Shortest Paths Among Obstacles in the Plane Revisited[*]

Haitao Wang[†]

Abstract

Given a set of pairwise disjoint polygonal obstacles in the plane, finding an obstacle-avoiding Euclidean shortest path between two points is a classical problem in computational geometry and has been studied extensively. The previous best algorithm was given by Hershberger and Suri [FOCS 1993, SIAM J. Comput. 1999] and the algorithm runs in $O(n \log n)$ time and $O(n \log n)$ space, where n is the total number of vertices of all obstacles. The algorithm is time-optimal because $\Omega(n \log n)$ is a lower bound. It has been an open problem for over two decades whether the space can be reduced to $O(n)$. In this paper, we settle it by solving the problem in $O(n \log n)$ time and $O(n)$ space, which is optimal in both time and space; we achieve this by modifying the algorithm of Hershberger and Suri. Our new algorithm can build a shortest path map for a source point s in $O(n \log n)$ time and $O(n)$ space, such that given any query point t, the length of a shortest path from s to t can be computed in $O(\log n)$ time and a shortest path can be produced in additional time linear in the number of edges of the path.

1 Introduction

Let $\mathcal{P}$ be a set of pairwise disjoint polygonal obstacles with a total of n vertices in the plane. The plane minus the interior of the obstacles is called the *free space*, denoted by $\mathcal{F}$. Given two points s and t in $\mathcal{F}$, we consider the problem of finding a Euclidean shortest path from s to t in $\mathcal{F}$. This is a classical problem in computational geometry and has been studied extensively, e.g., [?, ?, ?, 2, 7–10, 12, 13, 17, 19, 20].

Two main methods have been used to tackle the problem in the literature: the visibility graph method and the continuous Dijkstra method. The visibility graph method is to first construct the visibility graph of the vertices of $\mathcal{P}$ along with s and t, and then run Dijkstra's shortest path algorithm on the graph to find a shortest s-t path. The best algorithms for constructing the visibility graph run in $O(n \log n + K)$ time [7] or in $O(n + h \log^{1+\epsilon} h + K)$ time [3] for any constant $\epsilon > 0$, where h is the number of obstacles of $\mathcal{P}$ and K is the number of edges of the visibility graph. Because $K = \Omega(n^2)$ in the worst case, the visibility graph method inherently takes quadratic time to solve the shortest path problem. To achieve a sub-quadratic time solution, Mitchell [20] made a breakthrough and gave an $O(n^{3/2+\epsilon})$ time algorithm by using the continuous Dijkstra method. Also using the continuous Dijkstra method plus a novel conforming subdivision of the free space, Hershberger and Suri [13] presented an algorithm of $O(n \log n)$ time and $O(n \log n)$ space; the running time is optimal as $\Omega(n \log n)$ is a lower bound in the algebraic computation tree model. Hershberger and Suri [13] raised an open question whether the space of their algorithm can be reduced to $O(n)$.[1]

The continuous Dijkstra algorithms in both [20] and [13] actually construct the *shortest path map*, denoted by $SPM(s)$, for a source point s. The map is of $O(n)$ size and can be used to answer shortest path queries. By building a point location data structure [6, 16] on $SPM(s)$ in additional $O(n)$ time, given a query point t, the shortest path length from s to t can be computed in $O(\log n)$ time and a shortest s-t path can be reported in time linear in the number of edges of the path.

The problem setting for $\mathcal{P}$ is usually referred to as *polygonal domains* or *polygons with holes* in the literature. The shortest path problem in simple polygons is relatively easier [8–10, 12, 17]. Guibas et al. [9] gave an algorithm that can construct the shortest path map with respect to a source point in linear time. For two-point shortest path query problem where both s and t are query points, Guibas and Hershberger [8, 10] built a data structure in linear time such that each query can be answered in $O(\log n)$ time. In contrast, the two-point query problem in polygonal domains is much more challenging: to achieve $O(\log n)$ time queries, the current best result uses $O(n^{11})$ space [4].

1.1 Our result

In this paper, by modifying the algorithm of Hershberger and Suri [13] (referred to as the *HS algorithm*),

[*]This research was supported in part by NSF under Grant CCF-2005323.

[†]This research was supported in part by NSF under Grant CCF-2005323.

[1]An unrefereed report [15] announced an algorithm based on the continuous Dijkstra approach with $O(n + h \log h \log n)$ time and $O(n)$ space.

we show that the problem of finding a shortest path among obstacles in $\mathcal{P}$ is solvable in $O(n \log n)$ time and $O(n)$ space, which is optimal in both time and space. This answers the longstanding open question of Hershberger and Suri [13]. Our algorithm actually constructs the shortest path map $\mathit{SPM}(s)$ for a source point s in $O(n \log n)$ time and $O(n)$ space. We give an overview of our approach below.

The reason that the HS algorithm needs $O(n \log n)$ space is two-fold. First, it uses fully persistent binary trees (with the path-copying method) to represent wavefronts. Because there are $O(n)$ events in the wavefront propagation algorithm and each event costs $O(\log n)$ additional space on a persistent tree, the total space consumed in the algorithm is $O(n \log n)$. Second, in order to construct $\mathit{SPM}(s)$ after the propagation algorithm, some previous versions of the wavefronts are needed, which are maintained in those persistent trees. We resolve these two issues in the following way.

We still use persistent trees to represent wavefronts. However, since there are $O(n)$ events in the propagation algorithm, we divide the algorithm into $O(\log n)$ phases such that each phase has no more than $n/\log n$ events. The total additional space for processing the events using persistent trees in each phase is $O(n)$. At the end of each phase, we "reset" the space of the algorithm by only storing a "snapshot" of the algorithm (and discarding all other used space) so that (1) the snapshot contains sufficient information for the subsequent algorithm to proceed as usual, and (2) the total space of the snapshot is $O(n)$. More specifically, the HS algorithm relies on a *conforming subdivision* of the free space $\mathcal{F}$ to guide the wavefront propagation; our snapshot is composed of the wavefronts of a set of edges of the subdivision (intuitively, those are the edges in the forefront of the current wavefront). In this way, we reset the space to $O(n)$ at the end of each phase. As such, the total space of the propagation algorithm is bounded by $O(n)$. This resolves the first issue.

For the second issue of constructing $\mathit{SPM}(s)$, the HS algorithm relies on some historical wavefronts produced during the propagation algorithm, plus some marked wavelet generators (a wavelet generator is either s or a vertex of $\mathcal{P}$) for each cell of the conforming subdivision; the total number of marked generators for all cells of the subdivision is $O(n)$. Due to the space-reset, our algorithm does not maintain historical wavefronts anymore, and thus we need to somehow restore these wavefronts. To this end, a key observation is that by marking a total of $O(n)$ additional wavelet generators it is possible to restore all historical wavefronts that are needed for constructing $\mathit{SPM}(s)$. In this way, $\mathit{SPM}(s)$ can be constructed in $O(n \log n)$ time and $O(n)$ space.

Outline. The rest of the paper is organized as follows. Section 2 defines notation and introduces some concepts. As our algorithm is a modification of the HS algorithm, we briefly review the HS algorithm in Section 3 and refer the reader to [13] for details. Our modified algorithm is described in Section 4. Section 5 concludes with remarks on some problem extensions and other possible applications of our technique.

