A New Balanced Subdivision of a Simple Polygon for Time-Space Trade-Off Algorithms

Eunjin Oh · Hee-Kap Ahn

Received: 8 April 2018 / Accepted: 4 March 2019 / Published online: 11 March 2019 © Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract

Given a read-only memory for input and a write-only stream for output, an $s$-workspace algorithm, for a positive integer parameter $s$, is an algorithm using only $O(s)$ words of workspace in addition to the memory for the input. In this paper, we present an $O(n^2/s)$-time $s$-workspace algorithm for subdividing a simple $n$-gon 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 by adopting the proposed subdivision: (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 problem (2) further by applying different approaches depending on the size of the workspace.

Keywords Time-space trade-off · Balanced subdivision · Simple polygon · Shortest path · Shortest path tree · Triangulation

1 Introduction

In the algorithm design for a given task, we seek to construct an efficient algorithm with respect to the time and space complexities. However, it is not always possible to...
achieve both of these objectives. Consider an algorithm that requires a certain amount of auxiliary memory to solve a problem within a certain amount of time. In many cases, the amount of memory requirement is proportional to the input size of the problem. If the amount of memory allowed for the algorithm is less than the required amount, it may not work at all or may take much longer time for solving the problem.

Therefore, one often has to take the size of the available memory space into account in the design phase of the algorithm and make a compromise between the time and space complexities, considering the goal of the task and the system resources where the algorithm under design is performed. Such an algorithm is supposed to change its behavior adaptively to the available memory size. With this reason, a number of time-space trade-offs were considered even as early as in the 1980s. For example, Frederickson [13] 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. The 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 can access the input simultaneously. For a positive integer parameter $s$ which is determined by the user, a memory space of $O(s)$ words is available as workspace (read–write memory under a random access model) in addition to the memory for the input. We assume that a word is large enough to store a number or a pointer. During the process, the output is to be written to a write-only stream without repetition. An algorithm designed in this setting is called an $s$-workspace algorithm.

Many classical algorithms require workspace of at least the size of the input in addition to the memory for the input. However, this is not always possible because the amount of data collected and used by various applications has significantly increased over the last years and the memory resources available in the system get relatively smaller compared to the amount of input data they use. The $s$-workspace algorithms deal with the case that the size of workspace is limited. If $s$ is at least $n$, we can copy the input into the workspace and use algorithms devised for the usual model of computation. Thus we assume that $s$ is at most the size of the input throughout this paper.

### 1.1 Previous Work

There have been a lot of work on time-space trade-offs for basic problems in computational geometry. For example, the convex hull of a set of $n$ points in the plane can be computed in $O(n^2/(s \log n) + n \log s)$ time using $O(s)$ words of workspace [11], and a triangulation of a set of $n$ points in the plane can be computed in $O(n^2/s + n \log s)$ time using $O(s)$ words of working space [1]. The problems of computing the Voronoi diagrams and Euclidean minimum spanning trees have also been considered in the limited workspace model [7,17]. For more information, refer to [8].

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 using $O(n)$ words of workspace [15]. A triangulation of
a simple $n$-gon can also be computed in $O(n)$ time using $O(n)$ words of workspace
[10]. With $O(1)$ workspace, there are a few algorithms that compute in $O(n^2)$ time
the shortest path between any two points in a simple $n$-gon [5,6]. A triangulation of a
simple $n$-gon can be computed in $O(n^2)$ time using $O(1)$ workspace [3].

For a positive integer parameter $s$, there are $s$-workspace algorithms for the prob-
lems we consider in this paper as follows.

- **The Shortest Path Between Two Points Inside a Simple Polygon** Asano et al. [3]
gave an $s$-workspace algorithm for computing the shortest path between two points
in a simple $n$-gon. Their algorithm consists of two phases. In the first phase,
they subdivide the input 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, they asked whether the
first phase can be improved to take $O(n^2/s)$ time. This problem is still open while
there are several partial results.

Har-Peled [16] presented an $s$-workspace algorithm which computes the shortest
path between two points in a simple $n$-gon in $O(n^2/s + n \log s \log^4 (n/s))$ expected
time. Har-Peled’s algorithm takes $O(n^2/s)$ expected time for the case of $s =
O(n/\log^2 n)$. For the case that the input polygon is monotone, Barba et al. [9]
presented an $s$-workspace algorithm which takes $O(n^2/s + (n^2 \log n)/2^s)$ time.

- **The Shortest-Path Tree from a Point Inside a simple Polygon** The shortest-path
tree from a point $p$ inside a simple polygon is defined as the union of the shortest
paths from $p$ to all vertices of the simple polygon. Aronov et al. [2] 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. [2] 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. [9] presented an algorithm using $O(s \log s n)$
workspace for triangulating the polygon in $O(n \log s n)$ time. Later, Asano and
Kirkpatrick [4] showed how to reduce the workspace to $O(s)$ words without
increasing the running time.

- **Decomposition of the Polygon Into Pieces of Asymptotically Equal Size** Asano
et al. [3] presented an $O(n^2)$-time algorithm for subdividing the polygon into
$O(s)$ subpolygons of complexity $O(n/s)$. Their algorithm uses $O(s)$ non-
crossing diagonals to subdivide the polygon. Later, Har-Peled [16] presented an
$O(n^2/s + n \log s \log^4 (n/s))$-time algorithm for subdividing the polygon into
$O(n/s)$ subpolygons of complexity $O(s)$. His subdivision is defined by $O(s)$
line segments whose endpoints are not necessarily contained in the boundary of the polygon. Later, Aronov et al. [2] presented an $O(n^2/s + n \log s \log^5(n/s))$-time algorithm for subdividing the polygon into $O(s)$ subpolygons of complexity $O(n/s)$ using $O(s)$ non-crossing diagonals. This improves the algorithm by Asano et al. [3].

### 1.2 Our Results

In this paper, we present an $s$-workspace algorithm to subdivide a simple polygon $P$ 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 by vertical extensions from some vertices of $P$ (similar to the vertical extensions in the trapezoid map), and therefore a vertex of the subdivision is either a vertex of $P$ or the point where the vertical extension abuts on an edge of $P$.

The subdivision is obtained in three steps. First, we choose every $\max\{n/s, s\}$th vertex of the simple polygon. We call the resulting vertices the partition vertices. In the second step, for every pair of consecutive partition vertices along the polygon boundary, 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 polygon for the first time. These extensions subdivide the polygon into subpolygons. In the subdivision, however, some subpolygons may still have complexity $\omega(\max\{n/s, s\})$. In the third step, we subdivide each such subpolygon further 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, we improve the running times for 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 takes $O(n^2/s + n \log s \log^4(n/s))$ expected time [16].

- **The Shortest-Path Tree from a Point Inside a Simple Polygon** The previously best known $s$-workspace algorithm takes $O((n^2 \log n)/s + n \log s \log^5(n/s))$ expected time [2]. It uses the algorithm by Har-Peled [16] as a subroutine for computing the shortest path between two points. If the subroutine is replaced by our shortest path algorithm, the algorithm is improved to take $O((n^2 \log n)/s)$ expected time.

- **A Triangulation of a Simple Polygon** The previously best known $s$-workspace algorithm takes $O(n^2/s + n \log s \log^4(n/s))$ expected time [2], which uses the shortest path algorithm by Har-Peled [16] as a subroutine. If the subroutine is replaced by our shortest path algorithm, the triangulation algorithm is improved to take only $O(n^2/s)$ deterministic time.

- **Decomposition of the Polygon Into Pieces of Asymptotically Equal Size** Our subdivision is induced by a set of $O(\min\{n/s, s\})$ vertical line segments whose endpoints are contained in the boundary of the polygon. Note that the endpoints of these segments are not necessarily vertices of the polygon. Moreover, the complexity of each
subpolygon is different from the one by Asano et al. [3] for \( s = \Omega(\sqrt{n}) \). Thus it is unclear if our subdivision method directly improves the subdivision method of Asano et al. However, by combining several previous results in [2,4,16] with our subdivision method, we show that we can obtain the subdivision of Asano et al. in \( O(n^2/s) \) deterministic time, and thus we improve the subdivision algorithms by Asano and Kirkpatrick [4] and Aronov et al. [2].

We also improve the algorithm for computing the shortest-path tree from a given point \( p \) even further to take \( O(n^2/s + (n^2 \log n)/s^c) \) expected time for an arbitrary positive constant \( c \). The improved result is based on the constant-workspace algorithm by Aronov et al. [2] for computing the shortest-path tree rooted at a given point. Depending on the size of workspace, we use two different approaches. For the case of \( s = O(\sqrt{n}) \), we decompose the polygon into subpolygons, each associated with a vertex, using a number of shortest paths from \( p \). Since we use shortest paths from \( p \) to decompose \( P \), the part of the shortest-path tree from \( p \) contained in each subpolygon is the shortest-path tree of the subpolygon from its associated vertex. Using this property, for each subpolygon we compute the shortest-path tree rooted at its associated vertex inside the subpolygon recursively. Due to the workspace constraint, we stop the recursion at a constant depth once one of the stopping criteria is satisfied. Then we show how to report the edges of the shortest-path tree without repetition efficiently using \( O(s) \) words of workspace. For the case of \( s = \Omega(\sqrt{n}) \), we can store all edges of each subpolygon in the workspace. We decompose the polygon into subpolygons associated with vertices and solve each subproblem directly using the algorithm by Guibas et al. [15].

The rest of our paper is organized as follows. In Sect. 2, we provide some necessary definitions and preliminaries. In Sect. 3, we present our subdivision procedure in three steps, and show that the subdivision is balanced. In Sect. 4, we provide comparison for our balanced subdivision method with other subdivision methods, and present improved algorithms for three problems using our balanced subdivision method. In Sect. 5, we present an improved \( s \)-workspace algorithm for computing the shortest-path tree from a given point in a simple polygon. Section 6 ends the paper with concluding remarks.

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 \( \partial P \). The vertices of \( P \) are stored in read-only memory in this order. For a subpolygon \( S \) of \( P \), we use \( \partial S \) to denote the boundary of \( S \) and \( |S| \) to denote the number of vertices 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 \( P \). To ease the description, we assume that no two distinct vertices of \( P \) have the same \( x \)-coordinate. We can avoid this assumption by using a shear transformation [12, 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
we made above. We call the point of \( \partial P \) where an extension from \( v \) escapes from \( P \) for the first time a \textit{foot point} of \( v \). Note that a foot point of a vertex might be the vertex itself. The following two lemmas show how to compute and report the foot points of vertices using \( O(s) \) words of workspace.

