Article to appear in International Journal of Computational Geometry & Applications

IMPROVING SHORTEST PATHS IN THE DELAUNAY TRIANGULATION

MANUEL ABELLANAS

Departamento de Matemática Aplicada, Facultad de Informática, Universidad Politécnica de Madrid,
28660 Boadilla del Monte, Madrid, Spain
mabellanas@fi.upm.es

MERCÈ CLAVEROL

Departament de Matemàtica Aplicada IV, Universitat Politècnica de Catalunya,
Edifici VG1, Avda. Víctor Balaguer s/n, 08800 Vilanova i la Geltrú, Barcelona, Spain
merce@ma4.upc.edu

GREGORIO HERNÁNDEZ

Departamento de Matemática Aplicada, Facultad de Informática, Universidad Politécnica de Madrid,
28660 Boadilla del Monte, Madrid, Spain
gregorio@fi.upm.es

FERRAN HURTADO

Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya,
Edifici Omega, Campus Nord, Jordi Girona 1-3, 08034 Barcelona, Spain
ferran.hurtado@upc.edu

VERA SACRISTÁN

Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya,
Edifici Omega, Campus Nord, Jordi Girona 1-3, 08034 Barcelona, Spain
vera.sacristan@upc.edu

MARIA SAUMELL

Département d'Informatique, Université Libre de Bruxelles,
Boulevard du Triomphe - CP 212, 1050 Brussels, Belgium
maria.saumell.m@gmail.com

RODRIGO I. SILVEIRA

Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya,
Edifici Omega, Campus Nord, Jordi Girona 1-3, 08034 Barcelona, Spain
rodrigo.silveira@upc.edu

Received (received date)
Revised (revised date)
Communicated by (Name)

We study a problem about shortest paths in Delaunay triangulations. Given two nodes s, t in the Delaunay triangulation of a point set S, we look for a new point $p \notin S$ that can be added, such that the shortest path from s to t, in the Delaunay triangulation of $S \cup \{p\}$, improves as much as possible. We study several properties of the problem, and give efficient algorithms to find such a point when the graph-distance used is Euclidean and for the link-distance. Several other variations of the problem are also discussed.

Keywords: Shortest path; dilation; Delaunay triangulation.

1. Introduction

There are many applications involving communication networks where the underlying physical network topology is not known, too expensive to compute, or there are reasons to prefer to use a logical network instead. An example of an area where this occurs is ad-hoc networks. An ad-doc network is composed of a collection of nodes, which can communicate directly with each other when their distance is below some threshold. Often these nodes are not even static, and their positions vary with time. The routing is done locally, with each node being capable of relaying packets to other nodes. To avoid broadcasting to all neighbors every time a packet needs to be sent, some type of logical network topology and routing algorithm must be used.

Similar situations arise in application-layer routing. In applications related to IP multicast or peer-to-peer networks, sometimes a logical network is used on top of the actual, physical network (see for example Ref. 1). This logical network is assumed to have some overlay topology, whose choice can have an important impact on the overall performance of the network.

Often, the overlay topology is modeled by using the Delaunay triangulation.[2,3,1] The Delaunay triangulation of a set of points S is defined as a triangulation in which the vertices are the points in S, and the circumcircle of each triangle (the circle defined by the three vertices of each triangle) contains no other point from S. The Delaunay triangulation presents several advantages that make it appropriate for this context: it provides locality, scales well, and in general avoids high-degree vertices, which can create serious bottlenecks. In addition, it has been proved that several widely-used *localized* routing protocols guarantee to deliver the packets when the underlying network topology is the Delaunay triangulation.[4] It is also known that there exist localized routing protocols based on the Delaunay triangulation where the total distance traveled by any packet is never more than a small constant factor off the network distance between source and destination.[4] Moreover, the Delaunay triangulation is known to be a constant spanner:[5] the shortest path between any pair of nodes through the Delaunay triangulation is at most a constant times longer than their Euclidean distance (the precise value of the constant is a matter of active research, see e.g. Ref. 6 and 7). Thus in the case of geometric networks this guarantees that all packets travel at most a constant times the minimum travel time.

In this paper we consider the problem of improving a geometric network, with

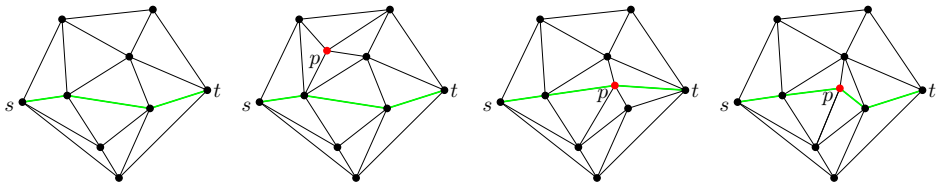

Fig. 1. Depending on where p is placed, the length of the shortest path between s and t in the resulting graph G'_p can be equal, shorter or longer than in G.

a Delaunay triangulation topology, by augmenting it with additional nodes. In particular, our main goal is to improve the shortest path in the Delaunay network between two given nodes s and t. Adding new nodes to a Delaunay network produces changes in the network topology that can result in equal, shorter, or longer shortest paths between s and t (see Figure 1), but we are only interested in the situation where the new shortest paths are shorter.

It turns out that effects of adding points to a Delaunay triangulation occur at a global level: it is sometimes the case that the new nodes inserted produce a new shortest path which uses many vertices that were not present in the former shortest path. This seems to make the problem quite complex, for which reason, as a first step, most of this paper is devoted to the more restricted scenario where at most one node can be added to the network.

We are not aware of previous work that attempts to add points to improve shortest paths in Delaunay triangulations. However, since the problem can be seen as one to improve the connections in a network, there is a relation with problems on graph augmentation for higher connectivity, which recently have received quite some attention (e.g. Ref. 8-10). Similarly, our problem is also related to the problem of adding edges to a network in order to improve its *dilation* (i.e. the maximum over all pairs of different vertices of the ratio between their network-distance and their geometric distance),[11,12] although here we are mostly concerned with improving the network distance between only one pair of points.

