

Steiner Point Removal in Graph Metrics

Amitabh Basu, Anupam Gupta

February 18, 2008

Abstract

Given a family of graphs \mathcal{F} , and graph $G \in \mathcal{F}$ with weights on the edges, the vertices of G are partitioned into *terminals* T and Steiner nodes S . The shortest paths (according to edge weights) define a metric on the set of vertices. We wish to embed the set T in a weighted graph $G' \in \mathcal{F}$ such that the distance between any two vertices $x, y \in T$ in the graph G' is “close” to their distance in G . More precisely, does there exist a graph G' on the set T , such that for every $x, y \in T$, $d_G(x, y) \leq d_{G'}(x, y) \leq \alpha d_G(x, y)$.

We obtain results for the family of outerplanar graphs. We show that we can remove Steiner nodes from any outerplanar graph G and embed the terminals in another outerplanar graph G' with constant α . Moreover, in our algorithm, G' is a minor of G . This strictly improves the class of graphs for which Steiner point removal can be done with constant distortion. The previously best known result was for trees due to Gupta [6].

1 Introduction

We consider a problem in low distortion embeddings of finite metric spaces. We recall that a *metric space* is a pair (X, d) , where X is a set and $d : X \times X \rightarrow [0, \infty)$ is a function satisfying the following conditions: $d(x, y) = 0$ iff $x = y$, $d(x, y) = d(y, x)$, and $d(x, y) + d(y, z) \geq d(x, z)$.

We consider the specific class of graph metrics, which are the shortest path metrics on weighted graphs. We are given a family of graphs \mathcal{F} . We are further given a graph $G = (V, E) \in \mathcal{F}$. A subset T of V is denoted as the set of terminals. The problem that we study in this paper is to construct a graph G' on the vertex set T such that $G' \in \mathcal{F}$, such that the distances in G' are not too distorted from the original distances in G . To be more precise,

$$d_G(x, y) \leq d_{G'}(x, y) \leq \alpha d_G(x, y) \quad \forall x, y \in T$$

Such an embedding is said to have a distortion of α . Note that as defined above, we want a dominating metric, i.e. none of the original distances should be contracted in the new graph. We define a bunch of problems that we wish to investigate in this general framework.

1.1 List of problems

Here we formulate a list of questions which is related to this line of research of removing Steiner points from graph metrics.

1. The most general question in this line of research would be when the family \mathcal{F} is a minor-closed family of graphs. Given a graph $G \in \mathcal{F}$, embed T into a graph $G' \in \mathcal{F}$. For example, the family \mathcal{F} could be the family of planar graph metrics.
2. More specifically, can we embed a T into a minor of the original graph G ? Clearly, if the answer to this question is yes, then the question 1 is also settled positively.
3. Consider specific families of graphs. Interesting families are trees, outerplanar graphs, planar graphs, graphs with a tree-width of k , series parallel graphs. Note that graphs with bounded tree-width subsumes the family of outerplanar and series parallel graphs.
4. We can consider embeddings into a distribution over a set of graphs from the family \mathcal{F} . In this case, we provide a probability distribution π over the family \mathcal{F} , such that the support of the distribution should be graphs on the vertex set T such that

$$d_G(x, y) \leq \mathbf{E}_{G' \leftarrow \pi}[d_{G'}(x, y)] \leq \alpha d_G(x, y)$$

We may even want the graphs in the support of π to be dominating metrics over d_G restricted to T . Additionally, it should be possible to efficiently sample from the distribution π . The size of the support of π should also be polynomial sized.

5. Is it possible to embed T into a distribution of minors of the original graph with the same guarantee as the previous question ?

1.2 Previous and Related Work

Gupta [6] studied the problem on \mathcal{F} being the family of trees. He showed that given a weighted tree, one can embed any subset of its vertices T into another weighted tree such that the distortion is at most 8. This was the best known result for question 3 before our work. Concerning question 4 and 5, we can use a result due to Emek et al. [4] which says that any graph metric can be embedded into a distribution over its *spanning subtrees* with an expected distortion of $O(\log^2 n \log \log n)$. Since subtrees are minors, combined with Gupta's result on trees, we immediately get a bound of $O(\log^2 n \log \log n)$ on the distortion for questions 4 and 5. As far as we know this is all that was previously known about the list of problems we outlined in the previous section.

1.3 Our Results

We show that for the family of outerplanar graphs, we can achieve Steiner point removal with only $O(1)$ distortion. More precisely, we can embed a subset T of vertices of any weighted outerplanar graph G into another outerplanar graph G' with $O(1)$ distortion. In fact, in our algorithm, G' is actually a minor of G . We present the results building up in complexity.

- We first show how to do Steiner point removal for the family of unweighted outerplanar graphs. This is done via a reduction of unweighted outerplanar graphs into a class of graphs that we call “tree of cycles”. This reduction is shown in subsection 1.5.
- Next we show how to do Steiner point removal in the family of “tree of cycles” such that the final graph is actually a *minor* of the original graph. This is explained in section 2.
- Finally we show how to handle general weighted outerplanar graphs. For this case too, we embed into a minor of the original graph. Section 3 shows how this is done.

All of our results achieve $O(1)$ distortion.

1.4 Applications

Steiner point removal in trees arises in the relevant and interesting problem of implementing Internet multicasting using unicasts. In a network, we typically have hosts and routers connected via physical links. For multicasting, a routing tree is defined on these hosts and routers, which is a spanning tree on the network. Once a host initiates a multicast, the packets are transmitted along the edges of this tree according to the routing protocol. The restriction of a tree helps to keep the protocols simple. For various reasons, several researchers [3, 5, 2] have argued that a virtual tree defined on just the hosts themselves would be very useful. Such a virtual tree of course should satisfy certain desirable properties; for example, the link distances in the virtual tree should not be drastically different from the original communication distance between hosts. If we imagine the hosts to be the set T in the Steiner point removal problem, then this is exactly what we want to achieve.

1.5 Definitions and Preliminaries

Notation : T is the set of terminals. $S = V \setminus T$ is the set of Steiner points, i.e. non-terminals. d denotes the distances in the original graph G . d^* denotes distances in the new graph G' . For any graph G , $n(G)$ denotes the number of vertices of G . $ecc(v)$ is the *eccentricity* of $v \in V$ and is defined as $\max_{x \neq v} d(v, x)$. $rad(G)$ is the radius of G . Radius is defined as $\min_{v \in V} ecc(v)$. The *center* of G is a vertex with minimum eccentricity.

1.5.1 Definitions

We will be using a special family of graphs in our discussion below, which is a proper subset of the family of outerplanar graphs. We refer to a graph from this family as a “tree of cycles”. A *tree of cycles* is a graph defined recursively as follows.

1. A single cycle is a tree of cycles.
2. Given a tree of cycles \mathcal{T} and cycle C , the graph formed by identifying a vertex of C with any vertex in \mathcal{T} is a tree of cycles.

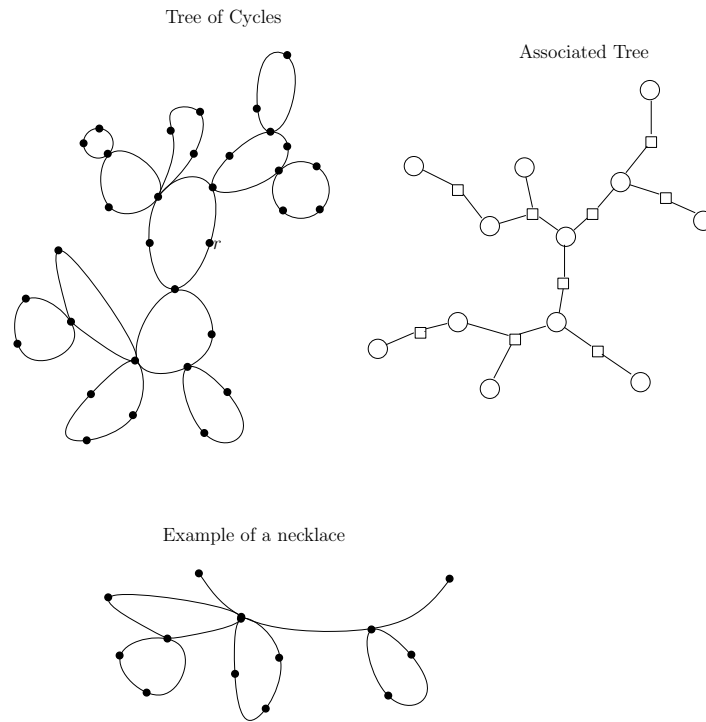


Figure 1: Tree of Cycles, its associated tree and Necklace formed by a cut

If we define meta-vertices for every cycle and for every cut-vertex in the tree of cycles, and add meta-edges between every cycle-vertex to the cut vertices on it, we should get a tree. See Figure 2.

A *necklace* is a path with a tree of cycles attached at each vertex on the path except the end-points. The end-points of a necklace will usually be referred to by the letters a and b . The path connecting a and b will be referred to as the *backbone* of the necklace.

For any Steiner point r , $C(r)$ is defined as the nearest terminal in T to it, breaking ties arbitrarily. Given a necklace N , let the backbone of N be $a, r_1, r_2 \dots r_k, b$. Let $C(r_i)$ be the closest terminal to r_i in the tree of cycles attached to r_i .

