SelectBestBoundaryPath($\beta_{k+1}(c_j, c_\ell)$,tmp)
13. **if** $\beta_{k+1}(c_j, c_\ell)$ stabs $\mathcal{S}$ **then**
14. $\Delta(\mathcal{S}) := \beta_{k+1}(c_j, c_\ell)$
15. done:=true
16. $k := k + 1$
17. **endwhile**;
18. **Return** ComputePath($\Delta(\mathcal{S}), \mathcal{S}$)

Consider each of the steps of the algorithm. Steps 1–6 of the algorithm require $\mathcal{O}(|\mathcal{C}|^2 n)$ time and step 18 requires $\mathcal{O}(n)$ time. It remains to study the body within the three loops in steps 11–12. Step 12 can easily be performed in linear time since one only has to decide which of the two boundary paths that stabs

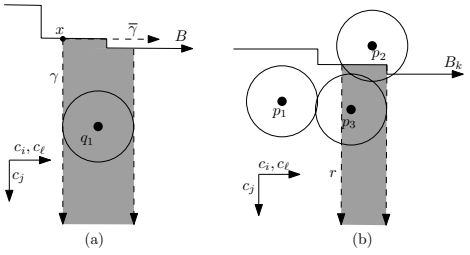

Fig. 3. (a) The boundary-restricted stabbing interval $I(\langle q_1 \rangle, c_j, B)$ (shaded). (b) The boundary-restricting stabbing interval $I(\langle p_2, p_3 \rangle, c_j, B_k)$ (shaded). The left bounding ray r of this interval is the backmost ray starting on B_k that ensures all points remain stabbed, and forms the first ray of B_{k+1}.

the most ε-circles. That leaves step 11, which is the step that will dominate the running time of the algorithm.

Guibas et al. [16] showed that one can compute a line that stabs the longest possible prefix of a sequence of unit discs in the correct order in $\mathcal{O}(n \log n)$ time. However, their problem has two main differences to ours. First, we only need to consider one direction. Second, when B_{k+1} is joined to the end of the boundary path $\beta_k(c_i, c_j)$ then B_k is truncated at its point of intersection with B_{k+1}. Hence we must ensure that after the truncation all points in S_k are either still stabbed by B_k or by B_{k+1}. Figure 1(c) illustrates this problem.

In the next section we will discuss how step ExpandBoundaryPath can be implemented to run in $\mathcal{O}(n^2)$ time.

2.2 Extending a Boundary Path with a Link Boundary

Consider the algorithm described in the previous section. Let $B_1, \ldots, B_k$ be the link boundaries in $\beta_k(c_i, c_j)$, and let $\mathcal{S}_k = \langle p_{f(k)}, \ldots, p_{l(k)} \rangle$ be the sequence of $\mathcal{S}$ stabbed by B_k. Thus the sequence of $\mathcal{S}$ that remains to be stabbed is $\mathcal{S}' = \langle p_{f(k+1)}, \ldots, p_n \rangle$, where $f(k+1) = l(k) + 1$.

We need to be very careful when joining two link boundaries. To facilitate this we define *boundary-restricted stabbing intervals* (Figs. 3(a), (b) give examples).

Definition 4. *Given a sequence of points $\mathcal{S}$, a direction c, and a link boundary B, the* boundary-restricted stabbing interval $I(\mathcal{S}, c, B)$ *is the region in the plane for which it holds that a c-directed ray starting on B lies inside the region if and only if it stabs $\mathcal{S}$ in order.*

Initialisation. The task of the initialisation step is to construct the first ray in B_{k+1}. If the turn from c_i to c_j is in the same direction as the turn from c_j to c_ℓ, e.g. both are left hand turns, then we add to B_{k+1} the c_j directed ray that is at ∞ according to c_i. If the turns are in different directions, i.e. one is a left hand turn and one is a right hand turn, then we need to find the c_j-directed ray that starts as far back as possible on B_k and ensures S_k remains stabbed. Note

that this ray starts as far back along B_k as possible, while still ensuring that all points are stabbed by either B_k or the ray itself. Hence, it forms the backmost bound for all possible rays in B_{k+1}. An example of this is illustrated in Fig. 3(b).