Notation. The input to the problem is a set of n points S, and two points $s, t \in S$. We will use G to denote the Delaunay graph of S. In this paper we assume that the points in S satisfy the two conditions ensuring that the Delaunay graph is a triangulation, i.e., no three points are collinear and no four points are concyclic. Moreover, we also assume that the edge st does not belong to G, otherwise the distance between s and t in G would be optimal.

The shortest path in G between two vertices $x, y \in S$ will be denoted by $\mathrm{SP}_G(x, y)$. For the sake of clarity, throughout the paper we assume that all shortest paths in G between two points from S, and also between a point from S and a new point p that is inserted, are unique. The extension of our results to the case where

there is multiplicity of shortest paths is straightforward (we make a few remarks about this at the end of Sections 2, 3, and 4).

The length of $\mathrm{SP}_G(x, y)$, defined as the sum of the Euclidean lengths of its edges, will be denoted by $|\mathrm{SP}_G(x, y)|$, although we will omit G if possible. The straight line segment between two points x and y will be denoted by $\overline{xy}$.

Finally, we will use G'_p to denote the Delaunay graph of $S \cup \{p\}$, for some $p \notin S$ (we will omit p when clear from the context).

Overview of the paper. The paper is organized as follows. We start by making a number of geometric observations about the general problem of improving a shortest path by adding one point. In Section 3 we give an algorithm that finds a point whose insertion gives the best possible improvement to the shortest path between s and t. Section 4 studies the same problem when using the link-distance as network metric. A brief discussion about the difficulty of the more general case of adding several points is presented in Section 5. Finally, we conclude the paper with some discussion about future work in Section 6.

2. Properties and Observations

We begin by analyzing some geometric properties of the problem of adding a point to improve the shortest path between s and t in the Delaunay triangulation G. Note that, in most of the paper, s and t are considered fixed vertices of G that are specified as part of the input.

When a new point p is inserted in S, the Delaunay triangulation changes: some edges might disappear and new edges, all incident to p, appear (see Figure 2). The edges of G that are affected by the insertion of p belong to Delaunay triangles whose circumcircles contain p. This is because triangle circumcircles that were empty in G, and do not contain p, are still empty in G'. If p is in the interior of the convex hull of S, the three edges from p to the vertices of the triangle in G that contains p are always Delaunay edges, thus are always in G'. In addition, all other triangles in G whose circumcircle contains p get new edges in G', connecting their vertices to p. If p is in the exterior of the convex hull of S, the situation is similar, because the vertices of the triangles in G whose circumcircle contains p also become adjacent to p. Additionally, all vertices of the convex hull of S that are visible from p get edges connecting them to p.

We are interested in adding a point p such that $\mathrm{SP}(s, t)$ improves. A first question that one may ask is whether it is always possible to improve a shortest path by adding one point to G. There are situations in which it is easy to obtain *some* improvement, as the following proposition shows.

Proposition 1. *Let e_1 and e_2 be two consecutive and non-aligned edges of $SP_G(s, t)$. Let C_1 and C_2 be two Delaunay circles through the endpoints of e_1 and e_2, respectively. If C_1 and C_2 are not tangent and $C_1 \cap C_2$ lies on the side of $SP_G(s, t)$ in which e_1 and e_2 form the smallest angle, then the length of $SP_G(s, t)$ can be*

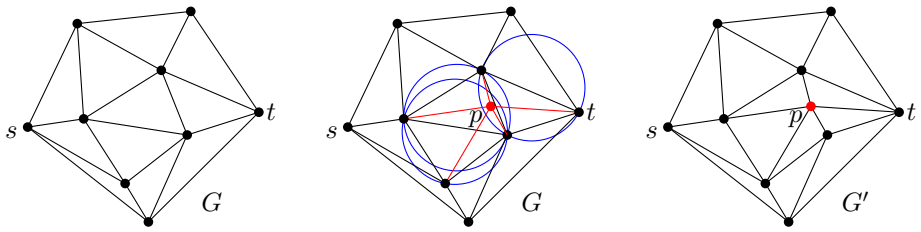

Fig. 2. Adding a new point p to the Delaunay triangulation.

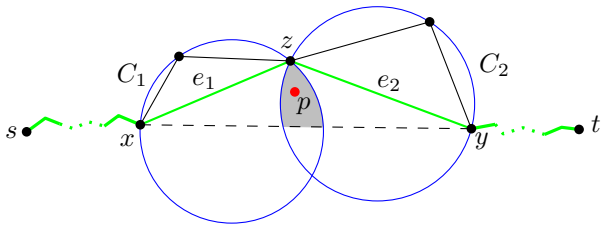

Fig. 3. Any point p inserted in the shaded region improves $\mathrm{SP}(s,t)$, shown in fat line.

reduced by inserting one point.

Proof. Insert the point p in the region delimited by $C_1 \cap C_2$ in the interior of the triangle formed by the endpoints of the edges e_1 and e_2 (see Figure 3). If $e_1 = xz$ and $e_2 = zy$, then xp and py will be Delaunay edges in G', shortening the portion of $\mathrm{SP}(s,t)$ between x and y.

Note that it could happen that the insertion of p makes some edge uv in $SP(s,x)$ (or $SP(y,t)$) disappear. However, in that case edges up and vp would be present in G'_p, creating an even shorter path between s and t through p, since p would be reachable directly from u and v. □

However, not all shortest paths have two edges as described in Proposition 1 which can be locally improved. In fact, as the next result shows, some shortest paths cannot be improved at all by adding a single point.

Proposition 2. *It is sometimes impossible to improve $SP_G(s,t)$ by inserting only one point.*

Proof. Consider the points shown in Figure 4, with s and t as in the figure. Note that edges e_1 and e_2 do not need to be part of the boundary of the convex hull of the point set—it is easy to add points around the depicted configuration without altering the essence of the example. The position of the points in S is such that C_1 and C_2 intersect on the side of $\mathrm{SP}_G(s,t)$ where the angle between e_1 and e_2 is largest. Hence Proposition 1 cannot be applied. Moreover, the angle between e_1

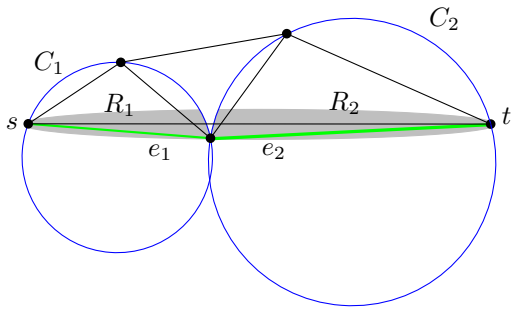

Fig. 4. Point set example where $\mathrm{SP}(s,t)$ cannot be improved by adding one single point. The insertion of a point p, no matter where it is placed, cannot reduce the length of $\mathrm{SP}_G(s,t)$.

and e_2 is close enough to π as to guarantee that the following condition holds: $\forall x \in S \setminus \{s,t\}$, $|sx| + |xt| \geq |e_1| + |e_2|$.

