Finding a Minimal Tree in a Polygon with its Medial Axis

Herman J. Haverkort*
hjhaverk@cs.uu.nl

Hans L. Bodlaender†
hansb@cs.uu.nl

Abstract

In order to solve a problem arising when generalizing topographical maps, we consider the following problem for simple polygons, i.e., coherent polygons without holes. Some edges of the polygon may be marked as *hard*, and at least two vertices of the polygon are marked as *terminals*. We show that the problem to find a tree of minimum total length, spanning the hard edges and terminals, using only edges of the polygon and its medial axis, can be stated as the problem to find a minimum Steiner tree in a Halin graph, and can be efficiently solved in linear time.

Keywords

map generalisation, minimum network Steiner tree, Halin graph, polygon elimination, polygon dissolution.

1 Introduction

Topographical maps show, among other things, land use by means of coloured polygons that cover the whole map. Different polygons represent different uses of land, e.g. woods, sea, buildings. Thus the area is divided into coherent polygons with the same land use. When going from a large scale to a small scale map, this planar subdivision often needs to be generalised or simplified. One of the things we would like to do is taking out faces which would be too small for practical purposes at the smaller scale. Of course, the elimination of a face should not create a hole in the map. The space left open after the elimination of the small face, must be divided among the neighbouring faces (see e.g. figure 1). Because of conceptual constraints there may be neighbours that cannot be enlarged by a piece of the eliminated face. For example, it may be undesirable to add a part of an area representing a wood to a neighbouring sea area.

In this paper, we consider the problem how to divide the small polygon among its neighbouring faces. The polygon should be cut at certain places, but where should we cut?

In topological terms, we need a tree structure (the cutting lines) with the following properties.

- The complete structure is inside or on the boundary of the eliminated polygon.
- When two neighbour faces meet at the boundary of the eliminated face, the boundary between the two faces continues along the cutting tree. That is: the cutting tree spans all boundary vertices that are adjacent to at least two other polygons.
- Each boundary between the eliminated face and a neighbouring face that cannot receive a part of the eliminated face, is part of the tree.

In geometrical terms, we would like to find a tree structure that maintains the 'character' of the neighbouring faces and does not introduce artifacts like narrow appendices or strange knobs. Moreover, since the object of the entire operation is to simplify the map, we would like to make sure that the new boundaries are less complex (in an optical sense) than the old boundaries.

Bader and Weibel proposed to use the medial axis for this purpose [2]. However, this approach can lead to strange knob-like artifacts (see figure 1).

An alternative method is to look for a cutting tree with the smallest possible total length. This gives us some kind of guarantee that, in a sense, a maximal reduction in complexity is achieved. Unfortunately, such a tree seems difficult to compute. It is a variant of

*The research of this author was done as part of a graduation project at the Dept. of Computer Science of Utrecht University, The Netherlands. The author wants to thank Marc van Kreveld for his support.

†Dept. of Computer Science, Utrecht University, The Netherlands

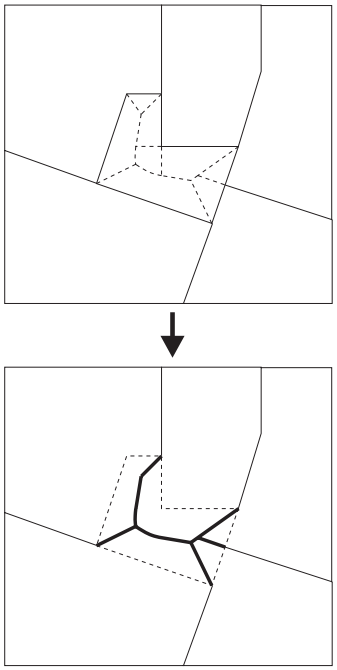

Figure 1: Using the medial axis to divide a polygon among its neighbours. On top is the situation before the division; below is the situation after. The cutting tree is drawn in fat lines.

the general planar Steiner tree problem: given a set of points in the plane, find a tree with minimum total length that spans the given points. Extra vertices may be added, i.e. the branches of the tree may meet each other anywhere in the plane, not only at the given points. Garey et al. have shown the general planar Steiner tree problem to be at least as difficult as any NP-complete problem [7]. However, our problem is restricted by the fact that all given points are on the boundary of a coherent polygon without holes. We demand that no part of the solution is outside that polygon. We do not know if a polynomial time algorithm exists for this restricted Steiner tree problem. Furthermore, we do not know how the complexity of the (restricted) Steiner tree problem is affected if we specify segments of the polygon boundary that must be present in every solution (like land-sea boundaries).