Extending B_{k+1} to Stab a New Point. Once the initialisation has been completed, the main loop commences. At each iteration, the algorithm attempts to extend the set of points stabbed by $\beta_k(c_i, c_j)$ and the link boundary B_{k+1}.

Consider the points in $\mathcal{S}$ not stabbed by $\beta_k(c_i, c_j)$, denoted $\mathcal{S}' = \langle q_1, q_2, \ldots \rangle$. Process the points in $\mathcal{S}'$ in order, starting with q_1. In a generic step assume we are about to process q_m. Compute the stabbing interval $I(\langle q_1, \ldots, q_m \rangle, c_j, B_k)$. If the interval is empty then we stop since there is no c_j-oriented line that can stab $q_1, \ldots, q_m$ in order. If the interval is non-empty let γ be the bounding line of the stabbing interval with the smallest c_ℓ-value. Find the intersection point x between γ and B_k, and let $\overline{\gamma}$ be the c_i-directed ray starting at x. Process every point p in $\mathcal{S}_k$ in reverse order as follows.

If $\overline{\gamma}$ cut the ε-circle of p into two pieces then B_k no longer stabs p, and it must be stabbed by B_{k+1}. Set γ to be the bounding line of $I(\langle p, \ldots, q_m \rangle, c_i, B_k)$ with the smallest c_ℓ-value. If the stabbing interval is empty then we are finished, as B_{k+1} cannot stab $p, \ldots, q_m$ or any further points. Otherwise, continue iterating.

Once the backtracking has completed, all points in the sequence $\mathcal{S}_k$ and $\langle q_1, \ldots, q_m \rangle$ are stabbed by either B_k or γ. Now consider the last ray γ' in B_{k+1}. If γ' is positioned at ∞, remove it from B_{k+1} and replace it with the ray along γ that starts on B_k. Otherwise, let t be the c_j-most c_ℓ-directed tangent to the ε-circle of p_m. Add t and γ to B_{k+1}, and truncate γ', t and γ at their pairwise intersection points. The point p_m is now stabbed by B_{k+1}, and can be removed from $\mathcal{S}'$ and added to $\mathcal{S}_k$. Iteration continues until $\mathcal{S}'$ is empty or a point is found that cannot be stabbed.

If each stabbing interval is computed from scratch, a straight-forward implementation would require $\mathcal{O}(n^2)$ time. This follows since the order restriction has to be checked for every pair of ε-discs in the sequence. However, since the above approach incrementally adds new points at the end of the sequence this can be improved as stated in Theorem 1 below (see [25] for details). The concluding result for this section follows in Theorem 2.

Theorem 1. *There is a data structure of size $\mathcal{O}(n)$ that answers boundary-restricted stabbing interval queries in constant time and allows additions of points to the end of the sequence in $\mathcal{O}(n)$ time.*

Theorem 2. ExpandBoundaryPath$(\beta_k(c_i, c_j), c_\ell, \mathcal{S})$ *can be computed in $\mathcal{O}((|\mathcal{S}_k| + |\mathcal{S}_{k+1}|)^2)$ time, where $\mathcal{S}_k$ and $\mathcal{S}_{k+1}$ are the sets of points stabbed by B_k and B_{k+1}, respectively, in the produced boundary path.*

