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

Bahareh Banyassady[1][0000−0002−3422−9028], Luis Barba[2], and Wolfgang Mulzer[1][0000−0002−1948−5840]

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

Abstract. In the limited-workspace model, we assume that 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 $s \in \{1, \ldots, n\}$ is a given parameter. In a time-space trade-off, we are interested in how the running time of an algorithm improves as s varies from 1 to n.

We present a time-space trade-off for computing the *Euclidean minimum spanning tree* (EMST) of a set V of n sites in the plane. We present an algorithm that computes $\mathrm{EMST}(V)$ using $O(n^3 \log s / s^2)$ time and $O(s)$ words of workspace. Our algorithm uses the fact that $\mathrm{EMST}(V)$ is a subgraph of the bounded-degree *relative neighborhood graph* of V, and applies Kruskal's MST algorithm on it. To achieve this with limited workspace, we introduce a compact representation of planar graphs, called an *s-net* which allows us to manipulate its component structure during the execution of the algorithm.

Keywords: Euclidean minimum spanning tree, relative neighborhood graph, time-space trade-off, limited workspace model, Kruskal's algorithm

1 Introduction

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. This problem is at the core of computational geometry and has been a classical problem taught in almost every first year lecture on the subject. Several classical algorithms are known that can compute $\mathrm{EMST}(V)$ in $O(n \log n)$ time using $O(n)$ words of workspace [11].

In this work, we revisit this problem, and design algorithms to compute the EMST in a memory-constrained model, where only few extra variables are allowed to be used during the execution of the algorithm. This kind of algorithms not only provides an interesting trade-off between running time and memory needed,

but also is very useful in portable devices where important hardware constraints are present.

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 substantially been 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. In addition, when working on inputs that do not fit in the local memory of our computer, it is often the case that data is simultaneously accessed by several devices. Moreover, even if a device is procured with a large memory, it might still be preferable to limit the number of write operations. Writing to flash memory is slow and costly, and may also reduce the lifetime of the memory. Additionally, if the input is stored on removable devices, write-access may not be allowed due to technical or security reasons. 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 algorithms are allowed to store a few variables that reside in a local memory and can be modified as needed to solve the problem (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 bits and is large enough to contain either an input item (such as a point coordinate) or a pointer into the input structure (of logarithmic size on the length of the input). The goal is then to design algorithms whose running time decreases as s increases, and that provide a nice trade-off between workspace size and running time.

Our results. For the case of EMST, Asano et al. [6] 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 revisit this problem and provide a time-space trade-off. Our algorithm computes the EMST in $O(n^3 \log s/s^2)$ time using $O(s)$ additional words of workspace. This algorithm provides a smooth transition between the $O(n^3)$ time algorithm [6] with constant words of workspace and the $O(n \log n)$ time algorithm [11] using a workspace of $O(n)$ words.

As the main tool to achieve this running time, we introduce a compact representation of planar graphs, called an *s-net*. The main idea is to carefully choose a "dense" set of s edges of the graph for which we remember their face incidences. That is, we store whether or not any of these edges are incident to the same face of the graph. Moreover, the density property of this s-net guarantees that no path can walk along a face of the graph for long without reaching an edge of the s-net. This allows us to "quickly" find the face of the graph that any given edge lies on. More specifically, we use this structure to speed up the implementation of Kruskal's EMST algorithm on planar graphs using limited workspace. Recall that in this algorithm, edges are added in increasing order to an auxiliary graph. Moreover, for each of them we need to find out whether or

not its endpoints lie on the same component of this auxiliary graph when the edge is inserted. If the original graph is planar, then this amounts to testing whether or not these endpoints are incident to the same face of the graph—a task for which the compact representation of the s-net allows us to obtain time-space trade-offs to compute the EMST of planar graphs. While the s-net is designed to speed up Kruskal's algorithm, this structure is of independent interest as it provides a compact way to represent planar graphs that can be exploited by other algorithms.