2 Preliminaries

For any two points p and q in the free space $\mathcal{F}$ of $\mathcal{P}$, we use $\pi(p,q)$ to denote a shortest path from p to q in $\mathcal{F}$. Note that $\pi(p,q)$ may not be unique, in which case $\pi(p,q)$ may refer to an arbitrary shortest path. Let $d(p,q)$ denote the length of $\pi(p,q)$; we call $d(p,q)$ the *geodesic distance* between p and q. For two line segments e and f in the free space $\mathcal{F}$, their *geodesic distance* is defined to be the minimum geodesic distance between any point on e and any point on f, i.e., $\min_{p\in e, q\in f} d(p,q)$; by slightly abusing the notation, we use $d(e,f)$ to denote their geodesic distance.

For any two points a and b in the plane, denote by $\overline{ab}$ the line segment with a and b as endpoints; denote by $|\overline{ab}|$ the length of the segment.

Throughout the paper, let s represent the source point. For any point p in the free space $\mathcal{F}$, the vertex prior to p in $\pi(s,p)$ is called the *predecessor* of p in $\pi(s,p)$. The *shortest path map* $\mathit{SPM}(s)$ of s is a decomposition of $\mathcal{F}$ into maximal regions such that each region R has an obstacle vertex that is the predecessor of all points in R; each edge of R is either a fragment of an obstacle edge or a portion of a *bisector*, which is the locus of points p with $d(s,u) + |\overline{pu}| = d(s,v) + |\overline{pv}|$ for two obstacle vertices u and v (we use $B(u,v)$ to denote their bisector). $B(u,v)$ is in general a hyperbola; a special case happens if one of u and v is the predecessor of the other, in which case $B(u,v)$ is on a straight line.

For any compact region A of the plane, let ∂A denote its boundary. We use $\partial\mathcal{P}$ to denote the union of the boundaries of all obstacles of $\mathcal{P}$. We call the vertices of $\mathcal{P}$ and s the *obstacle vertices* and call the edges of $\mathcal{P}$ the *obstacle edges*.

The conforming subdivision $\mathcal{S}'$ of the free space. A novel contribution of the paper [13] is a conforming subdivision $\mathcal{S}'$ of the free space $\mathcal{F}$. We briefly review it here by following the notation in [13]. Let V denote the set of all obstacle vertices of $\mathcal{P}$ plus the source point s.

The conforming subdivision $\mathcal{S}'$ is built upon a conforming subdivision $\mathcal{S}$ with respect to the points of V (ignoring the obstacle edges). The subdivision $\mathcal{S}$ is a quad-tree-style subdivision of the plane into $O(n)$ cells with the following properties. (1) Each cell is either a square or a square annulus (i.e., an outer square with an inner square hole). (2) Each cell of $\mathcal{S}$ contains at most one point of V (only square cells can contain points of V). (3) Each edge of $\mathcal{S}$ is axis-parallel. (3) Each edge e of $\mathcal{S}$ is *well-covered*, i.e., there exists a set of $O(1)$ cells whose union $\mathcal{U}(e)$ contains e with the following three properties: (a) the size of $\mathcal{U}(e)$ is $O(1)$; (b) for each edge f on $\partial\mathcal{U}(e)$, the Euclidean distance between e and f (i.e., the minimum $|\overline{pq}|$ among all points $p \in e$ and $q \in f$) is at least $2 \cdot \max\{|e|, |f|\}$; (c) $\mathcal{U}(e)$, which is called the *well-covering region* of e, contains at most one point of V.

The conforming subdivision $\mathcal{S}'$ of the free space $\mathcal{F}$ is built by inserting the obstacle edges into $\mathcal{S}$. More specifically, $\mathcal{S}'$ is a subdivision of $\mathcal{F}$ into $O(n)$ cells with the following properties. (1) Each cell of $\mathcal{S}'$ is one of the connected components formed by intersecting $\mathcal{F}$ with an axis-parallel rectangle (which is the union of a set of adjacent cells of $\mathcal{S}$) or a square annulus of $\mathcal{S}$. (2) Each cell of $\mathcal{S}'$ contains at most one point of V. (3) $\mathcal{S}'$ has two types of edges: *opaque edges*, which are fragments of obstacles edges, and *transparent edges*, which are introduced by the subdivision construction (these edges are in the free space $\mathcal{F}$); each transparent edge is axis-parallel. (4) Each point of V is incident to a transparent edge. (5) Each transparent edge e of $\mathcal{S}'$ is *well-covered*, i.e., there exists a set of $O(1)$ cells whose union $\mathcal{U}(e)$ contains e with the following three properties: (a) the size of $\mathcal{U}(e)$ is $O(1)$; (b) for each edge f on $\partial\mathcal{U}(e)$, the geodesic distance $d(e, f)$ between e and f is at least $2 \cdot \max\{|e|, |f|\}$; (c) $\mathcal{U}(e)$, which is called the *well-covering region* of e, contains at most one point of V.

Both subdivisions $\mathcal{S}$ and $\mathcal{S}'$ can be constructed in $O(n \log n)$ time and $O(n)$ space [13].

3 A brief review of the HS algorithm

In this section, we briefly review the HS algorithm. We refer the readers to [13] for details.

The algorithm starts generating a wavefront from the source point s and uses the conforming subdivision $\mathcal{S}'$ to guide the wavefront expansion. At any moment of the algorithm, the wavefront consists of all points of $\mathcal{F}$ with the same geodesic distance from s (e.g., see Fig. 1). The wavefront is composed of a sequence of *wavelets*, each centered at an obstacle vertex that is already covered by the wavefront (the obstacle vertex is called the *generator* of the wavelet). To simulate the wavefront expansion, the algorithm produces the wavefront passing through each transparent edge of $\mathcal{S}'$. As it is difficult to compute a true wavefront for each transparent edge e of $\mathcal{S}'$, a key idea of the HS algorithm is to compute two one-sided wavefronts (called *approximate wavefronts*) for e, each representing the wavefront coming from one side of e. Intuitively, an approximate

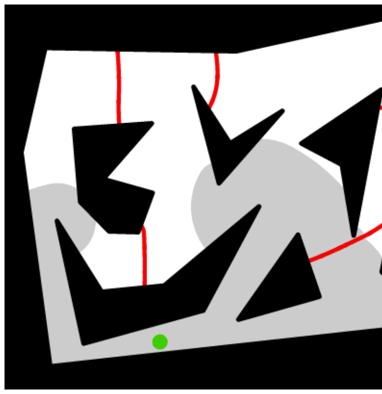

Figure 1: Illustrating the wavefront. The black region are obstacles. The green p
The gray region is the free space that has been covered by the wavefront. The
region is the wavefront. The figure is generated using the applet at [11].

wavefront from one side of e is what the true wavefront would be if the wavefront were blocked off at e by considering e as an (open) opaque obstacle segment. To limit the interaction between approximate wavefronts from different sides of e, when an approximate wavefront propagates across e, an artificial wavelet is created at each endpoint of e. This is a mechanism to eliminate an approximate wavefront from one side of e if it arrives at e later than the approximate wavefront from the other side of e.

In the following, unless otherwise stated, a wavefront at a transparent edge e of $\mathcal{S}'$ refers to an approximate wavefront. We use $W(e)$ to denote a wavefront at e. As e has two wavefronts, to make the description concise, depending on the context, $W(e)$ may refer to both wavefronts, i.e., the discussion on $W(e)$ applies to both wavefronts. For example, "construct the wavefronts $W(e)$" means construct the two wavefronts at e. Also, unless otherwise stated, an edge of $\mathcal{S}'$ refers to a transparent edge.

