Time-Space Trade-Offs for Computing Euclidean Minimum Spanning Trees[*]

Bahareh Banyassady[1], Luis Barba[2], and Wolfgang Mulzer[1]

1 Freie Universität Berlin, Berlin, Germany
[bahareh, mulzer]@inf.fu-berlin.de
2 ETH Zurich, Zurich, Switzerland
luis.barba@inf.ethz.ch

Abstract

Given n sites in the plane, their *Euclidean minimum spanning tree* (EMST), is the minimum spanning tree with the sites as vertices, where the weight of the edge between two sites is their Euclidean distance. In this paper, we revisit this problem, and design algorithms to compute the EMST in a limited-workspace model. In this model the input of size n lies in a random access read-only memory. The output has to be reported sequentially, and it cannot be accessed or modified. In addition, there is a read-write *workspace* of $O(s)$ words, where $s \in \{1, \dots, n\}$ is a given parameter. We present an algorithm that computes EMST using $O(n^3 \log s/s^2)$ time and $O(s)$ words of workspace. Using the fact that EMST is a subgraph of the bounded-degree *relative neighborhood graph* (RNG), we apply Kruskal's MST algorithm on RNG. To achieve this with limited workspace, we introduce a compact representation of planar graphs, called an s-*net* which allows us to manipulate RNG's component structure during the execution of the algorithm.

1 Introduction

A significant amount of research was focused on the design of algorithms using few variables. Many of them dating from the 1970s, when memory used to be an expensive commodity. While in recent days the cost has been substantially reduced, the amount of data has increased, and the size of some devices has been dramatically reduced. Sensors and small devices, where larger memories are neither possible nor desirable, have proliferated in recent years. Moreover, even if a device is procured with a large memory, it might still be preferable to limit the number of write operations, since they are slow and costly. Therefore, while many memory-constrained models exist, the general scheme is the following: the input resides in a read-only memory where data cannot be modified by the algorithm. The algorithm is allowed to store a few variables to solve the problem. These variables reside in a local memory and can be modified as needed (usually called *workspace*). Since the output may also not fit in our local memory, the model provides us with a write-only memory where the desired output is sequentially reported by the algorithm.

In general, one might consider algorithms that are allowed to use a workspace of $O(s)$ *words* for some parameter s, where a word is a collection of $\theta(\log n)$ bits. The goal is then to design algorithms whose running time decreases as s increases, and that provides a nice trade-off between workspace size and running time.

Asano et al. [1] proposed an algorithm to compute the EMST of a set of n given sites in $O(n^3)$ time using a workspace of $O(1)$ words. In this paper, we provide an algorithm that computes the EMST in $O(n^3 \log s/s^2)$ time using $O(s)$ words of workspace. This algorithm provides a smooth transition between the $O(n^3)$ time algorithm [1] with constant words of workspace and the $O(n \log n)$ time algorithm [2] using a workspace of $O(n)$ words.

[*] Supported in part by DFG project MU/3501/2 and by the ETH Postdoctoral Fellowship.

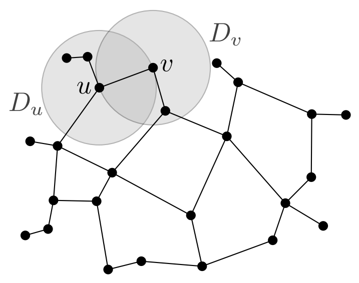

Figure 1 The graph RNG for a set of sites. The disk D_u (resp. D_v) is centered at u (resp. v) and passes through v (resp. u). The edge uv is in RNG, since there is no site in the lens $D_u \cap D_v$.

2 Preliminaries and Definitions

Let V be a set of n points (sites) in the plane. The *Euclidean minimum spanning tree* of V, EMST(V), is the minimum spanning tree of the complete graph G on V, where the edges are weighted by the Euclidean distance between their endpoints. We assume that V is in general position, i.e., the edge lengths in G are pairwise distinct, thus EMST(V) is unique. Given V, we can compute EMST(V) in $O(n \log n)$ time using $O(n)$ words of workspace [2].

