A New Balanced Subdivision of a Simple Polygon for Time-Space Trade-off Algorithms[*][†]

Eunjin Oh[1] and Hee-Kap Ahn[2]

1 **Department of Computer Science and Engineering, POSTECH, Korea**
`jin9082@postech.ac.kr`

2 **Department of Computer Science and Engineering, POSTECH, Korea**
`heekap@postech.ac.kr`

Abstract

We are given a read-only memory for input and a write-only stream for output. For a positive integer parameter s, an s-workspace algorithm is an algorithm using only $O(s)$ words of workspace in addition to the memory for input. In this paper, we present an $O(n^2/s)$-time s-workspace algorithm for subdividing a simple polygon into $O(\min\{n/s, s\})$ subpolygons of complexity $O(\max\{n/s, s\})$.

As applications of the subdivision, the previously best known time-space trade-offs for the following three geometric problems are improved immediately: (1) computing the shortest path between two points inside a simple n-gon, (2) computing the shortest path tree from a point inside a simple n-gon, (3) computing a triangulation of a simple n-gon. In addition, we improve the algorithm for the second problem even further.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Time-space trade-off, simple polygon, shortest path, shortest path tree

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.61

1 Introduction

In the algorithm design for a given task, we seek to achieve an efficient algorithm with respect to the time and space complexities. However, one cannot achieve both goals at the same time in many cases: one has to use more space to achieve a faster algorithm and spend more time to reduce the space consumption of the algorithm. Therefore, one has to make a compromise between the running time and the space consumption, considering the goal of the task and the system resources where the algorithm is performed. With this reason, a number of time-space trade-offs were considered even as early as in 1980s. For example, Frederickson [7] presented optimal time-space trade-offs for sorting and selection problems in 1987. After this work, a significant amount of research has been done for time-space trade-offs in the design of algorithms.

The model we consider in this paper is formally described as follows. An input is given in a read-only memory. For a positive integer parameter s which is determined by users, we are allowed to use $O(s)$ words as workspace in addition to the memory for input. We assume that a word is large enough to store a number and a pointer. While processing input,

[*] A full version of the paper is available at [10], `https://arxiv.org/abs/1709.09932`.

[†] This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the SW Starlab support program(IITP-2017-0-00905) supervised by the IITP(Institute for Information & communications Technology Promotion)

we write output to a write-only stream without repetition. An algorithm designed in this setting is called an *s-workspace algorithm*.

Most of previous fundamental algorithms and applications assume that they can use workspace without much constraint in size. Typically, they use workspace of at least the size of input. Although the memory is relatively cheap these days, this is not always the case as the amount of data collected and used by various applications has significantly increased over the last years and the memory resource available in the system is relatively smaller compared to the amount of data they use. This constraint implies some restriction in using workspace for the applications.

We assume that input is given in a *read-only memory* under a *random-access model*. The assumption on the read-only memory has been considered in applications where the input is required to be retained in its original state or more than one program may access the input simultaneously. Many time-space trade-offs for fundamental problems have been studied under this read-only assumption.

In this paper, we consider time-space trade-offs for constructing a few geometric structures inside a simple polygon: the shortest path between two points, the shortest path tree from a point, and a triangulation of a simple polygon. With linear-size workspace, optimal algorithms for these problems are known. The shortest path between two points and the shortest path tree from a point inside a simple n-gon can be computed in $O(n)$ time [8]. A triangulation of a simple n-gon can also be computed in $O(n)$ time [5].

For a positive integer parameter s, the following s-workspace algorithms are known.

- **The shortest path between two points inside a simple polygon:** The first non-trivial s-workspace algorithm for computing shortest paths between any two points in a simple n-gon was given by Asano et al. [2]. Their algorithm consists of two phases. In the first phase, they subdivide the input simple polygon into $O(s)$ subpolygons of complexity $O(n/s)$ in $O(n^2)$ time. In the second phase, they compute the shortest path between the two points in $O(n^2/s)$ time using the subdivision. In the paper (and the talk by Asano in a workshop in honor of his 65th birthday during SoCG 2014), they asked whether the first phase can be avoided and the running time can be improved to $O(n^2/s)$. This problem is still open while there are several partial results.

  Har-Peled [9] presented an s-workspace algorithm which takes $O(n^2/s + n \log s \log^4(n/s))$ expected time. Their algorithm takes $O(n^2/s)$ expected time for the case that $s = O(n/\log^2 n)$. For the case that the input polygon is monotone, Barba et al. [4] presented an s-workspace algorithm which takes $O(n^2/s + (n^2 \log n)/2^s)$ time. Their algorithm takes $O(n^2/s)$ time for $\log \log n \leq s < n$.
- **The shortest path tree from a point inside a simple polygon:** Aronov et al. [1] presented an s-workspace algorithm for computing the shortest path tree from a given point. Their algorithm reports the edges of the shortest path tree without repetition in an arbitrary order in $O((n^2 \log n)/s + n \log s \log^5(n/s))$ expected time.
- **A triangulation of a simple polygon:** Aronov et al. [1] presented an s-workspace algorithm for computing a triangulation of a simple n-gon. Their algorithm returns the edges of a triangulation without repetition in $O(n^2/s + n \log s \log^5 (n/s))$ expected time. Moreover, their algorithm can be modified to report the resulting triangles of a triangulation together with their adjacency information in the same time if $s \geq \log n$.

  For a monotone n-gon, Barba et al. [4] presented an $(s \log_s n)$-workspace algorithm for triangulating the polygon in $O(n \log_s n)$ time for a parameter $s \in \{1, \ldots, n\}$. Later, Asano and Kirkpatrick [3] showed how to reduce the workspace to $O(s)$ words without increasing the running time.