**Lemma 2.1** For a polygonal chain \( \gamma \subseteq \partial P \) of size \( O(s) \), we can compute the foot points of all vertices of \( \gamma \) in \( O(n) \) deterministic time using \( O(s) \) words of workspace.

**Proof** We show how to compute the foot point of every vertex \( v \) of \( \gamma \) lying above \( v \). The other foot points can be computed analogously. The foot point of a vertex \( v \) of \( \gamma \) might be \( v \) itself. We can determine whether the foot point of \( v \) is \( v \) itself or not in \( O(1) \) time by considering the two edges incident to \( v \).

We split the boundary of \( P \) into \( O(n/s) \) polygonal chains each of which contains \( O(s) \) vertices. Let \( \beta_0, \ldots, \beta_t \) be the resulting polygonal chains with \( t = O(n/s) \). For a vertex \( v \in \gamma \) whose foot point is not \( v \) itself, let \( \beta_i(v) \) denote the first point of \( \beta_i \cup \gamma \) (excluding \( v \)) hit by the upward vertical ray from \( v \) for each \( i = 0, \ldots, t \). If there is no such point, we let \( \beta_i(v) \) denote a point at infinity. We observe that the foot point of a vertex \( v \in \gamma \) is the one closest to \( v \) among \( \beta_i(v) \)'s for \( i = 0, \ldots, t \) unless its foot point is \( v \) itself.

For any fixed index \( i \in \{0, \ldots, t\} \), we can compute \( \beta_i(v) \) for all vertices \( v \in \gamma \) whose foot points are not themselves in \( O(s) \) time using \( O(s) \) words of workspace using the algorithm by Chazelle [10]. This algorithm computes the vertical decomposition of a simple polygon in linear time using linear space, but it can be modified to compute the vertical decomposition of any two non-crossing polygonal curves without increasing the time and space complexities. Since both \( \beta_i \) and \( \gamma \) have size of \( O(s) \), we can apply the vertical decomposition algorithm in [10] in \( O(s) \) time using \( O(s) \) words of workspace.

We apply this algorithm to \( \beta_0 \). For each vertex \( v \) of \( \gamma \) whose foot point is not \( v \) itself, we store \( \beta_0(v) \) in the workspace. Now we assume that we have the one closest to \( v \) among \( \beta_i(v) \)'s, for \( i = 0, \ldots, j - 1 \), stored in the workspace. To compute the one closest to \( v \) among \( \beta_i(v) \)'s for \( i = 1, \ldots, j \), we compute \( \beta_j(v) \). This can be done in \( O(s) \) time for all vertices on \( \gamma \) whose foot points are not themselves using the algorithm in [10]. Then we compare \( \beta_j(v) \) and the one stored in the workspace, choose the one closer to \( v \) between them and store it in the workspace.

Once we do this for all polygonal chains \( \beta_i \), we obtain the foot points of all vertices of \( \gamma \) by the observation. Since we spend \( O(s) \) time for each polygonal chain \( \beta_i \), the total running time is \( O(n) \).

**Lemma 2.2** We can report the foot points of all vertices of \( P \) in \( O(n^2/s) \) deterministic time using \( O(s) \) words of workspace.

**Proof** We apply the procedure in Lemma 2.1 to the first \( s \) vertices of \( P \), the next \( s \) vertices of \( P \), and so on. In this way we apply this procedure \( O(n/s) \) times. Thus we can find all foot points in \( O(n^2/s) \) time.

The extensions from some vertices of \( P \) induce a subdivision of \( P \) into subpolygons. Notice that the number of subpolygons in the subdivision is linear in the number of
extensions. In the following sections, we compute \( O(\min\{n/s, s\}) \) extensions from vertices of \( P \) and use them to subdivide \( P \) into \( O(\min\{n/s, s\}) \) subpolygons. We store the endpoints of the extensions of the subdivision in clockwise order along \( \partial P \) in the workspace and for each extension, we let its endpoints point to each other. Then we can traverse the boundary of the subpolygon starting from a given edge of the subpolygon in time linear in the complexity of the subpolygon.

3 Balanced Subdivision of a Simple Polygon

We say that a subdivision of \( P \) with \( n \) vertices balanced if it subdivides \( P \) into \( O(\min\{n/s, s\}) \) subpolygons of complexity \( O(\max\{n/s, s\}) \) using \( O(\min\{n/s, s\}) \) line segments. In this section, we present an \( s \)-workspace algorithm that computes a balanced subdivision using \( O(\min\{n/s, s\}) \) vertical extensions in \( O(n^2/s) \) time. We first present a subdivision procedure in three steps. Then we show that the resulting subdivision is balanced.

3.1 Subdivision in Three Steps

We first present an \( s \)-workspace algorithm to subdivide \( P \) into \( O(n/\triangle) \) subpolygons of complexity \( O(\triangle) \) using \( O(n/\triangle) \) extensions in \( O(n^2/s) \) time, where \( \triangle \) is a positive integer satisfying \( \max\{n/s, (s \log n)/n\} \leq \triangle \leq n \). Since \( n/s \leq \triangle \), we have \( n/\triangle \leq s \). Thus, we can keep all such extensions in workspace of size \( O(s) \). We will set the value of \( \triangle \) in Theorem 3.11 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, \( v_0, v_\triangle, v_{2\triangle}, \ldots, v_{[n/\triangle]\triangle} \). We call them the 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 using \( O(s) \) words of workspace by Lemma 2.2. We sort the foot points along \( \partial P \) 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 vertical extensions of the partition vertices subdivide \( P \) into \( O(n/\triangle) \) subpolygons. See Fig. 1a. However, there might be a subpolygon with complexity \( \omega(\triangle) \). See Fig. 1b. Recall that our goal is to subdivide \( P \) into \( O(n/\triangle) \) subpolygons each of complexity \( O(\triangle) \). To achieve this complexity, we subdivide each subpolygon further.

The Second Step: Subdivision by Extreme Vertices The \((L, C)\)-extreme vertex and \((L, CC)\)-extreme vertex of a polygonal chain \( \gamma \) of \( \partial P \) are defined as follows. Let \( V_\gamma \) be the set of all vertices of \( \gamma \), excluding its endpoints, both of whose foot points are on \( \partial P \setminus \gamma \) and whose extensions lie locally to the left of \( \gamma \). The \((L, C)\)-extreme vertex (or the \((L, CC)\)-extreme vertex) of \( \gamma \) is the vertex in \( V_\gamma \) defining the first extension we encounter\(^1\) while we traverse \( \partial P \) in clockwise (or counterclockwise) order.

\(^1\) We say we encounter an extension during the traversal of \( \partial P \) if we reach a foot point or the defining vertex of the extension.
Fig. 1  a Subdivision of polygons with $\Delta = 3$ induced by partition vertices. $v_0, v_3, v_6, v_9$, and $v_{12}$ are the partition vertices among the vertices of the polygon. There are six subpolygons in the subdivision, b the subpolygon (gray) in the middle is incident to $\lfloor n/\Delta \rfloor + 1$ vertical extensions of partition vertices with $\Delta = 6$, where $n$ is the number of vertices of the polygon, c the subdivision of the polygon in b after the second step.

Fig. 2  a Two chains $\gamma$ and $\gamma'$ connecting two vertices $p_1$ and $p_2$. The set $V_\gamma = \{u_2, u_4\}$. The $(L, C)$-extreme vertex of $\gamma$ is $u_2$, and the $(L, CC)$-extreme vertex of $\gamma$ is $u_4$. The $(R, C)$-extreme vertex of $\gamma'$ is $u_1$, and the $(R, CC)$-extreme vertex of $\gamma'$ is $u_3$, b in the third step, we compute the vertical extension $h$ for $(\ell_0, \ell_1, \ell_2)$ and the vertical extension $h'$ for $(\ell_2, \ell_3, \ell_4)$ that subdivide $Q$ into five subpolygons from $v_0$. See Fig. 2a for an illustration. Similarly, we define the $(R, C)$-extreme vertex and $(R, CC)$-extreme vertex of $\gamma$. In this case, we consider the vertices of $\gamma$ whose extensions lie locally to the right of $\gamma$. We simply call the $(L, C)$-, $(L, CC)$-, $(R, C)$- and $(R, CC)$-extreme vertices extreme vertices of $\gamma$. Note that $\gamma$ may have no extreme vertex.

In the second step, we consider each polygonal chain of $\partial P$ connecting two consecutive partition vertices along $\partial P$ and compute the extreme vertices of the chain. See Fig. 1c. Then we have $O(n/\Delta)$ extreme vertices. We compute the foot points of all extreme vertices and store them together with their vertical extensions using $O(n/\Delta) = O(s)$ words of workspace in $O(n^2/s)$ time using Lemmas 2.2 and 3.1.

**Lemma 3.1** We can find the extreme vertices of every polygonal chain of $\partial P$ connecting two consecutive partition vertices along $\partial P$ in $O(n^2/s)$ total time using $O(s)$ words of workspace.

**Proof** Let $\beta_i$ be the polygonal chain of $\partial P$ connecting two consecutive partition vertices $v_i\Delta$ and $v_{i+1}\Delta$ ($v_0$ if $i = \lfloor n/\Delta \rfloor$) along $\partial P$ for $i = 0, \ldots, \lfloor n/\Delta \rfloor$. We show how to compute the $(L, C)$-extreme vertices of $\beta_i$ for all $i$. The other types of extreme vertices can be computed analogously.
We apply the algorithm in Lemma 2.2 that reports both foot points of every vertex of \( P \). During the execution of the algorithm, for every \( i \), we store one vertex for \( \beta_i \) together with its foot points and the indices of the edges containing the foot points as a candidate of the \((L, c)\)-extreme vertex of \( \beta_i \). These vertices are updated during the execution of the algorithm so that the vertex stored for \( \beta_i \) is the \((L, c)\)-extreme vertex of \( \beta_i \) for every \( i \) from 0 to \( \lfloor n/\Delta \rfloor \) at the end of the execution.

Assume that the algorithm in Lemma 2.2 reports the foot points of a vertex \( v \in \beta_i \). If the extensions of \( v \) lie locally to the left of \( \beta_i \), we update the vertex for \( \beta_i \) as follows. We compare \( v \) and the vertex \( v' \) stored for \( \beta_i \). Specifically, we check if we encounter the extension of \( v \) before the extension of \( v' \) during the traversal of \( \partial P \) from \( v_0 \) in clockwise order. We can check this in constant time using the edge indices for the foot points of \( v' \) which are stored for \( \beta_i \) together with \( v' \). If so, we store \( v \) for \( \beta_i \) together with its foot points instead of \( v' \). Otherwise, we just keep \( v' \) for \( \beta_i \).