The *relative neighborhood graph* of V, RNG(V), is the undirected graph with vertex set V obtained by add an edge between any two sites $u, v \in V$ if and only if the intersection of the two disks centered at u or v and passing through the other one, which is called the *lens* of u and v, is empty of sites in V [7]; see Figure 1. A plane embedding of RNG(V) is obtained by the straight line drawing of the edges. Furthermore, the maximum degree of RNG(V) is six and so, the number of edges of RNG(V), which is denoted by m, is $O(n)$. It is well-known that EMST(V) is a subgraph of RNG(V). This implies that RNG(V) is connected. Given V, we can compute RNG(V) in $O(n \log n)$ time using $O(n)$ words of workspace [5–7].

Recall the classic algorithm by Kruskal to find EMST(V) [4]: start with an empty forest T, and consider the $m = O(n)$ edges of RNG(V) one by one, by increasing weight. In each step, insert the current edge $e = vw$ into T iff there is no path between v and w in T. In the end, T is EMST(V). This takes $O(n \log n)$ total time and $O(n)$ words of workspace.

Let $s \in \{1, \dots, n\}$ be a parameter, and let V be a set of n sites in general position (as above) in a read-only array. The goal is to find EMST(V), with $O(s)$ words of workspace. We use RNG(V) in order to compute EMST(V). By general position, the edge lengths in RNG(V) are pairwise distinct. Thus, we define $E_R = e_1, \dots, e_m$ to be the sorted sequence of the edges in RNG(V), in increasing order of length. For $i \in \{1, \dots, m\}$, we define RNG$_i$ to be the subgraph of RNG(V) with vertex set V and edge set $\{e_1, \dots, e_{i-1}\}$.

In the limited workspace model, we cannot store RNG$_i$ explicitly. Instead, we resort to the *computing instead of storing* paradigm [1]. That is, we completely compute the next batch of edges in E_R whenever we need new edges of RNG(V) in Kruskal's algorithm. To check whether a new edge $e_i \in E_R$ belongs to EMST(V), we need to check if e_i connects two distinct components of RNG$_i$. To do this with $O(s)$ words of workspace, we will use a succinct representation of its component structure; see below. In our algorithm, we represent each edge $e_i \in E_R$ by two directed *half-edges*. The two half-edges are oriented in opposite directions such that the face incident a half-edge lies to the left of it. Obviously, each half-edge in RNG$_i$ has an opposing partner. However, in our succinct representation, we will rely on individual half-edges. We denote directed half-edges as $\overrightarrow{e}$, and undirected edges as e. For a half-edge $\overrightarrow{e} = \overrightarrow{uv}$ with $u, v \in V$, we call v the *head* of $\overrightarrow{e}$, and u the *tail* of $\overrightarrow{e}$.

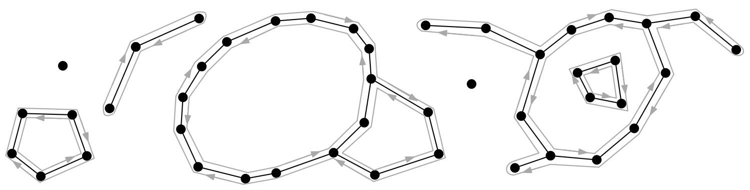

Figure 2 A schematic drawing of RNG_i is shown in black. The face-cycles of RNG_i are shown in gray. All the half-edges of a face-cycle are directed according to the arrows.

3 The Algorithm

In Lemma 3.1 we compute batches of edges of $\mathrm{RNG}(V)$ using $O(s)$ words of workspace. Then using this lemma we enumerate the edges of $\mathrm{RNG}(V)$ by increasing lengths, in Lemma 3.2.