Related work. The study of constant-workspace algorithm started with the introduction of the complexity class LOGSPACE [3]. After that, many classic problems were studied in this setting. Selection and sorting were among the first such problems [13, 20–22]. In graph theory, Reingold [23] solved a long standing problem, and showed that connectivity in an undirected graph can be tested using constant workspace. The model was made popular in computational geometry by Asano et al. [6] who presented several algorithms to compute classic geometric data structures in the constant-workspace model. Algorithms with time-space trade-off for many of these problems were presented in subsequent years [1, 2, 4, 5, 7–10, 15, 16, 18], with the notable exception of the problem of computing the EMST which is finally addressed in this paper.

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 [11].

The *relative neighborhood graph* of V, RNG(V), is the undirected graph with vertex set V obtained by connecting two sites $u, v \in V$ with an edge if and only if there is no site $w \in V \setminus \{u, v\}$ such that both $|uw|$ and $|vw|$ is less than $|uv|$, where $|uv|$ denotes the Euclidean distance between u and v [24]. This is also known as the *empty lens* property, where the *lens* between u and v is the intersection of the disks of radius $|uv|$ centered at both u and v; see Figure 1. One can show that a plane embedding of RNG(V) is obtained by drawing the edges as straight line segments between the corresponding sites in V. Furthermore, each vertex in RNG(V) has at most six neighbors, so that RNG(V) has $O(n)$ edges. We will denote the number of those edges by m. It is well-known that EMST(V) is a subgraph of RNG(V). In particular, 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 [17, 19, 24].

Recall the classic algorithm by Kruskal to find EMST(V) [14]: we start with an empty forest T, and we consider the edges of RNG(V) one by one, by increasing weight. In each step, we insert the current edge $e = vw$ into T if and only if there is no path between v and w in T. In the end, T will be EMST(V). Since
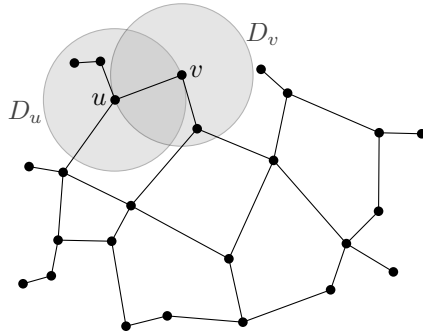
Fig. 1. The RNG for a set of sites V. The disks D_u and D_v have radius $|uv|$ and are centered at u and v, respectively. The edge uv is in $\mathrm{RNG}(V)$, since there is no site in V that lies in the lens $D_u \cap D_v$.

$\mathrm{EMST}(V)$ is a subgraph of $\mathrm{RNG}(V)$, it suffices to consider only the edges of $\mathrm{RNG}(V)$. Thus, Kruskal's algorithm needs to consider $m = O(n)$ edges and runs in $O(n \log n)$ time, using $O(n)$ words of workspace.

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

In the limited workspace model, we cannot store RNG_i explicitly. Instead, we resort to the *computing instead of storing* paradigm [6]. That is, we completely compute the next batch of edges in E_R whenever we need new edges of $\mathrm{RNG}(V)$ in Kruskal's algorithm. To check whether a new edge $e_i \in E_R$ belongs to $\mathrm{EMST}(V)$, 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. We call the endpoints of a half-edge the *head* and the *tail* such that the half-edge is directed from the tail endpoint to the head endpoint. Obviously, each half-edge in RNG_i has an opposing partner. However, in our succinct representation, we will rely on individual half-edges. Throughout the paper, directed half-edges will be denoted 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 $\overrightarrow{e}$.

3   The Algorithm

Before we discuss our algorithm, we explain how to compute batches of edges in $\mathrm{RNG}(V)$ using $O(s)$ words of workspace. A similar technique has been used previously in the context of Voronoi diagrams [8].