In this paper, we combine the medial axis approach and the Steiner tree approach: we look for a cutting tree, that only uses edges that belong to either an enhanced version of the original polygon or the medial axis, and that has minimum total length. The advantage of this mixed approach is that we can prove that such cutting trees can be found in linear time, as we will show in this article.

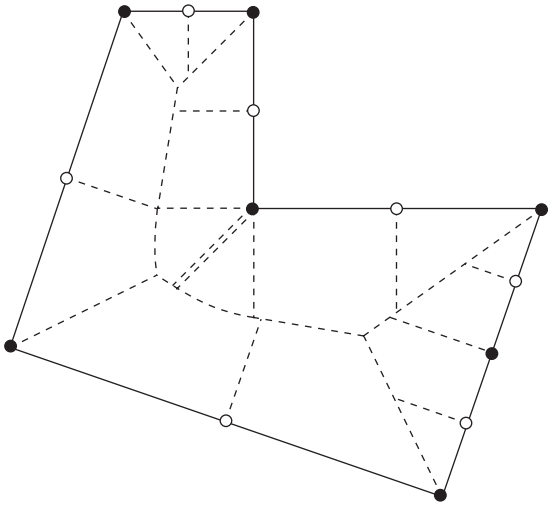

Figure 2: The enhanced polygon with its medial axis. Added vertices are shown as open dots. The actual medial axis is shown dashed. The added edges (concave corner bisectors) are printed double dashed.

2 Modelling the problem: Steiner trees in Halin graphs

We first define the graph we work with more precisely.

For this purpose the boundary of the face that must be eliminated is enhanced with n additional vertices half-way on all the edges (n being the number of vertices in the original polygon). The medial axis is defined as the Voronoi diagram of the boundary edges and vertices of the polygon (using only the part of the diagram that is inside of the polygon), with $O(n)$ additional edges dividing concave corners (see figure 2). Hopefully this choice of edges is sufficient to enable an algorithm to maintain the 'characteristics' of the neighbour faces reasonably well (see explanation given above; a detailed discussion of this idea is beyond the subject of this article).All edges are weighted according to their length.

Lemma 1 *Given Euclidian distance metric, the medial axis of a polygon without holes is a tree, except for edges to corners (i.e. concave corners) which meet in the corner.*

Proof: This is well known. Suppose the medial axis contains a cycle that does not contain a corner. By the definition of the Voronoi diagram, this cycle is the boundary of an area for which some boundary edge or vertex is closer than any other boundary edge or vertex. Let P be a point inside this area and P' the associated vertex or the closest point on the associated edge. Since P is inside a cycle inside the polygon, the segment PP' cuts the cycle in some point Q, which is not on a boundary edge or vertex. Consider any

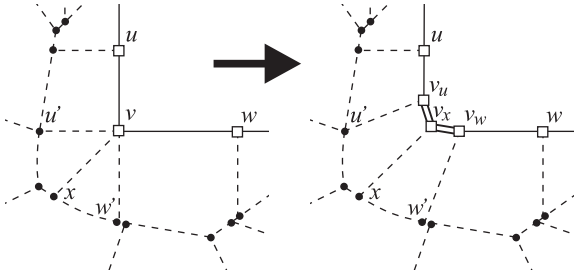

Figure 3: A corner transformed into multiple leafs of the medial axis. Double lines represent zero-length auxiliary edges.

point R on QP' ($R \neq Q$ and $R \neq P'$). Let R' be a point on the polygon boundary which is closest to R. Since R is not in the same Voronoi cell as P, R' is not on the same boundary segment as P' and it has to be true that $RR' < RP'$. By the triangular inequality, $PR' \leq PR + RR' < PR + RP' = PP'$. This means that the boundary point R' is closer to P than P', which contradicts the definition of P'. Therefore, the medial axis cannot contain a cycle that does not contain a corner.