▶ **Lemma 3.1.** *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \dots, n\}$ be a parameter. Given a set $Q \subseteq V$ of s sites, we can compute for each $u \in Q$ the at most six neighbors of u in $\mathrm{RNG}(V)$ in total time $O(n \log s)$, using $O(s)$ words of workspace.*

Proof. Let $V_j \subseteq V$, $j = 1, \dots \lceil n/s \rceil$, be the j-th *batch* of s sites of V. In the first step, we compute $\mathrm{RNG}(Q \cup V_1)$ with standard algorithms in $O(s \log s)$ time using $O(s)$ words of workspace. We store N_1, the set of all neighbors in $\mathrm{RNG}(Q \cup V_1)$ of all sites in Q. Then, in each step $j \neq 1$, we compute $\mathrm{RNG}(Q \cup V_j \cup N_{j-1})$ in $O(s \log s)$ time using $O(s)$ words of workspace. We store N_j, the set of all neighbors in $\mathrm{RNG}(Q \cup V_j \cup N_{j-1})$ of all sites in Q. Since the degree of sites in Q is at most six, $|N_j| = O(s)$. Notice that for a pair $u \in Q, v \in V$, if v is not among the neighbors of u in $N_{\lceil n/s \rceil}$, at some step there was a site in the lens of u and v. Thus, only the sites in $N_{\lceil n/s \rceil}$ define edges of $\mathrm{RNG}(V)$. However, all of them are not necessarily the neighbors of sites of Q in $\mathrm{RNG}(V)$. To filter these neighbors, we again scan V in batches of size s: for each $u \in Q$, we test if the lens between u and each of its neighbors in $N_{\lceil n/s \rceil}$ is empty of sites of V. After scanning V, the candidates with empty lens define neighbors of u in $\mathrm{RNG}(V)$. Since we use $O(s \log s)$ time per step, the claim follows. ◀

▶ **Lemma 3.2.** *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \dots, n\}$ be a parameter. Let $E_R = e_1, \dots, e_m$ be the sequence of edges in $\mathrm{RNG}(V)$, by increasing length. Let $i \geq 1$. Given e_{i-1} (or null, if $i = 1$), we can find $e_i, \dots, e_{i+s-1}$ (or $e_i, \dots, e_m$, if $i + s - 1 > m$), in $O(n^2 \log s / s)$ time using $O(s)$ words of workspace.*

Proof. We generate all the edges of $\mathrm{RNG}(V)$ by applying $O(n/s)$ times Lemma 3.1. Since, we obtain the edges in batches of size $O(s)$, each taking $O(n \log s)$ time, the total time amounts to $O(n^2 \log s / s)$. During this process, we find $e_i, \dots, e_{i+s-1}$ of E_R with a trick by Chan and Chen [3]. More precisely, whenever we produce new edges of $\mathrm{RNG}(V)$, we store the edges that are longer than e_{i-1} in an array A of size $O(s)$. Whenever A contains more than $2s$ elements, using a linear time selection procedure, we find the edge with rank s, and we remove all edges lorger than that [4]. This needs $O(s)$ operations per step, repeating for $O(n/s)$ steps, giving total time $O(n)$ for selecting the edges. In the end, we have $e_i, \dots, e_{i+s-1}$ in A, albeit not in sorted order. Thus, we sort the final A in $O(s \log s)$ time. The running time is dominated by the time needed to compute the edges of $\mathrm{RNG}(V)$, so the claim follows. ◀

For $i \in \{1, \dots, m\}$, a *face-cycle* in RNG_i is the circular sequence of half-edges that bounds a face in RNG_i. All half-edges in a face-cycle are oriented in the same direction, and RNG_i can be represented as a collection of face-cycles; see Figure 2. Asano et al. [1] observe that to run Kruskal's algorithm on $\mathrm{RNG}(V)$, it suffices to know the structure of the face-cycles.