1.1 Our Results

We present an s-workspace algorithm to subdivide a simple polygon with n vertices into $O(\min\{n/s, s\})$ subpolygons of complexity $O(\max\{n/s, s\})$ in $O(n^2/s)$ deterministic time. We obtain this subdivision in three steps. First, we choose every $\max\{n/s, s\}$th vertex of the simple polygon which we call *partition-vertices*. In the second step, for every pair of consecutive partition-vertices, we choose $O(1)$ vertices which we call *extreme-vertices*. Then we draw the vertical extensions from each partition-vertex and each extreme-vertex, one going upwards and one going downwards, until the extensions escape from the simple polygon. These extensions subdivide the polygon so that each subpolygon has complexity of $O(\max\{n/s, s\})$ or has a spiral-like structure. In the third step, we subdivide each spiral-like structure into subpolygons of complexity $O(\max\{n/s, s\})$. Then we show that the resulting subdivision has the desired complexity.

By using this subdivision method together with new ideas, we improve the running times of the following three problems without increasing the size of the workspace.

- **The shortest path between two points inside a simple polygon:** We can compute the shortest path between any two points inside a simple n-gon in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace. The previously best known s-workspace algorithm [9] takes $O(n^2/s + n \log s \log^4(n/s))$ expected time.
- **The shortest path tree from a point inside a simple polygon:** We can compute the shortest path tree from a given point inside a simple n-gon in $O(n^2/s + (n^2 \log n)/s^c)$ expected time for any constant $c > 0$. The previously best known s-workspace algorithm [1] takes $O((n^2 \log n)/s + n \log s \log^5 (n/s))$ expected time. The algorithm in [1] computes the shortest path between two points as a subprocedure. If one uses our shortest path algorithm for this subprocedure, the algorithm takes $O((n^2 \log n)/s)$ expected time.
- **A triangulation of a simple polygon:** The previously best known s-workspace algorithm [1] takes $O(n^2/s + n \log s \log^4(n/s))$ expected time. This algorithm computes the shortest path between two points as a subprocedure. If their algorithm uses our shortest path algorithm for this subprocedure, it takes $O(n^2/s)$ deterministic time.

All missing details and proofs can be found in the full version of this paper [10].

2 Preliminaries

Let P be a simple polygon with n vertices. Let $v_0, \ldots, v_{n-1}$ be the vertices of P in clockwise order along ∂P. The vertices of P are stored in a read-only memory in this order. For a subpolygon S of P, we use ∂S to denote the boundary of S and $|S|$ to denote the complexity of S. For any two points p and q in P, we use $\pi(p, q)$ to denote the shortest path between p and q contained in P. We assume that no two distinct vertices of P have the same x-coordinate. We can avoid this assumption by using a shear transformation [6, Chapter 6].

Let v be a vertex of P. We consider two vertical extensions from v, one going upwards and one going downwards, until they escape from P for the first time. A vertical extension from v contains no vertex of P other than v due to the assumption that no two distinct vertices of P have the same x-coordinate. We call the point of ∂P where an extension from v escapes from P for the first time a *foot point* of v. Note that a foot point of a vertex might be the vertex itself. We can compute (report) the foot points of all vertices of P in $O(n^2/s)$ time using $O(s)$ words of workspace.

▶ **Lemma 1.** *We can report the foot points of all vertices of P in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace.*

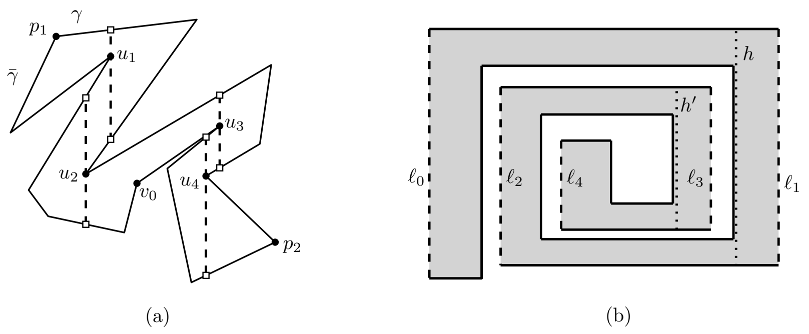

Figure 1 (a) Two chains γ and $\bar{\gamma}$ connecting two vertices p_1 and p_2. The set $V_\gamma = \{u_2, u_4\}$. The (L,C)-extreme-vertex of γ is u_2, and the (L,CC)-extreme-vertex of γ is u_4. The (R,C)-extreme-vertex of $\bar{\gamma}$ is u_1, and the (R,CC)-extreme-vertex of $\bar{\gamma}$ is u_3. (b) In the third step, we compute h for (ℓ_0, ℓ_1, ℓ_2) and h' for (ℓ_2, ℓ_3, ℓ_4).

In the following, we compute the extensions from some vertices of P. These extensions form a subdivision of P. We call each such extension a *wall*, and each subpolygon in the subdivision a *cell*. Given an edge of a cell, we can traverse the boundary of the cell starting from the given edge in time linear to the complexity of the cell once we store the walls of the subdivision and their endpoints in clockwise order along ∂P in the workspace.