1.5.2 Preliminaries

Tree of cycles are useful because unweighted outerplanar graphs embed easily into tree of cycles. Tree of cycles were first studied by Bateni et al. [1] who observed the following.

Lemma 1.1 *Any unweighted outerplanar graph G embeds into a tree of cycles with $O(1)$ distortion.*

If we can remove Steiner points from the family of tree of cycles with distortion D , we would be able to do Steiner point removal for unweighted outerplanar graphs with distortion $O(D)$. As it turns out we will show that for the family of tree of cycles, the distortion is $O(1)$. For the sake of completeness, we briefly describe the proof of the above lemma.

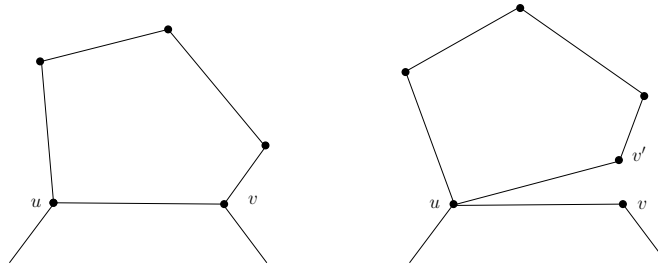


Figure 2: Converting an unweighted outerplanar graph into a tree of cycles

Proof: Given an outerplanar graph G , consider the dual graph of any outerplanar embedding of G . If we ignore the vertex corresponding to the outer face, the dual graph is a tree. Consider any leaf of this tree and the face in G corresponding to this leaf. Now imagine cutting along the edge u, v between this face and the rest of the graph from u to v . The face will now hinge off u with two copies of v in the new graph. Now ignoring this cycle which is hanging off u , recursively resolve the remaining graph. What we get finally is a tree of cycles. To see that distances were only distorted by a constant factor, note that any shortest path in the original graph which uses a duplicated vertex, now travels an additional 2 for each duplicated vertex. So the distance is changed by at most 3 times. \square

In conclusion, the total distortion we incur for resolving unweighted outerplanar graphs will be $O(1)$.

We now state some useful observations about the problem of removing Steiner points from graph metrics.

Observation 1.2 *If the set of Steiner points S is an independent set in G , then we can embed T into a minor of G with a distortion of at most 2.*

Proof: Consider any Steiner point s . By the hypothesis, all of its neighbours are terminals. Now take the shortest edge (say e) incident on s and contract it. On the other edges that were incident on s , put an additional weight of $w(e)$. Do this for all steiner points in G . Clearly, the new graph is a minor of G since all we did were edge contractions. Now note that the weight of any edge increased by at most a factor of 2. Therefore, $d^*(x, y) \leq 2d(x, y)$ for every $x, y \in T$. \square

Observation 1.3 *Let C_1, \dots, C_k be the connected components of $G \setminus T$ (i.e. the graph resulting from removing the terminals). Then we can embed T into a minor of G with distortion $O(\max_j \{\min\{n(C_j) - 1, 2rad(C_j)\})$.*

Proof: Consider a particular C_j and all the terminals $T' \subset T$ such that there is an edge between some terminal in T' and a vertex in C_j . (The set T' contains what one might visualize as the “boundary” terminals of C_j .) Create a minimum spanning tree on $C_j \cup T'$ or a shortest path tree from the center of this graph, whichever has smaller expansion on the distances between the vertices of this graph. It can be easily seen that the expansion incurred by doing this is at most $\min\{n(C_j) - 1, 2rad(C_j)\}$ and there is no contraction. Using Gupta’s [6] algorithm on this tree, we can embed T' into a minor of C_j with an additional expansion of 8. Now some vertices in T might occur on the boundary of more than one component. Simply identify all these copies of the vertex.

Consider any path from x to y in the original graph G , with $x, y \in T$. Let the path be $x = v_1, v_2, \dots, v_k = y$. Let the terminals be $x = v_{i_1}, v_{i_2}, \dots, v_{i_k} = y$. As noted earlier, for each part of the path between successive terminals $v_{i_j}, v_{i_{j+1}}$, the expansion is at most $8 \min\{n(C_j) - 1, 2rad(C_j)\}$, where C_j is the component which contains the path from v_{i_j} to $v_{i_{j+1}}$. So the total expansion of the path is $8 \max_j \{\min\{n(C_j) - 1, 2rad(C_j)\}\}$.

\square

2 Minor-closed Algorithm for Tree of Cycles

We present an algorithm which takes a tree of cycles and gives a minor of the graph with all the Steiner points removed and with low distortion. These ideas of embedding into a minor will be useful in the case of general outerplanar graphs. It should be noted that the following algorithm works on a tree of cycles with positive weights on the edges. For unweighted outerplanar graphs, this is not relevant. However, since these ideas are used for general outerplanar graphs, and for the sake of completeness, it needs to be pointed out that the algorithm can deal with weighted edges.

2.1 Preprocessing

We make a quick note that it is enough to consider graphs, where none of the terminals are cut-vertices (that is, vertices whose removal disconnects the tree of cycles). If there are such vertices, we use the idea behind Observation 1.3, to work on the components formed by removing the terminals. When we throw back the “boundary” terminals (the set T' defined in the proof of Observation 1.3), none of the terminals will be cut-vertices.

2.2 The Algorithm

There are two basic routines. **Main_routine(r)** is a recursive procedure, which breaks the graph into smaller subgraphs, recursively removes Steiner vertices from these smaller subgraphs and then stitches these new graphs to give the solution. It uses the second routine **Resolve_necklace(N)** to stitch the smaller solved graphs.

Main_routine(r) is invoked with some Steiner vertex r as a parameter. The inductive hypothesis for **Main_routine(r)** is that it returns a graph in which the *only* Steiner vertex is r . All other Steiner vertices have been removed. We show how to remove this final vertex in the analysis of the algorithm.

Main_routine(r)

1. If the graph is a single terminal node r , simply return the same node. Else do the following steps
2. Do a breadth first search from r and determine $C(r)$ and $d(r, C(r))$.
3. Make a “cut” from r at a distance of $\frac{1}{2}d(r, C(r))$, resulting in necklaces $N_1 \dots N_k$. This cut might be going through some edges in the graph. Introduce new Steiner vertices at these points. See Figure 3 for an illustration. In the diagram, vertices a_1, a_2, b_2, b_3, a_4 are examples of vertices which were introduced because the edges were cut.
4. For each necklace N_i , call **Main_routine** recursively with the roots of the trees hanging off of it and attach these new graphs back onto the necklace
5. Call procedure **Resolve_necklace(N_i)**, for each necklace N_i . Let the resolved necklace be N'_i
6. Do the following for each resolved necklace N'_i
7. Attach the two free edges at the ends of the backbone of N'_i to a single node. This is the copy of r in the new graph. This final operation can be thought of as contraction of the entire subgraph which fell *within* the cut made from r , into one single node r .