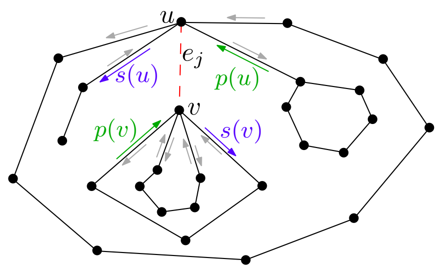

Figure 3 A schematic drawing of RNG_i. The endpoints u and v of e_j identify the predecessors of e_j, shown by $p(u)$ and $p(v)$ in green, and the successors of e_j, shown by $s(u)$ and $s(v)$ in blue.

▶ **Observation 3.3.** *Let $i \in \{1, \dots, m\}$. The edge $e_i \in E_R$ belongs to* $\mathrm{EMST}(V)$ *if and only if there is no face-cycle C in* RNG_i *such that both endpoints of e_i lie on C.*

For $j \geq i \geq 1$, we define *predecessor* (*successor*) of e_j in RNG_i, regarding each endpoint w of e_j, as the half-edge in RNG_i which has w as its head (tail) and is the first edge encountered in a counterclockwise (clockwise) sweep from e_j around w; see Figure 3. If there is no edge incident to w in RNG_i, we set null to the predecessor $p(w)$, and successor $s(w)$, of e_j. Here, we can already derive a simple time-space trade-off for computing $\mathrm{EMST}(V)$.

▶ **Theorem 3.4.** *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \dots, n\}$ be a parameter. We can output all the edges of* $\mathrm{EMST}(V)$*, in sorted order, in $O(n^3 \log s/s)$ time using $O(s)$ words of workspace.*

Proof. Let $E_R = e_1, \dots, e_m$ be the edges of $\mathrm{RNG}(V)$, sorted by length. We simulate Kruskal's algorithm on E_R: take batches of s edges of E_R and report the ones which are in $\mathrm{EMST}(V)$. More precisely, we use Lemma 3.2 to find a batch of s edges $e_i, \dots, e_{i+s-1}$, in $O(n^2 \log s/s)$ time. For each such edge e_j, we pick an endpoint $u_j \in V$ and we find first its incident edges in $\mathrm{RNG}(V)$ (Lemma 3.1), and then its incident edges in RNG_j (compare the edges from $\mathrm{RNG}(V)$ with e_j). Then, we identify the successor $s(u_j)$ of each e_j in RNG_j (if it exists), and we perform s parallel walks, where walk j takes place in RNG_j. In each step, we have s current half-edges and we advance each half-edge along its face-cycle, using Lemma 3.1 in $O(n \log s)$ time. A walk j continues until either it encounters the other endpoint of e_j or until it arrives at the predecessor $p(u_j)$ of e_j in RNG_j. Only in the latter case, e_j is in $\mathrm{EMST}(V)$, and we report it. Since there are $O(n)$ half-edges in $\mathrm{RNG}(V)$, it takes $O(n)$ steps to conclude all the walks. Thus, we can process a single batch of edges in $O(n^2 \log s)$ time, using $O(s)$ words of workspace. Since we have $O(n/s)$ many batches, the claim follows. ◀

For the case of linear space $s = n$, the running time of Theorem 3.4 is $O(n^2 \log n)$, while the classic algorithm takes $O(n \log n)$ time to find $\mathrm{EMST}(V)$. The bottleneck in Theorem 3.4 is performing the walks in RNG_j, that might take up to $\Omega(n)$ steps, leading to a running time of $\Omega(n^2 \log s)$ for processing a single batch. To avoid this, we maintain a compressed representation of RNG_j that allows us to reduce the number of steps in each walk to $O(n/s)$.

An s-*net* N for RNG_i, $i \in \{1, \dots, m\}$, is a collection of half-edges, called *net-edges*, in RNG_i such that: (i) each face-cycle in RNG_i with at least $\lfloor n/s \rfloor + 1$ half-edges contains at least one net-edge; and (ii) for any net-edge $\vec{e} \in N$, let C be the face-cycle of $\vec{e}$ in RNG_i. Then, between the head of $\vec{e}$ and the tail of the next net-edge on C, there are at least $\lfloor n/s \rfloor$ and at most $2\lfloor n/s \rfloor$ other half-edges on C. Note that the next net-edge on C after $\vec{e}$ could be possibly $\vec{e}$ itself. This implies that face-cycles with less than $\lfloor n/s \rfloor$ edges contain no net-edges. The following observation records two important properties of s-nets.