We can consider the medial axis to be a real tree if we consider concave corners to be multiple leafs; one leaf for each incoming edge of the medial axis (see figure 3). Note also that every vertex at the outside of the polygon is adjacent to an edge of the medial axis.

Thus, the cutting tree that we want to find is a subtree of a special (planar) graph: we have a tree, embedded in the plane, and then all leafs of the tree are connected by a cycle, in the order in which the leafs appear in the embedding. This kind of graph is known in graph theory as a Halin graph. Halin graphs are known to have treewidth three [10] (see also [5]), and hence it is known that many problems that are NP-hard for general graphs can be solved in linear time on these graphs with a general methodology (see e.g., [1, 4]). The disadvantage of the methodology is that usually large constants are hidden in the O-notation. Fortunately, for the problem considered here, the methodology can also be used, but the hidden constants are very reasonable, as discussed later in this paper.

In order to see how to cut the polygon, we have to find a subtree of the Halin graph consisting of the edges of the polygon and the medial axis. This subtree must fulfil certain conditions. In the graph, we have the following types of edges:

- Hard boundary edges. These are edges of the polygon that must be cut, i.e., the edge must be present in the subtree.
- Boundary edges that are not hard. These edges may be cut.

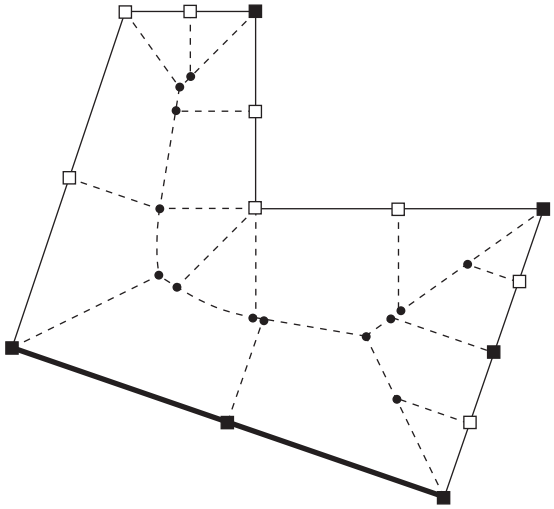

Figure 4: Types of edges and vertices. Suppose the polygon represents a piece of land, that cannot be added to a sea face which borders it from below. The figure shows hard boundary edges (fat lines), other boundary edges (thin lines), internal edges (dashed), terminals (black squares), other boundary vertices (white squares) and internal vertices (black dots).

- Edges of the medial axis. These also can be cut. These are called *internal edges*.

We have the following types of vertices:

- Vertices on the boundary of the polygon, adjacent to at least two other faces. These vertices are called the *terminals*. The subtree must span at least each of these vertices.
- Other vertices on the boundary of the polygon. The terminals and the vertices in this category are called *boundary* vertices.
- Internal vertices of the medial axis.

For an example, see figure 4.

Thus, the subtree of the Halin graph that we search for should contain all terminals and all hard edges. Note that if we replace the polygon by such a subtree, then we have a subdivision of the plane into faces again.

We first discuss how to handle hard edges. While all other edges are given a weight, equal to their normal Euclidean length, hard edges are given weight 0, and each endpoint of a hard edge is considered to be a terminal.

Note that if all edges of the polygon are hard, then there is no solution to the problem. We further assume that this degenerate case does not hold. Now, there is no cycle of weight 0 in the graph.

Given a graph $G = (V, E)$ and a set of vertices $W \subseteq V$, a tree $\mathcal{T}$ is *a Steiner tree spanning W in G*, if $\mathcal{T}$

is a subgraph of G (i.e., $\mathcal{T}$ can be obtained from G by removing vertices and edges) and every vertex $w \in W$ belongs to $\mathcal{T}$.

The following easy lemma shows that it is sufficient to find a Steiner tree of minimum total weight over all edges in the tree – every hard edge will automatically belong to the Steiner tree.