A shorter path between s and t resulting from the insertion of p must be of the form $\{s,p,t\}$, i.e., p must be inserted so that $|sp| + |pt| < |e_1| + |e_2|$, $sp \in G'_p$, and $pt \in G'_p$. The locus R_1 of all points p such that $|sp| + |pt| < |e_1| + |e_2|$ and $sp \in G'_p$, is a portion of C_1 not containing any point in $C_1 \cap C_2$. Analogously, the locus R_2 of all points p such that $|sp| + |pt| < |e_1| + |e_2|$ and $pt \in G'_p$, is a portion of C_2 not containing any point in $C_1 \cap C_2$. Therefore the three conditions $|sp| + |pt| < |e_1| + |e_2|$, $sp \in G'_p$, and $pt \in G'_p$ cannot be simultaneously satisfied. □

On the other hand, two points always suffice to improve the shortest path between s and t.

Proposition 3. *It is always possible to reduce the length of $SP_G(s,t)$ by inserting two points.*

Proof. The proof follows the same idea as in Proposition 1. Let $e_1 = xz$, $e_2 = zy$, C_1, and C_2 respectively be two consecutive edges of $\mathrm{SP}(s,t)$ and two Delaunay circles circumscribing them, such that Proposition 1 cannot be applied.

It is easy to verify that there always exists a circle C going through z such that (i) C intersects the interior of C_1 and C_2, (ii) C intersects the boundary of C_1 and C_2 in (z and) points lying in the interior of triangle $\triangle xzy$, and (iii) C is empty of points from S. See Figure 5 for an example. Then we can add two points p_1 and p_2 on the boundary of C, with p_1 inside C_1 and $\triangle xzy$, and p_2 inside C_2 and $\triangle xzy$. The previous conditions will ensure that $\triangle zp_1p_2$ will be a Delaunay triangle, and edges xp_1 and p_2y will be Delaunay edges. Furthermore, we place p_1 and p_2 such that they form a convex path with x and y (that is, such that $\{x,p_1,p_2,y\}$ defines a convex polygon). The convexity of the path, plus the fact that it lies inside $\triangle xzy$, guarantee that $|xp_1| + |p_1p_2| + |p_2y| < |e_1| + |e_2|$, so the path from x to y through p_1 and p_2 is shorter than the path through z.

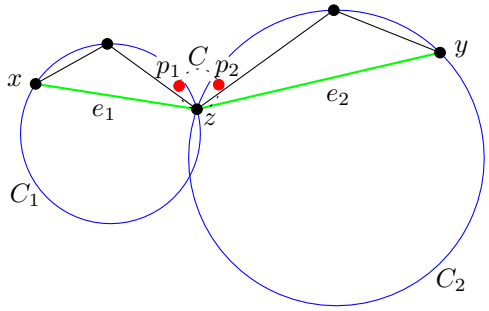

Fig. 5. The path $\{x, z, y\}$ is shortened if we insert p_1 and p_2.

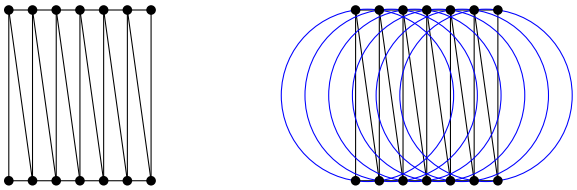

Fig. 6. A Delaunay triangulation (left) whose Delaunay circles (right) have a quadratic number of intersections. It is simple to perturbe the points such that no four of them are cocircular, no three of them are collinear, and the number of intersections is the same.

As with Proposition 1, it is worth noting that it could happen that the insertion of p_1 and p_2 makes some edge uv in $SP(s, x)$ (or $SP(y, t)$) disappear. However, in that case edges up_1 or up_2, and vp_1 or vp_2 would be present in the new triangulation, creating an even shorter path between s and t, since p_1 or p_2 would be reachable directly from u or v. □

Note that in case of having more than one shortest path $\mathrm{SP}_G(s, t)$, Propositions 1 and 3 hold for all of them.

3. Optimally Adding one Point

In this section we present an algorithm that finds a point that gives the optimal improvement. In other words, it computes a point p such that $\mathrm{SP}_{G'_p}(s, t)$ is minimum among all possible locations of p in the plane. In principle, p can be any point in the plane. However, the algorithm makes use of several geometric observations that allow to reduce the set of possible candidates to $O(I)$ points, where I is the number of pairs of Delaunay circles that intersect. Note that the maximum complexity of I is $\Theta(n^2)$ (an example with a quadratic number of intersections is shown in Figure 6). However, we expect that in practice the number of intersections is closer to $O(n)$.

3.1. *Preliminaries*

We begin with a simple but important observation, which follows from the fact that all new edges in G'_p (i.e., edges that were not present in G) are incident to p.

Remark 1. If the insertion of p improves $\mathrm{SP}_G(s,t)$, then $\mathrm{SP}_{G'_p}(s,t)$ goes through p.

The correctness of our algorithm is based on two technical lemmas. The first one allows us to narrow down the set of points that need to be considered as candidate points.

Lemma 1. *Let p be an optimal point, and let x, y be the points in G such that $SP_{G'_p}(s,t)$ includes xp and py as consecutive edges. Then p lies on the segment $\overline{xy}$, or is one of the two intersection points of two circumcircles of triangles in G. Furthermore, one of these circles goes through x, and the other one through y.*

Proof. If p is not on $\overline{xy}$, it must be because moving p from the optimal position closer to $\overline{xy}$ produces a change in the Delaunay triangulation that results in a worse shortest path. We show below that such topological change in the triangulation can only occur if p lies exactly on the intersection of two Delaunay circles.