For each transparent edge e of $\mathcal{S}'$, define $\mathit{input}(e)$ as the set of transparent edges on the boundary of $\mathcal{U}(e)$ and define $\mathit{output}(e) = \mathit{input}(e) \cup \{g \mid e \in \mathit{input}(g)\}$[2]. As the size of $\mathcal{U}(e')$ for each transparent edge e' of $\mathcal{S}'$ is $O(1)$, both $|\mathit{input}(e)|$ and $|\mathit{output}(e)|$ are $O(1)$. The wavefronts at e are computed from the wavefronts at all edges of $\mathit{input}(e)$; this guarantees the correctness because e is in $\mathcal{U}(e)$ (and thus the shortest path $\pi(s, p)$ passes through some edge $f \in \mathit{input}(e)$ for any point $p \in e$). After the wavefronts at e are computed, they will pass to the edges of $\mathit{output}(e)$. In the meanwhile, the geodesic distances from s to both endpoints of e will be computed. By the property (4) of $\mathcal{S}'$, each obstacle vertex is incident to a transparent edge of $\mathcal{S}'$. Hence, after the algorithm is finished, geodesic distances from s to all obstacle vertices will be available.

[2]The set $\mathit{input}(e)$ is included in $\mathit{output}(e)$ because the algorithm relies on the property that edges of $\mathit{output}(e)$ form a cycle enclosing e.

The process of passing the wavefronts $W(e)$ at e to all edges $g \in \mathit{output}(e)$ is called the *wavefront propagation procedure* (see Section 5 in [13]), which will compute the wavefront $W(e, g)$, the portion of $W(e)$ that passes to g through the well-covering region $\mathcal{U}(g)$ of g if $e \in \mathit{input}(g)$ and through $\mathcal{U}(e)$ otherwise; whenever the procedure is invoked on e, we say that e is *processed*. The wavefronts $W(e)$ at e are constructed by merging the wavefronts $W(f, e)$ for all edges $f \in \mathit{input}(e)$; this process is called the *wavefront merging procedure* (see Lemma 4.6 in [13]).

The transparent edges of $\mathcal{S}'$ are processed in a rough time order. The wavefronts $W(e)$ of each edge e are constructed at the time $\tilde{d}(s, e) + |e|$, where $\tilde{d}(s, e)$ is the minimum geodesic distance from s to the two endpoints of e. At the time $\tilde{d}(s, e) + |e|$, it is guaranteed that all edges $f \in \mathit{input}(e)$ whose wavefronts $W(f)$ contribute a wavelet to $W(e)$ have already been processed (see Lemma 4.2 [13]); this is due to the property 5(b) of $\mathcal{S}'$ that $d(e, f) \geq 2 \cdot \max\{|e|, |f|\}$ (because f is on $\partial\mathcal{U}(e)$).

Define $\mathit{covertime}(e) = \tilde{d}(s, e) + |e|$. The value $\mathit{covertime}(e)$ will be computed during the algorithm. Initially, for each edge e whose well-covering region contains s, $\mathit{covertime}(e)$ is computed directly. The algorithm maintains a timer τ and processes the transparent edges e of $\mathcal{S}'$ following the order of $\mathit{covertime}(e)$. The main loop of the algorithm works as follows. Recall that "processing an edge e" means invoking the wavefront propagation procedure on $W(e)$. As long as $\mathcal{S}'$ has an unprocessed edge, we do the following. First, among all unprocessed edges, choose the one e with minimum $\mathit{covertime}(e)$ and set $\tau = \mathit{covertime}(e)$. Second, call the wavefront merging procedure to construct the wavefronts $W(e)$ from $W(f, e)$ for all edges $f \in \mathit{input}(e)$ satisfying $\mathit{covertime}(f) < \mathit{covertime}(e)$; compute $d(s, v)$ from $W(e)$ for each endpoint v of e. Third, process e, i.e., call the wavefront propagation procedure on $W(e)$ to compute $W(e, g)$ for all edges $g \in \mathit{output}(e)$; in particular, compute the time τ_g when the wavefronts $W(e)$ first encounter an endpoint of g and set $\mathit{covertime}(g) = \min\{\mathit{covertime}(g), \tau_g + |g|\}$.

Each wavefront $W(e)$ is represented by a persistent balanced binary trees by path copying [5] so that we can operate on an old version of $W(e)$ even after it is used to construct wavefronts $W(e, g)$ for some $g \in \mathit{output}(e)$. Due to the path-copying of the persistent data structure, each operation on $W(e)$ will cost $O(\log n)$ additional space. Note that in the wavefront merging procedure, because we do not need the old versions of $W(f, e)$ after $W(e)$ is constructed, it is not necessary to use the path-copying method after each operation on $W(f, e)$ in the procedure. Hence, the total additional space in the wavefront merging procedure in the entire algorithm is $O(n)$. But $O(n \log n)$ additional space is needed in the wavefront propagation procedure because after it is used to compute $W(e, g)$ for some $g \in \mathit{output}(e)$, the old version of $W(e)$ needs to be kept for computing $W(e, g')$ for other $g' \in \mathit{output}(e)$. There are $O(n)$ *bisector events* in the wavefront propagation procedure in the entire algorithm. In particular, if two generators u and v are not adjacent in $W(e)$ but become adjacent during the propagation from e to an edge $g \in \mathit{output}(e)$, then there is a bisector event involving u and v. The intersection between a bisector and an obstacle edge will also cause a bisector event. There are a total of $O(n)$ operations on persistent trees during the wavefront propagation procedure (we call them *wavefront propagation operations*) and each such operation requires $O(\log n)$ additional space in the persistent trees; thus the total space needed in the wavefront propagation procedure is $O(n \log n)$.

The algorithm halts once all transparent edges of $\mathcal{S}'$ are processed. Then, geodesic distances from s to all obstacle vertices are computed. This is actually the first main step of the HS algorithm, which we refer to as the *wavefront expansion step* (we could call it the wavefront propagation step, but to distinguish it from the wavefront propagation procedure, we use "expansion" instead). The second main step of the HS algorithm is to construct the shortest path map $\mathit{SPM}(s)$, which we call the *SPM-construction* step.

During the above wavefront expansion step, some generators are marked so that if a generator v is involved in a true bisector event of $\mathit{SPM}(s)$ in a cell c of $\mathcal{S}'$, then v is guaranteed to be marked for c. Also, after the wavefront expansion step, thanks to the persistent binary trees, the wavefronts $W(e)$ for all transparent edges e of $\mathcal{S}'$ are still available. To construct $\mathit{SPM}(s)$, its vertices in each cell c of $\mathcal{S}'$ are computed. To this end, the marked generators for c and the wavefronts for the transparent edges on the boundary of c are utilized, as follows.

First, c is partitioned into *active* and *inactive* regions so that no vertices of $\mathit{SPM}(s)$ are in inactive regions. Because only marked generators possibly contribute to true bisector events in c, the portion of a bisector defined by a marked generator and an unmarked generator in c belongs to an edge of $\mathit{SPM}(s)$. All such bisectors, which must be disjoint in c, partition c into regions each of which is claimed either by only marked generators or by only unmarked generators; the former regions are active while the latter are inactive.