3 Balanced Subdivision of a Simple Polygon

In this section, we present an s-workspace algorithm to subdivide a simple polygon P into $O(\min\{n/s, s\})$ subpolygons of complexity $O(\max\{n/s, s\})$ using $O(\min\{n/s, s\})$ walls. To do this, we first present an s-workspace algorithm to subdivide P into $O(n/\triangle)$ subpolygons of complexity $O(\triangle)$ using $O(n/\triangle)$ walls in $O(n^2/s)$ time, where $\triangle$ is a positive integer with $\min\{n/s, (s \log n)/n\} \leq \triangle \leq n$ which is determined by s. Since $n/s \leq \triangle$, we have $n/\triangle \leq s$. Thus, we can keep all such walls in the workspace of size $O(s)$. We will set the value of $\triangle$ in Theorem 10 so that we can obtain a subdivision of our desired complexity.

The first step: Subdivision by partition-vertices. We first consider every $\triangle$th vertex of P from v_0 in clockwise order, that is, $v_0, v_\triangle, v_{2\triangle}, \ldots, v_{\lfloor n/\triangle \rfloor \triangle}$. We call them *partition-vertices*. The number of partition-vertices is $O(n/\triangle)$. We compute the foot points of each partition-vertex, which can be done for all partition-vertices in $O(n^2/s)$ time in total by Lemma 1. We sort the foot points along ∂P in $O((n/\triangle)\log(n/\triangle))$ time, which is $O(n^2/s)$ by the fact that $\triangle \geq (s \log n)/n$. We store them together with their vertical extensions using $O(n/\triangle) = O(s)$ words of workspace.

The second step: Subdivision by extreme-vertices between two partition-vertices. The (L,C)-*extreme-vertex* and (L,CC)-*extreme-vertex* of a polygonal curve $\gamma \subset \partial P$ are defined as follows. Let V_γ be the set of all vertices of γ both of whose foot points are on $\partial P \setminus \gamma$ and whose extensions lie locally to the *left* of γ. The (L,C)-extreme-vertex (or the (L,CC)-extreme-vertex) of γ is the vertex in V_γ defining the first extension we encounter while we traverse ∂P in clockwise (or counterclockwise) order from v_0. See Figure 1(a). Similarly, we define the (R,C)-*extreme-vertex* and (R,CC)-*extreme-vertex* of γ. In this case, we consider the vertices of γ whose extensions lie locally to the *right* of γ. We simply call the (L,C)-,(L,CC)-,(R,C)- and (R,CC)-extreme-vertices *extreme-vertices* of γ. Note that γ may not have any extreme-vertex.

In the second step, we compute the extreme-vertices of each polygonal curve connecting two consecutive partition-vertices along ∂P and containing no other partition-vertices. Then we have $O(n/\triangle)$ extreme-vertices. We compute the foot points of all extreme-vertices and store them together with their vertical extensions using $O(n/\triangle) = O(s)$ words of workspace in $O(n^2/s)$ time using Lemma 1 and Lemma 2.

▶ **Lemma 2.** *We can find the extreme-vertices of every polygonal curve connecting two consecutive partition-vertices along ∂P and containing no other partition-vertices in $O(n^2/s)$ total time using $O(s)$ words of workspace.*

The third step: Subdivision by a vertex on a chain connecting three extensions. After applying the first and second steps, we obtain the subdivision induced by the extensions from the partition- and extreme-vertices. Let S' be a subpolygon in this subdivision. We will see later in Lemma 4 that S' has the following property: every chain connecting two consecutive extensions along $\partial S'$, except for two of them, has no extreme-vertex. However, it is still possible that S' contains $\omega(1)$ extensions on its boundary. In this case, S' has a spiral-like structure due to the property of S' mentioned above. See Figure 1(b). In the third step, we subdivide each subpolygon further so that every subpolygon has $O(1)$ extensions on its boundary.

The boundary of S' consists of vertical extensions and polygonal chains from ∂P whose endpoints are partition-vertices, extreme-vertices, or their foot points. We treat the upper and lower extensions defined by one partition- or extreme-vertex (more precisely, the union of them) as one vertical extension.

For every triple (ℓ, ℓ', ℓ'') of consecutive vertical extensions on $\partial S'$ such that ℓ, ℓ' and ℓ'' appear on $\partial S'$ in clockwise order, we consider the part (polygonal curve) of $\partial S'$ from ℓ to ℓ'' in clockwise order (excluding ℓ and ℓ''). Let Γ be the set of all such polygonal curves. For every $\gamma \in \Gamma$, we find a vertex $v(\gamma)$ of $\partial S' \setminus \gamma$ such that one of its foot points lies in γ between ℓ and ℓ', and the other foot point lies in γ between ℓ' and ℓ'' if it exists. If there are more than one such vertex $v(\gamma)$, we choose an arbitrary one.

The extensions of $v(\gamma)$ subdivide S' into three subpolygons each of which contains one of ℓ, ℓ' and ℓ''. In other words, the extensions from $v(\gamma)$ *separate* ℓ, ℓ' and ℓ''. We can compute $v(\gamma)$ and their extensions for every $\gamma \in \Gamma$ in $O(|S'|^2/s + \triangle)$ time in total. See Figure 1(b). The sum of $|S'|$ over all subpolygons S' is $O(n)$ and the number of the subpolygons from the second step is $O(n/\triangle)$ since we construct $O(n/\triangle)$ extensions in the first and second steps. Therefore, we do this for all subpolygons in the subdivision from the second step in $O(n^2/s + n) = O(n^2/s)$ time using $O(s)$ words of workspace.

Analysis. We obtained $O(n/\triangle)$ vertical extensions in $O(n^2/s)$ time using $O(s)$ words of workspace. In the following, we show that these vertical extensions subdivide P into $O(n/\triangle)$ subpolygons of complexity $O(\triangle)$. We call this subdivision the *balanced subdivision* of S. For any two points a, b on ∂P, we use $P[a, b]$ to denote the polygonal curve from a to b (including a and b) in clockwise order along ∂P.