2.3 Complexity Analysis

Time Complexity. Recall from Section 2.1 that the running time of algorithm StabbingPath is dominated by step 11, i.e. ExpandBoundarypath. Assume

that a minimum link path contains k links, using Theorem 2 gives that the total running time is bounded by:

$$\mathcal{C}^3 \cdot \left(|\mathcal{S}_1| + \sum_{i=1}^{k-1} \left(|\mathcal{S}_i| + |\mathcal{S}_{i+1}| \right)^2 \right) < \mathcal{C}^3 \cdot 6 \left(\sum_{i=1}^{k-1} |\mathcal{S}_i|^2 + \sum_{i=1}^{k-1} |\mathcal{S}_{i+1}|^2 \right) = \mathcal{O}\left(\mathcal{C}^3 n^2 \right).$$

The final step of extracting a minimum-link $(\mathcal{C}, \varepsilon)$-simplification from the boundary path takes $\mathcal{O}(n)$ time, therefore the entire algorithm needs $\mathcal{O}(|\mathcal{C}|^3 n^2)$ time.

Space Complexity. At each iteration of the algorithm, $\mathcal{O}(|\mathcal{C}|^2)$ paths are stored, each containing up to k boundaries. Each link boundary uses linear space and every input point is stabbed by at most a constant number of link boundaries per boundary path, thus it follows that the algorithm requires $\mathcal{O}(|\mathcal{C}|^2 n)$ space.

2.4 Proof of Correctness

Lemma 1. *Given a starting point q, a preceding link boundary B_{i-1} and directions c_{i-1}, c_i and c_{i+1}, the pair of link boundaries (B_i, B_{i+1}) constructed by the algorithm stabs the furthest point of P possible.*

Theorem 3. *Algorithm* StabbingPath(P, ε) *computes a minimum link $(\mathcal{C}, \varepsilon)$-simplification of P.*

Lemma 1 and Theorem 3 are proved in [25].

3 Extension to Metro Map Layout

A major motivation for this paper is the automatic visualisation of metro maps. For this purpose, we extend the $\mathcal{C}$-directed path simplification algorithm of Section 2 to handle multiple intersecting paths.

We adopt the graph-based model of Hong et al. [18] to describe a metro network; a *metro map graph* consists of a graph G and a set of paths covering all vertices and edges of G. We assume we are given a metro map graph with a set of initial coordinates for each vertex, representing the geographical locations of stations in the metro network.

The first step taken by the algorithm is to sort the set of paths according to a given measure of *importance*. Importance may be manually defined, or calculated automatically by some heuristic function. For our purposes, we define the importance of a path to be the number of vertices on the path that are also a part of some other path in the metro map graph.

Once the paths are sorted, the most important path is taken from the set and simplified using the algorithm of Section 2. Once the path simplification has been computed, the points of the path must be placed on the simplification. This may be achieved by simply projecting each original point onto the link in the path simplification that stabs it. Alternatively, the points may be redistributed along

the line segment such that the distance between every adjacent pair of points is equal. This may result in a distance greater than ε between each original point and its counterpart on the simplification. However, it is useful for reducing clutter and making the final diagram clearer. Once placed, the points in the simplification are fixed in their positions.

The next most important path is then taken and split into subpaths around any sequences of fixed vertices, such that each subpath contains at most one fixed vertex at either end. Each of these subpaths is simplified in turn, and all of the points in the resulting simplifications are fixed.

A small modification must be made to the path simplification algorithm to ensure that the simplified path runs exactly through any fixed points. This is achieved by restricting the allowed interval for each fixed point to the single line in each direction that passes exactly through that point.

The algorithm repeats the above steps, each time taking the next most important path from the set, splitting it and simplifying, until there are no paths left to be simplified. A complete layout has then been generated.

We found that two extensions to the path simplification algorithm were useful to improve the results: restricting the maximum turning angle at bends, and enforcing a minimum link length.

Maximum bend angle. Allowing turns of large angles between links in a simplification can result in undesirable sharp bends. Given an angle α, one may want to ensure that no turn of more than α degrees is made in the simplified path. To implement this, simply ignore any pairs of consecutive directions separated by more than α degrees. Theorem 3 still holds in this case, i.e. the simplification produced will have the minimum number of links of any $(\mathcal{C}, \varepsilon)$-simplification subject to the same restriction.