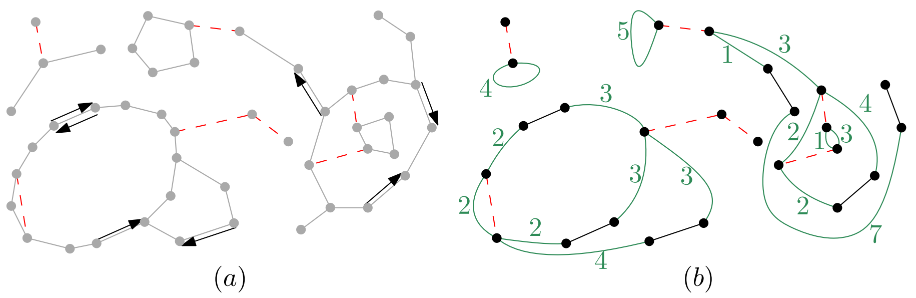

Figure 4 (a) A schematic drawing of RNG_i is shown in gray. The half-edges of N are in black and the edges of the next batch $E_{i,s}$ are dashed red segments. (b) The auxiliary graph H including the batch-edges (in red). The graph H contains the net-edges (in black), and the successors of the batch-edges and the compressed edges (which are combined in green paths in this picture).

▶ **Observation 3.5.** *Let $i \in \{1, \dots, m\}$, and N be an s-net for RNG_i. Then, (N1) $|N| = O(s)$; (N2) let $\overrightarrow{f}$ be a half-edge of RNG_i, and C be the face-cycle that contains it. Then, it takes at most $2\lfloor n/s \rfloor$ steps along C from the head of $\overrightarrow{f}$ until either a net-edge or the tail of $\overrightarrow{f}$.*

Proof. Property (ii) implies that only face-cycles of RNG_i with at least $\lfloor n/s \rfloor + 1$ half-edges contain net-edges. Furthermore, on these face-cycles, we can uniquely charge $\Theta(n/s)$ half-edges to each net-edge, again by (ii). Thus, since there are $O(n)$ half-edges in total, we have the first statement. For *(N2)*, note that if C contains less than $2\lfloor n/s \rfloor$ half-edges, the claim holds trivially. Otherwise, C contains at least one net-edge, by property (i). Now, property (ii) shows that we reach a net-edge in at most $2\lfloor n/s \rfloor$ steps from $\overrightarrow{f}$. ◀

Now, we show how to use the s-net in order to speed up the processing of a single batch.

▶ **Lemma 3.6.** *Let $i \in \{1, \dots, m\}$, and let $E_{i,s} = e_i, \dots, e_{i+s-1}$ be a batch of s edges of E_R. Suppose we have an s-net N for RNG_i in our workspace. Then, we can determine which edges from $E_{i,s}$ belong to $\mathrm{EMST}(V)$, using $O(n^2 \log s / s)$ time and $O(s)$ words of workspace.*

Proof. Let F be the set of half-edges that contains all net-edges from N, as well as, for each *batch-edge* $e_j \in E_{i,s}$, the two successors of e_j in RNG_i, one for each endpoint of e_j. By definition, we have $|F| = O(s)$, and it takes $O(n \log s)$ time to compute F, using Lemma 3.1. Now, we perform parallel walks through the face-cycles of RNG_i, as in Theorem 3.4. We have one walk for each half-edge in F, and each walk proceeds until it encounters the tail of a half-edge from F (including the starting half-edge itself). In each step of these parallel walks we need $O(n \log s)$ time to find the next edge on the face-cycle and then we need $O(s \log s)$ time to check whether these new edges are in F. Since F contains N, by property *(N2)*, each walk finishes after $O(n/s)$ steps. Thus, the total time for this procedure is $O(n^2 \log s / s)$.