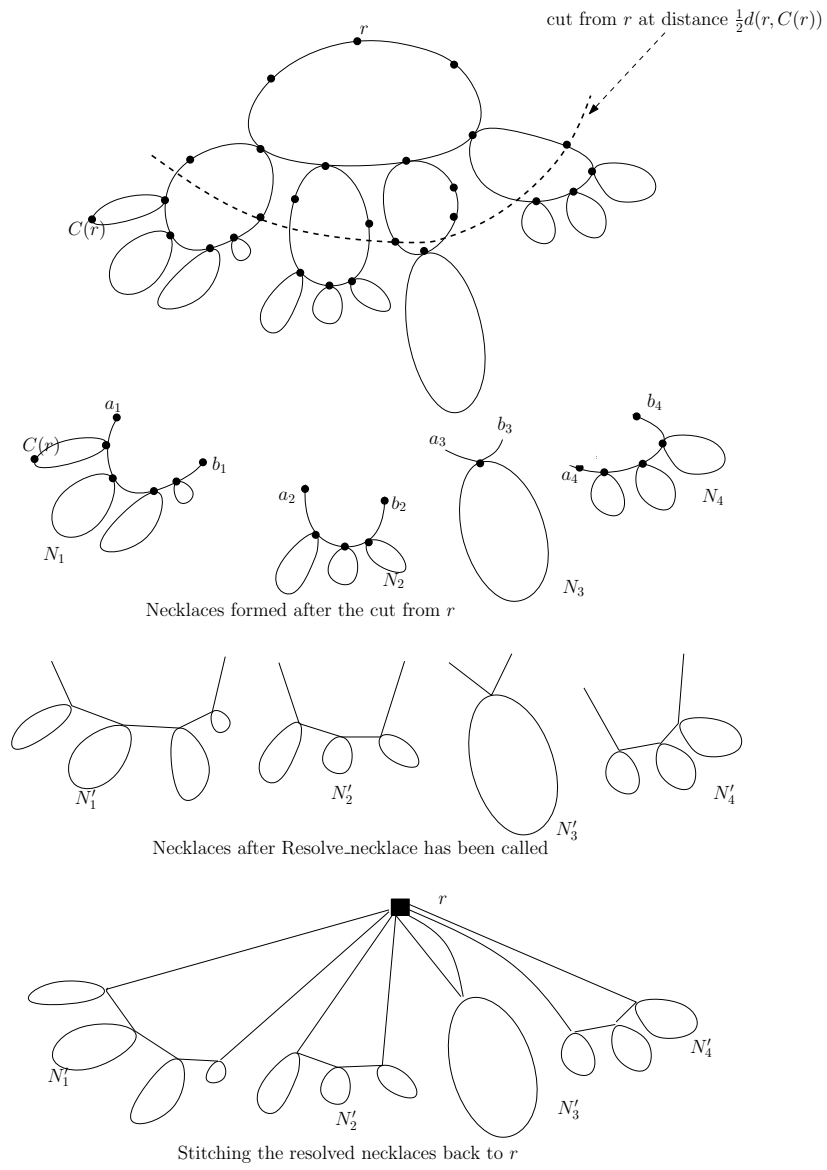


Figure 3: Different stages of the algorithm

In the next procedure, we impose the natural ordering on the trees in the necklace as the order in which their roots occur on the backbone. Note that after recursively solving the subtrees hanging off it, some distances (the ones on the backbone) are original distances d and some distances (in the solved subtrees) are new distances d^* . To refer to this hybrid of distances, we use the notation \hat{d} . The next procedure removes the Steiner vertices in this hybrid necklace, which are the roots of the trees hanging off the backbone (recall that, apart from the end-points of the backbone, these are the only Steiner vertices in the necklace now, by the induction hypothesis of **Main_routine**), and returns a necklace with only the end-points as Steiner vertices. Also, for any terminal t in the necklace, let $d_{frontier}(t) = \min\{d(t, a), d(t, b)\}$, which is the minimum distance to get from t to the “frontier” (i.e. a or b) of the cut, in the original graph.

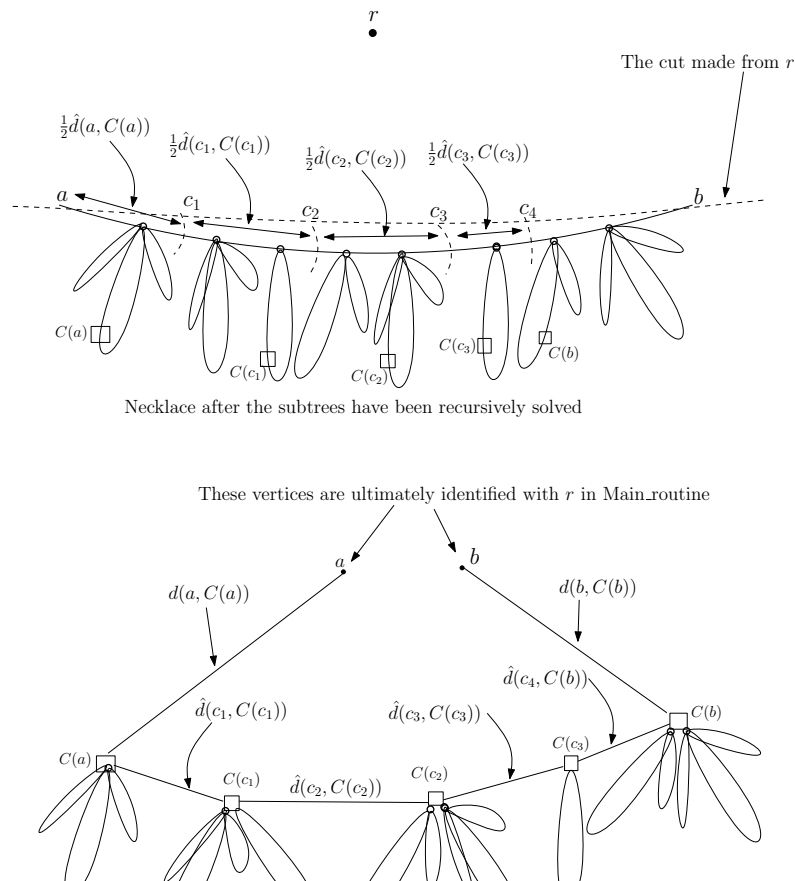


Figure 4: How `Resolve_necklace` works

See Figure 4 for an illustration of the above procedure.

Resolve_necklace(N)

1. Make a cut from a of length $\frac{1}{2}\hat{d}(a, C(a))$ on the backbone, where $C(a)$ is the nearest terminal on the backbone to a , under the hybrid distances \hat{d} . Say it cuts the backbone of N at c_1 .
2. Make a series of cuts in succession, cut c_{i+1} is made from cut c_i , of length $\frac{1}{2}\hat{d}(c_i, C(c_i))$, where $C(c_i)$ is the closest terminal to c_i amongst the trees hanging off the part of the backbone from c_i to b . Stop when either you fall off the end of the backbone, or you are about to cut off beyond the root containing $C(b)$, where $C(b)$ is the terminal closest to b under the hybrid distance \hat{d} . In the second case, put the final cut just before the root containing $C(b)$. Let the last cut be labeled c_m .
3. Consider the section of the backbone between two consecutive cuts made in the previous step, say c_i, c_{i+1} , $i = 1, \dots, m - 1$. Identify all the roots that fall within the cuts c_i, c_{i+1} and contract the edge between $C(c_i)$ and its root. Also identify the roots between a and c_1 , and contract the edge between $C(a)$ and its root. Similarly, identify all the roots between c_m and b and contract the edge between $C(b)$ and its root.
4. On the edge between $C(a)$ and $C(c_1)$, put a weight of $\hat{d}(c_1, C(c_1))$. On the edge of the backbone between $C(c_i)$ and $C(c_{i+1})$, for $i = 1, \dots, m - 1$, put a weight of $\hat{d}(c_{i+1}, C(c_{i+1}))$. On the edge between a and $C(a)$, put a weight of $d_{frontier}(C(a))$. Similarly, on the edge between b and $C(b)$, put a weight of $d_{frontier}(C(b))$. We finally have an edge remaining between $C(c_m)$ and $C(b)$. Put a weight of $\hat{d}(r_b, C(b))$ on this edge.

2.3 Analysis

As outlined in the previous section, the analysis relies on the inductive principle that the graph returned by **Main_routine**(\mathbf{r}) has only one Steiner node left in it, which is the vertex r , from where the routine was called and the BFS was started. We will show that the new graph has $O(1)$ distortion with respect to the original graph on which **Main_routine**(\mathbf{r}) was run. At the final level of recursion, we would obtain a graph with only one Steiner node. Using Observation 1.2, we can remove this vertex with only a distortion of at most 2 more. This keeps the distortion $O(1)$.

First we note a simple fact about the edges that are added from $C(a)$ and $C(b)$ to r for each necklace in the new graph. Note that these edges were joined to r in the last step of the main routine. There is a minor detail to be noted here. Note that the tree with $C(a)$ is always to the left of the tree with $C(b)$ (in the ordering assumed for each necklace). So, it is never the case that the tree with $C(a)$ got contracted into the tree with $C(b)$ during `Resolve_necklace`.

Fact 2.1 $\frac{1}{2}d(C(t), r) \leq d^*(C(t), r) \leq d(C(t), r)$ where t is an endpoint of the backbone of any necklace resulting from the cut from r .

Proof: Note that by construction, $d^*(C(t), r) = d_{\text{frontier}}(C(t))$. Also note that $d(C(t), r) = d_{\text{frontier}}(C(t)) + \frac{1}{2}d(r, C(r))$. So clearly, the second inequality is true. Also, note that $C(r)$ is closer to r than $C(t)$, so $d(C(r), r) \leq d(C(t), r) = d_{\text{frontier}}(C(t)) + \frac{1}{2}d(r, C(r))$. So $\frac{1}{2}d(r, C(r)) \leq d_{\text{frontier}}(C(t))$. Therefore, since $d(C(t), r) = d_{\text{frontier}}(C(t)) + \frac{1}{2}d(r, C(r))$, we get $d(C(t), r) \leq 2d_{\text{frontier}}(C(t))$. \square

We now make the following claim about this new algorithm.

Claim 2.2 *The graph returned by main_routine has the property that*

$$d^*(x, r) \leq 4d(x, r) - 2d(r, C(r))$$

Proof:

We prove this by induction on the recursion tree of `Main_routine`. The base case is when the graph is a single terminal, and there is nothing to prove. Note that the base case for `main_routine` is always a single terminal because of the preprocessing we did. Let us consider a graph where a cut was made and the graph was resolved as outlined in `main_routine`.