Lemma 2 *Suppose we have a graph $G = (V, E)$ with for every edge $e \in E$ a weight $l(e) \geq 0$. Suppose G has no cycle of weight 0. Let W be a set of vertices and suppose that every endpoint of an edge of weight 0 belongs to W. Then every Steiner tree spanning W in G of minimum total weight contains every edge of weight 0 in G.*

Proof: Suppose $l(e) = 0$ but e does not belong to a Steiner tree $\mathcal{T}$ spanning W, with the total weight of $\mathcal{T}$ minimum. If we add e to $\mathcal{T}$, we obtain a cycle. This cycle must have an edge e' of weight more than 0. Now $\mathcal{T} - e' + e$ is also a Steiner tree spanning W, but the total weight of the edges in $\mathcal{T} - e' + e$ is smaller than that of $\mathcal{T}$, contradiction.

Finding a minimum Steiner tree in a graph is an NP-complete problem, as was already shown by Karp [8] in 1972. However, in our case, we do not need to find a Steiner tree in an arbitrary graph, but we restrict inputs to Halin graphs. By the facts that Halin graphs have treewidth three [10, 5], and that the minimum Steiner tree problem is solvable in linear time on graphs of bounded treewidth [1], it follows directly that the problem we want to solve has a linear time algorithm. However, using the general approach from e.g. [1] would cause too large constant factors for the algorithm. A first step of such an algorithm is usually to find a tree decomposition of bounded treewidth of this input graph, but the known algorithms are complicated (see [3, 9]) and often not practical. Thus, instead, we give a tailor-made algorithm for the minimum Steiner tree on Halin graphs, exploiting the specific structure of this problem and these graphs, and avoid working explicitly with tree decompositions, using instead the medial axis tree as main structure. Our algorithm does not carry very high constant factors and can be expected to be fast in practical situations. In our case, we also may assume that all terminals are boundary vertices – this helps for slightly easier presentation and better constant factors.

3 Algorithmic solution

In this section, we describe the algorithm that solves the minimum Steiner tree problem on Halin graphs.

First note that we may assume that there are at least two terminals: if there is one terminal, then the Steiner tree is just the tree containing the one terminal; and if there is no terminal, the Steiner tree is empty. In both cases, this corresponds to the case that there is only one face around the face to be eliminated: the eliminated face is absorbed completely by it.

Thus, now we assume we have a Halin graph $G = (V, E)$, and a set $W \subset V$ of size at least two. Each edge $e \in E$ has a weight that is non-negative.

The first step of the algorithm is to transform the problem to one where every vertex has degree exactly three. Note that vertices of degree at most two cannot exist (by the manner of construction of G). We now discuss how to modify vertices of degree at least four.

Each boundary vertex v has degree 3 (convex corners and half-way vertices) or 5 (concave corners). In the latter case, the five adjacent edges are:

- a boundary edge (v, u);
- an internal edge (v, u') which is perpendicular to (v, u);
- a boundary edge (v, w);
- an internal edge (v, w') which is perpendicular to (v, w).
- an added bisector edge (v, x), which divides the angle between (v, u') and (v, w').

We replace each 5-degree vertex v by three vertices v_u, v_x and v_w and two zero-weight edges (v_u, v_x) and (v_x, v_w) that connect them. The edges coming from u and u' are now connected to v_u, the edge from x is connected to v_x, and the edges coming from w en w' are connected to v_w (see figure 3). If v was a terminal, v_u, v_x and v_w are terminals as well.

Internal vertices of degree $k > 3$ can be reduced in a similar way to $k - 2$ vertices of degree three: suppose the neighbours of vertex v are, in (e.g.) clockwise order: $w_1, \ldots, w_k$. Then split v into $v_1, \ldots, v_{k-2}$, and add edges (v_1, w_1), (v_1, w_2), (v_2, w_3), (v_3, w_4), …, (v_{k-2}, w_{k-1}), (v_{k-2}, w_k), and (v_i, v_{i+1}), $1 \leq i \leq k - 3$. Each edge (v_i, w_j) is given the weight of edge (v, w_j), and each edge (v_i, v_{i+1}) is given weight zero.

From now on, we assume that G is the graph resulting from this operation. Note that G is still a Halin graph.