Consider an active region R, whose boundary consists of $O(1)$ segments, each of which is a transparent edge fragment, an obstacle edge fragment, or a bisector portion in $\mathit{SPM}(s)$. For each transparent edge fragment e of ∂R, we use $W(e)$ to partition R into sub-regions, each with a unique predecessor in $W(e)$. This can be done in $O(|W(e)| \log |W(e)|)$ time by propagating the wavefront $W(e)$ into R using the wavefront propagation procedure (and ignoring the wavefronts of other transparent edges of ∂R). Let $S(e)$ denote the partition of R.

Remark. Since the wavefront $W(e)$ is not useful anymore after the above step, here in the wavefront propagation algorithm for computing $S(e)$ we do not need to use the path-copying method for each operation on the tree representing $W(e)$. Therefore, constructing the partition $S(e)$ takes only linear additional space. Consequently, the total additional space for constructing *SPM*(s) is $O(n)$. Although this does not matter for the HS algorithm, it helps in our new algorithm.

The partitions $S(e)$ for all transparent edges $e \in \partial R$ are merged in a similar way as the merge step in the standard divide-and-conquer Voronoi diagram algorithm [**?**]; during the merge process, the vertices of *SPM*(s) in R are computed (see Lemma 4.12 in [13]).

Applying the above for all active regions of c computes all vertices of *SPM*(s) in c.

After the vertices of *SPM*(s) in all cells $c \in \mathcal{S}'$ are computed, which in total takes $O(n \log n)$ time and $O(n)$ space as remarked above, *SPM*(s) can be constructed in $O(n \log n)$ additional time and $O(n)$ space by first computing all edges of *SPM*(s) and then assembling the edges using a standard plane sweep algorithm (see Lemma 4.13 in [13]).

In summary, the SPM-construction step of the HS algorithm runs in $O(n \log n)$ time and $O(n)$ space.

4 Our new algorithm

In this section, we present our modified algorithm, which uses $O(n \log n)$ time and $O(n)$ space. It turns out that we have to modify both main steps of the HS algorithm. We first give an overview of our approach in Section 4.1. Then we describe our modification to the wavefront expansion step in Section 4.2 while the proof of a key lemma is given in Section 4.3. Section 4.4 discusses the SPM-construction step.

4.1 Overview

Our modification to the wavefront expansion step is mainly on the wavefront propagation procedure. As there are a total of $O(n)$ wavefront propagation operations, we divide the algorithm into $O(\log n)$ phases such that each phase has no more than $n/\log n$ operations. We still use persistent binary trees to represent wavefronts $W(e)$. To reduce the space, at the end of each phase, we "reset" the space of the algorithm by storing a "snapshot" of the algorithm and discarding the rest of the used space. The snapshot consists of wavefronts $W(e)$ of a set of transparent edges e of $\mathcal{S}'$. We show that these wavefronts are sufficient for the subsequent algorithm to proceed as usual. We prove that the total space of the snapshot is $O(n)$; this is a main challenge of our approach and Section 4.3 is devoted to this. Since each phase has $n/\log n$ wavefront propagation operations, the total additional space in each phase is $O(n)$. Due to the space reset, the total space used in the algorithm is $O(n)$.

For the SPM-construction step, it considers each cell of $\mathcal{S}'$ individually. For each cell c, the algorithm has two sub-steps. First, compute the active regions of c. Second, for each active region R, compute the vertices of *SPM*(s) in R. For both sub-steps, the original HS algorithm utilizes the wavefronts of the transparent edges on the boundary of c. Due to the space reset, the wavefronts are not available anymore in our new algorithm. We use the following strategy to resolve the issue. First, to compute the active regions in c, we need to know the bisectors defined by an unmarked generator u and a marked generator v. We observe that u is a generator adjacent to v in the wavefronts along the boundary of c. Based on this observation, in our new algorithm we will mark the neighbors of the generators originally marked in the HS algorithm and we call them *newly-marked* generators (the generators marked in the original HS algorithm are called *originally-marked* generators). We show that the newly-marked generators and the originally-marked generators are sufficient for computing all active regions of each cell c, and the total number of all marked generators is still $O(n)$. Second, to compute the vertices of *SPM*(s) in each active region R of c, we need to restore the wavefronts of the transparent edges on the boundary of R. To this end, we observe that these wavefronts are exactly determined by the originally-marked generators. Consequently, the same algorithm as before can be applied to construct *SPM*(s) in $O(n \log n)$ time, but the space is only $O(n)$.

4.2 The wavefront expansion step

In the wavefront expansion step, our main modification is on the wavefront propagation procedure, which is to compute the wavefronts $W(e, g)$ for all edges $g \in$ *output*(e) using the wavefront $W(e)$.

We now maintain a counter *count* in the wavefront expansion step to record the number of wavefront propagation operations that have been executed so far since the last space reset; *count* $= 0$ initially. Consider a wavefront propagation procedure on a wavefront $W(e)$ of a transparent edge e. The algorithm will compute $W(e, g)$ for each edge $g \in$ *output*(e), by propagating $W(e)$ through the well-covering region $\mathcal{U}(g)$ of g if $e \in$ *input*(g) and through $\mathcal{U}(e)$ otherwise. We apply the same algorithm as before. For each wavefront propagation operation, we first do the same as before. Then, we increment *count* by one. If *count* $< n/\log n$, we proceed as before (i.e., process the next wavefront operation). Otherwise, we have reached the end of the current phase and start a new phase. To do so, we first reset *count* $= 0$ and then reset the space by constructing and storing a snapshot of the algorithm, as follows.

1. Let g refer to the edge of $output(e)$ whose $W(e,g)$ is currently being computed in the algorithm. We store the tree that is currently being used to compute $W(e,g)$ right after the above wavefront propagation operation. To do so, we can make a new tree by copying the newest version of the current persistent tree the algorithm is operating on, and thus the size of the tree is $O(n)$. We will use this tree to "resume" computing $W(e,g)$ in the subsequent algorithm.

2. For each $g' \in output(e) \setminus \{g\}$ whose $W(e,g')$ has been computed, we store the tree for $W(e,g')$. We will use the tree to compute the wavefronts $W(g')$ of g' in the subsequent algorithm.

3. We store the tree for the wavefront $W(e)$. Note that the tree may have many versions due to performing the wavefront propagation operations and we only keep its original version for $W(e)$. Hence, the size of the tree is $O(|W(e)|)$. This tree will be used in the subsequent algorithm to compute $W(e,g')$ for those edges $g' \in output(e) \setminus \{g\}$ whose $W(e,g')$ have not been computed.

4. We check every transparent edge e' of $\mathcal{S}'$ with $e' \neq e$. If e' has been processed (i.e., the wavefront propagation procedure has been called on $W(e')$) and there is an edge $g' \in output(e')$ that has *not* been processed, we know that $W(e',g')$ has been computed and is available; we store the tree for $W(e',g')$. We will use the tree to compute the wavefronts $W(g')$ of g' in the subsequent algorithm.

We refer to the above four steps as the *snapshot construction algorithm*. We refer to the wavefronts stored in the algorithm as the *snapshot*; intuitively, the snapshot contains all wavelets in the forefront of the current global wavefront.