We use the technical lemmas (Lemma 3 to Lemma 6) to show that each subpolygon in the final subdivision is incident to $O(1)$ walls and has complexity of $O(\triangle)$. Then we obtain Theorem 11 by setting a parameter $\triangle$.

▶ **Lemma 3.** *Both $P[a_1, v]$ and $P[v, a_2]$ contain partition-vertices for any extension a_1a_2 from a vertex v constructed from any of the three steps such that $P[a_1, a_2]$ contains v.*

Let S be a subpolygon in the final subdivision and S' be the subpolygon in the subdivision from the second step containing S. We again treat two vertical extensions defined by one vertex as one vertical extension. We label the extensions lying on ∂S as follows. Let ℓ_0 be the first extension on ∂S we encounter while we traverse ∂P from v_0 in clockwise order. We let $\ell_1, \ell_2, \ldots, \ell_k$ be the extensions appearing on ∂S in clockwise order along ∂S from ℓ_0. Similarly, we label the extensions lying on $\partial S'$ from ℓ'_0 to $\ell'_{k'}$ along $\partial S'$ in clockwise order such that ℓ'_0 is the first one we encounter while we traverse ∂P from v_0 in clockwise order. Then we have the following lemmas.

▶ **Lemma 4.** *For any $1 \leq i < k'$, let $a_1a_2 = \ell'_i$ and $b_1b_2 = \ell'_{i+1}$ such that a_1, a_2, b_1 and b_2 appear on ∂P (and $\partial S'$) in clockwise order. Then $P[a_2, b_1]$ has no extreme-vertex.*

▶ **Lemma 5.** *For any $1 \leq i < k-1$, one of ℓ_i, ℓ_{i+1} and ℓ_{i+2} is constructed in the third step.*

▶ **Lemma 6.** *The subpolygon S is incident to $O(1)$ extensions constructed in the third step.*

Proof. Consider an extension ℓ incident to S constructed in the third step. Let v be the vertex defining the extension ℓ. The boundary of S' consists of the walls $\ell'_0, \ldots, \ell'_{k'}$ and the polygonal curves connecting two consecutive walls. Let η_i be the polygonal curve of $\partial S'$ connecting ℓ'_i and ℓ'_{i+1} (excluding the walls) for $0 \leq i < k'$, and $\eta_{k'}$ be the polygonal curve connecting $\ell'_{k'}$ and ℓ'_0 (excluding the walls).

We also claim that there exist at most two vertices $u \in \eta_0$ such that the foot points of u are in $\partial S' \setminus \eta_0$ and the extension of u is incident to S. To see this, let $u_1, u_2 \in \eta_0$ be such vertices if they exist. Let h_1 and h_2 be the extensions from u_1 and u_2, respectively, incident to S. One polygonal chain connecting h_1 and h_2 along ∂S (but not containing them in its interior) is contained in η_0 since u_1 and u_2 are in η_0. The other polygonal chain along ∂S does not intersect η_0. This is because the foot points of u_1 and u_2 are not in η_0, and both h_1 and h_2 are incident to S. Therefore, no other vertex in η_0 with foot points in $\partial S \setminus \eta_0$ has extensions incident to S. This proves the claim. The same holds for $\eta_{k'}$.

Therefore, there are at most four extensions on ∂S constructed in the third step: two of them are extensions of vertices of η_0 and the others are extensions of vertices of $\eta_{k'}$. Thus the lemma holds. ◀

Due to Lemma 5 and Lemma 6, the following corollary holds.

▶ **Corollary 7.** *Every subpolygon in the final subdivision is incident to $O(1)$ extensions.*

▶ **Lemma 8.** *Every subpolygon in the final subdivision has complexity of $O(\triangle)$.*

Proof. Consider a subpolygon S in the final subdivision. By Corollary 7, the boundary of S consists of $O(1)$ vertical extensions and $O(1)$ polygonal curves from the boundary of P connecting two consecutive endpoints of vertical extensions of S. Each polygonal curve from the boundary of P contains at most one partition-vertex in its interior. Otherwise, a vertical extension intersecting the interior of S is constructed in the first or second step, which contradicts that S is a subpolygon in the final subdivision. The number of vertices between two consecutive partition-vertices is $O(\triangle)$. Therefore, S has $O(\triangle)$ vertices on its boundary. ◀

Therefore, we have the following lemma.

▶ **Lemma 9.** *Given a simple n-gon and a parameter $\triangle$ with $\min\{n/s, (s \log n)/n\} \leq \triangle \leq n$, we can compute a set of $O(n/\triangle)$ walls which subdivides the polygon into $O(n/\triangle)$ cells of complexity $O(\triangle)$ in $O(n^2/s)$ time using $O(s)$ words of workspace.*

By setting a parameter, we can obtain the following theorem.

▶ **Theorem 10.** *Given a simple n-gon, we can compute a set of $O(\min\{n/s, s\})$ walls which subdivides the polygon into $O(\min\{n/s, s\})$ cells of complexity $O(\max\{n/s, s\})$ in $O(n^2/s)$ time using $O(s)$ words of workspace.*

Proof. We set $\triangle$ to n/s if $s \leq \sqrt{n}$. We set $\triangle$ to s, otherwise. ◀

4 Applications

Comparison with other subdivision methods. There are several subdivision methods which are used for computing the shortest path between two points in the context of time-space trade-offs. Asano et al. [2] presented a subdivision method that subdivides a simple polygon into $O(s)$ subpolygons of complexity $O(n/s)$ using $O(s)$ chords. Then they showed that the shortest path can be computed in $O(n^2/s)$ time using $O(s)$ words of workspace. However, their algorithm takes $O(n^2)$ time to compute such a subdivision, which dominates the overall running time. Thus, computing such a subdivision is a bottleneck of this problem. In fact, in the paper, they asked whether such a subdivision can be computed more efficiently.