Assume without loss of generality that $\overline{xy}$ is horizontal, and p is above $\overline{xy}$.

First notice that x and y are points on $\mathrm{SP}_{G'}(s,t)$ that are connected through p. Since p does not lie on the straight line segment $\overline{xy}$, the path from x to y makes a turn at p. Moving p infinitesimally vertically down reduces the length of $\mathrm{SP}_{G'}(s,t)$ as long as the combinatorial structure of $\mathrm{SP}_{G'}(s,t)$ (that is, the set of edges of the triangulation) stays the same. Thus if the optimal solution does not have p on $\overline{xy}$, it must be because any movement toward $\overline{xy}$ results in a combinatorial change in the triangulation. This only happens if p, when moved, crosses a circumcircle of a triangle in G, which causes a flip removing xp, py or some other edge in $\mathrm{SP}_{G'}(s,t)$. There are two possibilities when crossing a disk boundary: entering or leaving the disk.

Consider first the case in which p enters the disk. Let $st(p)$ be the star-shaped polygon formed by the union of all triangles incident to p in G'_p. The only change in $\mathrm{SP}_{G'}(s,t)$ occurs in $st(p)$: a flip such that an edge e on the boundary of $st(p)$ disappears, and is replaced by an edge e' incident to p. The current shortest path $\mathrm{SP}_{G'}(s,t)$ can only be affected by the removal of e. However, since e is part of the boundary of $st(p)$, it cannot be part of $\mathrm{SP}_{G'}(s,t)$. This yields a contradiction.

Therefore we are left with the event of leaving the disk. Then a flip occurs in the opposite way: an edge e currently incident to p disappears, and is replaced by another edge e' on the boundary of the new $st(p)$ (see Figure 7). The only part of $\mathrm{SP}_{G'}(s,t)$ that can be affected by such a change is the subpath $\{x,p,y\}$. In particular, a change in the path may happen only if $e = xp$ or $e = py$. Assume without loss of generality that $e = xp$. Let C be the circle that p will cross if moved down. If crossing C removes xp, it must be the case that x lies on C as well. Even though moving p vertically down would make it cross C, in general p could still

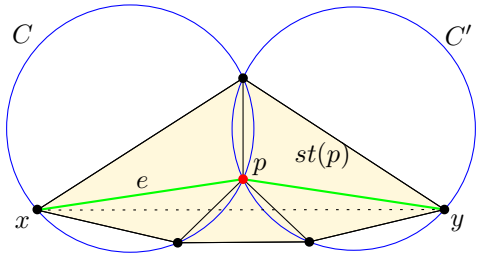

Fig. 7. Situation in proof of Lemma 1: moving p further towards $\overline{xy}$ results in p leaving two circles.

be moved by some small amount along the circle C—thus without crossing it—toward $\overline{xy}$. This would be enough to get p closer to $\overline{xy}$ and thus reduce the length of $\mathrm{SP}_{G'}(s,t)$. But that cannot occur, because p is optimal. Therefore it must be the case that such a movement of p on C cannot be done without affecting the combinatorial structure of $\mathrm{SP}_{G'}(s,t)$. This can only occur if there is a second circle C' through p and y. The result follows. □

Next we show that, if $\mathrm{SP}_{G'_p}(s,t)$ includes an edge xp, then the original shortest path in G between s and x is still present in G'_p.

Lemma 2. *Let p be a point, and let x be a neighbor of p in G'_p. If $SP_{G'_p}(s,p)$ includes xp, then the path $SP_G(s,x)$ exists in G'_p.*

Proof. Let us assume that $\mathrm{SP}_{G'_p}(s,p)$ includes xp and suppose, for the sake of contradiction, that $\mathrm{SP}_G(s,x)$ does not exist in G'_p (i.e. at least one edge of $\mathrm{SP}_G(s,x)$ is not part of G'_p). Consider the shortest path $\mathrm{SP}_{G'_p}(s,p)$ between s and p in G'_p. Clearly, $\mathrm{SP}_{G'_p}(s,p) = \mathrm{SP}_{G'_p}(s,x) \cup xp$. Moreover, $\mathrm{SP}_{G'_p}(s,x)$ is not shorter than $\mathrm{SP}_G(s,x)$ because $\mathrm{SP}_{G'_p}(s,x)$ uses edges that are all in G. If $\mathrm{SP}_G(s,x)$ is not contained in G'_p, it uses some edges that are not present in G'_p. Let e be the first of these edges, when going from s to x. If e is not in G'_p, it is necessarily a diagonal of $st(p)$ (that is, an edge between two non-neighboring vertices of $st(p)$), because no edge outside of $st(p)$ is affected by the insertion of p. Let $e = uv$, where u comes before v when going from s to x. We have the following

$$\begin{aligned}|\mathrm{SP}_{G'_p}(s,p)| &= |\mathrm{SP}_{G'_p}(s,x)| + |xp| \geq |\mathrm{SP}_G(s,x)| + |xp| \\ &= |\mathrm{SP}_G(s,u)| + |\mathrm{SP}_G(u,x)| + |xp| > |\mathrm{SP}_G(s,u)| + |up|,\end{aligned}$$

where the last strict inequality follows from the following argument. In general, $|\mathrm{SP}_G(u,x)| + |xp| \geq |up|$, while the equality holds only if all points in $\mathrm{SP}_G(u,x)$ lie on the segment $\overline{up}$. If u, v, x are all distinct, this is not possible because points in S are in general position, and thus u, v, x cannot be collinear. Otherwise, notice that $u \neq x$ and $u \neq v$, but it could happen that $v = x$. However, in this case v cannot lie on the segment $\overline{up}$, because if uv belongs to G but not to G'_p, it must be because p

lies in all empty circles through u and v in S, and that cannot occur if $u, v = x, p$ are collinear.

Notice that $\mathrm{SP}_G(s,u) \cup up$ is a path between s and p in G'_p by the definition of e. The fact that it is shorter than $\mathrm{SP}_{G'_p}(s,p)$ yields the contradiction.