Say x is in necklace N with end-points a and b . Number the cuts that were made in `Resolve_necklace` as $c_1, c_2 \dots c_m$. Say x is in the tree between cuts c_i and c_{i+1} , rooted at r' . Also assume that the shortest path from x to r (in the original graph) is through a . There are two cases to consider :

Case 1 : r' is before the tree containing $C(b)$ in the natural ordering. In the new graph, by construction,

$$d^*(x, r) = d^*(x, r') + \hat{d}(C(c_i), c_i) + \sum_{k=1}^{i-1} \hat{d}(c_k, C(c_k)) + d(a, C(a)) \quad (1)$$

Now the edges that bridge the cuts $c_1 \dots c_i$ are of total weight $\sum_{k=1}^{i-1} \hat{d}(c_k, C(c_k))$ which is at most $2d(c_1, c_i)$, because $\hat{d}(c_k, C(c_k)) = 2d(c_k, c_{k+1})$. See Figure 4. We also have the final edge from $C(a)$ to r of length $d(a, C(a))$. Also $d(a, c_1) = \frac{1}{2}\hat{d}(a, C(a)) \geq \frac{1}{4}d(a, C(a))$, using fact 2.1. Fact 2.1 also gives $\hat{d}(C(c_i), c_i) \leq d(C(c_i), c_i)$. Plugging all these bounds into Equation (1), we get

$$\begin{aligned}
d^*(x, r) &\leq d^*(x, r') + d(C(c_i), c_i) + 2d(c_i, c_1) + 4d(c_1, a) \\
&\leq 4d(x, r') - 2d(r', C(r')) + d(C(c_i), c_i) + 4d(c_i, a) \\
&\leq 4d(x, r') - 2d(r', C(r')) + d(C(r'), c_i) + 4d(c_i, a) \\
&= 4d(x, r') - 2d(r', C(r')) + d(C(r'), r') + d(r', c_i) + 4d(c_i, a) \\
&= 4d(x, r') - d(r', C(r')) - 3d(r', c_i) + 4d(r', a) \\
&\leq 4d(x, a) - d(r', C(r')) - 3d(r', c_i) \\
&\leq 4d(x, a) \\
&\leq 4[d(x, r) - d(a, r)] \\
&= 4d(x, r) - 2d(C(r), r)
\end{aligned}$$

Case 2 : r' comes after the tree containing $C(b)$ in the natural ordering. In this case, we added 2 edges for bridging the last cut made. That is, to bridge the cut c_{m-1}, c_m , we used edges of weight $\hat{d}(C(b), r_b)$ and $\hat{d}(C(c_{m-1}), c_{m-1})$. The final thing to note is that $\hat{d}(C(b), r_b) \leq \hat{d}(C(r'), r')$ and $\hat{d}(C(c_{m-1}), c_{m-1}) \leq \hat{d}(C(r'), c_{m-1})$. Using this fact, we can carry out a very similar calculation as above and get the claim (in fact the reader might note that the extra slack term of $d(r', C(r'))$ in the sixth line of the calculation above will now cancel the extra term that we have due to $\hat{d}(C(b), r_b)$).

When the shortest path from x to r goes through b , there are some very similar cases to handle. The argument then takes the cuts $c_i \dots c_m$ and argues similarly, where c_i is the cut which is crossed first when going from x to b .

□

Now we turn to bound the expansion in the graph.

Theorem 2.3 For any two terminal x, y , $d^*(x, y) \leq 4d(x, y)$

Proof: We again proceed by induction on the recursion tree of Main_routine. If x and y are in the same subtree of a necklace, we can use the induction hypothesis, and distances are only contracted further during Resolve_necklace (when a particular terminal is promoted to become the representative for its root in Step 3).

Now we look at the case when x and y are not in the same subtree of a necklace. There are 2 cases to consider.

Case 1 : x and y are in different necklaces. WLOG, assume x and y both go through a (of their corresponding necklaces - call them x_a and y_a) to get to r . Then, $d^*(x, y) = d^*(x, r) + d^*(y, r)$. Using Claim 2.2, $d^*(x, y) \leq 4d(x, r) - 2d(C(r), r) + 4d(y, r) - 2d(C(r), r) = 4[d(x, x_a) + \frac{1}{2}d(C(r), r)] - 2d(C(r), r) + 4[d(y, y_a) + \frac{1}{2}d(C(r), r)] - 2d(C(r), r) = 4(d(x, x_a) + d(y, y_a))$. Now note that $d(x, x_a) + d(y, y_a)$ is a lower bound on $d(x, y)$. So we are done.

Case 2 : x and y are in the same necklace, but in different subtrees. There are two subcases.

Case 2a : The shortest path from x to y uses the backbone. Let the roots of the trees for x and y be r' and r'' . In the final graph, consider path from x to y which goes x, r', r'', y . So the distance traveled is $d^*(x, r') + d_{bridge} + d^*(r'', y)$, where d_{bridge} is the distance in the resolved necklace to go from r' to r'' . (Technically, after `Resolve_necklace`, r' and r'' are no longer present, but we look the nodes that they were contracted into). Now by a similar argument as the proof of the claim, we can show that d_{bridge} is at most $2d(r', r'')$. Furthermore, using the claim to bound $d^*(x, r')$ and $d^*(r'', y)$, we get the desired bound.

Case 2b : The shortest path from x to y in the original graph is going from x to a , then to b and then coming to y . Note that this also means that the shortest path from x to r is through a and from y to r is through b . But in the final graph, we have a path going from x to r to y which is of length $d^*(x, r) + d^*(y, r) \leq 4d(x, r) - 2d(r, C(r)) + 4d(y, r) - 2d(r, C(r)) = 4[d(x, a) + \frac{1}{2}d(r, C(r))] - 2d(r, C(r)) + 4[d(y, b) + \frac{1}{2}d(r, C(r))] - 2d(r, C(r))$ (since the shortest paths from x and y to r go through a and b respectively). Simplifying, we get $d^*(x, r) + d^*(y, r) \leq 4[d(x, a) + d(y, b)]$. And then we again note that $d(x, y) \geq d(x, a) + d(y, b)$ as the shortest path from x to y in the original graph is $x \rightsquigarrow a \rightsquigarrow b \rightsquigarrow y$. \square

The next theorem bounds the contraction incurred in the algorithm.

Theorem 2.4 For any two terminals x and y , $\frac{1}{7}d(x, y) \leq d^*(x, y)$

Proof: We bound the expansion for the edges of the new graph, i.e the expansion when going from d^* to d . The edges that exist in the final graph are of two types.

Type 1 : Edges between $C(a)$ or $C(b)$ of a necklace and r (the root from which we make the cut to obtain the necklace). Because of Fact 2.1, $d(C(a), r) \leq 2d^*(C(a), r)$ and so the expansion for these edges is at most 2.

Type 2 : Edges created during `Resolve_necklace`. Again there are two kinds of edges.

Type 2a) Edges created in Step 3 of `Resolve_necklace`, while contracting all the roots falling within two consecutive cuts. Consider the segment between c_i, c_{i+1} . The edges that come from processing this cut are between $C(c_i)$ and terminals in the trees contracted into $C(c_i)$. Consider some such terminal x . The weight of the edge between x and $C(c_i)$ is $d^*(x, x_r)$, where x_r is the root of the tree containing x . Note that this edge is actually of Type 1, with respect to the previous level of recursion of `Main_routine`. Therefore,

$$d(x, x_r) \leq 2d^*(x, x_r) \quad (2)$$

The maximum distance between $C(c_i)$ and x in the original graph can be at most

$$d(x, x_r) + d(x_r, c_i) + d(c_i, C(c_i)) \quad (3)$$

Now since x_r fell within c_i, c_{i+1} ,

$$d(x_r, c_i) \leq \frac{1}{2}\hat{d}(c_i, C(c_i))$$

Moreover, since $C(c_i)$ is closer to c_i than x (in terms of \hat{d}),

$$\begin{aligned}\hat{d}(c_i, C(c_i)) &\leq \hat{d}(c_i, x) \\ &= d(c_i, x_r) + d^*(x, x_r) \\ &\leq \frac{1}{2}\hat{d}(c_i, C(c_i)) + d^*(x, x_r) \\ \Rightarrow \frac{1}{2}\hat{d}(c_i, C(c_i)) &\leq d^*(x, x_r)\end{aligned}$$

Using Fact 2.1 again, its easy to show $d(c_i, C(c_i)) \leq 2\hat{d}(c_i, C(c_i))$, and so the maximum distance from (3) becomes at most $d(x, x_r) + \frac{5}{2}\hat{d}(c_i, C(c_i))$. Using Equation(2) and the inequality derived in the previous paragraph, we get the maximum distance in the original graph is at most $2d^*(x, x_r) + 5d^*(x, x_r) = 7d^*(x, x_r)$. Recall that the weight on the edge being considered was exactly $d^*(x, x_r)$. So we get an expansion factor of at most 7 when mapping back to the original graph.

Type 2b) : These are the edges added between $C(c_i)$ and $C(c_{i+1})$, $i = 1, \dots, m - 1$. The weight on such an edge is $\hat{d}(c_{i+1}, C(c_{i+1}))$. The maximum distance in the original graph between $C(c_i)$ and $C(c_{i+1})$ can be at most

$$d(C(c_i), c_i) + d(c_i, c_{i+1}) + d(c_{i+1}, C(c_{i+1})) \quad (4)$$

By construction,

$$d(c_i, c_{i+1}) = \frac{1}{2}\hat{d}(c_i, C(c_i)) \quad (5)$$

Also, note that since $C(c_i)$ is closer to c_i than $C(c_{i+1})$, $\hat{d}(c_i, C(c_i)) \leq \hat{d}(c_i, C(c_{i+1}))$. So

$$\begin{aligned}\hat{d}(c_i, C(c_i)) &\leq \hat{d}(c_i, c_{i+1}) + \hat{d}(c_{i+1}, C(c_{i+1})) \\ \Rightarrow \frac{1}{2}\hat{d}(c_i, C(c_i)) &\leq \hat{d}(c_{i+1}, C(c_{i+1}))\end{aligned}$$

Putting this and (5) into (4), we get the maximum distance in the original graph between $C(c_i)$ and $C(c_{i+1})$ is at most

$$2\hat{d}(c_{i+1}, C(c_{i+1})) + \hat{d}(c_{i+1}, C(c_{i+1})) + d(c_{i+1}, C(c_{i+1})) \leq 5\hat{d}(c_{i+1}, C(c_{i+1}))$$

The last inequality follows because Fact 2.1 implies $d(c_{i+1}, C(c_{i+1})) \leq 2\hat{d}(c_{i+1}, C(c_{i+1}))$. So in this case the expansion is at most 5.