The following lemma shows that if we discard all persistent trees currently used in the algorithm and instead store the snapshot, then the algorithm will proceed without any issues.

Lemma 1 *The snapshot stores sufficient information for the subsequent algorithm to proceed as usual.*

Proof: Let ξ denote the moment of the algorithm when the snapshot construction algorithm starts. Let ξ' be any moment of the algorithm after the snapshot is constructed. In the following, we show that if the algorithm needs any information that was computed before ξ in order to perform certain operation at ξ', then that information is guaranteed to be stored at the snapshot. This will prove the lemma.

At ξ, the transparent edges of $\mathcal{S}'$ excluding e can be classified into two categories: those that have already been processed at ξ and those that have not been processed. Let E_1 and E_2 denote the sets of the edges in these two categories, respectively. The edge e is special in the sense that it is currently being processed at ξ.

At ξ', the algorithm may be either in the wavefront merging procedure or in the wavefront propagation procedure. We discuss the two cases separately.

The wavefront merging procedure case. Suppose the algorithm is in the wavefront merging procedure at ξ', which is to compute $W(e')$ for some edge e' by merging all wavefronts $W(f',e')$ for $f' \in input(e')$. Because at ξ' we need some information that was computed before ξ, that information must be the wavefront $W(f',e')$ for some edge $f' \in input(e')$. Depending on whether $f' = e$, there are two subcases.

- If $f' = e$, then since $W(e,e')$ was computed before ξ, $W(e,e')$ is stored in the snapshot by step (2) of the snapshot construction algorithm.
- If $f' \neq e$, then since $W(f',e')$ was computed before ξ, the edge f' must have been processed at ξ. Because the algorithm is computing $W(e')$ at ξ', e' has not been processed at ξ. Therefore, $W(f',e')$ is stored in the snapshot by step (4) of the snapshot construction algorithm.

Hence, in either subcase, the information needed by the algorithm at ξ' is stored at the snapshot.

The wavefront propagation procedure case. Suppose the algorithm is in the wavefront propagation procedure at ξ', which is to process a transparent edge e', i.e., compute $W(e',g')$ for some $g' \in output(e')$. Because at ξ' the algorithm needs some information that was computed before ξ to compute $W(e',g')$, $W(e')$ must have been computed before ξ (since $W(e',g')$ relies on $W(e')$).

We claim that e' must be e. Indeed, if $e' \in E_1$, then e' has been processed before ξ and thus the wavefront propagation procedure cannot happen to e' at ξ', a contradiction. If $e' \in E_2$, then e' has not been processed at ξ. According to our algorithm, for any transparent edge e'', $W(e'')$ is computed during the wavefront merging procedure for e'', which is immediately followed by the wavefront propagation procedure to process e''. Since at ξ the algorithm is in the wavefront propagation procedure to process e, the wavefront merging procedure for e' must have not been invoked at ξ, and thus $W(e')$ must have not been computed at

ξ. This contradicts with the fact that $W(e')$ has been computed at ξ. Therefore, e' must be e.

Depending on whether g' is g, there are two subcases.

- If $g' = g$, then the tree at the moment ξ during the propagation for computing $W(e,g)$ from $W(e)$ is stored in the snapshot by the step (1) of the snapshot construction algorithm, and thus we can use the tree to "resume" computing $W(e,g)$ at ξ'.
- If $g' \neq g$, then in order to compute $W(e,g')$ at ξ', we need the wavefront $W(e)$, which is stored in the snapshot by step (3) of the snapshot construction algorithm.

Hence, in either subcase, the information needed by the algorithm at ξ' is stored at the snapshot.

The lemma thus follows. $\square$

A challenging problem is to bound the space of the snapshot, which is established in the following lemma. As the proof is lengthy and technical, we devote the next subsection to it.

Lemma 2 *The total space of the snapshot is $O(n)$.*

Since each phase has no more than $n/\log n$ wavefront propagation operations, the total extra space introduced by the persistent trees in each phase is $O(n)$. Due to the space-reset and Lemma 2, the total space of the algorithm is $O(n)$.

For the running time of our new algorithm, comparing with the original HS algorithm we spend extra time on constructing the snapshot at the end of each phase. In light of Lemma 2, each call of the snapshot construction algorithm takes $O(n)$ time. As there are $O(\log n)$ phases, the total time on constructing the snapshots in the entire algorithm is $O(n \log n)$. Hence, the running time of our new algorithm is still bounded by $O(n \log n)$.

4.3 Proof of Lemma 2

In this subsection, we prove Lemma 2, i.e., prove that the space introduced by the four steps of the snapshot construction algorithm is $O(n)$.

For the first step, as each wavefront has $O(n)$ generators, the size of one tree is $O(n)$. For the second step, since $|output(e)| = O(1)$ and the size of each $W(e,g')$ is $O(n)$, the total space is $O(n)$. For the third step, since $|W(e)| = O(n)$, the space is $O(n)$. Below we focus on the fourth step.

Let Π denote the collection of pairs (e,g) whose wavefront $W(e,g)$ is stored in the fourth step of the snapshot construction algorithm. Our goal is to show that $\sum_{(e,g)\in\Pi} |W(e,g)| = O(n)$.

For an edge e, Π may have multiple pairs with e as the first element and the second elements of all these pairs are in *output*(e); among all those pairs, we only keep the pair (e,g) such that $|W(e,g)|$ is the largest in Π and remove all other pairs from Π. Since $|output(e)| = O(1)$, it suffices to show that the total sum of $|W(e,g)|$ for all pairs (e,g) in the new Π is $O(n)$. Now in the new Π, no two pairs have the same first element. However, for an edge g, it is possible that there are multiple pairs in Π whose second elements are all g and their first elements are all from *input*(g); among all those pairs, we only keep the pair (e,g) such that $|W(e,g)|$ is the largest in Π and remove all other pairs from Π. Since $|input(g)| = O(1)$, it suffices to show that the total sum of $|W(e,g)|$ for all pairs (e,g) in the new Π is $O(n)$. Now in the new Π, no two pairs have the same first element and no two pairs have the same second element. Hence, the size of Π is $O(n)$.

For each pair $(e,g) \in \Pi$, recall that $W(e,g)$ may have up to two artificial generators (i.e., the endpoints of e). Since $|\Pi| = O(n)$, there are a total of $O(n)$ artificial generators in $W(e,g)$ for all pairs $(e,g) \in \Pi$. In the following discussion we ignore these artificial generators and by slightly abusing the notation we use $W(e,g)$ to refer to the remaining wavefront without the artificial generators. Hence, each generator of $W(e,g)$ is an obstacle vertex. It suffices to prove $\sum_{(e,g)\in\Pi} |W(e,g)| = O(n)$.

For any three adjacent generators u, v, and w that are obstacle vertices in a wavefront, we call (u,v,w) an *adjacent-generator-triple.* Two adjacent-generator-triples (u_1,v_1,w_1) and (u_2,v_2,w_2) are *distinct* if $u_1 \neq u_2$, or $v_1 \neq v_2$, or $w_1 \neq w_2$. We have the following observation.

Observation 1 *The total number of distinct adjacent-generator-triples in all wavefronts involved in the entire wavefront expansion step of the HS algorithm is $O(n)$.*

Proof: Initially, the algorithm starts a wavefront from s with only one generator s. Whenever a distinct adjacent-generator-triple is produced during the algorithm, a bisector event must happen. As there are $O(n)$ bisector events [13], the observation follows. $\square$

Lemma 2 is almost an immediate consequence of the following lemma.