Instead of answering this question directly, Har-Peled [9] presented a way to subdivide a simple polygon into subpolygons of slightly different complexity and show that this subdivision has a structural property similar to the one for the subdivision of Asano et al. The subdivision of Har-Peled consists of $O(n/s)$ subpolygons of complexity $O(s)$ using $O(n/s)$ line segments. The number of line segments defining this subdivision is larger than $\Omega(s)$, so they cannot keep the whole subdivision in the workspace. Instead, they gave a procedure to find the subpolygon containing a query point in $O(n + s \log s \log^4(n/s))$ expected time. They showed that one can find the shortest path between two points using this subdivision in a way similar to the algorithm by Asano et al.

The balanced subdivision that we propose has a structural property similar to the ones by Asano et al. and Har-Peled, so we can use our balanced subdivision to compute the shortest path between any two points. Moreover, our subdivision method has several advantages compared to the ones of Asano et al. and Har-Peled: our balanced subdivision can be computed faster than the one by Asano et al., and we can keep the whole subdivision in the workspace unlike the one by Har-Peled. Due to the first advantage, ours can be used to improve a number of algorithms. Due to the second advantage, we can solve a few other application problems. A specific example is to compute the shortest path between a query point and a fixed point after a preprocessing for the fixed point. See Lemma 17.

Computing the shortest path between two points. Given any two points p and q in P, we can report the edges of the shortest path $\pi(p, q)$ in order in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace. This improves the s-workspace randomized algorithm by Har-Peled [9] which takes $O(n^2/s + n \log s \log^4(n/s))$ expected time.

As mentioned earlier, our subdivision method has properties similar to the ones by Asano et al. [2] and Har-Peled [9]. For $s \geq \sqrt{n}$, we have the subdivision consisting of $O(n/s)$ cells of complexity $O(s)$. We use the algorithm by Har-Peled [9]. His algorithm takes $O(n(T(n) + n)/s)$ time, where $T(n)$ is the time for finding the cell containing a query point. In our case, $T(n) = O(n)$, thus we can compute the shortest path between any two points in $O(n^2/s)$ deterministic time.

For $s < \sqrt{n}$, we have the subdivision consisting of $O(s)$ cells of complexity $O(n/s)$. We use the algorithm by Asano et al. [2] that computes the shortest path between any two points assuming that we are given a subdivision consisting of $O(s)$ cells of complexity $O(n/s)$.

▶ **Theorem 11.** *Given any two points in a simple polygon with n vertices, we can compute the shortest path between them in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace.*

Computing the shortest path tree from a point. The *shortest path tree* rooted at p is defined to be the union of $\pi(p, v)$ over all vertices v of P. Aronov et al. [1] gave an s-workspace randomized algorithm for computing the shortest path tree rooted at a given point. It takes $O((n^2 \log n)/s + n \log s \log^5 (n/s))$ expected time and uses the algorithm by Har-Peled [9] as a subprocedure. If one uses Theorem 11 instead of Har-Peled's algorithm, the running time is improved to $O((n^2 \log n)/s)$ expected time. In Section 5, we improve this algorithm even further using our balanced subdivision.

Computing a triangulation of a simple polygon. Aronov et al. [1] presented an s-workspace algorithm for computing a triangulation of a simple polygon by using the shortest path algorithm by Har-Peled [9] as a subprocedure. By replacing this algorithm with ours in Theorem 11, we can obtain a triangulation of a simple polygon in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace.

▶ **Theorem 12.** *Given a simple polygon with n vertices, we can compute a triangulation of the simple polygon in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace.*

5 Improved Algorithm for Computing the Shortest Path Tree

In this section, we improve the algorithm for computing the shortest path tree from a given point even further to $O(n^2/s + (n^2 \log n)/s^c)$ expected time for an arbitrary constant $c > 0$. We use the following lemma given by Aronov et al. [1].

▶ **Lemma 13** ([1, Lemma 6]). *For any point p in a simple n-gon, we can compute the shortest path tree rooted at p in $O(n^2 \log n)$ expected time using $O(1)$ words of workspace.*

We apply two different algorithms depending on the size of the workspace: $s = O(\sqrt{n})$ or $s = \Omega(\sqrt{n})$. In this paper, we consider the case of $s = O(\sqrt{n})$ only. The other case can be handled analogously. A main difference is that we do not use Theorem 11 and Lemma 13 for $s = \Omega(\sqrt{n})$. Instead, we use the fact that we can store all edges of each cell in the workspace. The details for the case of $s = \Omega(\sqrt{n})$ can be found in the full version of this paper.

Given a point $p \in P$, we want to report all edges of the shortest path tree rooted at p. For every wall a_1a_2 of the balanced subdivision, we first compute the edges of $\pi(p, a_1) \cup \pi(p, a_2)$ crossing some walls in $O(n^2/s^2)$ time in Section 5.1. We show that the number of such edges is $O(s)$ in total. These edges allow us to compute the shortest path $\pi(p, q)$ for any point q of P in $O(n^2/s^2)$ time. We call an edge a *w-edge* if it crosses a wall.