In this way, for every chain \( \beta_i \), we consider the foot points of all vertices on \( \beta_i \) whose extensions lie to the left of \( \beta_i \), and keep the extension which comes first from \( v_0 \) in clockwise order. Thus, at the end of the algorithm, we have the \((L, c)\)-extreme vertex of every polygonal chain \( \beta_i \) by definition. This takes \( O(n^2/s) \) time in total, which is the time for computing the foot points of all vertices of \( P \) by Lemma 2.2. \( \square \)

**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 \( Q \) be a subpolygon in this subdivision. We will see later in Lemma 3.5 that \( Q \) has the following property: every chain connecting two consecutive extensions along \( \partial Q \) has no extreme vertex, except for two such chains.

However, it is still possible that \( Q \) contains more than a constant number of extensions on its boundary. For instance, Fig. 2b shows a spiral-like subpolygon in the subdivision constructed after the first and second steps that has five extensions, \( \ell_0, \ldots, \ell_4 \), on its boundary. The input polygon can easily be modified to have more than a constant number of extensions on the boundary of such a spiral-like subpolygon. Then the subpolygon has complexity \( \omega(\Delta) \).

In the third step, we subdivide each subpolygon further so that every subpolygon has \( O(1) \) extensions on its boundary. The boundary of \( Q \) consists of vertical extensions and polygonal chains from \( \partial P \) whose endpoints are partition vertices, extreme vertices, or their foot points. We treat the upward and downward extensions defined by a common partition or extreme vertex (more precisely, the union of them) as one vertical extension.

For every triple \((\ell, \ell', \ell'')\) of consecutive vertical extensions appearing along \( \partial Q \) in clockwise order, we consider the part (polygonal chain) of \( \partial Q \) from \( \ell \) to \( \ell'' \) in clockwise order (excluding \( \ell \) and \( \ell'' \)). Let \( \Gamma \) be the set of all such polygonal chains. For every \( \gamma \in \Gamma \), we find a vertex, denoted by \( v(\gamma) \), of \( \partial Q \setminus \gamma \) such that one of its foot points lies in \( \gamma \) between \( \ell \) and \( \ell' \), and the other foot point lies in \( \gamma \) between \( \ell' \) and \( \ell'' \) if it exists. If there is more than one such vertex, we choose an arbitrary one.

The extensions of \( v(\gamma) \) subdivide \( Q \) into three subpolygons each of which contains one of \( \ell, \ell' \) and \( \ell'' \) on its boundary. In other words, the extensions from \( v(\gamma) \) separate \( \ell \), \( \ell' \) and \( \ell'' \). In Fig. 2b, the vertical extension \( h \) for \((\ell_0, \ell_1, \ell_2)\) and the vertical extension
h’ for (ℓ₂, ℓ₃, ℓ₄) together subdivide Q into five subpolygons. We can compute v(γ) and their extensions for every γ ∈ Γ in O(|Q|^2/s + m(Q)) time in total, where m(Q) denotes the number of extensions on the boundary of Q.

Lemma 3.2 We can find v(γ) for every γ ∈ Γ in O(|Q|^2/s + m(Q)) total time using O(s) words of workspace.

Proof The algorithm is similar to the one in Lemma 3.1. We apply the algorithm in Lemma 2.2 to compute the foot points of every vertex of Q with respect to Q. Assume that the algorithm in Lemma 2.2 reports the foot points of a vertex v of Q. We find the polygonal chains γ ∈ Γ containing both foot points of v if they exist. There are at most two such polygonal chains by the construction of Γ. We can find them in constant time after an O(m(Q))-time preprocessing for Q by Lemma 3.3, below. Let ℓ, ℓ’ and ℓ” be the three extensions defining γ. Then we check whether one foot point of v lies on the part of γ between ℓ and ℓ’, and the other foot point of v lies on the part of γ between ℓ’ and ℓ”. If so, we denote this vertex by v(γ) and keep it for γ. Otherwise, we do nothing. In this way, we can find v(γ) if it exists since we consider every vertex whose foot points lie on γ. This takes O(|Q|^2/s + m(Q)) time in total, which is the time for computing the foot points of all vertices of Q plus the preprocessing time for Q.

Now we analyze the space complexity of this algorithm. The algorithms in Lemmas 2.2 and 3.3 use O(s) words of workspace. Additionally, the algorithm in this proof keeps one vertex for each polygonal chain of Γ. Thus it uses additional O(|Γ|) words of workspace. Note that |Γ| is O(s) since the total number of extensions we obtained from the first and second steps is O(n/Δ) = O(s). Therefore, we use O(s) words of workspace in total.

Lemma 3.3 Given any point p on ∂Q ∩ ∂P with the index of the edge of P containing p, we can find the polygonal chains in Γ containing p in constant time, if they exist, after an O(m(Q))-time preprocessing for Q using O(s) words of workspace, where m(Q) denotes the number of extensions on the boundary of Q.

Proof Imagine that we subdivide ∂P with respect to the partition vertices of P into O(n/Δ) chains. Each such chain β intersects at most two chains f₁(β), f₂(β) ∈ Γ by the construction of Γ. As a preprocessing, for each chain β in the subdivision of ∂P by the partition vertices, we store f₁(β) and f₂(β). Among O(n/Δ) chains of ∂P, only O(m(Q)) of them intersect chains f₁(⋅) and f₂(⋅) of Γ. Thus, we can find and store for all such chains their f₁(⋅) and f₂(⋅) in O(m(Q)) time as follows. For each γ ∈ Γ, we find two chains β₁ and β₂ of ∂P containing γ in constant time, and set f₁(β₁) = γ and f₂(β₂) = γ for i = 1, 2, accordingly. We use O(m(Q)) words of workspace in the preprocessing. Note that m(Q) is O(s) since the total number of extensions we obtained from the first and second steps is O(n/Δ) = O(s).

Given any point p on ∂P with the index of the edge of P containing p, we can find the subchain β in the subdivision of ∂P containing p in constant time because the partition vertices are distributed uniformly at intervals of Δ vertices along ∂P. Then we check whether f₁(β) and f₂(β) contain p in constant time.
Each gray region has a partition vertex on its boundary, if $P[a_2, b_1]$ has an (L, C)- or (L, CC)-extreme vertex, $v_0$ lies on $P[a_1, b_2]$, which implies that $i = 0$ or $i = k'$, if $v_0$ lies on $P[f, a_1]$, an extension which separates $\ell'_i$ and $\ell'_{i+1}$ is constructed in the second step.

The sum of $|Q|$ over all subpolygons $Q$ is $O(n)$ and the number of the subpolygons from the second step is $O(n/\Delta)$ since we construct $O(n/\Delta)$ extensions in the first and second steps. Therefore, we can apply the third step of the subdivision for all subpolygons in the subdivision from the second step in $O(\frac{n^2}{s} + n) = O(\frac{n^2}{s})$ time using $O(s)$ words of workspace.

3.2 Balancedness of the Subdivision

We obtained $O(n/\Delta)$ vertical extensions in $O(\frac{n^2}{s})$ time using $O(s)$ words of workspace. In this section, we show that these vertical extensions subdivide $P$ into $O(n/\Delta)$ subpolygons of complexity $O(\Delta)$. We call this subdivision the balanced subdivision of $P$. For any two distinct points $a, b$ on $\partial P$, we use $P[a, b]$ to denote the polygonal chain from $a$ to $b$ (including $a$ and $b$) in clockwise order along $\partial P$.

We use a few technical lemmas (Lemmas 3.4–3.7) to show that each subpolygon in the final subdivision is incident to $O(1)$ extensions and has complexity $O(\Delta)$. Then we obtain Theorem 3.11 by setting the parameter $\Delta$.

**Lemma 3.4** Let $a_1a_2$ be any extension constructed from a vertex $v$ during any of the three steps such that $P[a_1, a_2]$ contains $v$. Then both $P[a_1, v]$ and $P[v, a_2]$ contain partition vertices.

**Proof** If $a_1a_2$ is constructed in the first step, $v$ is a partition vertex and lies on $P[a_1, v]$ and $P[v, a_2]$, and we are done. If $a_1a_2$ is constructed in the second step, $v$ is an extreme vertex of a polygonal chain which connects two consecutive partition vertices. One of the two partition vertices lies on $P[a_1, v]$ and the other lies on $P[v, a_2]$, and thus the claim holds.

Now, consider the case that $a_1a_2$ is constructed in the third step. In this case, $a_1a_2$ separates three consecutive extensions $\ell, \ell'$ and $\ell''$ which are constructed in the first or second step of the subdivision. See Fig. 3a.

Let $Q$ be the subpolygon of $P$ bounded by the three extensions. Then every connected component of $P \setminus Q$ contains a partition vertex on its boundary contained in $\partial P$ because it is incident to an extension constructed in the first or second step. Since $P[a_1, a_2]$ contains $v$, $P[a_1, v]$ contains the foot points of one of $\ell, \ell'$ and $\ell''$, say $\ell$. 

---

Fig. 3  a Each gray region has a partition vertex on its boundary, b if $P[a_2, b_1]$ has an (L, C)- or (L, CC)-extreme vertex, $v_0$ lies on $P[a_1, b_2]$, which implies that $i = 0$ or $i = k'$, c if $v_0$ lies on $P[f, a_1]$, an extension which separates $\ell'_i$ and $\ell'_{i+1}$ is constructed in the second step.
Then the component of $P \setminus Q$ incident to $\ell$ has a partition vertex on its boundary contained in $P[a_1, v]$. Similarly, $P[v, a_1]$ contains the foot points of one of $\ell'$ and $\ell''$, say $\ell'$, and thus the component of $P \setminus Q$ incident to $\ell'$ has a partition vertex on its boundary contained in $P[v, a_2]$. Thus, both $P[a_1, v]$ and $P[v, a_2]$ contain partition vertices. □

Let $S$ be a sub-polygon in the final subdivision and let $Q$ be the sub-polygon in the subdivision from the second step containing $S$. We again treat the two (upward and downward) vertical extensions defined by one vertex as one vertical extension. We label the extensions lying on $\partial S$ as follows. Let $\ell_0$ be the first extension on $\partial S$ we encounter while we traverse $\partial P$ from $v_0$ in clockwise order. Let us let $\ell_1, \ell_2, \ldots, \ell_k$ be the extensions appearing on $\partial S$ in clockwise order along $\partial S$ from $\ell_0$. Similarly, we label the extensions lying on $\partial Q$ from $\ell_0'$ to $\ell_k'$ in clockwise order along $\partial Q$ such that $\ell_0'$ is the first one we encounter while we traverse $\partial P$ from $v_0$ in clockwise order. Then we have the following lemmas.