We choose an arbitrary terminal t. From t there is exactly one internal edge, (t, t'). Let T be the tree obtained by taking all internal edges except (t, t'). Note that every vertex in T, except the leafs and t' has degree exactly three. t' is taken as the root of T. (see figure 5.) Each internal vertex v now has a unique well-defined left child $l(v)$, a right child $r(v)$, and a parent $p(v)$, t being the parent of t'. (I.e., t' is the only node whose parent is outside T.)

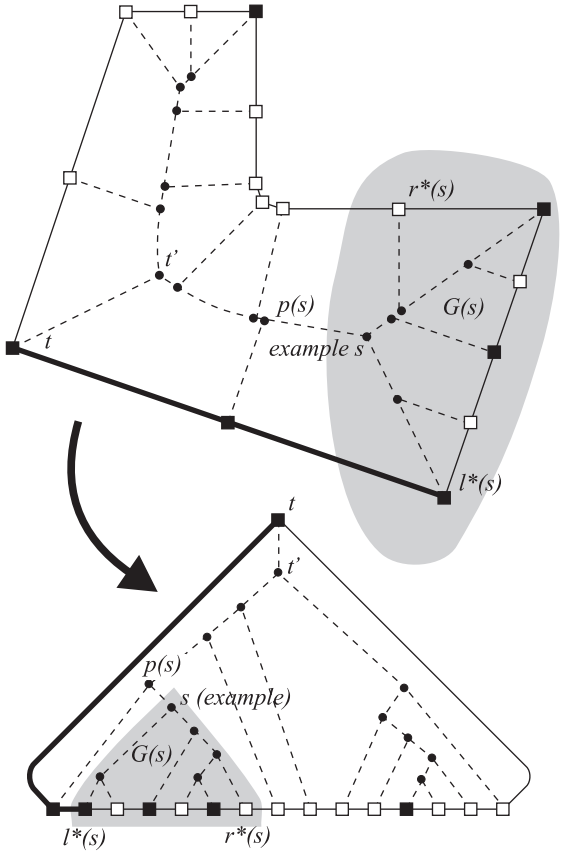

Figure 5: A polygon with its medial axis, redrawn as a tree rooted at t', a non-tree edge (t, t') and boundary edges connecting the leafs and t. Black squares are terminal vertices.

For each vertex $s \neq t$, we define the subtree $T(s)$ of T as the subtree of T with root s, i.e., it is formed by s and all descendants of s, and the subgraph $G(s)$ of G as the subgraph, consisting of $T(s)$, and all boundary edges (u, v), where u and v are leafs of $T(s)$. Going down into the subtree, starting from s, we find a unique leftmost leaf $l^*(s)$ and a unique rightmost leaf $r^*(s)$. (This can be defined inductively as follows: if s is a leaf, then $l^*(s) = r^*(s) = s$; otherwise, $l^*(s) = l^*(l(s))$ and $r^*(s) = r^*(r(s))$.)

Only three edges not in $G(s)$, are adjacent to a vertex of $G(s)$. These are:

- the internal edge $(s, p(s))$
- the boundary edges connecting $l^*(s)$ and $r^*(s)$ to their respective neighbours not in $G(s)$

We call s, $l^*(s)$ and $r^*(s)$ the top, left and right corners of $G(s)$.

If we have a steiner tree $\mathcal{T}$ in G, then the intersection of $\mathcal{T}$ with $G(s)$ forms a forest. Such a forest should meet the following conditions:

- every terminal in $G(s)$ belongs to one of the connected components of the forest;
- every component contains at least one of the corners (otherwise there would be no way to construct a path in $\mathcal{T}$ from that component to terminal t, which is outside $G(s)$).

(Components that link two or three corners, but contain no terminals, are of interest, because they may be useful as a part of a path between terminals in neighbouring subgraphs.)

Call a forest, fulfilling these conditions an S-forest in $G(s)$. Note that from the conditions above, it follows that an S-forest has at most three components. We partition the possible S-forests in $G(s)$ into classes. For each subgraph $G(s)$, there are twenty different classes of S-forests, depending on which corner vertices are connected by the components and which components contain a terminal.

The forest classes are the following (see figure 6 for an illustration).

- Each component contains exactly one corner. Each of the components can contain a terminal vertex or not, which makes $2^3 = 8$ subclasses.
- There is one component with two corners, and one component with one corner. There are a number of different cases: there are three choices which two corners can belong to the same component. Also, this component either contains a terminal or not. Then, the other component can contain a terminal or not. This would give $3 \times 2 \times 2 = 12$ subclasses.