Then we decompose P into subpolygons associated with vertices in Section 5.2. For each subpolygon, we compute the shortest path tree rooted at its associated vertex inside the subpolygon recursively. If a subpolygon satisfies one of stopping conditions (to be defined later), we compute the shortest path tree inside the subpolygon in different ways. Because of the space restriction, we restrict the depth of the recurrence to be a constant instead of applying the procedure recursively until the problem size becomes $O(s)$.

5.1 Computing w-edges

We compute all w-edges of the shortest paths between p and the endpoints of the walls. The following lemma implies that there are $O(s)$ such w-edges. For any three points x, y and z in P, we call a point x' the *junction* of $\pi(x, y)$ and $\pi(x, z)$ if $\pi(x, x')$ is the maximal common path of $\pi(x, y)$ and $\pi(x, z)$.

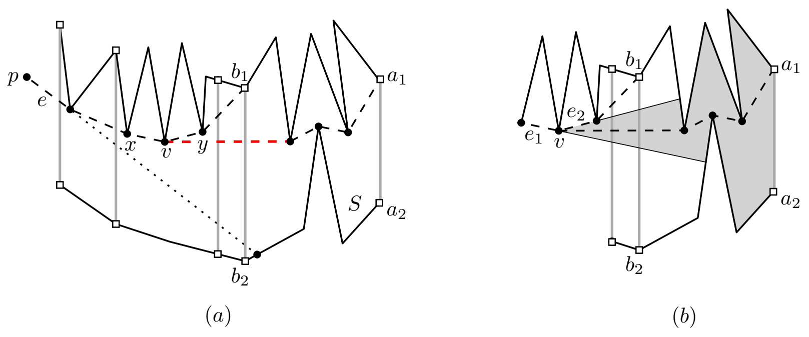

Figure 2 (a) We compute the junction v of $\pi(p, a_1)$ and $\pi(p, b_1)$ by applying binary search on the w-edges of $\pi(p, b_1)$. (b) We extend e_1 and e_2 towards b_1. The gray region contains the edge of $\pi(v, a_1)$ incident to v and has complexity of $O(n/s)$.

▶ **Lemma 14.** *For every wall a_1a_2, there is at most one w-edge of $\pi(p, a_i)$ for $i = 1, 2$ which is not a w-edge of $\pi(p, b) \cup \pi(p, b')$ for any wall bb' crossed by $\pi(p, a_i)$.*

We consider the walls one by one in a specific order and compute such w-edges one by one. To decide the order for considering the walls, we define a *wall-tree* T as follows. Each node α of T corresponds to a wall $d(\alpha)$ of the balanced subdivision of P, except for the root. The root of T corresponds to p and has children each of which corresponds to a wall incident to the cell containing p. A non-root node β of T is the parent of a node α if and only if $d(\beta)$ is the first wall that we encounter during the traversal of $\pi(p, a_1)$ from a_1 for an endpoint a_1 of $d(\alpha)$. We can compute T in $O(n)$ time.

▶ **Lemma 15.** *The wall-tree can be built in $O(n)$ time using $O(s)$ words of workspace.*

After constructing T, we apply depth-first search on T. Let $\mathcal{D}$ be an empty set. When we visit a node α with $a_1a_2 = d(\alpha)$, we compute the w-edges of $\pi(p, a_1) \cup \pi(p, a_2)$ which are not in $\mathcal{D}$ yet, and put them in $\mathcal{D}$. Each w-edge in $\mathcal{D}$ has information on the node of T defining it and the cells of the balanced subdivision containing its endpoints. Due to this information, we can compute the w-edges of $\pi(p, a)$ in order from a in $O(s)$ time for any endpoint a of $d(\alpha)$ and any node α we visited before. Once the traversal is done, $\mathcal{D}$ contains all w-edges in the shortest paths between p and the endpoints of the walls.

We show how to compute the w-edge of $\pi(p, a_1)$ which is not in $\mathcal{D}$ yet. The case for $\pi(p, a_2)$ can be handled analogously. By Lemma 14, there is at most one such edge of $\pi(p, a_1)$. Moreover, by its proof, such an edge is incident to v on $\pi(v, a_1)$. Here, v is the one of the two junctions closer to a_1 than the other among the junction of $\pi(p, b_1)$ and $\pi(p, a_1)$, and the junction of $\pi(p, b_2)$ and $\pi(p, a_1)$, where b_1b_2 is the wall corresponding the parent of α.

Computing junctions. We show how to compute the junction v_1 of $\pi(p, b_1)$ and $\pi(p, a_1)$ in $O(n^2/s^2)$ time assuming that $s = O(\sqrt{n})$. The junction of $\pi(p, b_2)$ and $\pi(p, a_1)$ can be computed analogously. Then we can compute v in the same time.

To do this, we find the w-edges in $\mathcal{D}$ lying on $\pi(p, b_i)$ in order from b_i in $O(s)$ time for $i = 1, 2$ and denote the set of them by $\mathcal{D}(b_i)$. Note that these are the w-edges of $\pi(p, b_i)$. We find two consecutive edges in $\mathcal{D}(b_1)$ containing v_1 between them along $\pi(p, b_1)$ by applying binary search on the edges in $\mathcal{D}(b_1)$.