□

Combining Theorem 2.3 and Theorem 2.4, we conclude that the overall distortion is $O(1)$. It is also fairly easy to see that the new graph is a minor of the original graph.

Theorem 2.5 *The final graph is a minor of the original graph.*

Proof: Since all operations that were done were edge contractions, the final graph by construction is a minor. □

Remark 2.6 *It should be noted that even though the algorithm in this section embeds T into a minor of the original tree of cycles, the unweighted outerplanar graph construction does not guarantee a minor because the conversion to a tree of cycles is not minor-preserving. However, the ideas in this section are very similar, and therefore relevant, to the ones we use for general outerplanar graphs, where we do embed into minors. Moreover, we remind the reader that the above algorithm works for weighted as well as unweighted tree of cycles.*

3 General Outerplanar Graphs

We now show how to do Steiner point removal for outerplanar graphs. As in the previous section, we give an algorithm to embed T into a minor of the original graph G .

By 1.2 we can assume that there is at least one edge in the graph with Steiner points on both end-points.

3.1 The Algorithm

For general outerplanar graphs, we again proceed in a similar manner as a tree of cycles. However, instead of making a cut from a single vertex r as earlier, we now make “cuts” from an edge with Steiner vertices as end-points. In the following, a, b are Steiner points connected by an edge from where we make a “cut” in the graph.

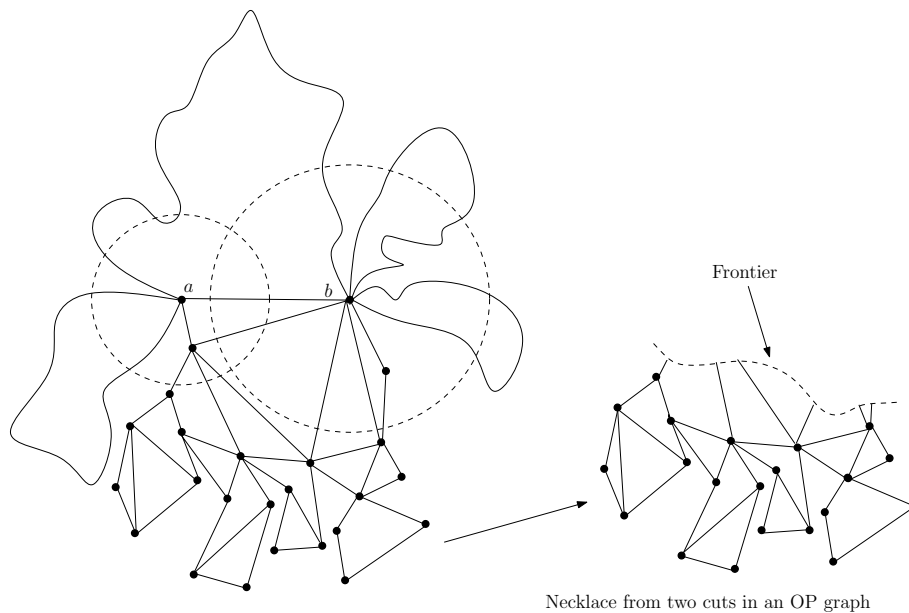


Figure 5: Necklace formed by cuts in general outerplanar graphs

More formally, we define a “cut” from an edge a, b as follows. Given two distances, d_a and d_b , do a breadth first search from a up to a distance of d_a and from b up to a distance of d_b . As

before, the BFS might end in the middle of some edges like in the tree of cycles, because we go out *exactly* d_a and d_b . As before, put a Steiner vertex at the point where the BFS “cut” ends if the BFS ends in the middle of an edge. Now remove all vertices and edges (and partial edges) that fall within the union of the two BFS cuts. We will have a bunch of connected components left. These components are graphs similar in structure to the necklaces defined earlier. See Figure 5.

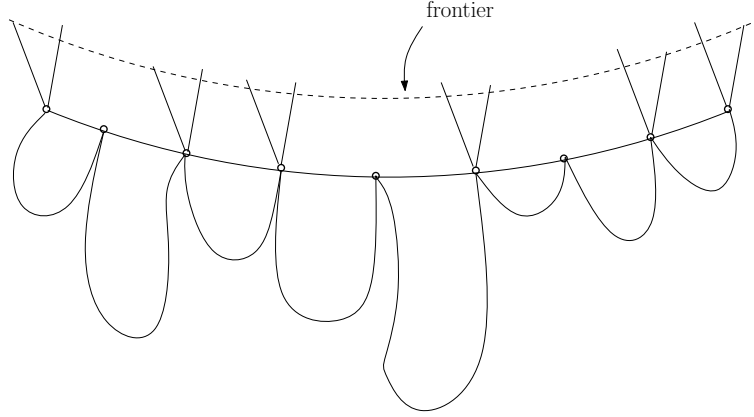


Figure 6: General structure of the new necklaces

We now define the new kind of necklace more precisely. We have a backbone (i.e. a path) as in the earlier case. The main difference is that we have outerplanar subgraphs of the original graph hanging off each *edge* of the backbone; whereas, in the necklaces of the earlier section, the subgraphs were hanging off the vertices of the backbone. Moreover, we might have edges attached to internal vertices of the backbone also (the edges that were cut partially), and not just at the end-points as earlier. Such edges will be referred to as *stitch edges*. So the necklaces in the tree of cycles had only 2 stitch edges.

Abusing notation, we will refer to these graphs as necklaces throughout the rest of this section. See Figure 6.

The points on the cut that was made will be referred to as the *frontier* in the necklaces.

As before, we impose the natural left-right ordering of the vertices on the backbone of the necklace. Moreover, d , d^* and \hat{d} refer to the same distances as before in the previous section. \hat{d} refers to the hybrid distances after recursively solving the subgraphs hanging off the necklaces.

3.2 Analysis

3.2.1 Minor closed property

In this subsection, we show that the graph returned by `Main_routine` is a minor of the original graph.

Lemma 3.1 *At the end of `main_routine`, there are only 2 Steiner points left in the graph, which are the end-points of the edge from which the cut was made.*

Main_routine($e = \{a, b\}$)

1. Make a cut of size $\frac{1}{2}d(a, C(a))$ from a and a cut of size $\frac{1}{2}d(b, C(b))$ from b as in the previous algorithms. Break ties by choosing a over b , i.e. if a point is on the frontier of both the cut from a and b , assume it is on the cut from a .
2. Consider each necklace formed from the above step in turn. Call **Main_routine** recursively to solve the outerplanar graphs hanging off the backbone of the necklace
3. Call **Resolve_necklace** for each of the necklaces with the cleaned up graphs
4. After **Resolve_necklace**, each necklace returns with edges which need to be attached to a or b . Attach them to their corresponding vertex.

Resolve_necklace(N)

1. Start from the left-most vertex on the backbone and make cuts of length half of the distance to the nearest terminal. This is identical to the previous algorithm for tree of cycles.
2. The difference from the previous algorithm is that when we look for the nearest terminal from a cut, we look at the entire backbone and not just in the part that is left after the cut. Say we made a cut from c_i at c_{i+1} . When looking for the nearest terminal to c_{i+1} , we might find it in one of the graphs which hangs off between c_i and c_{i+1} .
3. As before, we contract all the vertices on the backbone within cuts and promote the nearest terminal to the cut-point. However, now there is a difference because some of the nearest terminals might be in a graph which needs to be contracted into the previous cut. To take care of this problem we use a trivial fact when we are dealing with the vertices on the backbone between two cuts, say c_i and c_{i+1} . If both $C(c_i)$ and $C(c_{i+1})$ are in graphs hanging off between c_i and c_{i+1} , $C(c_i)$ will be to the left of $C(c_{i+1})$. In this situation, we contract together only those vertices which occur before the vertices of the graph containing $C(c_{i+1})$ and the “left” vertex of the graph containing $C(c_{i+1})$ and promote $C(c_i)$ as their representative. The remaining vertices between c_i and c_{i+1} are contracted together with $C(c_{i+1})$ as their representative. See Figure 7.
4. Finally we see if the vertices on the backbone that were contracted into $C(c_i)$ for some i had stitch edges attached to them. If yes, edges are added on $C(c_i)$ of length equal to its minimum distance (in the original graph) to the frontier through these stitch edges. These are the edges that are finally attached to a or b depending on whether the cut through which $C(c_i)$ reaches the frontier was made by a or by b .