Next, we build an auxiliary *undirected* graph H as follows: the vertices of H are the endpoints of the half-edges in F. Furthermore, H contains undirected edges for all the half-edges in F and additional *compressed edges* representing the outcomes of the walks: if a walk started from the head u of a half-edge in F and ended at the tail v of a half-edge in F, we add an edge from u to v in H, and we label it with the number of steps during the walk. Thus, H contains F*-edges* and *compressed edges*; see Figure 4. After all the walks have been performed, we can construct H in $O(s)$ time, using $O(s)$ words of workspace.

Next, using Kruskal's algorithm we insert the batch-edges of $E_{i,s}$ into H: we determine the connected components of H, in $O(s)$ time using depth-first search. Then, we insert the batch-edges into H, one after another, in sorted order and we keep track of how the connected components of H change, using a union-find data structure [4]. Whenever a batch-edge connects two different connected components, we output it as an edge of $\text{EMST}(V)$. Otherwise, we do nothing. Note that even though H may have a lot more components than RNG_i, the algorithm is still correct, by Observation 3.3. This execution of Kruskal's algorithm, and updating the structure of connected components of H takes $O(s \log s)$ time, which is dominated by the running time of $O(n^2 \log s/s)$ from the first phase of the algorithm. ◀

The following lemma shows how to compute an s-net for RNG_{i+s}, having an s-net for RNG_i and the graph H described in the proof of Lemma 3.6, for each $i \in \{1, \dots, m\}$.

▶ **Lemma 3.7.** *Let $i \in \{1, \dots, m\}$, and suppose we have the graph H derived from RNG_i as above, such that all batch-edges have been inserted into H. Then, we can compute an s-net N for RNG_{i+s} in time $O(n^2 \log s/s)$, using $O(s)$ words of workspace.*

Proof. By construction, all *big* face-cycles of RNG_{i+s}, which are the faces with at least $\lfloor n/s \rfloor + 1$ half-edges appear as faces in H. Thus, by walking along all faces in H, and taking into account the labels of the compressed edges, we can determine these big face-cycles in $O(s)$ time. The big face-cycles are represented through sequences of F-edges, compressed edges, and batch-edges. For each such sequence, we determine the positions of the half-edges for the new s-net N, by spreading the half-edges equally at distance $\lfloor n/s \rfloor$ along the sequence, again taking the labels of the compressed edges into account. Since the compressed edges have length $O(n/s)$, for each of them, we create at most $O(1)$ new net-edges. Now that we have determined the positions of the new net-edges on the face-cycles of RNG_{i+s}, we perform $O(s)$ parallel walks in RNG_{i+s} to actually find them. As it was explained in Theorem 3.4, this can be done in $O(n^2 \log s/s)$ time using Lemma 3.1. ◀

The following theorem provides a smooth trade-off between the cubic time constant workspace algorithm and the classical $O(n \log n)$ time algorithm with $O(n)$ words of workspace.

▶ **Theorem 3.8.** *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \dots, n\}$ be a parameter. We can output all the edges of $\text{EMST}(V)$, in sorted order of length, in $O(n^3 \log s/s^2)$ time using $O(s)$ words of workspace.*

Proof. This follows immediately from Lemma 3.6 and Lemma 3.7, because we need to process $O(n/s)$ batches of edges from E_R. ◀

References

1 T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *J. Comput. Geom.*, 2(1):46–68, 2011.

2 M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars. *Computational Geometry: Algorithms and applications.* Springer-Verlag, third edition, 2008.

3 T. M. Chan and E. Y. Chen. Multi-pass geometric algorithms. *Discrete Comput. Geom.*, 37(1):79–102, 2007.

4 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT Press, third edition, 2009.

5 J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80:1502–1517, 1992.

6 J. S. B. Mitchell and W. Mulzer. Proximity algorithms. In J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, page to appear. CRC Press, third edition, 2017.

7 G. T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.