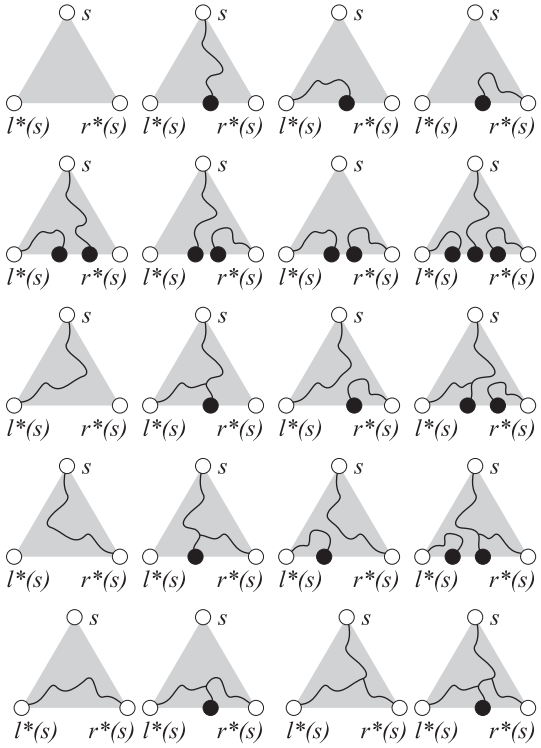

Figure 6: The 20 classes of forests, with terminals shown as black dots

However, since all terminals are boundary vertices, any path from the top corner to a terminal will cross or meet any path from the left corner to the right. Therefore there are no two-component forests with the left and right corners in one component, and the top and a terminal in another. This reduces the number of two-component classes to ten.

- There is one component containing all three corners. There are two subclasses: one for components with a terminal vertex, and one for components without a terminal.

Note that forests in the same class are equivalent with respect to their ability to connect parts of neighbouring subgraphs. They are also equivalent with respect to the need to connect certain corners to establish paths to all terminals. To be precise, if S-forests F_1 and F_2 belong to the same subclass, then if we have a Steiner tree whose intersection with $G(s)$ is F_1, we get another Steiner tree by replacing F_1 by F_2. So whenever a forest is part of a minimum Steiner tree as required, the forest must be a least weight forest in its class. Otherwise it would be possible to replace this part of the solution by an alternative forest of lesser weight.

For each vertex s, except t, and for each of the twenty classes, we calculate the minimum total weight of a S-forest in $G(s)$ in that class, if one exists.

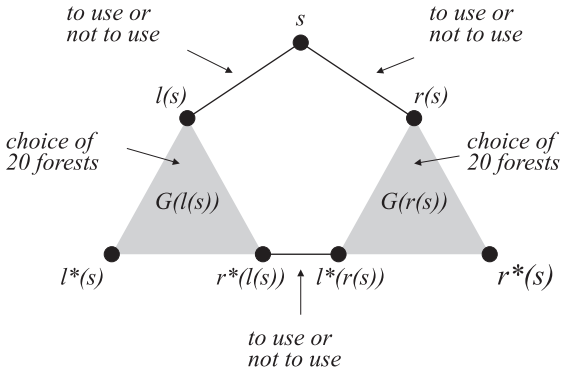

Figure 7: $20^2 \times 2^3$ possibilities to create a forest for $G(s)$

For boundary vertices (leafs) there is only one possibility. The one-component forest consisting of that sole vertex. It necessarily connects all three corners, since all corners are at the only vertex. The class of the forest depends on whether or not that vertex is a terminal. The other 19 classes are empty.

For internal vertices s, forests can be constructed by choosing:

- one of the minimal forests of $l(s)$;
- one of the minimal forests of $r(s)$;
- whether or not to include $(s, l(s))$;
- whether or not to include $(s, r(s))$;
- whether or not to include $(r^*(l(s)), l^*(r(s)))$ (this is the boundary edge that connects the subgraphs rooted at the children of s).