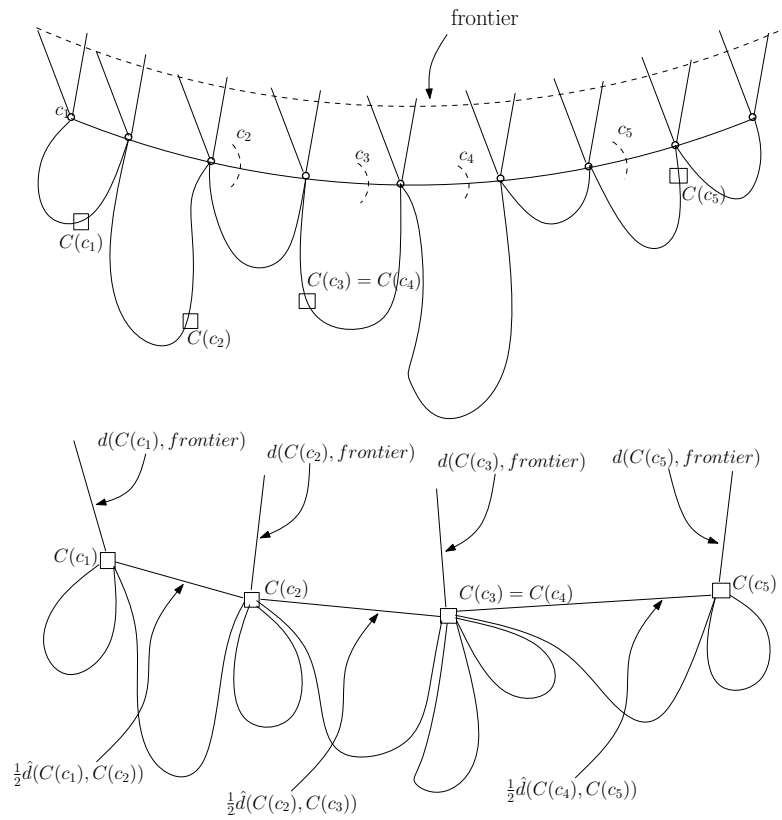


Figure 7: The Resolve_necklace procedure

Proof: By induction on the recursion tree of the algorithm. Before `Resolve_necklace` is invoked, the only Steiner points in a necklace are the ones on the backbone, by the inductive hypothesis. After `resolve_necklace`, all these are removed. So the only steiner points that are left are a and b . \square

We now prove some structural lemmas about necklaces. These will ultimately help us to show that the new graph is a minor of the original graph. Note that the frontier of a necklace consists of points which are cut by a and points which are cut by b .

Lemma 3.2 *On the frontier of a necklace, all the points that are cut by a are consecutive and all the points cut by b are also consecutive.*

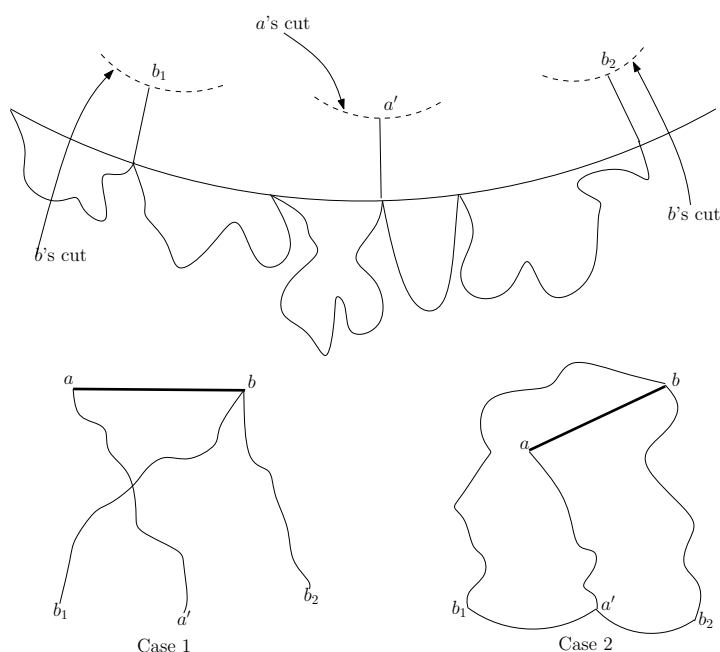


Figure 8: Structure of Necklaces

Proof: We prove by contradiction. Without loss of generality, suppose there is a necklace which has a frontier with a vertex cut by b , then some vertex cut by a and then one again by b . See Figure 8. The vertices at the frontier are denoted by b_1 , b_2 and a' as shown in the figure. Now consider the paths from b to b_1, b_2 and the path from a to a' . There can be two cases for the topology of these paths, as shown in the figure. Case 1 occurs when the path b, b_1 intersects the path a, a' , say at x . This cannot happen because unless $d(x, b_1) = d(x, a')$ (otherwise one of them would be within some cut and hence not on the frontier). Hence, $d(a, b_1) = d(a, a')$. But we broke ties by choosing a over b . So case 1 cannot happen.

In case 2, we can see that the graph is a subdivision of $K_{2,3}$. Which means $K_{2,3}$ is a minor of the original graph G . But then G is not outerplanar. This is a contradiction. \square

The above lemma shows that a necklace's frontier always looks as in figure 9 (with the roles of a and b possibly interchanged).

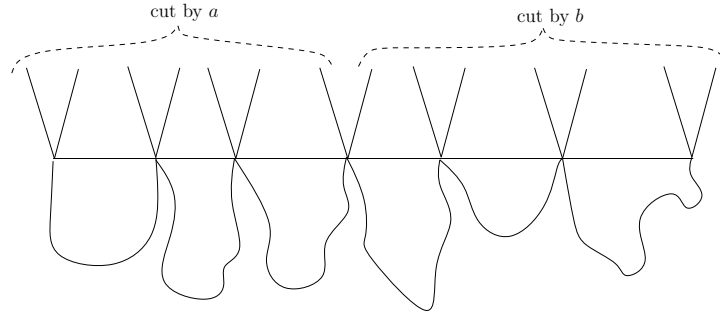


Figure 9: Structure of Frontier

The next lemma shows that a similar structure applies *across* necklaces.

Lemma 3.3 *Consider any embedding of the outerplanar graph G . After the cuts from a and b are made, the necklaces and the frontier points can be given a clockwise ordering around the edge $e = \{a, b\}$. In this clockwise ordering, all the frontier points cut by a are consecutive and all the frontier points cut by b are consecutive.*

Proof: Consider the largest 2-connected component of G containing the edge a, b , denote it by C_{ab} . The remaining 2-connected components are either attached to a or attached to b or attached to some vertex of C_{ab} . The components attached to a will be all consecutive in any embedding of G (similarly with the components attached to b). Moreover, these components will be cut by a only (and b only). So we have to worry about C_{ab} along with the graphs hanging off it. We can actually consider them as one composite piece, i.e. G without the 2-connected components attached to a and b . The most general embedding looks like figure 10. In general, there will be 2 leaves of C_{ab} in the embedding as shown in the figure. By 3.2, the necklace formed on the 2 leaves will have a frontier with points cut by a consecutively placed and points cut by b consecutively placed. In the lower leaf, going clockwise, this should be a 's frontier followed by b . Otherwise, we get a minor of K_4 sitting in the lower leaf (as shown in figure 10). Similarly in the upper leaf, going clockwise the frontier will have b 's frontier followed by a 's. Hence the lemma is proved. \square

Theorem 3.4 *The final graph returned by Main_routine is a minor of the original outerplanar graph.*

Proof: By induction on the recursion tree of the algorithm. By the induction hypothesis, before Resolve_necklace is called, the subgraphs hanging off the backbone are minors of the original subgraphs hanging off the backbone. Now the operations in Resolve_necklace that remove the Steiner vertices on the backbone were defined to be edge contractions. Finally the edges that are added to a and b from the necklaces can be seen as edge contractions on the shortest paths to a and b from the frontier, in light of lemmas 3.2 and 3.3. \square

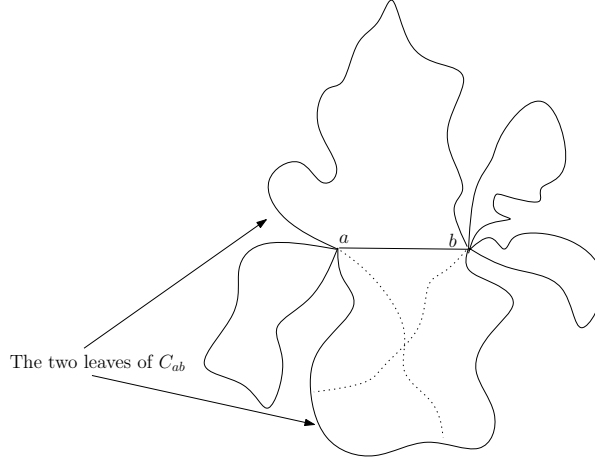


Figure 10: Structure of Frontier across necklaces

3.2.2 Bounding the distortion

We now show why the distortion is $O(1)$. We follow a similar proof strategy as for the tree of cycles case. We will first show a bound on the new distance d^* of the terminals to the vertices from which the cuts were made, and then use that to bound the expansion. We first mention a fact similar to fact 2.1.