**Lemma 3.5** 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 $\partial P$ (and on $\partial Q$) in clockwise order. Then $P[a_2, b_1]$ has no extreme vertex.

**Proof** Assume to the contrary that for some $1 \leq i < k'$, $P[a_2, b_1]$ has an extreme vertex. For an illustration, see Fig. 3b. By definition, no partition vertex lies on $P[a_2, b_1]\setminus(a_2, b_1)$. Consider the maximal polygonal chain $\gamma \subset \partial P$ containing no partition vertex in its interior and containing $P[a_2, b_1]$. Note that $\gamma \subseteq P[a_1, b_2]$ since both $P[a_1, a_2]$ and $P[b_1, b_2]$ contain partition vertices by Lemma 3.4.

Let $v$ be an extreme vertex of $P[a_2, b_1]$. (Recall that $v$ exists by the assumption made in the beginning of the proof.) Without loss of generality, we assume that $P[a_2, b_1]$ lies locally to the right of the extension of $v$. The foot points of $v$ lie on $\partial P \setminus \gamma$ while $v$ lies on $\gamma$. Therefore, $\gamma$ has an extreme vertex. (But $v$ is not necessarily an extreme vertex of $\gamma$ by definition.) The extension of $v$ subdivides $P$ into three subpolygons. Let $f$ be the foot point of $v$ incident to the subpolygon containing $\ell_i'$ on its boundary and $f'$ be the other foot point of $v$, as shown in Fig. 3b.

Since $1 \leq i < k'$, $v_0$ lies on $P[f', f], P[f, a_1]$ or $P[b_2, f']$. (Recall that the vertices of $P$ are labeled from $v_0$ to $v_{n-1}$ in clockwise order.) We show that in each case, there is an extreme vertex on $\gamma$ whose extension separates $\ell_i'$ and $\ell_{i+1}'$. Note that these extensions are constructed in the second step, which contradicts the assumption that $Q$ contains both $\ell_i'$ and $\ell_{i+1}'$ on its boundary.

- **Case 1.** $v_0$ is in $P[f', f]$: Then $v$ is the $(L, C)$- and $(L, CC)$-extreme vertex of $\gamma$ by definition. The extension of $v$ separates $\ell_i'$ and $\ell_{i+1}'$, which is a contradiction.
- **Case 2.** $v_0$ is in $P[f, a_1]$: By definition, the foot points of the $(L, CC)$-extreme vertex $u$ of $\gamma$ lie on $P[f, v_0]$. See Fig. 3c. Moreover, $u$ lies on $P[a_2, v]$. Thus, the extension of $u$ separates $\ell_i'$ and $\ell_{i+1}'$, which is a contradiction.
- **Case 3.** $v_0$ is in $P[b_2, f']$: A contradiction can be shown in a way similar to Case 2. The only difference is that we consider the $(L, C)$-extreme vertex instead of the $(L, CC)$-extreme vertex.

Therefore, $P[a_2, b_1]$ has no extreme vertex. □
We need a few more technical lemmas, which are given in the following, to conclude that the subdivision proposed in the previous section is balanced.

**Lemma 3.6** For any \(1 \leq i < k-1\), one of \(\ell_i, \ell_{i+1}\) and \(\ell_{i+2}\) is constructed in the third step.

**Proof** Assume to the contrary that all of \(\ell_i, \ell_{i+1}\) and \(\ell_{i+2}\) are constructed prior to the third step for some index \(1 \leq i < k-1\). Then the three extensions are consecutive along \(\partial Q\) as well since there is no vertical extensions added to the part of \(\partial Q\) from \(\ell_i\) to \(\ell_{i+2}\) in clockwise order in the third step. Consider the polygonal chain, denoted by \(\gamma\) of \(\partial Q\) from \(\ell_i\) to \(\ell_{i+2}\) in clockwise order (excluding \(\ell_i\) and \(\ell_{i+2}\)). Let \(\gamma_1\) be the part of \(\gamma\) lying between \(\ell_i\) and \(\ell_{i+1}\) excluding the two extensions, and let \(\gamma_2\) be the part of \(\gamma\) lying between \(\ell_{i+1}\) and \(\ell_{i+2}\) excluding the two extensions. By Lemma 3.5, \(\gamma_1\) and \(\gamma_2\) have no extreme vertex. Thus, \(\gamma\) has no extreme vertex.

We claim that \(v(\gamma)\) exists. Recall that \(v(\gamma)\) is a vertex of \(\partial Q\setminus\gamma\) such that one of its foot points lies in \(\gamma\) between \(\ell_i\) and \(\ell_{i+1}\), and the other foot point lies in \(\gamma\) between \(\ell_{i+1}\) and \(\ell_{i+2}\). Consider the point \(x \in \gamma\) closest to an endpoint of \(\ell_i\) along \(\gamma_1\) among the points in \(\gamma_1\) one of whose foot points is on \(\gamma_2\). Let \(y\) be the foot point of \(x\) lying on \(\gamma_2\). See Fig. 4a, b. If \(xy\) intersects some point in \(\partial Q\setminus\gamma\) (and therefore in \(\partial S\setminus\gamma\)) in its interior, such a point is \(v(\gamma)\). Otherwise, \(xy\) coincides with \(\ell_i\) or \(\ell_{i+2}\). See Fig. 4c. This means that \(\ell_i\) separates \(\ell_{i+1}\) and \(\ell_{i+2}\), or \(\ell_{i+2}\) separates \(\ell_i\) and \(\ell_{i+1}\). This contradicts that \(\ell_i, \ell_{i+1}\) and \(\ell_{i+2}\) appear on \(\partial S\) (and on \(\partial Q\)). Thus, the claim holds.

In the third step, we construct the extensions of \(v(\gamma)\), which separate \(\ell_i, \ell_{i+1}\) and \(\ell_{i+2}\). This is a contradiction. \(\square\)

**Lemma 3.7** Every subpolygon in the final subdivision has \(O(1)\) extensions constructed in the third step on its boundary.

**Proof** Consider a subpolygon \(S\) in the final subdivision. Let \(Q\) be the subpolygon in the subdivision from the second step containing \(S\). Consider an extension \(\ell\) incident to \(S\) constructed in the third step. Let \(v\) be the vertex defining the extension \(\ell\). Recall that the boundary of \(Q\) consists of the extensions \(\ell'_0, \ldots, \ell'_k\) and the polygonal chains of \(\partial P\) connecting the pairs of the extensions in consecutive order. Let \(\eta_i\) be the polygonal
Fig. 5  a $v$ lies on a polygonal chain $\eta$ and its foot points lies on $\partial Q \setminus \eta$. By Lemma 3.5, $\eta$ is either $\eta_0$ or $\eta_{k'}$. b one of the two polygonal chains connecting $h_1$ and $h_2$ along $\partial S$ is contained in $\eta_0$ for any two extensions $h_1$ and $h_2$ of vertices of $\eta_0$ constructed in the third step. Therefore, there are at most two such extensions on the boundary of $S$. 

chain of $\partial P$ connecting $\ell_i'$ and $\ell_{i+1}'$, excluding the extensions, for $0 \leq i < k'$, and $\eta_{k'}$ be the polygonal chain connecting $\ell_{k}'$ and $\ell_0'$, excluding the extensions.

We claim that $v$ is contained in $\eta_0$ or $\eta_{k'}$. Let $\eta$ be the polygonal chain containing $v$ among the polygonal chains $\eta_i$ for $0 \leq i \leq k'$. See Fig. 5a. Since $\ell$ is constructed in the third step, it separates three extensions on $\partial Q$. Therefore, the foot points of $v$ lie outside of $\eta$. Thus, $\eta$ has an extreme vertex, and $\eta$ is either $\eta_0$ or $\eta_{k'}$ by Lemma 3.5. (In Fig. 2b, both $v$ and $v'$ are contained in $\eta_4$, that is, the polygonal chain connecting $\ell_0$ and $\ell_4$ does not contain any other extensions.)

We also claim that there exist at most two vertices in $\eta_0$ that have both foot points in $\partial Q \setminus \eta_0$ and an extension 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 $u_2$ incident to $S$, respectively.

Since no foot point of $u_1$ and $u_2$ is in $\eta_0$, one of the two polygonal chains connecting $h_1$ and $h_2$ along $\partial S$ (but not containing them in its interior) is contained in $\eta_0$ and the other is disjoint with $\eta_0$. See Fig. 5b. Therefore, no other vertex in $\eta_0$ that has both foot points in $\partial S \setminus \eta_0$ and has an extension incident to $S$. This proves the claim. The same holds for $\eta_{k'}$.

Therefore, there are at most four extensions on $\partial S$ constructed in the third step: two of them are extensions of vertices of $\eta_0$ and the other two are extensions of vertices of $\eta_{k'}$. Thus the lemma holds.

Due to Lemmas 3.6 and 3.7, the following corollary holds.

Corollary 3.8 Every subpolygon in the final subdivision has $O(1)$ extensions on its boundary.

Lemma 3.9 Every subpolygon in the final subdivision has complexity $O(\Delta)$.

Proof Consider a subpolygon $S$ in the final subdivision. By Corollary 3.8, the boundary of $S$ consists of $O(1)$ vertical extensions and $O(1)$ polygonal chains from the boundary of $P$ connecting two consecutive endpoints of vertical extensions along $\partial S$. Each polygonal chain 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 along $\partial S$ is $O(\Delta)$. Therefore, $S$ has $O(\Delta)$ vertices on its boundary. \hfill $\square$
Therefore, we have the following lemma and theorem.

**Lemma 3.10** Given a simple n-gon and a parameter $\Delta$ with $\max\{n/s, (s \log n)/n\} \leq \Delta \leq n$, we can compute a set of $O(n/\Delta)$ extensions which subdivides the polygon into $O(n/\Delta)$ subpolygons of complexity $O(\Delta)$ in $O(n^2/s)$ time using $O(s)$ words of workspace.

**Theorem 3.11** Given a simple n-gon, we can compute a set of $O(\min\{n/s, s\})$ extensions which subdivides the polygon into $O(\min\{n/s, s\})$ subpolygons of complexity $O(\max\{n/s, s\})$ in $O(n^2/s)$ time using $O(s)$ words of workspace.