Lemma 3 *Any adjacent-generator-triple (u,v,w) can appear in the wavefront $W(e,g)$ for at most $O(1)$ pairs (e,g) of Π.*

Before proving Lemma 3, we prove Lemma 2 with the help of Lemma 3. Indeed, there are $O(n)$ distinct adjacent-generator-triples in the entire algorithm by Observation 1. Now that each such triple can only appear in a constant number of wavefronts $W(e,g)$ for all $(e,g) \in \Pi$, the total number of generators in all wavefronts $W(e,g)$ for all $(e,g) \in \Pi$ is bounded by $O(n)$. This leads to Lemma 2.

Proving Lemma 3. We prove Lemma 3 in the rest of this subsection. Assume to the contrary that an adjacent-generator-triple (u, v, w) appears in the wavefront $W(e, g)$ for more than $O(1)$ pairs (e, g) of Π [3]; let Π' denote the set of all such pairs.

Consider a pair $(e, g) \in \Pi'$. Recall that $W(e, g)$ is obtained by propagating $W(e)$ from e to g through the well-covering region $\mathcal{U}(g)$ if $e \in input(g)$ and through $\mathcal{U}(e)$ otherwise. If one of u, v, and w is in $\mathcal{U}(g) \cup \mathcal{U}(e)$, then we call (e, g) a *special pair*. The properties of the subdivision $\mathcal{S}'$ guarantee that each obstacle vertex is in $\mathcal{U}(f)$ for at most $O(1)$ edges f of $\mathcal{S}'$ [13], and thus Π' has at most $O(1)$ special pairs. We remove all special pairs from Π', after which Π' still has more than $O(1)$ pairs and for each pair $(e, g) \in \Pi'$ the three generators u, v, and w are all outside $\mathcal{U}(g) \cup \mathcal{U}(e)$. Therefore, u, v, and w are also generators in $W(e)$ in this order (although they may not be adjacent in $W(e)$). Because (u, v, w) is an adjacent-generator-triple in $W(e, g)$, $B(u, v)$ (resp., $B(v, w)$) intersects g. Also, note that u, v, and w are on the same side of the supporting line of e; since they are generators of both $W(e)$ and $W(e, g)$, the bisector $B(u, v)$ (resp., $B(v, w)$) intersects e as well (although the intersection may not appear in $W(e)$, i.e., the intersection may be covered by a different generator in $W(e)$). Further, v is closer to the intersection of $B(u, v)$ (resp., $B(v, w)$) with e that with g. Let (e_1, g_1) be the pair in Π' such that the intersection of $B(u, v)$ with its first element is closest to v among the intersections of $B(u, v)$ with the first elements of all pairs of Π'.

By the property 5(a) of $\mathcal{S}'$, the size of $\mathcal{U}(g_1) \cup \mathcal{U}(e_1)$ is $O(1)$. Because $|\Pi'|$ is not $O(1)$, Π' must have a pair, denoted by (e_2, g_2), such that e_2 is outside $\mathcal{U}(g_1) \cup \mathcal{U}(e_1)$. Recall that $W(e_1, g_1)$ is obtained by propagating $W(e_1)$ from e_1 to g_1 inside $\mathcal{U}(g_1)$ or $\mathcal{U}(e_1)$. Hence, if we move along the bisector $B(u, v)$ from its intersection with e_1 to its intersection with g_1, all encountered edges of $\mathcal{S}'$ are in $\mathcal{U}(g_1) \cup \mathcal{U}(e_1)$. Therefore, by the definitions of e_1 and e_2, $B(u, v)$ intersects e_1, g_1, e_2, and g_2 in this order following their distances from v (e.g., see Fig. 2).

By definition (i.e., the fourth step of our snapshot construction algorithm), e_2 has been processed but g_1 has not. According to the wavefront expansion algorithm, $covertime(e_2) \leq covertime(g_1)$. In the following we will obtain $covertime(e_2) > covertime(g_1)$, which leads to a contradiction.

Let b be the intersection between the bisector $B(u, v)$ and g_2 (e.g., see Fig. 2). Since u and v are two adjacent generators in $W(e_2, g_2)$, $\overline{vb}$ is in the free space $\mathcal{F}$. As $W(e_2, g_2)$ is obtained from $W(e_2)$ and v is also a generator in $W(e_2)$, $\overline{vb}$ must intersect e_2, say, at a point q (e.g., see Fig. 2). Further, we have the following observation.

Observation 2 *$\overline{vb}$ intersects g_1 (e.g., see Fig. 2).*

Proof: Since (u, v, w) is an adjacent-generator-triple in both wavefronts $W(e_1, g_1)$ and $W(e_2, g_2)$, each of the bisectors $B(u, v)$ and $B(v, w)$ intersects both g_1 and g_2. Let z_1 and z_2 be the intersections of g_1 with $B(u, v)$ and $B(v, w)$, respectively. Let b' be the intersection of g_2 and $B(v, w)$. Then, $\overline{z_1 z_2}$, which is a subsegment of g_1, is *claimed* by v in $W(e_1, g_1)$ (i.e., among all wavelets of $W(e_1, g_1)$, the wavelet at v reaches p the earliest, for all points $p \in \overline{z_1 z_2}$). Similarly, $\overline{bb'}$, which is a subsegment of g_2, is claimed by v in $W(e_2, g_2)$. Recall that v is closer to z_1 than to b. Hence, for any point $p \in \overline{bb'}$, $\overline{pv}$ must intersect $\overline{z_1 z_2}$. This leads to the observation, for $\overline{z_1 z_2} \subseteq g_2$ and $b \in \overline{bb'}$. $\square$

In light of the above observation, let z be an intersection point of $\overline{vb}$ and g_1 (e.g., see Fig. 2). Using the properties of the well-covering regions of $\mathcal{S}'$, we have the following observation.

Observation 3 *$|\overline{qz}| \geq 2 \cdot |g_1|$.*

Proof: Since $q \in e_2$ is outside the well-covering region $\mathcal{U}(g_1)$ of g_1, $z \in g_1$, and $\overline{qz}$ is in the free space $\mathcal{F}$, $\overline{qz}$ must cross the boundary of $\mathcal{U}(g_1)$, say, at a point p. By the property 5(b) of the conforming subdivision $\mathcal{S}'$, $|\overline{pz}| \geq 2 \cdot |g_1|$. As $|\overline{qz}| \geq |\overline{pz}|$, the observation follows. $\square$

The following lemma, which is a consequence of Observation 3, leads to a contradiction, for $covertime(g_1) \geq covertime(e_2)$, and thus Lemma 3 is proved.