Fact 3.5 *Let e be an edge added in step 4 of `Resolve_necklace`. Say the terminal on e is t and it is attached to a . Then $\frac{1}{2}d(t, a) \leq w(e) \leq d(t, a)$*

Proof: Note that by construction, $w(e) = d_{\text{frontier}}(t)$. Also note that $d(t, a) = d_{\text{frontier}}(t) + d(\text{frontier}, a)$. So clearly, the second inequality is true. Also, note that $d(\text{frontier}, a) \leq d_{\text{frontier}}(t)$, because $d(\text{frontier}, a) = \frac{1}{2}d(C(a), a)$ and $C(a)$ is closer to a than t , so $d(C(a), a) \leq d(t, a) = d_{\text{frontier}}(t) + d(\text{frontier}, a)$. So $d(\text{frontier}, a) \leq d_{\text{frontier}}(t)$. Hence, $d(t, a) \leq 2d_{\text{frontier}}(t) = 2w(e)$. \square

Lemma 3.6 *In the graph returned by `Main_routine`, the following holds*

- $d^*(x, a) \leq 14d(x, a) - 7d(a, C(a))$
- $d^*(x, b) \leq 14d(x, b) - 7d(b, C(b))$

Let us show that given this lemma, we get an expansion of at most 15.

Theorem 3.7 $d^*(x, y) \leq 15d(x, y) \quad \forall x, y \in T$

Proof: Case 1 : x and y are in different necklaces N_x and N_y .

Case 1a : N_x and N_y both have frontiers cut by a (or b). WLOG, assume it is a . Then $d^*(x, y) = d^*(x, a) + d^*(y, a)$. By lemma 3.6, $d^*(x, a) \leq 14d(x, a) - 7d(a, C(a)) = 14d(x, \text{frontier})$ and $d^*(y, a) \leq 14d(y, a) - 7d(a, C(a)) = 14d(y, \text{frontier})$. Therefore, $d^*(x, y) \leq 14d(x, \text{frontier}) + 14d(y, \text{frontier})$. Now using the fact that $d(x, y) \geq d(x, \text{frontier}) + d(y, \text{frontier})$, we have $d^*(x, y) \leq 14d(x, y) \leq 15d(x, y)$.

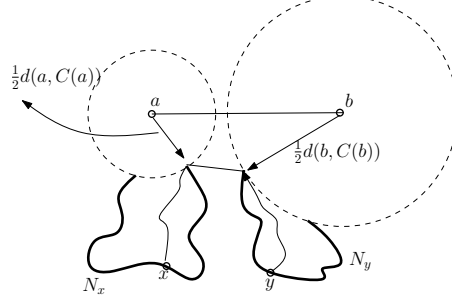


Figure 11: Bounding expansion

Case 1b : N_x has a frontier cut by only a and N_y has a frontier cut by only b . See Figure 11. Now $d^*(x, y) = d^*(x, a) + d(a, b) + d^*(y, b)$. Using lemma 3.6, $d^*(x, a) \leq 14d(x, a) - 7d(a, C(a)) = 14d(x, \text{frontier}(N_x))$ and similarly $d^*(y, b) \leq 14d(y, \text{frontier}(N_y))$.

Moreover, note that by the triangle inequality,

$$d(a, b) \leq d(a, \text{frontier}(N_x)) + d(\text{frontier}(N_x), \text{frontier}(N_y)) + d(\text{frontier}(N_y), b)$$

See Figure 11. Also $d(a, \text{frontier}(N_x)) = \frac{1}{2}d(a, C(a))$ and similarly for b . So we get

$$d(a, b) \leq \frac{1}{2}d(a, C(a)) + d(\text{frontier}(N_x), \text{frontier}(N_y)) + \frac{1}{2}d(b, C(b)).$$

Therefore, $d^*(x, y) \leq 14d(x, \text{frontier}(N_x)) + [\frac{1}{2}d(a, C(a)) + d(\text{frontier}(N_x), \text{frontier}(N_y)) + \frac{1}{2}d(b, C(b))] + 14d(y, \text{frontier}(N_y))$. Now we also know that $\frac{1}{2}d(a, C(a)) \leq d(x, \text{frontier}(N_x))$ and $\frac{1}{2}d(b, C(b)) \leq d(y, \text{frontier}(N_y))$, because $C(a)$ is closer to a than x and similarly for b and y . So $d^*(x, y) \leq 15d(x, \text{frontier}(N_x)) + d(\text{frontier}(N_x), \text{frontier}(N_y)) + 15d(y, \text{frontier}(N_y))$. Finally note that $d(x, y) \geq d(x, \text{frontier}(N_x)) + d(\text{frontier}(N_x), \text{frontier}(N_y)) + d(y, \text{frontier}(N_y))$. Therefore, $d^*(x, y) \leq 15d(x, y)$.

Case 1c : One of x or y is in a necklace with a frontier cut by both a and b , and the other is in a necklace with a frontier cut only by a (or b). Say x is in a necklace with a frontier cut by a only, and y is in a necklace with both frontiers. There are two subcases in this. Say the shortest path from y to x leaves y 's necklace through the frontier cut by a . then the analysis is the same as Case 1a. If it leaves through b , the analysis is like Case 1b.

Case 1d : Both x and y are in necklaces with frontiers cut by both a and b . There will be four subcases depending on which frontiers x and y escape through. The proofs are identical to the previous cases in all of them.

Case 2: x and y are in the same necklace but in different subgraphs hanging off the backbone. We again have several subcases depending on the topology of the shortest path between x, y in the original graph.

Case 2a : The shortest path goes through the backbone. For x and y they need to exit their subgraphs on the backbone through either a or b or that graph. We call these the escape vertices $v_{esc}(x)$ and $v_{esc}(y)$ respectively. Say the situation looks like in Figure 12. Moreover, say $v_{esc}(x)$ is between cuts c_i and c_{i+1} AND $v_{esc}(y)$ is between cuts c_j and c_{j+1} .

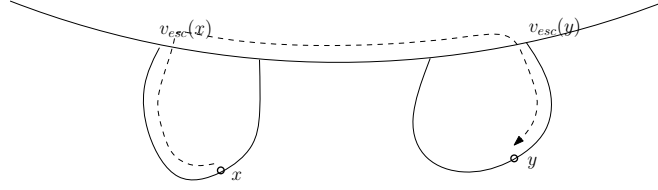


Figure 12: Bounding Expansion II

The worst case is if $v_{esc}(x)$ gets assigned $C(c_i)$ and the other end-point of the subgraph is assigned to $C(c_{i+1})$, because the other end-point was also the end-point of a graph containing $C(c_{i+1})$ and a similar thing happens with $v_{esc}(y)$. Then, $d^*(x, y) = d^*(x, v_{esc}(x)) + d_{bridge}^* + d^*(y, v_{esc}(y))$.

$$\text{Now, } d_{bridge}^* = \frac{1}{2}\hat{d}(C(c_i), C(c_{i+1})) + \frac{1}{2}\hat{d}(C(c_{i+1}), C(c_j)) + \frac{1}{2}\hat{d}(C(c_j), C(c_{j+1})).$$

Now also note that

$$\hat{d}(C(c_i), C(c_{i+1})) \leq \hat{d}(C(c_i), c_i) + d(c_i, c_{i+1}) + \hat{d}(c_{i+1}, C(c_{i+1}))$$

Denote by $C(v_{esc}(x))$ the terminal in the subgraph containing x closest to $v_{esc}(x)$. Also, since $v_{esc}(x)$ is within c_i and c_{i+1} , $d(c_i, v_{esc}(x)) \leq d(c_i, c_{i+1})$. Moreover $C(c_i)$ is closer to c_i than $C(v_{esc}(x))$, so $\hat{d}(c_i, C(c_i)) \leq d(c_i, v_{esc}(x)) + \hat{d}(v_{esc}(x), C(v_{esc}(x)))$. And recall that $d(c_i, c_{i+1}) \leq \frac{1}{2}\hat{d}(c_i, C(c_i))$. This yields $\frac{1}{2}\hat{d}(c_i, C(c_i)) \leq \hat{d}(v_{esc}(x), C(v_{esc}(x)))$. Finally, note that $\hat{d}(c_{i+1}, C(c_{i+1})) \leq \hat{d}(c_{i+1}, C(v_{esc}(x)))$.

This gives us

$$\hat{d}(C(c_i), C(c_{i+1})) \leq 3\hat{d}(v_{esc}(x), C(v_{esc}(x))) + \hat{d}(c_{i+1}, C(v_{esc}(x)))$$

Using Fact 3.5, the above expression is at most $3d(v_{esc}(x), C(v_{esc}(x))) + d(c_{i+1}, C(v_{esc}(x)))$.

Similarly with $v_{esc}(y)$ and $C(c_{j+1})$. Also note that $\frac{1}{2}\hat{d}(C(c_{i+1}), C(c_j)) \leq 2d(c_{i+1}, c_j)$. Finally invoking lemma 3.6, we have that $d^*(x, v_{esc}(x)) \leq 14d(x, v_{esc}(x)) - 7d(v_{esc}(x), C(v_{esc}(x)))$ and similarly for y . Putting it all together, and working out the algebra, we get $d^*(x, y) \leq 14d(x, y)$.