Therefore $\mathrm{SP}_G(s,x)$ must exist in G'_p. □

3.2. *Algorithm*

Lemma 1 shows that there are two types of candidates for an optimal p, namely, intersection points between two circumcircles of triangles in G, and points on segments with endpoints in S. Suppose that p lies on the segment $\overline{xy}$, where $x, y \in S$, and $\mathrm{SP}_{G'}(s,t)$ contains the edges xp and py. Then p lies in the interior of some Delaunay disk through x, and the same for y, so p is in the intersection of the interiors of two Delaunay disks. This implies that the two types of candidates for p will be considered if we analyze each pair of intersecting Delaunay disks.

We describe an algorithm that solves a more general problem: Instead of fixing s and t, only s is fixed, and we compute for each $t \in S$ the point p_t whose insertion gives the optimal improvement of $\mathrm{SP}(s,t)$.

Description of the algorithm. We first augment G by creating a new graph $H = (S, F)$. This graph has the same set of vertices as G, and has all its edges plus a number of additional *shortcut* edges, which correspond to new paths that we can obtain by inserting one point. We denote by *2-path*(x, y) a path $\{x, v, y\}$, where $v \notin S$. To find such shortcut edges, we proceed as follows. Let $\{C_1, C_2\}$ be a pair of intersecting Delaunay disks. Each disk boundary is the circumcircle of a Delaunay triangle from G. Let the two triangles be t_1, t_2. For each pair of vertices $x \in t_1$ and $y \in t_2$, we first check whether $\overline{xy}$ intersects $C_1 \cap C_2$. If it does, we take as v any point on $(\overline{xy} \cap C_1 \cap C_2)$. Otherwise, among the two points where the boundaries of C_1 and C_2 intersect, we take v as the one minimizing the sum of distances to x and y (see Figure 8). We add a shortcut edge xy with weight equal to the length of the 2-path $\{x, v, y\}$, and store the point v (whose insertion creates the 2-path). The resulting graph H has n vertices and $O(n + I)$ edges, where I is the number of pairs of intersecting Delaunay circles.

We next compute a shortest path tree of s in H (that is, all shortest paths from s in H) with the following restriction: every shortest path in the tree contains at most one shortcut edge. After computing this tree, for every vertex $t \in S$ there are two situations. If the shortest path from s to t does not contain any shortcut edge, then it is impossible to improve $\mathrm{SP}(s,t)$ by inserting only one point. If the shortest path from s to t contains a shortcut edge corresponding to a 2-path $\{x, v, y\}$, we report v as the point whose insertion gives the maximum improvement in $\mathrm{SP}(s,t)$.

Correctness. We need to argue that finding a point that gives the largest improvement to the path from s to t indeed comes down to finding a shortest path

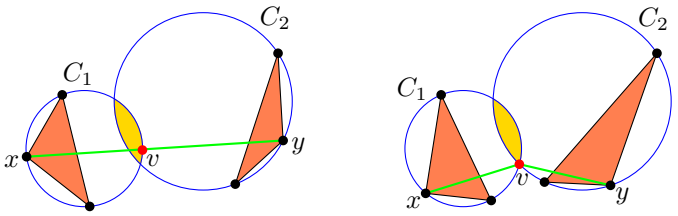

Fig. 8. Two optimal ways to connect x and y: by a point v on $\overline{xy}$ (left), and by a point v on the boundary of $C_1 \cap C_2$ (right).

γ between s and t in H, with the restriction that such path can use at most one shortcut edge. This is straightforward in the situation where it is impossible to improve $\mathrm{SP}(s,t)$ by inserting only one point, so we focus on the other case.

Let p be a point giving the maximum improvement in $\mathrm{SP}(s,t)$. We first show that $|\gamma| \leq |\mathrm{SP}_{G'_p}(s,t)|$, where $|\gamma|$ denotes the length of γ. Let x, y be the points in G such that $\mathrm{SP}_{G'_p}(s,t)$ includes xp and py as consecutive edges. By Lemma 1, there exist a Delaunay disk C_1 through x and a Delaunay disk C_2 through y such that p lies either in $(\overline{xy} \cap C_1 \cap C_2)$ or on the point where the boundaries of C_1 and C_2 intersect that minimizes the sum of distances to x and y. Therefore H contains a shortcut edge corresponding to the 2-path $\{x, p, y\}$. The edges of $\mathrm{SP}_{G'_p}(s,t)$ different from xp and py belong to G and thus to H as well. So $\mathrm{SP}_{G'_p}(s,t)$ is a path in H with one shortcut edge, from which we can conclude that $|\gamma| \leq |\mathrm{SP}_{G'_p}(s,t)|$.

For the reverse inequality, it suffices to show that either γ is a path in G or γ is a path in G'_v for some v. If γ does not contain any shortcut edge, it is trivially a path in G. Otherwise suppose that γ contains a shortcut edge corresponding to the 2-path $\{u, w, z\}$. Edges uw and wz clearly belong to G'_w. Notice that the portion of γ between s and u equals $\mathrm{SP}_G(s,u)$, and the portion of γ between z and t equals $\mathrm{SP}_G(z,t)$. If we show that uw is included in $\mathrm{SP}_{G'_w}(s,w)$ (and symmetrically, wz is included in $\mathrm{SP}_{G'_w}(w,t)$), then we can apply Lemma 2, which ensures that $\mathrm{SP}_G(s,u)$ and $\mathrm{SP}_G(z,t)$ exist in G'_w. This yields that γ is a path in G'_w. Thus the last part of this proof is devoted to arguing that uw is included in $\mathrm{SP}_{G'_w}(s,w)$. Suppose that this is not the case; then w had a different predecessor u' in $\mathrm{SP}_{G'_w}(s,w)$. If $\{u', w, z\}$ is a 2-path in H, then H contains a shorter path between s and w through u' and thus γ is not optimal, a contradiction. Otherwise, since $u'w$ and wz belong to G'_w, there exists a pair of Delaunay circles $\{C_3, C_4\}$ with non-empty intersection such that C_3 goes through u' and C_4 goes through z. Then H must contain a 2-path $\{u', w', z\}$ whose length is smaller than that of $\{u', w, z\}$. This again contradicts the optimality of γ. The correctness of the algorithm follows.