Lemma 3.1. *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \ldots, n\}$ 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. The algorithm uses $\lceil n/s \rceil$ *steps*. In each step, we process a *batch* of s sites of $V = V_1 \cup \ldots \cup V_{\lceil n/s \rceil}$, and produce at most six candidates for each site of Q to be in $\mathrm{RNG}(V)$. In the first step, we take the first batch $V_1 \subseteq V$ of s sites, and we compute $\mathrm{RNG}(Q \cup V_1)$. Because both Q and V_1 have at most s sites, we can do this in $O(s \log s)$ time using $O(s)$ words of workspace using standard algorithms. For each $u \in Q$, we remember the at most six neighbors of u in $\mathrm{RNG}(Q \cup V_1)$. Notice that for each pair $u \in Q, v \in V_1$, if the edge uv is not in $\mathrm{RNG}(Q \cup V_1)$, then the lens of u and v is non-empty. That is, there is a witness among the points of $Q \cup V_1$ that certifies that uv is not an edge of $\mathrm{RNG}(V)$. Let N_1 be the set containing all neighbors in $\mathrm{RNG}(Q \cup V_1)$ of all sites in Q. Storing N_1, the set of candidate neighbors requires $O(s)$ words of workspace.

Then, in each step $j = 2, \ldots, O(n/s)$, we take next batch $V_j \subseteq V$ of s sites, and compute $\mathrm{RNG}(Q \cup V_j \cup N_{j-1})$ in $O(s \log s)$ time using $O(s)$ words of space. For each $u \in Q$, we store the set of at most six neighbors in this computed graph. Additionally, we let N_j be the set containing all neighbors in $\mathrm{RNG}(Q \cup V_j \cup N_{j-1})$ of all sites in Q. Note that N_j, the set of candidate neighbors, consists of $O(s)$ sites as each site in Q has degree at most six in the computed graph.

Therefore, after $\lceil n/s \rceil$ steps, we are left with at most six candidate neighbors for each site in Q. As mentioned above, for a pair $u \in Q, v \in V$, if v is not among the candidate neighbors of u, then at some point in the construction there was a site witnessing that the lens of u and v is non-empty. Therefore, only the sites which are in the set of candidate neighbors can define edges of $\mathrm{RNG}(V)$. However, all the candidate neighbors are not necessarily the neighbors in $\mathrm{RNG}(V)$ of sites in Q.

To obtain the edges of $\mathrm{RNG}(V)$ incident to the sites of Q, we take each site in Q and its corresponding neighbors in $N_{\lceil n/s \rceil}$. Then, we go again through the entire set $V = V_1 \cup \ldots \cup V_{\lceil n/s \rceil}$ in batches of size s: for each $u \in Q$, we test the at most six candidate neighbors in $N_{\lceil n/s \rceil}$ against all elements of the current batch to test the empty-lens property. After going through all sites, the candidates that maintained the empty-lens property throughout define the edges of $\mathrm{RNG}(V)$ incident to the sites of Q. Since we use $O(s \log s)$ time per step, and since there are $\lceil n/s \rceil$ steps, the total running time is $O(n \log s)$ using $O(s)$ words of workspace. $\qquad\square$

Through repeated application of Lemma 3.1, we can enumerate the edges of $\mathrm{RNG}(V)$ by increasing lengths.

Lemma 3.2. *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \ldots, n\}$ be a parameter. Let $E_R = e_1, e_2, \ldots, e_m$ be the sequence of edges in $\mathrm{RNG}(V)$, by increasing length. Let $i \geq 1$. Given e_{i-1} (or $\perp$, if $i = 1$), we can find the edges $e_i, \ldots, e_{i+s-1}$ in $O(n^2 \log s/s)$ time using $O(s)$ words of workspace.*[3]