Minimum link length. The algorithm enforces no restriction on the length of links in the simplification. This can cause a restriction on the maximum bend angle to be less effective, as the extra links produced may be of zero length. We enforce a minimum length l_{min} for each link when constructing a link boundary, by trimming all boundary-restricted stabbing intervals to a distance of l_{min} from the start of the previous link boundary. Problems can occur where a link needs to be shorter than l_{min} in order to stab a point. To avoid this, we decrease l_{min} on any iteration in which no further points are stabbed by any link boundary. If this continues to occur for several iterations, we set l_{min} to zero. This approach ensures that a solution will be computed, but the number of links in the solution may not be the minimum.

3.1 Results

We applied the method of Section 3 to the central part of the Sydney Cityrail railway network [9], and the London Underground network [22]. The Sydney graph has 173 vertices and 182 edges, and London has 266 vertices and 308 edges. We created a Java implementation as a plugin to the graph editor *jjGraph* [12].

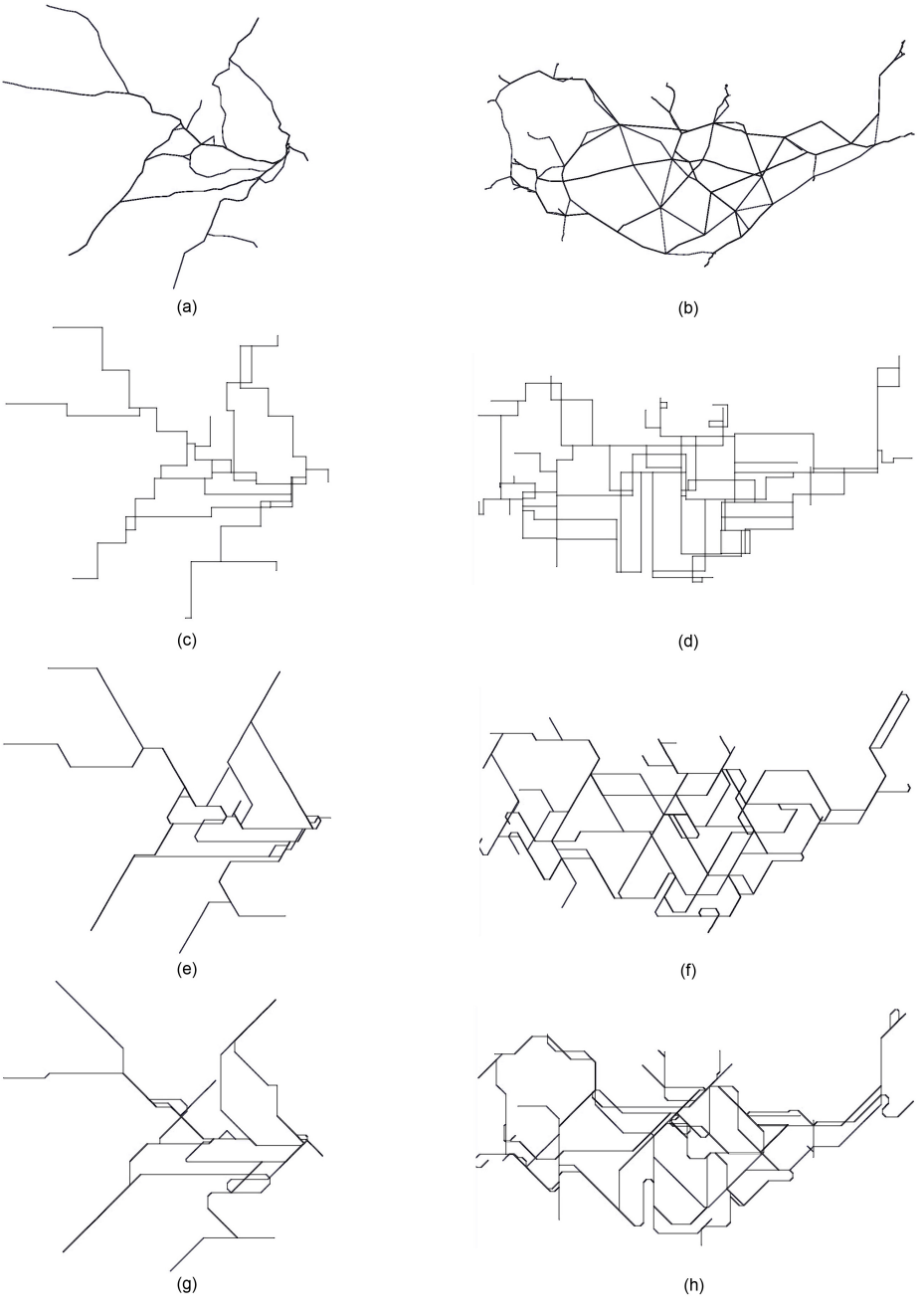

Fig. 4. (a) The original Cityrail geometry, and simplifications using (c) 4, (e) 6 and (g) 8 directions. (b) The original London geometry, and simplifications using (d) 4, (f) 6 and (h) 8 directions.

The input geometry for the Cityrail network was taken directly from its geographical layout, shown in Fig. 4(a). The original geography of the London network is quite dense in the centre and sparse in the exterior, so we scaled it using the centrality-based scaling technique of Merrick and Gudmundsson [24] before applying our algorithm. The scaled geometry is shown in Fig. 4(b). The scaling used betweenness centrality, with parameters $\alpha = 10, \beta = 50$ and $\gamma = 2$, and the scaling took 6445 ms to execute. Refer to the literature for details on the scaling method and its associated parameters [24].