For each combination, we can easily determine its class (it depends directly on the five choices made above; no further information needed), and its size (assuming that the sizes of the minimal forests for $l(s)$ and $r(s)$ have already been established). We can also check easily whether the result fulfils the conditions of being an S-forest.

The five choices establish a fixed number of combinations for each node. There are 20 possibilities for the first choice, 20 for the second, and 2 for each of the others. This makes $20^2 \times 2^3 = 3200$ in total. However, 2632 combinations are useless because they contain loops, dead-ends without a terminal, components that do not contain a corner, or unconnected non-corner terminals. Tables containing the remaining 568 valid combinations will be used in the implementation, so that only valid combinations have to be checked.

For each internal vertex, we examine all valid combinations. For each of the 20 result classes, we store the following information about a minimum forest in that class for that vertex:

- its total weight;
- the five construction choices that lead to this forest.

For the computations of the total weights of the forests, we need to have the weight of the boundary edge $(r^*(l(s)), l^*(r(s)))$ quickly. In order to do this, we also maintain for every s the weight of the boundary edge, connecting $r^*(s)$ to its right neighbour vertex outside $G(s)$. This costs $O(1)$ time per vertex. Now we can conclude that an internal vertex can be handled in $O(1)$ time.

Processing the tree T bottom-up, we obtain in linear time for each of the classes the minimum total weight of a forest in the class for $G(t')$.

Each of these 20 minimal forests can be extended to a Steiner tree according to our specifications as follows. For each connected component in the forest, add the least weight edge that connects one of the corners to t. Note that all corners of $G(t')$ are adjacent to a vertex not in $G(t')$. There is only one such vertex: it is the terminal t.

After all components have been connected to t, the forests have become trees. We still have only an implicit representation of the trees, but we can construct them easily in $O(n)$ time by processing the graph top-down, following the construction choices recorded in the bottom-up phase of the algorithm.

Out of the 20 Steiner trees that can be constructed, we take the tree of minimum total weight, and output that tree.

(The number of forests that really need to be checked is actually smaller than 20. We know that there is at least one terminal in $G(t')$. Furthermore, we can forget about components containing multiple corners, but no terminals. 13 forests for $G(t')$ remain.)

Note that the medial axis has linear complexity and can be calculated in linear time [6]. The complete minimum Steiner tree algorithm as explained above, runs in $O(n)$ time as well (n being the number of vertices in the input polygon).

4 Conclusions

In this paper, we applied dynamic programming techniques, established for graphs of bounded treewidth, to give an efficient algorithm for a problem arising in map generalization. Besides the contribution made here for this map generalization problem, one can also note that a nice example is given how the methodology established for graphs of bounded treewidth (sometimes deemed of only theoretical interest) can give rise to practical algorithms for real-life applications.

An interesting open problem that remains is to find a polynomial time algorithm for the variant Steiner tree problem that was mentioned in the introduction. Given a polygon, with some vertices and some edges marked as 'hard', can we find efficiently a Steiner tree of minimum total length that connects at least all hard vertices and edges, with no part of the tree outside the polygon?

References

[1] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340, 1991.

[2] M. Bader and R. Weibel. Detecting and resolving size and proximity conflicts in the generalization of polygonal maps. In *Proc. 18th Int. Cartographic Conference*, Stockholm, 1997.

[3] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[4] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In I. Privara and P. Ruzicka, editors, *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS'97, Lecture Notes in Computer Science, volume 1295*, pages 19–36, Berlin, 1997. Springer-Verlag.

[5] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sc.*, 209:1–45, 1998.

[6] F. Chin, J. Snoeyink, and C.-A. Wang. Finding the medial axis of a simple polygon in linear time. In *Proceedings 6th Annual International Symposium on Algorithms and Computation, ISAAC '95*, pages 382–391, Berlin, 1995. Springer Verlag, Lecture Notes in Computer Science, vol. 1004.

[7] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing steiner minimal trees. *SIAM J. Appl. Math.*, 32:835–859, 1977.

[8] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85 – 104. Plenum Press, 1972.

[9] J. Matoušek and R. Thomas. Algorithms finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22, 1991.

[10] T. V. Wimer. *Linear Algorithms on k-Terminal Graphs*. PhD thesis, Dept. of Computer Science, Clemson University, 1987.