Proof. By applying Lemma 3.1 $O(n/s)$ times, we can generate all the edges of $\mathrm{RNG}(V)$. Because we obtain the edges in batches of size $O(s)$, each taking $O(n \log s)$ time, the total time to compute all the edges amounts to $O(n^2 \log s/s)$. During this process, we find the edges $e_i, \ldots, e_{i+s-1}$ of E_R. This can be done with a trick by Chan and Chen [12], similar to the procedure in the second algorithm in [7]. 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, we use a linear time selection procedure to remove all edges of rank larger than s [14]. This needs $O(s)$ operations per step. We repeat this procedure for $O(n/s)$ steps, giving total time $O(n)$ for selecting the edges. In the end, we have $e_i, \ldots, e_{i+s-1}$ 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. □

Lemma 3.2, together with the techniques from the original constant workspace EMST-algorithm by Asano et al. [6], already leads to a simple time-space trade-off for computing $\mathrm{EMST}(V)$. Recall that we represent the edges of $\mathrm{RNG}(V)$ as pairs of opposing half-edges, such that the face incident to a half-edge lies to its left. For $i \in \{1, \ldots, m\}$, 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. [6] observe that to run Kruskal's algorithm on $\mathrm{RNG}(V)$, it suffices to know the structure of the face-cycles.
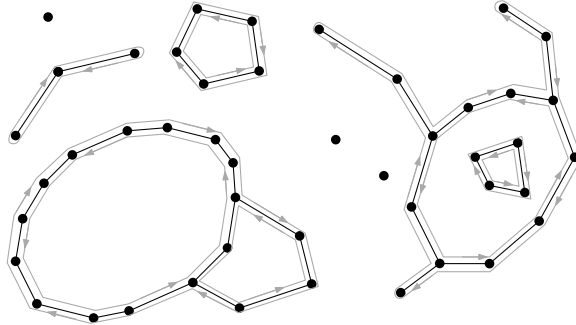
Fig. 2. A schematic drawing of RNG_i is shown in black. The face-cycles of this graph are shown in gray. All the half-edges of a face-cycle are directed according to the arrows.

---

[3] Naturally, if $i + s - 1 > m$, we report the edges $e_i, \ldots, e_m$.

Observation 3.3. *Let $i \in \{1, \ldots, m\}$. 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.*

Proof. Let u and v be the endpoints of e_i. If there is a face-cycle C in RNG_i that contains both u and v, then e_i clearly does not belong to $\mathrm{EMST}(V)$. Conversely, suppose there is no face-cycle in RNG_i containing both u and v. Thus, any two face-cycles C_u and C_v such that u lies on C_u and v lies on C_v must be distinct. Since $\mathrm{RNG}(V)$ is plane, C_u and C_v must belong to two different connected components of RNG_i, and e_i is an edge of $\mathrm{EMST}(V)$. □

Observation 3.3 tells us that we can identify the edges of $\mathrm{EMST}(V)$ if we can determine, for each $i \in \{1, \ldots, m\}$, the face-cycles of RNG_i that contain the endpoints of e_i. To accomplish this task, we use the next lemma to traverse the face-cycles.

Lemma 3.4. *Let $i \in \{1, \ldots, m\}$. Suppose we are given $e_i \in E_R$ and a half-edge $\overrightarrow{f} \in \mathrm{RNG}_i$, as well as the at most six edges incident to the head of $\overrightarrow{f}$ in $\mathrm{RNG}(V)$. Let C be the face-cycle of RNG_i that $\overrightarrow{f}$ lies on. We can find the half-edge $\overrightarrow{f'}$ that comes after $\overrightarrow{f}$ on C, in $O(1)$ time using $O(1)$ words of workspace.*

Proof. Let w be the head of $\overrightarrow{f}$. By comparing the edges incident to w with e_i, we identify the incident half-edges of w in RNG_i, in $O(1)$ time. Then, among them we pick the half-edge $\overrightarrow{f'}$ which has the smallest clockwise angle with $\overrightarrow{f}$ around w and has w as its tail. This takes $O(1)$ time using $O(1)$ words of workspace. □

For $j \geq i \geq 1$, we define *predecessor* and *successor* of e_j in RNG_i regarding each endpoint w of e_j as follows: the predecessor $\overrightarrow{p_w}$ of e_j is the half-edge in RNG_i which has w as its head and is the first half-edge encountered in a counterclockwise sweep from e_j around w. The successor $\overrightarrow{s_w}$ of e_j is the half-edge in RNG_i which has w as its tail and is the first half-edge encountered in a clockwise sweep from e_j around w; see Figure 3. If there is no edge incident to w in RNG_i, we set $p_w, s_w = \perp$.