We ran our implementation on both networks with varying parameters, and show selected results. Figs. 4(c), (e) and (g) show the Cityrail network simplified with $|\mathcal{C}| = 4$, 6 and 8 respectively. The respective running times were 154, 201 and 268 ms. Figs. 4(d), (f) and (h) show the London network simplified with $|\mathcal{C}| = 4$, 6 and 8 respectively. The running times to obtain these results were 282, 423 and 573 ms. All results were produced on a computer with a Pentium 4 3.0 GHz processor and 1GB of RAM, running Windows XP. Running times were averaged over 10 runs. Values of ε were chosen by trial and error and varied between the datasets; at this stage the possibility of automatically choosing an appropriate value for ε remains as future work.

4 Concluding Remarks and Acknowledgements

Consider a k-link $(\mathcal{C}, \varepsilon)$-simplification of a path P produced by the algorithm of Section 2. It might be that there exists a k-link $(\mathcal{C}, \varepsilon')$-simplification of P where ε' is much smaller than ε. In [25] we present a fully polynomial-time approximation scheme (FPTAS) to the dual of the problem, i.e. given k compute a $(\mathcal{C}, \varepsilon)$-simplification of P that minimises ε. The FPTAS produces a $(1 + \delta)$-approximation for any given $\delta > 0$.

We thank Martin Nöllenburg and Alexander Wolff for the London data.

References

1. H. Alt and M. Godau. Measuring the resemblance of polygonal curves. In Proc. 8th Annual Symposium on Computational Geometry, p. 102–109, 1992.
2. P. K. Agarwal and K. R. Varadarajan. Efficient Algorithms for Approximating Polygonal Chains. Discrete & Computational Geometry, 23(2):273–291, 2000.
3. J. Bose, O. Cheong, S. Cabello, J. Gudmundsson, M. van Kreveld and B. Speckmann. Area-preserving approximations of polygonal paths. J. Discrete Alg., 2006.
4. G. Barequet, D. Z. Chen, O. Daescu, M. T. Goodrich and J. Snoeyink. Efficiently approximating polygonal paths in three and higher dimensions. Algorithmica, 33(2):150–167, 2002.
5. S. Cabello, M. de Berg, and M. van Kreveld. Schematization of networks. Computational Geometry and Applications, 30:223–238, 2005.
6. W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. IJCGA, 6:59–77, 1996.
7. D. Z. Chen and O. Daescu. Space-efficient algorithms for approximating polygonal curves in two-dimensional space. IJCGA, 13:95–111, 2003.