**Proof** If $s \leq \sqrt{n}$, we set $\Delta$ to $n/s$. In this case, we can subdivide the polygon into $O(s)$ subpolygons of complexity $O(n/s)$ by Lemma 3.10. If $s > \sqrt{n}$, we set $\Delta$ to $s$. Note that $\max\{n/s, (s \log n)/n\} \leq \Delta \leq n$ in both cases. We can subdivide the polygon into $O(n/s)$ subpolygons of complexity $O(s)$. Therefore, the theorem holds. $\square$

### 4 Applications

We first introduce other subdivision methods frequently used for $s$-workspace algorithms and provide comparison for our balanced subdivision method with them. Then we will present $s$-workspace algorithms that improve the previously best known results for three problems without increasing the size of the workspace.

#### 4.1 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. [3] presented a subdivision method that subdivides a simple $n$-gon into $O(s)$ subpolygons of complexity $O(n/s)$ using $O(s)$ non-crossing diagonals. They showed that the shortest path between any two points in the polygon 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 the subdivision, which dominates the overall running time. In fact, in the paper they asked whether a subdivision for computing shortest paths can be computed more efficiently using $O(s)$ words of workspace.

Instead of answering this question directly, Har-Peled [16] presented a way to subdivide a simple $n$-gon into $O(n/s)$ subpolygons of complexity $O(s)$. That is, this subdivision is defined by a set of non-crossing line segments whose endpoints are not necessarily contained in the boundary of the polygon. The number of segments defining this subdivision can be $\omega(s)$, for $s = \omega(\sqrt{n})$, and therefore the whole subdivision may not be stored in the $O(s)$ words of workspace. Instead, they gave a procedure to find the subpolygon of the subdivision containing a query point in $O(n + s \log s \log^4(n/s))$ expected time without maintaining the subdivision explicitly. They showed that one can find the shortest path between any two points using this subdivision in a way similar to the algorithm by Asano et al. in $O(n^2/s + (n/s)T(n, s))$ time, where $T(n, s)$ is the time for computing the subpolygon of the subdivision containing a query point. Therefore, the running time is $O(n^2/s + s \log s \log^4(n/s))$. 
The balanced subdivision that we propose can replace the subdivision methods in the algorithms by Asano et al. and Har-Peled for computing the shortest path between any two points. Moreover, our subdivision method has two advantages compared to the subdivision methods by Asano et al. and Har-Peled: (1) the subdivision can be computed faster than the one by Asano et al., and (2) we can keep the whole subdivision in the workspace unlike the one by Har-Peled. By using our balanced subdivision, we can improve the running times of trade-offs that use a subroutine of computing the shortest path between two points. Moreover, we can solve other application problems efficiently using $O(s)$ words of workspace. An example is to compute the shortest path between a query point and a fixed point after preprocessing the input polygon for the fixed point. See Lemma 5.5.

Note that our balanced subdivision is defined by a set of vertical extensions while the subdivision by Asano and Kirkpatrick [4] is defined by non-crossing diagonals. We would like to mention that we can also obtain $\Theta(s)$ non-crossing diagonals that subdivide a simple $n$-gon into subpolygons with $\Theta(n/s)$ vertices each in $O(n^2/s)$ deterministic time. This can be done by combining Theorem 4.3 and the algorithm by Aronov et al. [2, Theorem 8]. More specifically, the algorithm in [2] triangulates a simple polygon using $O(s)$ words of workspace. Whenever an edge of a triangulation is reported, we check if the edge subdivides the input polygon into two subpolygons of asymptotically equal size. If it is the case, the edge is a diagonal for the subdivision. This can be done in constant time by considering the indices of the endpoints of the edge. We do this for each subpolygon until every subpolygon has size $\Theta(n/s)$, which takes $O(n^2/s)$ deterministic time if we can triangulate a simple $n$-gon in $O(n^2/s)$ deterministic time.

### 4.2 Time-Space Trade-Offs Based on the Balanced Subdivision Method

By using our balanced subdivision method, we improve the previously best known running times for the following three problems without increasing the size of the workspace.

**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 [16] which takes $O(n^2/s + n \log s \log^4(n/s))$ expected time.

We can compute the shortest path between two query points using our balanced subdivision as follows. For $s = \Omega(\sqrt{n})$, we have the subdivision consisting of $O(n/s)$ subpolygons of complexity $O(s)$. Thus we use the algorithm by Har-Peled [16] described in the following lemma. Har-Peled presented an algorithm that for a given query point $q$ computes the subpolygon of the subdivision containing $q$ in $O(n + s \log s \log^4(n/s))$ expected time [16, Lemma 3.2]. It is shown in the paper that the shortest path between any two points can be computed using the algorithm as stated in the following lemma.

**Lemma 4.1** (Implied by [16, Lemma 4.1 and Theorem 4.3]) For a subdivision of a simple polygon consisting of $O(n/s)$ subpolygons, each of complexity $O(s)$, if the
subpolygon containing a query point can be computed in \( T(n) \) time using \( O(s) \) words of workspace, the shortest path between any two points can be computed in \( O((n/s)(T(n) + n)) \) time using \( O(s) \) words of workspace.

In our case, we can find the subpolygon of the balanced subdivision containing a query point in \( O(n) \) deterministic time. Combining this result with the lemma, we can compute the shortest path between any two points in \( O(n^2/s) \) deterministic time.

For \( s = O(\sqrt{n}) \), we have the subdivision consisting of \( O(s) \) subpolygons of complexity \( O(n/s) \). Instead of the algorithm by Har-Peled, we use the algorithm by Asano et al. [3] to compute the shortest path between any two points in the polygon, which still works for the subdivision by vertical extensions.

**Theorem 4.2** 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. [2] gave an \( s \)-workspace randomized algorithm for computing the shortest-path tree rooted at a given point. It uses the algorithm by Har-Peled [16] as a subroutine and takes \( O((n^2 \log n)/s + n \log s \log^5 (n/s)) \) expected time. If one uses Theorem 4.2 instead of Har-Peled’s algorithm, the running time improves to \( O((n^2 \log n)/s) \) expected time. In Sect. 5, we improve this algorithm even further using properties of our balanced subdivision.

**Computing a Triangulation of a Simple Polygon** Aronov et al. [2] 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. It uses the shortest path algorithm by Har-Peled [16] as a subroutine, which takes \( O(n^2/s + n \log s \log^5 (n/s)) \) expected time. By replacing this shortest path algorithm with ours in Theorem 4.2, we can obtain a triangulation of a simple polygon in \( O(n^2/s) \) deterministic time using \( O(s) \) words of workspace.

**Theorem 4.3** Given a simple polygon with \( n \) vertices, we can compute a triangulation of the simple polygon by returning the edges of the triangulation without repetition in \( O(n^2/s) \) deterministic time using \( O(s) \) words of workspace.

As mentioned by Aronov et al. [2], the algorithm can be modified to report the resulting triangles of a triangulation together with their adjacency information in the same time if \( s \geq \log n \).

**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 \( O(n^2/s + (n^2 \log n)/sc) \) expected time for an arbitrary positive constant \( c \). We use the following lemma given by Aronov et al. [2].
Lemma 5.1 [2, Lemma 6] For any point \( p \) in a simple \( n \)-gon, we can compute the shortest-path tree rooted at a point in the polygon 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 \( s = \Omega(\sqrt{n}) \). We consider the case of \( s = O(\sqrt{n}) \) first. For the case of \( s = \Omega(\sqrt{n}) \), we can store all edges of each subpolygon in the workspace.

5.1 Case of \( s = O(\sqrt{n}) \)

Given a point \( p \in P \), we want to report all edges of the shortest-path tree rooted at \( p \). Recall that there are \( O(s) \) extensions in the balanced subdivision in this case. We call an edge of a path a \( w \)-edge if it crosses an extension. For every extension \( a_1a_2 \) of the balanced subdivision, we first compute the \( w \)-edges of \( \pi(p, a_1) \) and \( \pi(p, a_2) \) in \( O(n^2/s^2) \) time in Sect. 5.1.1. We show that the total number of \( w \)-edges for all extensions is \( O(s) \). These \( w \)-edges allow us to compute the shortest path \( \pi(p, q) \) for any point \( q \) of \( P \) in \( O(n^2/s^2) \) time.

Then we decompose \( P \) into subpolygons associated with vertices in Sect. 5.1.2. For each subpolygon, we compute the shortest-path tree rooted at its associated vertex inside the subpolygon recursively. If a subpolygon satisfies one of the stopping criteria (to be defined later), we stop the recursion but proceed further to complete the shortest-path tree inside the subpolygon if necessary. Because of the space constraint, we restrict the depth of the recursion to be a constant.

5.1.1 Computing \( w \)-Edges

We compute all \( w \)-edges of the shortest paths between \( p \) and the endpoints of the extensions. The following lemma implies that there are \( O(s) \) \( w \)-edges of the shortest paths. 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 \( \pi(x, z) \).

Lemma 5.2 For an extension \( a_1a_2 \), there is at most one \( w \)-edge of \( \pi(p, a_i) \) for \( i = 1, 2 \) which is not a \( w \)-edge of \( \pi(p, b) \) or \( \pi(p, b') \) for any other extension \( bb' \) crossed by \( \pi(p, a_i) \).

Proof Let \( b_1 \) and \( b_2 \) be the endpoints of the first extension that we encounter during the traversal of \( \pi(p, a_1) \) from \( a_1 \) towards \( p \). See Fig. 6a. Consider the two junctions, one of \( \pi(p, a_1) \) and \( \pi(p, b_1) \), and one of \( \pi(p, a_1) \) and \( \pi(p, b_2) \). Let \( v \) be the junction closer to \( a_1 \). Note that \( \pi(p, a_1) \) is the concatenation of \( \pi(p, v) \) and \( \pi(v, a_1) \). The vertices of \( \pi(v, a_1) \) other than \( v \) lie in the subpolygon incident to \( a_1a_2 \) and \( b_1b_2 \). Thus every edge of \( \pi(v, a_1) \) not incident to \( v \) is contained in this subpolygon, and does not cross any extension. Therefore, the \( w \)-edge of \( \pi(p, a_1) \) which is not a \( w \)-edge of \( \pi(p, b) \) or \( \pi(p, b') \) for any extension \( bb' \) crossed by \( \pi(p, a_1) \) is unique: the edge of \( \pi(v, a_1) \) incident to \( v \). \( \Box \)

We consider the extensions one by one in a specific order and compute such \( w \)-edges one by one. To decide the order for considering the extensions, we define a \( w \)-tree \( T \),
We compute the junction $v$ of $\pi(p, a_1)$ and $\pi(p, b_1)$ by applying binary search on the $w$-edges of $\pi(p, b_1)$, 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 $O(n/s)$.

which is a dual graph of the balanced subdivision. Each node $\alpha$ of $T$ corresponds to a subpolygon of the balanced subdivision, and two nodes are connected by an edge in $T$ if their corresponding subpolygons share a common extension. Notice that $T$ is a tree. We let the node of $T$ whose corresponding subpolygon contains $p$ be the root of $T$. For a non-root node $\alpha$ of $T$, we let $d(\alpha)$ denote the extension which is shared by $\alpha$ and the parent of $\alpha$. For the root of $T$, we do not define $d(\alpha)$. Then we use the following property: a non-root node $\beta$ of $T$ is the parent of a node $\alpha$ if and only if $d(\beta)$ is the first extension 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 5.3** The $w$-tree can be built in $O(n)$ time using $O(s)$ words of workspace.