From our observations so far, we can already derive a simple time-space trade-off for computing $\mathrm{EMST}(V)$.

Theorem 3.5. *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \ldots, n\}$ 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. We simulate Kruskal's algorithm on $\mathrm{RNG}(V)$. For this, we take batches of s edges, sorted by increasing length, and we report the edges of $\mathrm{EMST}(V)$ in each batch. Let $E_R = e_1, \ldots, e_m$ be the edges of $\mathrm{RNG}(V)$, sorted by length. To determine whether an edge $e_i \in E_R$ is in $\mathrm{EMST}(V)$, we apply Observation 3.3, i.e., we determine whether the endpoints of e_i are on two distinct face-cycles of the corresponding RNG_i. To do this, we process E_R in batches of s edges, and for each edge, we perform a walk along the face-cycle that contains one endpoint
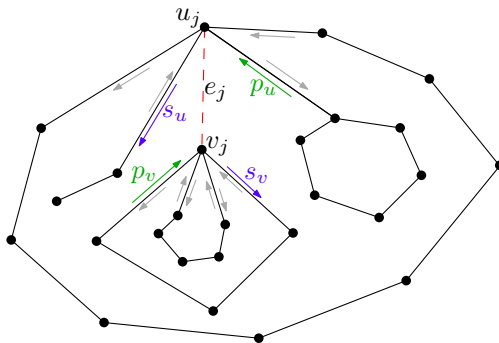
Fig. 3. A schematic drawing of RNG_i is shown in black. The endpoint $w = u, v$ of e_j identifies the half-edges p_w and s_w as the predecessor and the successor of e_j. They are shown in green and blue, respectively.

of e_i until we either encounter the other endpoint of e_i or until we are back at the starting point of our walk.

More precisely, we proceed as follows: first, we use Lemma 3.2 to find the next batch $e_i, \ldots, e_{i+s-1}$ of s edges in E_R, in $O(n^2 \log s / s)$ time. For each such edge e_j, we pick an endpoint $u_j \in V$. Using Lemma 3.1, we find for each u_j first the incident edges in $\mathrm{RNG}(V)$, and then the incident edges in RNG_j (by comparing the edges from $\mathrm{RNG}(V)$ with e_j). Then, we identify the successor 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 use Lemma 3.1 and Lemma 3.4 to advance each half-edge along its face-cycle. This takes $O(n \log s)$ operations. A walk j continues until we either encounter the other endpoint of e_j or until we arrive at the predecessor of e_j in RNG_j. In the latter case, e_j is in $\mathrm{EMST}(V)$, and we report it. In the former case, e_j is not in $\mathrm{EMST}(V)$. Since there are $O(n)$ half-edges in $\mathrm{RNG}(V)$, it takes $O(n)$ steps to conclude all the walks. If follows that we can process a single batch of edges in $O(n^2 \log s)$ time. We have $O(n/s)$ many batches, so the total running time of the algorithm is $O(n^3 \log s / s)$, using $O(s)$ words of workspace. $\qquad\square$

Theorem 3.5 is clearly not optimal: for the case of linear space $s = n$, we get a running time of $O(n^2 \log n)$, although we know that it should take $O(n \log n)$ time to find $\mathrm{EMST}(V)$. Can we do better? The bottleneck in Theorem 3.5 is the time needed to perform the walks in the partial relative neighborhood graphs RNG_j. In particular, such a walk 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 will maintain a compressed representation of the partial relative neighborhood graphs that allow us to reduce the number of steps in each walk to $O(n/s)$.