Given any edge e in $\mathcal{D}(b_1)$, we can determine which side of e along $\pi(p, b_1)$ contains v_1 in $O(n/s)$ time as follows. We first check whether e is also contained in $\pi(p, b_2)$ in constant time using $\mathcal{D}(b_2)$. If so, v_1 is contained in the side of e along $\pi(p, b_1)$ containing b_1. Thus we are done. Otherwise, we extend e towards b_1 until it escapes from S, where S is the cell incident to both a_1a_2 and b_1b_2. See Figure 2(a). Note that the extension crosses b_1b_2 since both $\pi(b_1, v_e)$ and $\pi(b_2, v_e)$ are concave for an endpoint v_e of e. We can compute the point where the extension escapes from S in $O(n/s)$ time by traversing the boundary of S once. If an endpoint of the extension lies on the part of ∂S between a_1 and b_1 not containing a_2, v_1 lies in the side of e containing p along $\pi(p, b_1)$. Otherwise, the junction v_1 is contained in the other side of e. Therefore, we can find two consecutive w-edges in $\mathcal{D}(b_1)$ containing v_1 between them along $\pi(p, b_1)$ in $O((n/s)\log s)$ time since the size of $\mathcal{D}(b_1) = O(s)$.

The edges of $\pi(p, b_1)$ lying between the two consecutive w-edges are contained in the same cell. Let x and y be the endpoints of the two consecutive edges of $\mathcal{D}(b_1)$ contained in the same cell. Then we compute the edges of $\pi(x, y)$ one by one from x to y inside the cell containing x and y. By Theorem 11, we can compute $\pi(x, y)$ in $O(n^2/s^3)$ time since the size of the cell is $O(n/s)$. Here, we use extra $O(s)$ words of workspace for computing $\pi(x, y)$. When the algorithm in Theorem 11 reports an edge f of $\pi(x, y)$, we check which side of f along $\pi(x, y)$ contains v_1 in $O(n/s)$ time as we did before. We do this until we find v_1. This takes $O((n/s)^2)$ time since there are $O(n/s)$ edges in $\pi(x, y)$. Therefore, in total, we can compute the junction v_1 in $O(s + (n/s)\log s + n^2/s^2) = O(n^2/s^2)$ time since $s = O(\sqrt{n})$.

Computing the edge of $\pi(v, a_1)$ incident to the junction v. In the following, we compute the edge of $\pi(v, a_1)$ incident to v. Let e_1 and e_2 be two edges of $\pi(p, b_1)$ incident to v, which can be obtained while we compute v. See Figure 2(b). We extend e_1 and e_2 towards b_1 until they escape from the cell incident to both a_1a_2 and b_1b_2. We consider the subpolygon bounded by the two extensions and containing a_1 on its boundary. Its boundary consists of parts of the boundary of P and three extra line segments: the extensions of e_1 and e_2, and the part of the wall a_1a_2. Thus, the subpolygon can be represented using $O(1)$ words and the number of vertices in the subpolygon is $O(n/s)$. Note that $\pi(v, a_1)$ is contained in the subpolygon. Thus, the edge of $\pi(v, a_1)$ incident to v inside the subpolygon is the edge we want to compute. We can compute it in $O(n^2/s^3)$ time by applying Theorem 11 to this cell.

In summary, we presented a procedure to compute the w-edge of $\pi(p, a_1)$ which is not computed before in $O(n^2/s^2)$ time assuming that we have done this for every node we have visited so far. More specifically, computing the junction of $\pi(p, a_1)$ and $\pi(p, b_i)$ takes $O(n^2/s^2)$ time for $i = 1, 2$, and computing the edge incident to each junction takes $O(n^2/s^3)$ time. One of the edges is the w-edge that we want to compute. Since the size of the wall-tree is $O(s)$, we can do this for every node in $O(n^2/s)$ time in total. Thus we have the following lemma.

▶ **Lemma 16.** *Given a point p in a simple polygon with n vertices, we can compute all w-edges of the shortest paths between p and the endpoints of the walls in $O(n^2/s)$ time using $O(s)$ words of workspace for $s = O(\sqrt{n})$.*

Due to the w-edges, we can compute the shortest path $\pi(p, q)$ in $O(n^2/s^2)$ time for any point q in P. Note that n^2/s^2 is at least n for $s = O(\sqrt{n})$. For a proof, see Section M.

▶ **Lemma 17.** *Given a fixed point p in P and a parameter $s = O(\sqrt{n})$, we can compute $\pi(p, q)$ in $O(n^2/s^2)$ time for any point q in P using $O(s)$ words of workspace after an $O(n^2/s)$-time preprocessing for P and p.*

5.2 Decomposing the Shortest Path Tree into Smaller Trees

We subdivide P into subpolygons each of which is associated with a vertex of it in a way different from the one for the balanced subdivision. Then inside each such subpolygon, we report all edges of the shortest path tree rooted at its associated vertex recursively. We

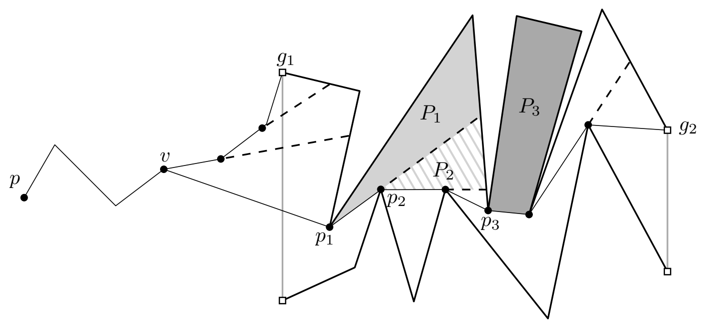

Figure 3 Subdivision of the region bounded by $\pi(v, g_1) \cup \pi(v, g_2)$ and the part of ∂P from g_1 to g_2 in clockwise order along ∂P by extending the edges of $\pi(v, g_1) \cup \pi(v, g_2)$. (P_i, p_i)'s are three of the subproblems of (P, p) for $i = 1, 2, 3$.

guarantee that the edges reported in this way are the edges of the shortest path tree rooted at p. We also guarantee that all edges of the shortest path tree rooted at p are reported. We use a pair (P', p') to denote the problem of reporting the shortest path tree rooted at a point p' inside a simple polygon $P' \subseteq P$. Initially, we are given the problem (P, p).