Case 2b : The shortest path goes out of the necklace and comes back in. Firstly, we ignore all parts

of the shortest path in the necklace except the ones with x and y in them. We show a bound on just these parts and hence we would show a bound on the entire shortest path. Now the proof is exactly like Case 1 and goes through identically. \square

Now to prove Lemma 3.6.

Proof: We prove by induction on the recursion tree of the algorithm. First we look at the case when x is in a necklace whose frontier is cut only by a (or b). We refer to the subgraph in the necklace containing x by S_x . Consider the shortest path from x to a (or b) and say it escapes S_x through $v_{esc}(x)$. Moreover, let us say that r is the vertex through which the shortest path *first* leaves the necklace. Let us say x is between cuts c_i and c_{i+1} made during `main_routine` after returning from `Resolve_Necklace` and r is between c_j and c_{j+1} . See Figure 13.

We break the new path from x to a (or b) into 3 parts. First part is from x to $C(c_i)$ or $C(c_{i+1})$ depending on where $v_{esc}(x)$ is assigned to. Second part is the path from $C(c_{i+1})$ to $C(c_j)$. Third part is $C(c_j)$ or $C(c_{j+1})$ to the frontier, depending on where r gets assigned to.

We first have the following bound on the second part.

Claim 3.8 $d^*(C(c_{i+1}), C(c_j)) \leq 5d(c_{i+1}, c_j) + \hat{d}(C(c_j), c_j) - \hat{d}(C(c_{i+1}), c_{i+1})$

Proof: For any $k \in i + 1, \dots, j - 1$, $d^*(C(c_k), C(c_{k+1})) \leq \hat{d}(C(c_k), c_k) + d(c_k, c_{k+1}) + \hat{d}(C(c_{k+1}), c_{k+1})$.

Now $\hat{d}(C(c_k), c_k) = 2d(c_k, c_{k+1})$ for every k . So adding $d^*(C(c_k), C(c_{k+1}))$ over all k , and we get the $d^*(C(c_{i+1}), C(c_j)) \leq 5d(c_{i+1}, c_j) - \hat{d}(C(c_{i+1}), c_{i+1}) + \hat{d}(C(c_j), c_j)$. \square

We now wish to bound the first and third parts of the path. We have four cases depending on how $v_{esc}(x)$ and r get assigned.

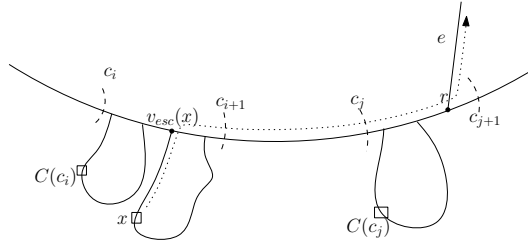


Figure 13: Proof of Main Lemma - Case 1

Case 1 : $v_{esc}(x)$ gets assigned to $C(c_i)$ and r gets assigned to $C(c_j)$. Then in the new graph, for the first part we have, $d^*(x, v_{esc}(x)) + \hat{d}(C(c_i), C(c_{i+1}))$. And note that $\hat{d}(C(c_i), C(c_{i+1})) \leq d^*(C(c_i), c_i) + d(c_i, c_{i+1}) + d^*(C(c_{i+1}), c_{i+1})$.

By the induction hypothesis, we have $d^*(x, v_{esc}(x)) \leq 14d(x, v_{esc}(x)) - 7d(v_{esc}(x), C(v_{esc}(x)))$. By construction of the cut c_{i+1} , we have $d^*(C(c_i), c_i) + d(c_i, c_{i+1}) = \frac{3}{2}d^*(C(c_i), c_i)$. And since $v_{esc}(x)$ falls within the cut, $\frac{1}{2}d^*(C(c_i), c_i) \leq d(v_{esc}(x), C(v_{esc}(x)))$.

Now for the last part of the path, $C(c_j)$ to the frontier is at most $\hat{d}(C(c_j), c_j) + d(c_j, r) + w(e)$ where e is the edge attached to r that was cut. See Figure 13. Now use the fact that $\hat{d}(C(c_j), c_j) \leq \hat{d}(c_j, C(v_{esc}(x))) \leq d(v_{esc}(x), C(v_{esc}(x))) + d(v_{esc}(x), c_j)$ by Fact 3.5.

Putting all the three parts together and simplifying, we would get that the total length of the new path is $\leq 7d(x, \text{frontier}) + 7d(x, v_{esc}(x)) - 2d(v_{esc}(x), C(v_{esc}(x)))$. This slack is going to be important in necklaces where the frontier is cut by both a and b . For now, it suffices to observe that

$$7d(x, \text{frontier}) + 7d(x, v_{esc}(x)) \leq 14d(x, \text{frontier}) \leq 14d(x, a) - 7(a, C(a))$$

There are three other cases depending on where $v_{esc}(x)$ and r are assigned to and all of them are along identical lines.

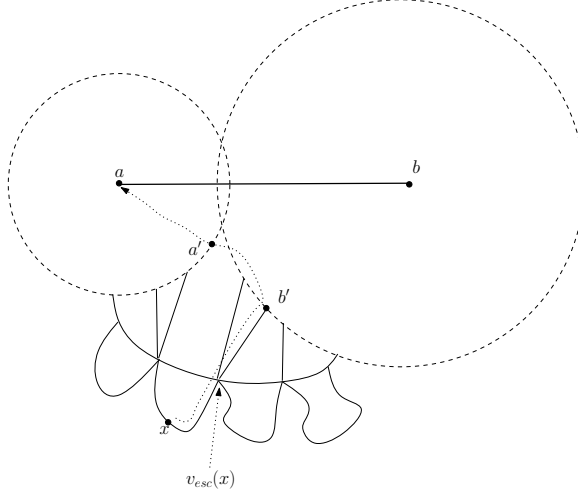


Figure 14: Proof of Main Lemma - Case 2

Now to look at the case when x is in a necklace whose frontier has vertices cut by both a and b . The only difference from the previous case comes when we are trying to bound the distance, say $d^*(x, a)$ but the shortest path escapes through a vertex cut by b . See Figure 14. Call this escape vertex b' . Moreover, to get to a , the path must enter the cut made by a at some vertex a' . The path in the new graph would have an additional $d(a, b)$ apart from the three parts defined earlier. This can be bounded as follows -

$$d(a, b) \leq d(a, a') + d(a', b') + d(b', b) \quad (6)$$

Now from the previous analysis, we would get that apart from the final edge a, b , the first 3 parts of the path in the new graph total $\leq 7d(x, b') + 7d(x, v_{esc}(x)) - 2d(v_{esc}(x), C(v_{esc}(x)))$.

We will rewrite the first two terms in the above expression. First

$$7d(x, b') = 7d(x, a') - 7(b', a')$$

Second, $7d(x, v_{esc}(x)) = 6d(x, v_{esc}(x)) + d(x, v_{esc}(x))$. Also,

$$d(x, v_{esc}(x)) = d(x, a') - d(a', v_{esc}(x))$$

And,

$$d(x, v_{esc}(x)) = d(x, b') - d(b', v_{esc}(x))$$

So

$$7d(x, v_{esc}(x)) = 6d(x, a') - 6d(a', v_{esc}(x)) + d(x, b') - d(b', v_{esc}(x))$$

So the first three parts of the new path total

$$\begin{aligned} &\leq 7d(x, a') - 7d(b', a') \\ &\quad + 6d(x, a') - 6d(a', v_{esc}(x)) \\ &\quad + d(x, b') - d(b', v_{esc}(x)) \\ &\quad - 2d(v_{esc}(x), C(v_{esc}(x))) \\ &\leq 14d(x, a') - 7d(b', a') - d(C(v_{esc}(x)), a') - d(C(v_{esc}(x)), b') \end{aligned}$$

Now look at the terms in equation (3.2.2). $d(a, a') \leq d(a', C(v_{esc}(x)))$ and $d(b, b') \leq d(b', C(v_{esc}(x)))$.

So adding $d(a, b)$ to the first three parts of the new path gives a total of at most $14d(x, a') = 14d(x, a) - 7d(a, C(a))$.

□

Theorem 3.9 *The contraction incurred by the algorithm is $O(1)$.*

Proof: The proof is identical to the proof for the contraction in the minor-closed algorithm for tree of cycles. □

Combining theorems 3.7 and 3.9, we can see that the distortion for the algorithm is $O(1)$.

References

- [1] M. Bateni, E. D. Demaine, M. Hajiaghayi, and M. Moharrami. Plane embeddings of planar graph metrics. In *ACM Symposium on Computational Geometry*, June 2006.
- [2] Y. Chawathe. An architecture for internet broadcast distribution as an infrastructure service. In *Ph.D. Thesis, University of California, Berkeley*, December 2000.
- [3] Y.-H. Chu, S. Rao, and H. Zhang. A case for end-system multicast. In *ACM Sigmetrics*, 2000.
- [4] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng. Low-stretch spanning trees. In *ACM Symposium on Theory of Computing*, 2005.
- [5] P. Francis. Yallcast: Extending the internet multicast architecture. In <http://www.yallcast.com>.
- [6] A. Gupta. Steiner points in trees metrics don't (really) help. In *SODA*, 2001.