8. D. Z. Chen, O. Daescu, J. Hershberger, P. M. Kogge, N. Mi and J. Snoeyink. Polygonal path simplification with angle constraints. CGTA, 32(3):173–187, 2005.
9. CityRail network map. Web page: http://www.cityrail.info/networkmaps/mainmap.jsp (Accessed 6th Sept 2006).
10. D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer 10(2):112–122, 1973.
11. D. Eu and G. T. Toussaint. On Approximating Polygonal Curves in Two and Three Dimensions. CVGIP: Graphical Model and Image Processing, 56(3):231–246, 1994.
12. C. Friedrich. jjGraph. Personal communication.
13. M. T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. Discrete & Computational Geometry, 14:445–462, 1995.
14. J. Gudmundsson, M. van Kreveld and D. Merrick. Schematisation of Tree Drawings. Submitted to Graph Drawing, June 2006.
15. J. Gudmundsson, G. Narasimhan and M. H. M. Smid. Distance-Preserving Approximations of Polygonal Paths. To appear in CGTA, 2006.
16. L. J. Guibas, J. Hershberger, J. S. B. Mitchell and J. Snoeyink. Approximating Polygons and Subdivisions with Minimum Link Paths. IJCGA, 3(4):383–415, 1993.
17. J. Hershberger and J. Snoeyink. Cartographic line simplification and polygon CSG formulæ in $O(n \log^* n)$ time. Computational Geometry – Theory & Applications, 11(3-4):175–185, 1998.
18. S.-H. Hong, D. Merrick, and H. A. D. do Nascimento. The metro map layout problem. In Proc. Graph Drawing 2004, p. 482–491, 2005.
19. H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. Comp. Vision, Graphics and Image Processing, 36:31–41, 1986.
20. H. Imai and M. Iri. An optimal algorithm for approximating a piecewise linear function. Journal of Information Processing, 9(3):159–162, 1986.
21. H. Imai and M. Iri. Polygonal approximations of a curve-formulations and algorithms. In Computational Morphology, G. T. Toussaint (ed.), North-Holland, Amsterdam, 1988.
22. London Underground network map. Web page: http://www.tfl.gov.uk/tube/maps/ (Accessed 6th Sept 2006).
23. A. Melkman and J. O'Rourke. On polygonal chain approximation. In Computational Morphology, G. T. Toussaint (ed.), North-Holland, Amsterdam, 1988.
24. D. Merrick and J. Gudmundsson. Increasing the readability of graph drawings with centrality-based scaling. In Proc. APVIS 2006, p. 67–76, 2006.
25. D. Merrick and J. Gudmundsson. $\mathcal{C}$-Directed Path Simplification for Metro Map Layout. http://www.dmist.net/metromap.pdf (Accessed 6th Sept 2006).
26. G. Neyer. Line simplification with restricted orientations. In Proc. 6th International Workshop on Algorithms and Data Structures, p. 13–24, 1999.
27. M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In Proc. 13th International Symposium on Graph Drawing, 2005.
28. J. Stott and P. Rodgers. Metro map layout using multicriteria optimization. In Proc. 8th Interational Conference on Information Visualisation, p. 355–362, 2004.
29. G. T. Toussaint. On the Complexity of Approximating Polygonal Curves in the Plane. In Proc. of the International Symposium on Robotics and Automation (IASTED), pages 311–318, 1985.