Structural properties of the decomposition. We use the following two steps of the decomposition. In the first step, we decompose P into a number of subpolygons by the shortest path $\pi(p, a)$ for every endpoint a of the walls. The boundary of each subpolygon consists of polygonal curves from ∂P with endpoints g_1, g_2 and shortest paths $\pi(v, g_1)$ and $\pi(v, g_2)$, where v is the junction of $\pi(p, g_1)$ and $\pi(p, g_2)$. In the second step, we decompose each subpolygon into smaller subpolygons by extending the edges of the shortest paths $\pi(v, g_1)$ and $\pi(v, g_2)$ towards g_1 and g_2, respectively. See Figure 3.

Consider a subpolygon P_i in the resulting subdivision. Its boundary consists of a polygonal curve from ∂P and two line segments sharing a common endpoint p_i. We can represent P_i using $O(1)$ words. Moreover, P_i has complexity of $O(n/s)$. For any point q in P_i, $\pi(p, q)$ is the concatenation of $\pi(p, p_i)$ and $\pi(p_i, q)$. Therefore, the shortest path rooted at p_i of P_i coincides with the shortest path tree rooted at p inside P restricted to P_i. We can obtain the entire shortest path tree rooted at p inside P by solving (P_i, p_i) for every subpolygon P_i in the resulting subdivision and its associated vertex p_i.

The procedure for obtaining this decomposition is described in the full version of this paper. We decompose each problem recursively unless the problem satisfies one of the three stopping conditions in Definition 18. Then we directly solve each base problem (that is, we report the edges of the shortest path tree.) But for non-base problems, we do not report any edge of the shortest path tree. In this way, we report each edge of the shortest path tree at most twice. We can report each edge without repetition using an orientation of each edge.

▶ **Definition 18** (Stopping conditions)**.** There are three stopping conditions for (P_i, p_i): (1) P_i has $O(s)$ vertices, (2) $s \geq \sqrt{|P_i|}$, where $|P_i|$ is the complexity of P_i, and (3) the depth of the recurrence is c, where $c > 0$ is a fixed constant.

When stopping condition (1) holds, we compute the shortest path tree directly using the algorithm by Guibas et al. [8]. When stopping condition (2) holds, we apply the algorithm described in the full version of this paper that computes the shortest path tree rooted at p_i inside P_i in $O(|P_i|^2/s)$ time for the case that $s \geq \sqrt{|P_i|}$, where $|P_i|$ is the complexity of P_i. When stopping condition (3) holds, we compute the shortest path tree using Lemma 13.

For each maximal polygonal curve with endpoints g_1 and g_2 containing no endpoints of walls in its interior, we spend $O(n^2/s^2 + nk/s)$ time, where k is the number of edges of $\pi(v, g_1) \cup \pi(v, g_2)$ for the junction v of $\pi(p, g_1)$ and $\pi(p, g_2)$. Since there are $O(s)$ such maximal polygonal curves and the sum of k over all such maximal polygonal curves is $O(n)$, the running time for decomposing the problem (P, p) into smaller problems is $O(n^2/s)$.

The total time complexity is $O(cn^2/s + (n^2 \log n)/s^c) = O(n^2/s + (n^2 \log n)/s^c)$, and the space complexity is $O(cs) = O(s)$.

▶ **Theorem 19.** *Given a point p in a simple polygon with n vertices, we can compute the shortest path tree rooted at p in $O(n^2/s + (n^2 \log n)/s^c)$ expected time using $O(s)$ words of workspace for an arbitrary constant $c > 0$.*

By setting c to the size of workspace and s to 2, we have the following theorem.

▶ **Theorem 20.** *Given a point p in a simple polygon with n vertices, we can compute the shortest path tree rooted at p in $O((n^2 \log n)/2^s)$ expected time using $O(s)$ words of workspace for $s \leq \log \log n$.*

References

1 Boris Aronov, Matias Korman, Simon Pratt, André van Renssen, and Marcel Roeloffzen. Time-space trade-offs for triangulating a simple polygon. *Journal of Computational Geometry*, 8(1):105–124, 2017.

2 Tetsuo Asano, Kevin Buchin, Maike Buchin, Matias Korman, Wolfgang Mulzer, Günter Rote, and André Schulz. Memory-constrained algorithms for simple polygons. *Computational Geometry*, 46(8):959–969, 2013.

3 Tetsuo Asano and David Kirkpatrick. Time-space tradeoffs for all-nearest-larger-neighbors problems. In *Proceedings of the 13th Algorithms and Data Strucutres Symposium (WADS 2013)*, pages 61–72, 2013.

4 Luis Barba, Matias Korman, Stefan Langerman, Kunihiko Sadakane, and Rodrigo I. Silveira. Space-time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015.

5 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

6 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, 2008.

7 Greg N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *Journal of Computer and System*, 34(1):19–26, 1987.

8 Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.

9 Sariel Har-Peled. Shortest path in a polygon using sublinear space. *Journal of Computational Geometry*, 7(2):19–45, 2015.

10 Eunjin Oh and Hee-Kap Ahn. A new balanced subdivision of a simple polygon for time-space trade-off algorithms, 2017. `arXiv:1709.09932`.