**Proof** We create the root, and its children by traversing the boundary of the subpolygon $S_p$ containing $p$. Then for each subpolygon $S$ incident to $S_p$, we traverse its boundary. Let $\alpha$ be the node of the tree corresponding to the extension incident to both $S$ and $S_p$. We create nodes for the extensions incident to $S$ other than $d(\alpha)$ as children of $\alpha$. We repeat this until we visit every extension of the balanced subdivision. In this way, we traverse the boundary of each subpolygon exactly once, thus the total running time is $O(n)$.

After constructing $T$, we apply depth-first search on $T$. We initialize $D$ to be the empty set. When we visit a node $\alpha$ of $T$, we compute the $w$-edges of $\pi(p, a_1)$ and $\pi(p, a_2)$ which are not in $D$ yet, where $a_1a_2 = d(\alpha)$, and insert them in $D$. Each $w$-edge in $D$ has information on the node of $T$ defining it and the subpolygons 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 $\alpha$ we visited before. Once the traversal is done, $D$ contains all $w$-edges in the shortest paths between $p$ and the endpoints of the extensions.

We show how to compute the $w$-edge of $\pi(p, a_1)$ which is not in $D$ yet. We can compute the $w$-edge of $\pi(p, a_2)$ not in $D$ yet analogously. By Lemma 5.2, there is at most one such edge of $\pi(p, a_1)$. Moreover, by its proof, it is the edge of $\pi(v, a_1)$ that is incident to $v$. Consider the two junctions, one of $\pi(p, b_1)$ and $\pi(p, a_1)$, and one...
of \(\pi(p, b_2)\) and \(\pi(p, a_1)\), where \(b_1b_2\) is the extension corresponding the parent of \(a\). Here, \(v\) is the junction closer to \(a_1\). Thus, to compute the \(w\)-edge, we first compute the junction of \(\pi(p, b_1)\) and \(\pi(p, a_1)\) and the junction of \(\pi(p, b_2)\) and \(\pi(p, a_1)\).

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 for \(s = O(\sqrt{n})\). The junction \(v_2\) of \(\pi(p, b_2)\) and \(\pi(p, a_1)\) can be computed analogously in the same time. Then we choose the one between \(v_1\) and \(v_2\) that is closer to \(a_1\).

To do this, we compute the set of the \(w\)-edges in \(D\) appearing on \(\pi(p, b_i)\) in order from \(b_i\) in \(O(s)\) time for \(i = 1, 2\). We denote the set by \(D(b_i)\). Note that the edges in \(D(b_1)\) are the \(w\)-edges of \(\pi(p, b_1)\). For an edge contained in both \(D(b_1)\) and \(D(b_2)\), we let the element in each set point to the element in the other set. We find two consecutive \(w\)-edges in \(D(b_1)\) containing \(v_1\) between them along \(\pi(p, b_1)\) by applying binary search on the edges in \(D(b_1)\).

Given any edge \(e\) in \(D(b_1)\), we can determine which connected component of \(\pi(p, b_1)\) 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 \(D(b_2)\). More specifically, if \(e\) is also contained in \(\pi(p, b_2)\), it is also a \(w\)-edge contained in \(D(b_2)\). Thus the element in \(D(b_2)\) corresponding to \(e\) has a pointer pointing to an element in \(D(b_1)\). This can be checked in constant time. If so, \(v_1\) is contained in the connected component of \(\pi(p, b_1)\) containing \(b_1\). Thus we are done. Otherwise, we extend \(e\) towards \(b_1\) until it escapes from \(S\), where \(S\) is the subpolygon incident to both \(a_1b_2\) and \(b_1b_2\). See Fig. 6a. Note that the extension crosses \(b_1b_2\) since both \(\pi(b_1, v_e)\) and \(\pi(b_2, v_e)\) are concave, where \(v_e\) is an endpoint 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 \(\partial S\) between \(a_1\) and \(b_1\) not containing \(a_2\), \(v_1\) lies in the connected component of \(\pi(p, b_1)\) containing \(p\) along \(\pi(p, b_1)\). Otherwise, \(v_1\) is contained in the other connected component. Therefore, we can find two consecutive \(w\)-edges in \(D(b_1)\) containing \(v_1\) between them along \(\pi(p, b_1)\) in \(O((n/s)\log s)\) time since the size of \(D(b_1) = O(s)\).

The edges of \(\pi(p, b_1)\) lying between the two consecutive \(w\)-edges are contained in the same subpolygon. Let \(x\) and \(y\) be the endpoints of the two consecutive edges of \(D(b_1)\) contained in the same subpolygon. Then we compute the edges of \(\pi(x, y)\) one by one from \(x\) to \(y\) inside the subpolygon containing \(x\) and \(y\). By Theorem 4.2, we can compute \(\pi(x, y)\) in \(O(n^2/s^3)\) time since the size of the subpolygon is \(O(n/s)\). Here, we use extra \(O(s)\) words of workspace for computing \(\pi(x, y)\). When the algorithm in Theorem 4.2 reports an edge \(f\) of \(\pi(x, y)\), we check which connected component \(\pi(x, y)\) containing \(v_1\) in \(O(n/s)\) time as we did before. We repeat 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 \(v\). We assume that \(v\) is the junction of \(\pi(v, a_1)\) and \(\pi(v, b_1)\). The case that \(v\) is the junction of \(\pi(v, a_1)\) and \(\pi(v, b_2)\) can be handled analogously. 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 Fig. 6b. We extend \(e_1\) and \(e_2\) towards \(b_1\) until they
escape from $P$ for the first time. The two extensions and $a_1a_2$ subdivide $P$ into regions. Consider the region bounded by the two extensions and $a_1a_2$. Note that the region can be represented using $O(1)$ words as the boundary consists of three line segments, one from each of the two extensions and $a_1a_2$, and two boundary (possibly empty) chains of $P$ connecting the segment of $a_1a_2$ to the other segments. The number of polygon vertices on the boundary of the region is $O(n/s)$. Moreover, $\pi(v, a_1)$ is contained in the region. Thus, the edge of $\pi(v, a_1)$ incident to $v$ inside the region is the edge we want to compute. We can compute it in $O(n^2/s^3)$ time by applying Theorem 4.2 to this region using $O(s)$ words of workspace.

In summary, we compute the w-edge of $\pi(p, a_1)$ which has not been computed yet 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 w-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 5.4** 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 extensions 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 $s = O(\sqrt{n})$.

**Lemma 5.5** 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 $p$.

**Proof** As a preprocessing, we compute the balanced subdivision of $P$. Then we compute all w-edges of the shortest paths between $p$ and the endpoints of the extensions in $O(n^2/s)$ time using Lemma 5.4.

To compute $\pi(p, q)$, we first find the subpolygon of the balanced subdivision containing $q$ in $O(n)$ time. The subpolygon is incident to $O(1)$ extensions due to Corollary 3.8. Consider the nodes in the w-tree corresponding to these extensions. One of the nodes is the parent of the others. We find the extension corresponding to the parent and denote it by $a_1a_2$. This extension is the first extension crossed by $\pi(p, q)$ we encounter during the traversal of $\pi(p, q)$ from $q$.

Then we compute the w-edge $e$ of $\pi(p, q)$ which is not in $D$ in $O(n^2/s^2)$ time as we did before, where $D$ is the set of all w-edges of the shortest paths between $p$ and the endpoints of the extensions. Let $v$ be the endpoint of $e$ closer to $q$. We report the edges of $\pi(q, v)$ from $q$ one by one using the algorithm in Theorem 4.2. Note that $\pi(q, v)$ is contained in a single subpolygon of the balanced subdivision. We can report them in $O(n^2/s^3)$ time since the subpolygon has complexity $O(n/s)$. Then we report $e$ as an edge of $\pi(p, q)$.

The remaining procedure is to report the edges of $\pi(p, q')$, where $q'$ is the endpoint of $e$ other than $v$. Note that $q'$ lies on $\pi(p, a_1) \cup \pi(p, a_2)$. Without loss of generality, we assume that it lies on $\pi(p, a_1)$. We can find all w-edges of $\pi(p, q')$ by computing
Subdivision of the region bounded by $\pi(v, g_1) \cup \pi(v, g_2)$ and the polygonal chain of $\partial P$ from $g_1$ to $g_2$ in clockwise order along $\partial P$ by the extensions of edges of $\pi(v, g_1)$ and $\pi(v, g_2)$ towards $g_1$ and $g_2$, respectively. We extend an edge of the paths if at least one of its endpoints are not on $\gamma$

all w-edges of $\pi(p, a_1)$ in $O(s)$ time. We consider the w-edges one by one from the one closest to $v'$ to the one farthest from $v'$. For two consecutive w-edges $e$ and $e'$ along $\pi(p, v')$, we report the edges of $\pi(p, v)$ lying between $e$ and $e'$. This takes $O(n^2/s^3)$ time using $O(s)$ words of workspace since all such edges are contained in a single subpolygon of complexity $O(n/s)$. Since there are $O(s)$ w-edges, we can report all edges of $\pi(p, q)$ in $O(n^2/s^2)$ time in total.  