Computing shortest path trees in H with at most one shortcut edge per path. We can modify Dijkstra's algorithm to compute the single-source shortest paths from s in H, ensuring that no path uses more than one shortcut edge. We

next give the details of the algorithm.

First we compute the shortest path tree from s in G, so that we know the distance from s to all other vertices in S before adding any point. Then we run a modified version of Dijkstra's algorithm to compute shortest paths that allow up to one 2-path per path. As in the usual version of the algorithm, we maintain a queue Q with vertices of G where the priorities are the distance to s, denoted $\phi(\ )$ (initially, $\phi(v) = +\infty$ for all $v \in S$ distinct from s, and $\phi(s) = 0$). The distances used for Q consider all paths that use at most one 2-path. In addition to the distance to s, for each vertex we also store a pointer to the neighbor of the node in the shortest path and a flag indicating if the shortest path to the node uses a 2-path or not.

An iteration of the adaptation of Dijkstra's algorithm consists in extracting the currently closest node v from Q, adding it to the shortest path tree from s, and updating certain information (distances, path pointers, and flags) about the neighbors of v in H that are still in Q. This is explained in the following.

If a neighbor, say u, is connected to v via an ordinary edge and $\phi(u) > \phi(v)+|uv|$, we decrease $\phi(u)$ to $\phi(v) + |uv|$, assign to u the 2-path flag of v, and set the path pointer of u to v. If u is connected using a 2-path, the 2-path flag of v is off, and the length of the 2-path plus $\phi(v)$ is smaller than $\phi(u)$, we decrease $\phi(u)$ to the length of the 2-path plus $\phi(v)$, set the flag of u on, and set the path pointer of u to v. Finally, if u is connected using a 2-path and v has the flag on, we cannot append the 2-path to the path of length $\phi(v)$ because it would result in two 2-paths used. In that case we only update $\phi(u)$, and set its flag on, if it decreases when going from s to v using the regular shortest path (without 2-paths), and then using the 2-path between v and u. In all other cases we do not make any update.

The algorithm finishes when Q is empty.

To prove that this algorithm is correct, suppose that some node has the wrong shortest path computed. Among all such nodes, let v be the one that is processed first in the algorithm. Since the algorithm only returns paths containing at most one 2-path, the path reported can only be wrong because there is a shorter path ρ. Since this path is not "found" by the algorithm, at least one node in ρ is processed after v. Let y be the first node in ρ (i.e., closest to s) that is processed after v, and let x be the node immediately before (so $\rho = \{s, \ldots, x, y, \ldots, v\}$). The path stored at x is correct by hypothesis, so the subpath of ρ between s and y is found by the algorithm after processing y. Since the distance between s and y is smaller than the distance between s and v, y is processed before v. This yields a contradiction.

Running time. The running time of the algorithm to compute H is dominated by the time needed to compute all pairs of intersecting Delaunay circles, since for each pair of circles the remaining computations take constant time. Therefore using an algorithm like Balaban's (see Ref. 13) we obtain an $O(n \log n + I)$ running time.

After computing H, we run the modified version of Dijkstra's algorithm. If Q is implemented using Fibonacci heaps (see Ref. 14), the operations of inserting nodes to Q, finding the node from Q that is closest to s and decreasing $\phi(\ )$ are done

in $O(1)$ amortized time, while the extractions are done in $O(\log n)$ time. Since the number of insertions and extractions is $O(|S|)$, and the number of times that some $\phi(\ )$ is decreased is $O(|F|)$, the total running time of the algorithm is $O(n \log n + I)$, using $O(n + I) = O(I)$ space (since in the case of Delaunay circles, $I = \Omega(n)$) .

In a similar way, this approach can be used to compute, for each s and each t, the point p_{st} that gives the optimal improvement for s and t, in $O(n^2 \log n + nI)$ time. Indeed it suffices to apply the algorithm above to all possible origins $s \in S$.

Theorem 1. *Given the Delaunay triangulation of n points and a vertex s, the problem of finding, for each $t \in S$, a point p_t whose insertion gives the maximum improvement in $SP(s,t)$ can be solved in $O(n \log n + I)$ time, where I is the number of intersecting pairs of Delaunay circles. The problem of computing, for each s and each t, the point p_{st} that gives the optimal improvement in $SP(s,t)$ can be solved in $O(n^2 \log n + nI)$ time.*

The implementation of the algorithm above is not straightforward, since to achieve the running time of $O(n \log n + I)$ it is necessary to use a rather involved algorithm to find all the intersections between Delaunay circles in $O(n \log n + I)$ time, and additionally implement Dijkstra's algorithm with Fibonacci heaps. The intersections between Delaunay circles can be computed in $O(n \log n + I \log n) = O(I \log n)$ time by means of a simple plane sweep algorithm (such as Bentley-Ottmann's in Ref. 15). On the other hand, if Q is implemented with a simpler priority queue (e.g. with a binary heap), then the modified version of Dijkstra's algorithm runs in $O(I \log n)$ time. These alternatives yield a slower $O(I \log n)$ time algorithm which could nevertheless be more interesting for practical purposes due to its ease of implementation.

Note that the algorithm in this section does not need any important modification to handle multiplicity of shortest paths (i.e. having more than one shortest path between two points). In particular, Lemma 1 still holds, and Lemma 2 can be easily adapted to hold for *all* shortest paths between x and s.

4. Optimally Adding one Point, Using the Link-Distance

In this section we consider the variant of the problem where the metric used to measure distances on G is the link-distance. This metric is also interesting in networking applications, since it measures the number of hops. We use $d_l(s,t)$ to denote the link-distance between s and t, formally defined as the minimum number of edges of any path in G connecting s and t. As before, we are interested in adding one new point to G such that the link-distance between s and t is minimized. See Figure 9 for some simple examples.