Lemma 4 *$covertime(g_1) < covertime(e_2)$.*

Proof: Let x_1 and y_1 be the two endpoints of g_1, respectively. Let x_2 and y_2 be the two endpoints of e_2, respectively. Refer to Fig. 3. As $\overline{zv} \subseteq \overline{qv}$ is in the free space $\mathcal{F}$, we have the following for $covertime(g_1)$:

$$\begin{aligned} covertime(g_1) &= \min\{d(s, x_1), d(s, y_1)\} + |g_1| \\ &\leq d(s, v) + |\overline{vz}| + |g_1|/2 + |g_1| \\ &= d(s, v) + |\overline{vz}| + 3|g_1|/2. \end{aligned}$$

On the other hand, because v claims b in $W(e_2, g_2)$ (as explained in the proof of Observation 2), v is also a generator in $W(e_2)$, and $\overline{bv}$ intersects e_2 at q, v must claim q in $W(e_2)$. Hence, $\min\{d(s, x_2), d(s, y_2)\} + |e_2| \geq d(s, v) + |\overline{vq}|$ must hold, since otherwise the artificial wavelets at the endpoints of e_2 would have prevented v from claiming q [13]. Therefore, we have the

[3]By "more than $O(1)$ pairs", we intend to say "more than c pairs for a constant c to be fixed later". To simplify the discussion, we use "more than $O(1)$ pairs" instead with the understanding that such a constant c can be fixed.

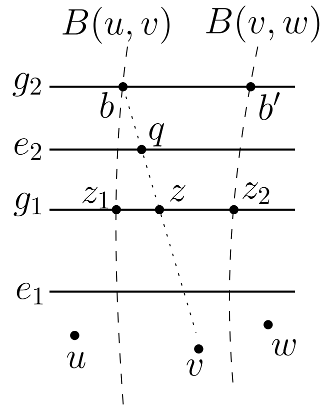

Figure 2: Illustrating the two bisectors $B(u,v)$ and $B(v,w)$ as well as the four edges e_1, g_1, e_2, and g_2 (each of these edges may also be vertical).

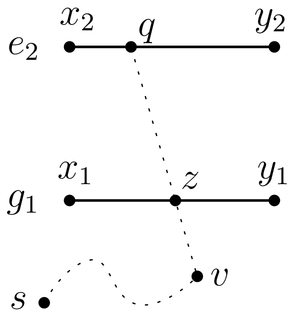

Figure 3: Illustrating the proof of Lemma 4.

following for *covertime*(e_2):

$$\begin{aligned} \mathit{covertime}(e_2) &= \min\{d(s,x_2), d(s,y_2)\} + |e_2| \\ &\geq d(s,v) + |\overline{vq}| \\ &= d(s,v) + |\overline{vz}| + |\overline{zq}| \\ &\geq d(s,v) + |\overline{vz}| + 2\cdot|g_1|. \end{aligned}$$

The last inequality is due to Observation 3. As $|g_1| > 0$, we obtain *covertime*$(g_1) <$ *covertime*(e_2). □

4.4 The SPM-construction step

The above shows that the wavefront expansion step of the HS algorithm can be implemented in $O(n \log n)$ time and $O(n)$ space. As discussed in Section 3, the algorithm computes geodesic distances from s to the endpoints of all transparent edges of $\mathcal{S}'$. As each obstacle vertex is incident to a transparent edge of $\mathcal{S}'$, geodesic distances from s to all obstacle vertices are known. In this subsection, we discuss our modification to the SPM-construction step of the HS algorithm so that the shortest path map *SPM*(s) can be constructed in $O(n \log n)$ time and $O(n)$ space. As discussed before, the key is to compute the vertices of *SPM*(s) in each cell c of $\mathcal{S}'$.

Consider a cell c of $\mathcal{S}'$. To compute the vertices of *SPM*(s) in c, there are two major steps. First, partition c into active and inactive regions. Second, compute the vertices of *SPM*(s) in each active region. The original HS algorithm achieves this using the wavefronts $W(e)$ of the transparent edges e on the boundary of c as well as the marked generators for c, which are marked during the wavefront expansion step. As our new algorithm does not maintain the wavefronts $W(e)$ anymore due to the space-reset, we cannot use the same algorithm as before. We instead propose the following approach.

The first step. The first step is to compute the active regions of c. Recall that c is partitioned into active and inactive regions by bisectors defined by an unmarked generator u and a marked generator v. Consider such a bisector $B(u,v)$. Observe that u and v must be from the same wavefront $W(e)$ of an edge e of ∂c. Indeed, assume to the contrary that u is from $W(e_u)$ and v is from $W(e_v)$ of two different edges e_u and e_v of ∂c. Then, because u and v are adjacent generators along ∂c, the wavelet of u in $W(e_u)$ must claim an endpoint of e_u and the wavelet of v in $W(e_v)$ must claim an endpoint of e_v. Hence, both u and v are marked by Rule 2(a) of the generator marking rules [13]. But this incurs a contradiction as u is unmarked. Therefore, u and v must be from the same wavefront $W(e)$ and they are actually neighboring in $W(e)$.

Based on the above observation, in addition to the marking rules of the original HS algorithm (see Section 4.1.2 of [13]), we add the following new rule: If a generator v in a wavefront $W(e)$ is marked for a cell by the original rules during the wavefront expansion step of the HS algorithm, we also mark both neighboring generators of v in $W(e)$. We call the generators marked by the new rule *newly-marked* generators, while the generators marked by the original rules are called *originally-marked* generators. If a generator is both originally-marked and newly-marked, we consider it originally-marked. Since the total number of originally-marked generators is $O(n)$ [13], the total number of newly-marked generators is $O(n)$. Also the new mark rules can be easily incorporated into the wavefront expansion step without affecting its running time asymptotically.

Now consider the bisector $B(u,v)$ discussed in the above observation. The generator u must be a newly-marked generator. This suggests the following method to compute the active regions of c. We collect all marked generators (including both newly-marked and originally-marked) for the transparent edges of ∂c and order them around the boundary of c. For each pair of adjacent generators u and v in the wavefront $W(e)$ of a transparent edge e such that u is newly-marked and v is originally marked, we compute their bisector in c. These bisectors are disjoint and partition c into regions, and the regions claimed by marked generators only are active regions [13]. The time for computing all active regions is bounded by $O(m \log m)$, where m is the number of all marked generators for c. Therefore, the total time of this step for all cells of $\mathcal{S}'$ is bounded by $O(n \log n)$ since the total number of marked generators is $O(n)$.

The second step. The second step is to compute the vertices of $\mathit{SPM}(s)$ in each active region R of c. For each transparent edge e of ∂R, which is on ∂c, we need to restore its wavefront $W(e)$, because it is not maintained by our new algorithm. To this end, the algorithm in the above first step determines a list $L(e)$ of originally-marked generators that claim e. It turns out that $L(e)$ is exactly $W(e)$, as shown below.

Observation 4 $L(e) = W(e)$.

Proof: Since each generator of $L(e)$ is marked for e and c, $L(e)$ is a subset of $W(e)$. On the other hand, because R is an active region, R is claimed by originally-marked generators only [13]. Since each generator of $W(e)$ claims at least one point of e and $e \subseteq R$, all generators are originally-marked. Therefore, all generators of $W(e)$ are in $L(e)$. The observation thus follows. $\square$

By the above observation, the wavefronts $W(e)$ for all transparent edges of all active regions of c can be restored once all active regions of c are computed in the first step. Subsequently, using the same method as the original HS algorithm, the vertices of $\mathit{SPM}(s)$ in c can be computed in $O(m \log m)$ time, where m is the total number of marked generators of c. As remarked in Section 3, the space is $O(m)$. As the total number of marked generators for all cells of $\mathcal{S}'$ is $O(n)$, the overall time for computing all vertices of $\mathit{SPM}(s)$ is $O(n \log n)$ and the space is $O(n)$. Finally, the edges of $\mathit{SPM}(s)$ can be computed separately and then $\mathit{SPM}(s)$ can be assembled by a plane sweep algorithm (see Lemma 4.13 [13] for details); this step takes $O(n \log n)$ time and $O(n)$ space.