5.1.2 Decomposing the Shortest-Path Tree Into Smaller Trees

To compute the shortest-path tree rooted at $p$ in $P$, we subdivide $P$ into subpolygons by shortest paths from $p$ to certain points on the boundary of $P$ such that each of the subpolygons is associated with a vertex of it. For each edge of the shortest paths, we report it if both of its’ endpoints are vertices of $P$ (or incident to $p$). For each such subpolygon $P'$, we do this recursively by shortest paths from $p'$ in $P'$, where $p'$ is the vertex associated with $P'$. Then the reported edges are indeed the edges of the shortest-path tree rooted at $p$ in $P$ by the construction of subpolygons. We will show that all edges of the shortest-path tree rooted at $p$ in $P$ are reported. We use a pair $(P', p')$ to denote the problem of reporting the edges of the shortest-path tree rooted at $p'$ in a subpolygon $P'$. 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 extensions. The boundary of each subpolygon consists of a polygonal chain of $\partial P$ with endpoints $g_1, g_2$ and the shortest paths $\pi(v, g_1)$ and $\pi(v, g_2)$, where $g_1, g_2$ are endpoints of extensions and $v$ is the junction of $\pi(p, g_1)$ and $\pi(p, g_2)$. In the second step, we decompose each subpolygon further 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 Fig. 7.
Consider a subpolygon $P'$ in the resulting subdivision. Its boundary consists of a polygonal chain of $\partial P$ and two line segments sharing a common endpoint $p'$. We can represent $P'$ using $O(1)$ words. Moreover, $P'$ has complexity $O(n/s)$. For any point $q$ in $P'$, $\pi(p, q)$ is the concatenation of $\pi(p, p')$ and $\pi(p', q)$. Therefore, the shortest-path tree rooted at $p'$ inside $P'$ coincides with the shortest-path tree rooted at $p$ inside $P$ restricted to $P'$. We can obtain the entire shortest-path tree rooted at $p$ inside $P$ by computing it on $(P', p')$ for every subpolygon $P'$ in the resulting subdivision and its associated vertex $p'$.

We define the orientation of an edge of the shortest-path tree using the indices of its endpoints (for example, from a smaller index to a larger index.) Note that the endpoints of an edge of the shortest-path tree are vertices of $P$ labeled from $v_0$ to $v_{n-1}$. We do not report an edge $e$ of the shortest path if $P'$ contains $e$ on its boundary and lies locally to the right of $e$ for a base problem $(P', p')$. Then every edge is reported exactly once.

**Computing the Subpolygons with Their Associated Vertices** In the following, we show how to obtain this subdivision using $O(s)$ words of workspace. Recall that the boundary of a subpolygon $S$ in the balanced subdivision consists of extensions and polygonal chains from $\partial P$. For each maximal polygonal chain $\gamma$ of $\partial S$ containing no endpoint of extensions in its interior, we do the following. Let $g_1$ and $g_2$ be the endpoints of $\gamma$. We compute the junction $v$ of $\pi(p, g_1)$ and $\pi(p, g_2)$ in $O(n^2/s^2)$ time as we did in Sect. 5.1.1.

Consider the region (subpolygon) of $P$ bounded by $\pi(v, g_1)$, $\pi(v, g_2)$ and $\gamma$. We compute the edges of $\pi(p, g_i)$ lying between $v$ and $g_i$ in order in $O(n^2/s^2)$ time using Lemma 5.5 for $i = 1, 2$. Clearly, these edges are the edges of $\pi(v, g_i)$. Whenever we compute an edge $e$ of $\pi(v, g_i)$, we check whether the endpoints of $e$ are on $\gamma$ or not, and obtain a subproblem $(P_e, v_e)$ as follows. Let $v_e$ be the endpoint closer to $v$. See Fig. 7 for an illustration.

- Both endpoints are on $\gamma$: $P_e$ is the subpolygon bounded by $e$ and a part of $\gamma$ connecting the two endpoints of $e$. (See $e_4$ in Fig. 7.)
- Exactly one of the endpoints are on $\gamma$: If $v_e$ is not on $\gamma$ we extend the edge incident to $v_e$ other than $e$ towards $g_i$ until it hits $\gamma$ in $O(n/s)$ time. (See $e_3$ in Fig. 7.) If $v_e$ is on $\gamma$, we extend $e$ towards $g_i$ until it hits $\gamma$ in $O(n/s)$ time. (See $e_1$ in Fig. 7.) Let $P_e$ be the subpolygon bounded by the extension (including $e$) and the part of $\gamma$ connecting the endpoint of $e$ and the endpoint of the extension lying on $\gamma$.
- No endpoint is on $\gamma$: We extend both edges of $\pi(v, g_2)$ incident to $v_e$ towards $g_i$ in $O(n/s)$ time. Let $P_e$ be the subpolygon bounded by the two extensions (including $e$) and the part of $\gamma$ connecting the endpoints of the extensions lying on $\gamma$. (See $e_2$ in Fig. 7.)

Therefore, we can compute the decomposition of the region of $P$ bounded by $\pi(v, g_1)$, $\pi(v, g_2)$ and $\gamma$ in $O(n^2/s^2 + nk/s)$ time using $O(s)$ words of workspace, 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 chains containing no endpoint of extensions in their interiors and the sum of $k$ over all such maximal polygonal chains is $O(n)$, the running time for decomposing the problem $(P, p)$ into smaller problems is $O(n^2/s)$. 
Lemma 5.6 We can decompose the problem \((P, p)\) into smaller problems in \(O(n^2/s)\) time using \(O(s)\) words of workspace.

Recursion We decompose each problem recursively unless the problem satisfies one of the three stopping criteria in Definition 5.7. Then we solve each base problem directly, that is, we report the edges of the shortest-path tree. For non-base problems, we do not report any edge of the shortest-path tree. More specifically, once the decomposition algorithm in Lemma 5.6 on \((P', p')\) reports a subproblem \((P'', p'')\), we do this recursively on \((P'', p'')\) until the recursion terminates. Then we go back to \((P', p')\) and continue to apply the decomposition algorithm in Lemma 5.6 on \((P', p')\). To continue this, we have to keep track of the balanced subdivision of \(P'\) while the recursion on \((P'', p'')\) is executed. Thus we have to analyze the space complexity in the recursion carefully. The analysis can be found at the end of Sect. 5.1.3.

Definition 5.7 (Stopping criteria) There are three stopping criteria for \((P', p')\):

1. \(P'\) has \(O(s)\) vertices,
2. \(s \geq \sqrt{|P'|}\), where \(|P'|\) is the complexity of \(P'\), and
3. the depth of the recursion is a positive constant \(c\).

When stopping criterion (1) holds, we compute the shortest-path tree directly using the algorithm by Guibas et al. [15]. When stopping criterion (2) holds, we apply the algorithm described in Sect. 5.2 that computes the shortest-path tree rooted at \(p'\) inside \(P'\) in \(O(|P'|^2/s)\) time for the case that \(s \geq \sqrt{|P'|}\), where \(|P'|\) is the complexity of \(P'\). When stopping criterion (3) holds, we compute the shortest-path tree using Lemma 5.1.

5.1.3 Analysis of the Recursion

Time Complexity Consider the base problems. All base problems induced by stopping criterion (1) can be handled in \(O(n)\) time in total because the subpolygons corresponding to them are pairwise interior-disjoint. For the base problems induced by stopping criterion (2), the subpolygons corresponding to them are also pairwise interior-disjoint. The time for handling these problems is the sum of \(O(|P'|^2/s)\) over all the problems \((P', p')\). The running time is \(O(n^2/s)\) because we have \(O(\sum |P'|^2/s) = O(n/s \sum |P'|) = O(n^2/s)\).

Now we consider the base problems induced by stopping criterion (3). For depth \(c\) of the recursion, every subpolygon has complexity \(O(n/s^c)\). Moreover, the total complexity of all subpolygons at depth \(c\) is \(O(n)\). By Lemma 5.1, the expected time for computing the shortest-path trees in all subpolygons is the sum of \(O(|P'|^2 \log n)\) over all subpolygons \(P'\) at depth \(c\). Therefore, we can solve all problems at depth \(c\) in \(O((n^2 \log n)/s^c)\) time because we have \(O(\sum_i |P_i|^2 \log n) = O(n/s^c \sum_i |P_i| \log n) = O((n^2 \log n)/s^c)\).

We analyze the running time for decomposing a problem into smaller problems. Consider depth \(k\) for \(1 \leq k < c\). Let \((P_1, p_1), \ldots, (P_t, p_t)\) be the problems at depth \(k\). Note that the sum of \(|P_i|\) over all indices from 1 to \(t\) is \(O(n)\). For each \(P_i\), we construct the balanced subdivision of \(P_i\) in \(O(|P_i|^2/s)\) time, compute \(O(s)\) \(w\)-edges of the shortest paths between \(p_i\) and the endpoints of the extensions in \(O(|P_i|^2/s^2)\) time, and decompose the problem into smaller problems in \(O(|P_i|^2/s)\) time. Thus,
the decomposition takes $O(\sum_{i} |P_i|^2/s) = O(n^2/s)$ time for the problems at depth $k$. Since $c$ is a constant, the decomposition over the $c$ depths takes $O(n^2/s)$ time.

Therefore, the total running time is $O(n^2/s + (n^2 \log n)/s^c)$ for an arbitrary constant $c > 0$.

Space Complexity To handle each problem $(P', p')$, we maintain the balanced subdivision of $P'$ using $O(s)$ words of workspace. Until all subproblems of $(P', p')$ for all depths are handled, we keep this balanced subdivision. However, we do not keep the subdivision for two distinct problems in the same depth at the same time. Therefore, the total space complexity is $O(cs)$, which is $O(s)$.

**Lemma 5.8** 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 $s = O(\sqrt{n})$, where $c$ is an arbitrary positive constant.

### 5.2 Case of $s = \Omega(\sqrt{n})$

For the case of $s = \Omega(\sqrt{n})$, the balanced subdivision consists of $O(n/s)$ subpolygons of complexity $O(s)$. The algorithm for this case is similar to the one for the case of $s = O(\sqrt{n})$, except that we do not use Theorem 4.2 and Lemma 5.1. Instead, we use the fact that we can store all edges of each subpolygon in the workspace.

As we did before, we compute all $w$-edges of the shortest paths between $p$ and the endpoints of the extensions. Using them, we decompose $(P, p)$ into a number of subproblems. In this case, we will see that every subproblem of $(P, p)$ is a base problem due to stopping criterion (1) in Definition 5.7. Then we solve each subproblem directly using the algorithm by Guibas et al. [15].

**Lemma 5.9** We can compute all $w$-edges of the shortest paths between $p$ and the endpoints of the extensions in $O(n)$ time.

**Proof** As we did in Sect. 5.1.1, we apply depth-first search on the $w$-tree and compute the $w$-edges one by one. When we reach a node $\alpha$ of the $w$-tree, we compute the $w$-edges of $\pi(p, a_1)$ and $\pi(p, a_2)$ which are not computed yet, where $a_1$ and $a_2$ are the endpoints of the extension $d(\alpha)$. We show how to compute the edges of $\pi(p, a_1)$ only. The case for $\pi(p, a_2)$ can be handled analogously. Consider the two junctions, one of $\pi(p, b_1)$ and $\pi(p, a_1)$, and one of $\pi(p, b_2)$ and $\pi(p, a_1)$, where $b_1b_2$ is the extension corresponding to the parent of $\alpha$. Let $v$ be the one closer to $a_1$. By Lemma 5.2, there is at most one such $w$-edge of $\pi(p, a_1)$. Moreover, by the proof of the lemma, such an edge is incident to $v$ on $\pi(v, a_1)$.