The main difference with the previous problem is that now, if two vertices are incident to Delaunay disks that intersect, then no matter where p is placed inside the intersection of the disks, the length of the resulting path $\{xp, py\}$ is always two. This allows us to design a faster algorithm, which can determine more efficiently

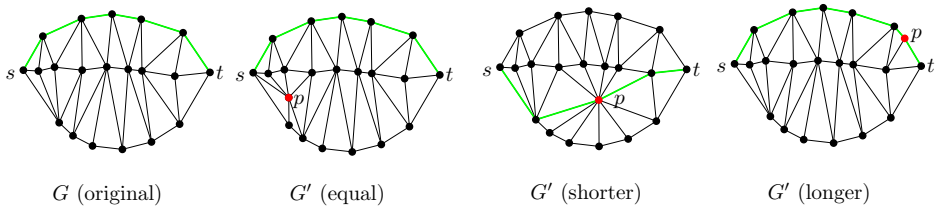

Fig. 9. As occurs with the Euclidean distance, for the link-distance it is also the case that the addition of one point p can improve, make worse or leave unchanged the shortest path between s and t.

whether there is a shortcut from a vertex x that improves the current shortest path. The algorithm uses a special data structure described in the following lines.

Data structure. We design a data structure D that allows us to answer the following type of query in $O(\log^2 n)$ time: given a circle C, find a Delaunay circle C^* of G that intersects C and has minimum link-distance to t. The link-distance from a circumcircle C—corresponding to a Delaunay triangle $\triangle(a, b, c)$—to t is defined as $\min\{d_l(a, t), d_l(b, t), d_l(c, t)\}$.

Let m be the total number of Delaunay circumcircles of G (notice that $m = 2n - 2 - |\mathcal{H}|$, where $|\mathcal{H}|$ is the number of vertices of the convex hull of S). The data structure consists of a balanced binary tree where each node represents the union of a subset of the Delaunay disks. The root of the tree represents the union of all disks. Its right child represents the union of all disks at link-distance at most μ from t, where μ is the median of all distances. The left child represents the union of all disks at distance larger than μ. If the left node contains the union of less than $\lfloor m/2 \rfloor$ disks, we move disks whose link-distance to t is μ from the right node to the left one until each node contains the union of at least $\lfloor m/2 \rfloor$ disks. This structure is repeated recursively, producing a tree of depth $O(\log m)$, where each level of the tree has total size $O(m)$.

Within one node of the tree, the union of the disks is not explicitly computed. Instead, we store the Voronoi diagram of the Delaunay disks, preprocessed for point-location queries. This allows us to determine in $O(\log m)$ time whether the query circle C intersects the circles represented in the node. Since $m \in O(n)$, each level of the tree has total size $O(n)$, and the Voronoi diagram of the circles, or equivalently, the additively-weighted Voronoi diagram of the circle centers (where weights are proportional to radii) can be computed in $O(n \log n)$ time using standard algorithms (e.g. see Ref. 16). Consequently, D requires $O(n \log n)$ space and can be built in $O(n \log^2 n)$ time.

To answer a query for a given circle C, we go down the tree, checking whether C intersects the union of disks stored at the node. If the root of the tree answers negatively, then C does not intersect any Delaunay circle and we stop. The procedure at interior nodes of the tree is as follows. We query the right child of the

node; if the answer is positive, we recurse there; otherwise, we continue in the left child. Since if possible we always recurse on the right child (with circles closer to t than the ones of the left child), the last visited node (a leaf) gives the circle with shortest distance to t, among the ones that intersect C. The total query time for C is therefore $O(\log^2 m)$ or, equivalently, $O(\log^2 n)$.

Algorithm. The algorithm proceeds as follows. First the shortest paths from s and t to all other nodes are precomputed using, for example, Dijkstra's algorithm in $O(n \log n)$ time. Then we go over each point in S and test "how good" it would be as the predecessor of the point that might be inserted. That is, for each possible point $x \in S$ we query the data structure D with each disk that goes through x, and keep track of the lowest value returned. Notice that, if this lowest value is attained by a disk C through x that intersects a disks C^* at link-distance $d_l(C^*)$ from t, then we could insert a point in $C \cap C^*$ and obtain a path between s and t with link-distance $d_l(s, x) + 2 + d_l(C^*)$. After doing this for all points in S, we return a point in the intersection associated with the circles that gave the minimum distance, provided that this distance improves over the original $d_l(s, t)$. Since we perform $O(n)$ queries and each query takes $O(\log^2 n)$ time, the total running time of this part is also $O(n \log^2 n)$.

The correctness of the algorithm follows from Lemma 2, which also holds for the link-distance. The total running time is $O(n \log^2 n)$.

Theorem 2. *Given the Delaunay triangulation of n points, and two vertices s and t, a point whose insertion gives the maximum improvement in $SP(s, t)$, under the link-distance, can be found in $O(n \log^2 n)$ time.*

As in the previous section, we note that the presence of multiple shortest paths between two points does not require any major modification to the algorithm in this section.

5. Optimally Adding More Than One Point

The previous results focus on the simpler variant of the problem where only one point is added to improve $\mathrm{SP}(s, t)$. However, in practice it would be more interesting to efficiently solve the problem in which one can add more than one point. The previous results give insight into the difficulty of this general problem.

Already for inserting two points, it is not evident how to adapt our algorithms. A natural first attempt could be to greedily solve the problem for one point, and afterward add the second point to the obtained graph. Unfortunately, such simple 2-phase algorithm does not lead to any guarantee with respect to the optimal solution: this is because there are point sets that cannot be improved at all with one point, whereas all point sets can be improved with two.

When looking for two points that give the optimal solution, it seems important to distinguish two cases, depending on whether the two inserted points will be

neighbors in the final Delaunay triangulation or not. The latter case allows to search for the two points independently, and it seems possible to use a similar approach to the one used in Section 3.2 (e.g. the algorithm based on 2-paths can be adapted to allow up to two different two-paths, instead of one). However, this does not work in cases in which the two optimal points are neighbors (e.g. in the situation of Figure 4).

To account for these cases, it would be necessary to consider *3-paths*. However, the number of 3-paths seems to grow substantially: a rough estimate is that up to $O(n^4)$ different 3-paths would have to be taken into account. This follows from the fact that different placements for the first point p_1 within one single cell of the arrangement of Delaunay circles can allow or prevent connections to different second points (i.e. different positions within one cell can cause different intersections between circles through p_1 and other circles, where it can be convenient to insert the second point). Therefore, in principle, it would be necessary to subdivide each of $O(I)$ cells in $O(n^2)$ smaller subcells, resulting in a subdivision of complexity $O(n^4)$. Clearly, such an approach does not scale to the more interesting cases of adding $3, 4, \ldots$ points.