Let $i \in \{1, \ldots, m\}$. An *s-net* N for RNG_i is a collection of half-edges, called *net-edges*, in RNG_i that has the following two properties: (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 $\overrightarrow{e} \in N$, let C be the face-cycle of RNG_i with $\overrightarrow{e}$. Then, between

the head of $\overrightarrow{e}$ 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 $\overrightarrow{e}$ could be possibly $\overrightarrow{e}$ itself. In particular, this implies that face-cycles with less than $\lfloor n/s \rfloor$ edges contain no net-edge. The following observation records two important properties of s-nets.

Observation 3.6. *Let $i \in \{1, \ldots, m\}$, and let N be an s-net for RNG_i. Then, (N1) N has $O(s)$ half-edges; and (N2) let $\overrightarrow{f}$ be a half-edge of RNG_i, and let 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 we either reach a net-edge or the tail of $\overrightarrow{f}$.*

Proof. Property (ii) implies that only face-cycle 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 $|N| = O(s)$.

For the second statement, we first 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 $\overrightarrow{f}$. □

By Observation 3.6, we can store an s-net in $O(s)$ words of workspace. This makes the concept of s-net useful in our time-space trade-off. Now, we can use the s-net in order to speed up the processing of a single batch. The next lemma shows how this is done:

Lemma 3.7. *Let $i \in \{1, \ldots, m\}$, and let $E_{i,s} = e_i, \ldots, e_{i+s-1}$ be a batch of s edges from 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, using Lemma 3.1 and Lemma 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). By Lemma 3.4, 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. Because F contains the net-edges of N, by property (N2), each walk finishes after $O(n/s)$ steps, and 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*, that represent 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 that were needed for the walk. Thus, H contains F-*edges*, and *compressed edges*; see Figure 4. Clearly, after all the walks have been performed, we can construct H in $O(s)$ time, using $O(s)$ words of workspace.
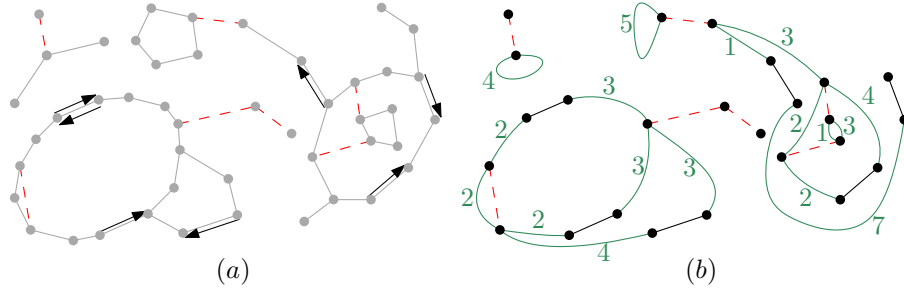
Fig. 4. (a) A schematic drawing of RNG_i is shown in gray. The half-edges of N are in black and the edges of the next batch $E_{i,s}$ are dashed red segments. (b) The auxiliary graph H including the batch-edges (in red). The graph H contains the net-edges (in black), and the successors of batch-edges and the compressed edges (which are combined in green paths in this picture).

Next, we use Kruskal's algorithm to insert the batch-edges of $E_{i,s}$ into H. This is done as follows: 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. As we do this, we keep track of how the connected components of H change, using a union-find data structure [14]. Whenever a new batch-edge connects two different connected components, we output it as an edge of $\mathrm{EMST}(V)$. 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. □

Finally, we need to explain how to maintain the s-net during the algorithm. The following lemma shows how we can compute an s-net for RNG_{i+s}, provided that we have an s-net for RNG_i and the graph H described in the proof of Lemma 3.7, for each $i \in \{1, \ldots, m\}$.