The following theorem summarizes our main result.

Theorem 1 *Given a source point s and a set of pairwise disjoint polygonal obstacles of n vertices in the plane, the shortest path map of s can be constructed in $O(n \log n)$ time and $O(n)$ space.*

By building a point location data structure on $\mathit{SPM}(s)$ in $O(n)$ time [6, 16], given any query point t, the geodesic distance $d(s,t)$ can be computed in $O(\log n)$ time and a shortest path $\pi(s,t)$ can be produced in additional time linear in the number of edges of the path.

5 Concluding remarks

In this paper, by modifying the HS algorithm [13] we solve the Euclidean shortest path problem among polygonal obstacles in the plane in $O(n \log n)$ time and $O(n)$ space, reducing the space complexity of the HS algorithm by a logarithmic factor and settling the longstanding open question of Hershberger and Suri [13]. The new algorithm is now optimal in both time and space. Our main technique is to divide the HS algorithm into phases and perform a space-reset procedure after each phase by constructing a linear-space snapshot that maintains sufficient information for the subsequent algorithm.

Like the original HS algorithm, our new algorithm can also handle some extensions of the problem. For example, if the source is not a point but a more complex object like a line segment or a disk, then as discussed in [13], except for initialization and propagating the initial wavelets, the rest of the algorithm does not change. As such, our new algorithm can solve this case in $O(n \log n)$ time and $O(n)$ space. For the multiple source case where there are m source points and the goal is essentially to compute the *geodesic Voronoi diagram* of these points, i.e., partition the free space into regions so that all points in the same region have the same nearest source point, our algorithm can solve this case in $O((n+m)\log(n+m))$ time and $O(n+m)$ space. Also, as discussed in [13], the algorithm still works if each source point has an initial "delay". Note that the geodesic Voronoi diagram problem in simple polygons has been particularly considered; other than some earlier work [1, 26], the problem has recently received increased attention [18,24,25]. Oh [24] finally proposed an algorithm of $O(n + m \log m)$ time, which is optimal.

Our technique might be able to find applications elsewhere. For example, Hershberger et al. [14] considered the shortest path problem among curved obstacles in the plane. They gave an algorithm to construct a data structure for a source point s so that each shortest path query can be answered in $O(\log n)$ time. Their algorithm runs in $O(n \log n)$ time and $O(n \log n)$ space, plus $O(n)$ calls to a *bisector oracle*, which is to compute the intersection of two bisectors defined by pairs of curved obstacle boundary segments. Their data structure is not a shortest path map, but consists of all wavefronts (represented by persistent binary trees) produced during the algorithm after each event; the size of the data structure is $O(n \log n)$. If we are looking for a shortest path from s to a single point t, then it might be possible that our technique can be applied to reduce the space of their algorithm to $O(n)$ because the high-level scheme of their algorithm is the same as the wavefront expansion step of the HS algorithm for polygonal obstacles. For answering queries, however, the data structure proposed in [14] is inherently of size

$\Omega(n \log n)$ because all wavefronts need to be (implicitly) maintained. To have a data structure of size $O(n)$, one possible way is to use a shortest path map as in the polygonal obstacle case. However, this will require an additional oracle to explicitly compute the bisector of two curved obstacle boundary segments; this oracle was not used by Hershberger et al. [14] because they wanted to minimize the number of oracles their algorithm relies on.

Another possible application of our technique is on finding shortest paths on the surface of a convex polytope in three dimensions [**?**, **?**, 21–23]. The current best algorithm was given by Schreiber and Sharir [**?**] and their algorithm uses $O(n \log n)$ time and $O(n \log n)$ space. Although many details are different, Schreiber and Sharir [**?**] proposed an oct-tree-like 3-dimensional axis-parallel subdivision of the space, which is similar in spirit to the planar conforming subdivision in [14], and used it to guide their continuous Dijkstra algorithm to propagate wavefronts, in a similar manner as the HS algorithm. Schreiber and Sharir [**?**] raised an open question whether the space of their algorithm can be reduced to $O(n)$. It would be interesting to see whether our technique can be adapted to their algorithm.

References

[1] B. Aronov. On the geodesic voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4:109–140, 1989.

[2] D.Z. Chen, J. Hershberger, and H. Wang. Computing shortest paths amid convex pseudodisks. *SIAM Journal on Computing*, 42(3):1158–1184, 2013.

[3] D.Z. Chen and H. Wang. A new algorithm for computing visibility graphs of polygonal obstacles in the plane. *Journal of Computational Geometry*, 6:316–345, 2015.

[4] Y.-J. Chiang and J.S.B. Mitchell. Two-point euclidean shortest path queries in the plane. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 215–224, 1999.

[5] J. Driscoll, N. Sarnak, D. Sleator, and R.E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.

[6] H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.

[7] S.K. Ghosh and D.M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991.

[8] L.J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

[9] L.J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R.E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

[10] J. Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, 38(5):231–235, 1991.

[11] J. Hershberger, V. Polishchuk, B. Speckmann, and T. Talvitie. Geometric kth shortest paths. In *Proceedings of the 30th Annual Symposium on Computational Geometry (SoCG)*, pages 96:96–96:97, 2014.

[12] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry: Theory and Applications*, 4(2):63–97, 1994.

[13] J. Hershberger and S. Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

[14] J. Hershberger, S. Suri, and H. Yıldız. A near-optimal algorithm for shortest paths among curved obstacles in the plane. In *Proceedings of the 29th Annual Symposium on Computational Geometry (SoCG)*, pages 359–368, 2013.

[15] R. Inkulu, S. Kapoor, and S.N. Maheshwari. A near optimal algorithm for finding euclidean shortest path in polygonal domain. *arXiv:1011.6481v1*, 2010.

[16] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[17] D.T. Lee and F.P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

[18] C.-H. Liu. A nearly optimal algorithm for the geodesic voronoi diagram of points in a simple polygon. *Algorithmica*, 82:915–937, 2020.

[19] J.S.B. Mitchell. A new algorithm for shortest paths among obstacles in the plane. *Annals of Mathematics and Artificial Intelligence*, 3(1):83–105, 1991.

[20] J.S.B. Mitchell. Shortest paths among obstacles in the plane. *International Journal of Computational Geometry and Applications*, 6(3):309–332, 1996.

[21] J.S.B. Mitchell, D.M. Mount, and C.H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16:647–668, 1987.

[22] D.M. Mount. On finding shortest paths on convex polyhedra. Technical report, University of Maryland, College Park, MD 20742, 1985.

[23] D.M. Mount. Storing the subdivision of a polyhedral surface. *Discrete and Computational Geometry*, 2:153–174, 1987.

[24] E. Oh. Optimal algorithm for geodesic nearest-point voronoi diagrams in simple polygons. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–409, 2019.

[25] E. Oh and H.-K. Ahn. Voronoi diagrams for a moderate-sized point-set in a simple polygon. *Discrete and Computational Geometry*, 63:418–454, 2020.

[26] E. Papadopoulou and D. T. Lee. A new approach for nearest-neighbor searching in polygonal domains. *Journal of Algorithms*, 27(2):341–372, 1998.