We compute the junction $v_1$ of $\pi(p, b_1)$ and $\pi(p, a_1)$ as follows. Consider the endpoints of the $w$-edges of $\pi(p, b_1)$ sorted along $\pi(p, b_1)$ from $p$. We connect the endpoints of the $w$-edges by line segments in this order to form a polygonal chain. We denote the resulting polygonal chain by $\mu_1$. Notice that it might intersect the boundary of $P$. We also do this for $b_2$ and denote the resulting polygonal chain by $\mu_2$. We can compute $\mu_1$ and $\mu_2$ in $O(n/s) = O(s)$ time.

Consider the union of the subpolygon of the balanced subdivision incident to both $a_1a_2$ and $b_1b_2$, and the region (funnel) bounded by $\mu_1, \mu_2$ and $b_1b_2$. The complexity of
this union is $O(s)$. Thus, we can compute the shortest-path tree rooted at $p$ restricted in this union using an algorithm by Guibas et al. [15]. This algorithm computes the shortest-path tree rooted at a given point in time linear in the complexity of the input simple polygon using linear space. We find the maximal subchain of $\mu_1$ which is a part of $\pi(p, a_1)$. One endpoint of the subchain is $p$. Let $v'$ be the other endpoint. We find two consecutive w-edges $e$ and $e'$ of $\pi(p, b_1)$ containing $v'$ between them along $\pi(p, b_1)$. Then they also contain the junction $v_1$ between them along $\pi(p, b_1)$.

Let $x$ and $y$ be the endpoints of $e$ and $e'$, respectively, that are contained in the same subpolygon. We compute $\pi(x, y)$ one by one using the algorithm by Guibas et al. We compute the extensions of the edges of $\pi(x, y)$ towards the subpolygon containing $a_1a_2$ and $b_1b_2$ on its boundary in $O(s)$ time. Then we can decide which vertex of $\pi(x, y)$ is the junction $v_1$. This takes $O(s)$ time in total.

Moreover, while we compute $v_1$, we can obtain the edge of $\pi(v_1, a_1)$ incident to $v_1$. Thus, we can obtain the w-edge of $\pi(p, a_1)$ which is not computed yet in $O(s)$ time. Since there are $O(n/s)$ nodes in the w-tree, we can compute all w-edges of the shortest paths between $p$ and the endpoints of the extensions in $O(n)$ time in total. □

We decompose the problem $(P, p)$ into smaller problems in $O(n^2/s)$ time in a way similar to the one in Sect. 5.1.2.

**Lemma 5.10** We can decompose the problem $(P, p)$ into smaller problems defined in Sect. 5.1.2 in $O(n^2/s)$ time.

**Proof** Recall that the boundary of a subpolygon $S$ in the balanced subdivision consists of extensions and polygonal chains from $\partial P$. For each maximal polygonal chain $\eta$ of $\partial S$ containing no endpoint of extensions in its interior, we do the followings. Let $g_1$ and $g_2$ be the endpoints of $\eta$. We compute the junction $v$ of $\pi(p, g_1)$ and $\pi(p, g_2)$ in $O(s)$ time as we showed in the proof of Lemma 5.9.

Then we compute the w-edges of $\pi(v, g_1)$ and $\pi(v, g_2)$ in $O(n/s) = O(s)$ time. We are to compute the first point hit by the extension (ray) of each w-edge towards $\eta$. See Fig. 8a. To do this, we compute the first point hit by the extension of the w-edge of $\pi(v, g_1)$ closest to $g_1$ towards $\mu$ in $O(s)$ time. Then we connect $v$, the w-edges of $\pi(v, g_1)$ and the first point hit by the extension to form a polygonal chain connecting $v$ and a point in $\eta$. We also do this for $g_2$. We let $\mu$ be the union of the two polygonal chains. We compute the union of $\mu$ and the part of $\eta$ connecting the two endpoints of $\mu$, which forms a simple polygon. Note that the polygon might intersects the boundary of $P$. See Fig. 8b. Then we apply the shortest-path tree algorithm by Guibas et al. [15]. This takes $O(s)$ time since there are $O(s)$ such w-edges and $\eta$ has complexity $O(s)$.

For the edges of $\pi(v, g_1)$ and $\pi(v, g_2)$ lying between two consecutive w-edges, we observe that they are contained in the same subpolygon of the balanced subdivision. Thus we can compute such edges and extend them towards $\eta$ by applying the algorithm by Guibas et al. See Fig. 8c. For a pair of consecutive w-edges, we can do this in $O(s)$ time. Since there are $O(n/s)$ such pairs, this takes $O(n)$ time for each maximal polygonal chain $\eta$. There are $O(n/s)$ maximal polygonal chains $\eta$, and thus the total running time is $O(n^2/s)$.

Note that the boundary of each subpolygon $P'$ consists of two line segments and a part of $\partial P$ containing no endpoint of extensions in its interior. Thus, the complexity
We want to compute all points on \( \eta \) marked with circles. But the complexity of the subpolygon induced by \( \pi(v, g_1), \pi(v, g_2) \) and \( \eta \) might exceed the size of workspace, \( b \) there are three \( w \)-edges of \( \pi(v, g_1) \) and \( \pi(v, g_2) \) (thick line segments). We first compute the extensions of them in a simplified polygonal region (dashed), \( c \) to compute the extensions of the edges lying between two consecutive \( w \)-edges, we consider a subpolygon of complexity \( O(s) \) (the gray region), and apply the algorithm by Guibas et al

of \( P' \) is \( O(s) \). This means that all subproblems of \( (P, p) \) are base problems due to stopping criterion (1) in Definition 5.7. We can solve all base problems in \( O(n) \) time in total. Therefore, we can compute the shortest-path tree in \( O(n^2/s) \) deterministic time in total.

**Lemma 5.11** Given a point \( p \) in a simple polygon with \( n \) vertices, we can compute the shortest-path tree rooted at \( p \) in \( O(n^2/s) \) deterministic time using \( O(s) \) words of workspace for \( s = \Omega(\sqrt{n}) \).

Combining the algorithm for case of \( s = O(\sqrt{n}) \) in Sect. 5.1 with the lemma above, we have the following theorem.

**Theorem 5.12** 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 positive constant \( c \).

Here, the size of the workspace is \( O(cs) \). Thus, by changing the roles of \( c \) and \( s \), we can achieve another \( s \)-workspace algorithm. In particular, by setting \( c \) to the size of workspace and \( s \) to 2, we have the following theorem.

**Theorem 5.13** 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 \( s \leq \log \log n \).

### 6 Conclusion

We present an \( s \)-workspace algorithm for computing a balanced subdivision of a simple polygon consisting of \( O(\min\{n/s, s\}) \) subpolygons of complexity \( O(\max\{n/s, s\}) \). This subdivision can be computed more efficiently than other subdivisions suggested in the context of time-space trade-offs, and therefore can be used for solving several fundamental problems in a simple polygon more efficiently. Since our subdivision method keeps all extensions of the balanced subdivision in the workspace, it has a few other application problems, including the problem for answering a single-source
shortest path query. We also believe that we can preprocess a simple polygon and maintain a data structure of size \( O(s) \) so that for any two points \( x \) and \( y \) in a simple polygon \( \pi(x, y) \) can be computed in \( o(n^2/s) \) time with \( O(s) \) words of workspace by combining the ideas from Guibas and Hershberger [14] with our subdivision method. We leave this as a future work.

References

1. Ahn, H.-K., Baraldo, N., Oh, E., Silvestri, F.: A time-space trade-off for triangulations of points in the plane. In: Proceedings of the 23rd Annual International Computing and Combinatorics Conference (COCOON 2017), pp. 3–12 (2017)
2. Aronov, B., Korman, M., Pratt, S., van Renssen, A., Roeloffzen, M.: Time-space trade-offs for triangulating a simple polygon. J. Comput. Geom. 8(1), 105–124 (2017)
3. Asano, T., Buchin, K., Buchin, M., Korman, M., Mulzer, W., Rote, G., Schulz, A.: Memory-constrained algorithms for simple polygons. Comput. Geom. 46(8), 959–969 (2013)
4. Asano, T., Kirkpatrick, D.: Time-space tradeoffs for all-nearest-larger-neighbors problems. In: Proceedings of the 13th Algorithms and Data Structures Symposium (WADS 2013), pp. 61–72 (2013)
5. Asano, T., Mulzer, W., Rote, G., Wang, Y.: Constant-work-space algorithms for geometric problems. J. Comput. Geom. 2(1), 46–68 (2011)
6. Asano, T., Mulzer, W., Wang, Y.: Constant-work-space algorithms for shortest paths in trees and simple polygons. J. Graph Algorithms Appl. 15(5), 569–586 (2011)
7. Banyassady, B., Barba, L., Mulzer, W.: Time-space trade-offs for computing Euclidean minimum spanning trees. In: Proceedings of the 13th Latin American Theoretical Informatics Symposium (LATIN 2018), pp. 108–119 (2018)
8. Banyassady, B., Korman, M., Mulzer, W.: Computational geometry column 67. SIGACT News 49(2), 77–94 (2018)
9. Barba, L., Korman, M., Langerman, S., Sadakane, K., Silveira, R.I.: Space-time trade-offs for stack-based algorithms. Algorithmica 72(4), 1097–1129 (2015)
10. Chazelle, B.: Triangulating a simple polygon in linear time. Discrete Comput. Geom. 6(3), 485–524 (1991)
11. Darwish, O., Elmasry, A.: Optimal time-space tradeoff for the 2D convex-hull problem. In: Proceedings of the 22nd Annual European Symposium on Algorithms (ESA 2014), pp. 284–295 (2014)
12. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer TELOS, Berlin (2008)
13. Frederickson, G.N.: Upper bounds for time-space trade-offs in sorting and selection. J. Comput. Syst. 34(1), 19–26 (1987)
14. Guibas, L., Hershberger, J.: Optimal shortest path queries in a simple polygon. J. Comput. Syst. Sci. 39(2), 126–152 (1989)
15. Guibas, L., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.E.: Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. Algorithmica 2(1), 209–233 (1987)
16. Har-Peled, S.: Shortest path in a polygon using sublinear space. J. Comput. Geom. 7(2), 19–45 (2015)
17. Korman, M., Mulzer, W., van Renssen, A., Roeloffzen, M., Seiferth, P., Stein, Y.: Time-space trade-offs for triangulations and Voronoi diagrams. Comput. Geom. 73, 35–45 (2018)

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.