Lemma 3.8. *Let $i \in \{1, \ldots, m\}$, 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. Using Lemma 3.4, this takes $O(n^2 \log s/s)$ time. $\qquad\square$

We now have all the ingredients for our main result which provides a smooth trade-off between the cubic time algorithm in constant workspace and the classical $O(n \log n)$ time algorithm with $O(n)$ words of workspaces.

Theorem 3.9. *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 $O(n^3 \log s/s^2)$ time using $O(s)$ words of workspace.*

Proof. This follows immediately from Lemma 3.7 and Lemma 3.8, because we need to process $O(n/s)$ batches of edges from E_R. $\qquad\square$

For our algorithm, it suffices to update the s-net every time that a new batch is considered. It is however possible to maintain the s-net and the auxiliary graph H through insertions of single edges. This allows us to handle graphs constructed incrementally and maintain their compact representation using $O(s)$ workspace words. We believe this is of independent interest and can be used by other algorithms for planar graphs in the limited-workspace model.

References

1. H.-K. Ahn, N. Baraldo, E. Oh, and F. Silvestri. A time-space trade-off for triangulations of points in the plane. In *Proc. 23rd Internat. Comput. and Combinat. Conf. (COCOON)*, pages 3–12, 2017.
2. B. Aronov, M. Korman, S. Pratt, A. van Renssen, and M. Roeloffzen. Time-space trade-offs for triangulating a simple polygon. In *Proc. 15th Scand. Symp. Work. Alg. Theory (SWAT)*, pages 30:1–30:12, 2016.
3. S. Arora and B. Barak. *Computational complexity: A modern approach*. Cambridge University Press, 2009.
4. T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *Comput. Geom.*, 46(8):959–969, 2013.

5. T. Asano and D. G. Kirkpatrick. Time-space tradeoffs for all-nearest-larger-neighbors problems. In *Proc. 13th Algorithms and Data Structures Symposium (WADS)*, pages 61–72, 2013.

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

7. Y. Bahoo, B. Banyassady, P. Bose, S. Durocher, and W. Mulzer. Time-space trade-off for finding the *k*-visibility region of a point in a polygon. In *Proc. 11th Workshop Alg. Comp. (WALCOM)*, pages 308–319. Springer-Verlag, 2017.

8. B. Banyassady, M. Korman, W. Mulzer, A. van Renssen, M. Roeloffzen, P. Seiferth, and Y. Stein. Improved time-space trade-offs for computing Voronoi diagrams. In *Proc. 34th Sympos. Theoret. Aspects Comput. Sci. (STACS)*, pages 9:1–9:14, 2017.

9. L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. I. Silveira. Space–time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015.

10. L. Barba, M. Korman, S. Langerman, and R. I. Silveira. Computing a visibility polygon using few variables. *Comput. Geom.*, 47(9):918–926, 2014.

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

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

13. T. M. Chan, J. I. Munro, and V. Raman. Selection and sorting in the "restore" model. In *Proc. 25th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 995–1004, 2014.

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

15. O. Darwish and A. Elmasry. Optimal time-space tradeoff for the 2d convex-hull problem. In *Proc. 22nd Annu. European Sympos. Algorithms (ESA)*, pages 284–295, 2014.

16. S. Har-Peled. Shortest path in a polygon using sublinear space. *J. Comput. Geom.*, 7(2):19–45, 2016.

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

18. M. Korman, W. Mulzer, A. van Renssen, M. Roeloffzen, P. Seiferth, and Y. Stein. Time–space trade-offs for triangulations and Voronoi diagrams. *Comput. Geom.*, page to appear, 2017.

19. 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*, pages 849–874. CRC Press, third edition, 2017.

20. J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoret. Comput. Sci.*, 12(3):315–323, 1980.

21. J. I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theoretical Computer Science*, 165(2):311–323, 1996.

22. J. Pagter and T. Rauhe. Optimal time-space trade-offs for sorting. In *Proc. 39th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 264–268, 1998.

23. O. Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17, 2008.

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