Therefore it seems difficult to adapt our current methods to yield efficient optimal algorithms for inserting several points. Whether there exists an algorithm to optimally insert $O(1)$ points in an efficient way is, without any doubts, the main open question arising from this paper.

6. Discussion and Further Work

We have studied the problem of adding a point to a Delaunay triangulation, such that it improves a shortest path as much as possible. We have considered this problem under two metrics, the Euclidean and the link-distance, and discussed some variants. With the use of several geometric observations, we have been able to give relatively simple and efficient algorithms to find the optimal point in the distinct situations.

The main question left open is whether there are efficient algorithms to allow inserting more than one point, as discussed in the previous section. Still, several improvements to our algorithms are possible. A particularly relevant question is whether it is possible to find a point that gives the optimal improvement without computing all intersecting circles. Another intriguing question is whether the decision problem (*Is there a point that improves* $SP(s,t)$*?*) can be solved faster than the optimization problem.

Throughout the paper we have assumed that the Delaunay graph models the network connecting the points in S. However, it would be interesting to consider how much the problem differs when other reasonable proximity graphs are used, such as nearest neighbor graphs, Gabriel graphs, minimum spanning trees, or others. A general approach that works for the Delaunay graph, and also for the proximity graphs that are a subset of it, is to partition the plane into regions such that

inserting p anywhere inside a region produces the same topological change to the graph. For example, in the case of the Delaunay graph this subdivision is given by the arrangement of Delaunay circles, and the subdivision for the minimum spanning tree has been described in Ref. 17. Once this subdivision is computed, one can find the best point p inside each region by optimizing the corresponding function. Let us point out, though, that for the Delaunay graph the resulting algorithm derived from this technique is much less efficient than the one given in Section 3.

Finally, another related line of research worth studying is to find a good definition of a point that can be inserted to globally improve the shortest paths in the network, not only for a particular pair of vertices, and finding algorithms to obtain such point(s).

Acknowledgements

This research was initiated during the 6th Iberian Workshop on Computational Geometry, held in Aveiro, Portugal. M.A., M.C., G.H., F.H., V.S. and R.I.S were partially supported by the ESF EUROCORES programme EuroGIGA - ComPoSe IP04 - MICINN Project EUI-EURC-2011-4306. M.A. and G.H. were partially supported by project MTM2008-05043. M.C., F.H., V.S. and M.S. were partially supported by projects MTM2009-07242 and Gen. Cat. DGR 2009SGR1040. M.C., F.H., V.S. and R.I.S. were partially supported by project MINECO MTM2012-30951. M.S. was supported by GraDR EUROGIGA project No. GIG/11/E023 and ESF EUROCORES programme EuroGIGA, CRP ComPoSe: Fonds National de la Recherche Scientique (F.R.S.-FNRS) - EUROGIGA NR 13604. R.I.S. was supported by the FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235. Finally, we thank the anonymous reviewers for their detailed suggestions, which helped to improve the presentation of this paper.

References

1. J. Liebeherr, M. Nahas and W. Si, Application-layer multicasting with Delaunay triangulation overlays, *IEEE J. Sel. Areas Commun.* **20** (2002) 1472–1488.
2. E. Buyukkaya and M. Abdallah, Efficient triangulation for P2P networked virtual environments, in *Proc. 7th ACM SIGCOMM Workshop on Network and System Support for Games*, Worcester, MA (Oct. 2008) pp. 34–39.
3. H.-Y. Kang, B.-J. Lim and K.-J. Li, P2P spatial query processing by Delaunay triangulation, in *Proc. 4th International Conf. on Web and Wireless Geographical Information Systems*, Goyang, Korea (Nov. 2004) pp. 136–150.
4. P. Bose and P. Morin, Online routing in triangulations, *SIAM J. Comput.* **33** (2004) 937–951.
5. D. P. Dobkin, S. J. Friedman and K. J. Supowit, Delaunay graphs are almost as good as complete graphs, *Discrete Comput. Geom.* **5** (1990) 399–407.
6. P. Bose, L. Devroye, M. Löffler, J. Snoeyink and V. Verma, Almost all Delaunay triangulations have stretch factor greater than $\pi/2$, *Comput. Geom.* **44** (2011) 121–127.
7. G. Xia, Improved upper bound on the stretch factor of Delaunay triangulations, in

Proc. 27th ACM Symp. on Computational Geometry, Paris, France (June 2011) pp. 264–273.
8. A. García, F. Hurtado, M. Noy and J. Tejel, Augmenting the connectivity of outerplanar graphs, *Algorithmica* **56** (2010) 160–179.
9. I. Rutter and A. Wolff, Augmenting the connectivity of planar and geometric graphs, *Electron. Notes Discrete Math.* **31** (2008) 53–56.
10. C. D. Tóth, Connectivity augmentation in plane straight line graphs, *Electron. Notes Discrete Math.* **31** (2008) 49–52.
11. M. Farshi, P. Giannopoulos and J. Gudmundsson, Improving the stretch factor of a geometric network by edge augmentation, *SIAM J. Comput.* **38** (2008) 226–240.
12. J. Luo and C. Wulff-Nilsen, Computing best and worst shortcuts of graphs embedded in metric spaces, in *Proc. 19th International Symp. on Algorithms and Computation*, Gold Coast, Australia (Dec. 2008) pp. 764–775.
13. I. J. Balaban, An optimal algorithm for finding segments intersections, in *Proc. 11th ACM Symp. on Computational Geometry*, Vancouver, BC (June 1995) pp. 211–219.
14. T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms* (MIT press, USA, 2001).
15. J. L. Bentley and T. A. Ottmann, Algorithms for reporting and counting geometric intersections, *IEEE Trans. Comput.* **C-28** (1979) 643–647.
16. S. J. Fortune, A sweepline algorithm for Voronoi diagrams, *Algorithmica* **2** (1987) 153–174.
17. C. L. Monma and S. Suri, Transitions in geometric minimum spanning trees, *Discrete Comput. Geom.* **8** (1992) 265–293.