Minimum Cuts in Geometric Intersection Graphs

Sergio Cabello∗ Wolfgang Mulzer†

May 8, 2020

Abstract

Let $\mathcal{D}$ be a set of $n$ disks in the plane. The disk graph $G_\mathcal{D}$ for $\mathcal{D}$ is the undirected graph with vertex set $\mathcal{D}$ in which two disks are joined by an edge if and only if they intersect. The directed transmission graph $G_\mathcal{D}^\rightarrow$ for $\mathcal{D}$ is the directed graph with vertex set $\mathcal{D}$ in which there is an edge from a disk $D_1 \in \mathcal{D}$ to a disk $D_2 \in \mathcal{D}$ if and only if $D_1$ contains the center of $D_2$.

Given $\mathcal{D}$ and two non-intersecting disks $s, t \in \mathcal{D}$, we show that a minimum $s$-$t$ vertex cut in $G_\mathcal{D}$ or in $G_\mathcal{D}^\rightarrow$ can be found in $O(n^{3/2}/\text{polylog } n)$ expected time. To obtain our result, we combine an algorithm for the maximum flow problem in general graphs with dynamic geometric data structures to manipulate the disks.

As an application, we consider the barrier resilience problem in a rectangular domain. In this problem, we have a vertical strip $S$ bounded by two vertical lines, $L_\ell$ and $L_r$, and a collection $\mathcal{D}$ of disks. Let $a$ be a point in $S$ above all disks of $\mathcal{D}$, and let $b$ a point in $S$ below all disks of $\mathcal{D}$. The task is to find a curve from $a$ to $b$ that lies in $S$ and that intersects as few disks of $\mathcal{D}$ as possible. Using our improved algorithm for minimum cuts in disk graphs, we can solve the barrier resilience problem in $O(n^{3/2}/\text{polylog } n)$ expected time.

Keywords: computational geometry, geometric intersection graph, disk graph, unit-disk graph, vertex-disjoint paths, barrier resilience.

Acknowledgments. Parts of this work were initiated at the Fifth Annual Workshop on Geometry and Graphs that took place March 5–10, 2017, at the Bellairs Research Institute. We thank the organizers and all participants for the productive and positive atmosphere.

∗Faculty of Mathematics and Physics, University of Ljubljana, Slovenia, and Institute of Mathematics, Physics and Mechanics, Slovenia. Supported by the Slovenian Research Agency (P1-0297, J1-9109, J1-8130, J1-8155, J1-1693.). Email address: sergio.cabello@fmf.uni-lj.si

†Institut für Informatik, Freie Universität Berlin, Germany. Supported in part by ERC StG 757609. Email address: mulzer@inf.fu-berlin.de
1 Introduction

Let $\mathcal{D}$ be a family of $n$ disks in the plane. The disk graph $G_{\mathcal{D}}$ for $\mathcal{D}$ is the undirected graph with vertex set $\mathcal{D}$ and edge set

$$E(G_{\mathcal{D}}) = \{D_1D_2 \mid D_1, D_2 \in \mathcal{D}, D_1 \cap D_2 \neq \emptyset\}.$$  

If the disks in $\mathcal{D}$ are partitioned into two sets $\mathcal{D}_A$ and $\mathcal{D}_B$, one can also define a bipartite intersection graph by considering only the edges that come from an intersection between a disk in $\mathcal{D}_A$ and a disk in $\mathcal{D}_B$. If all disks in $\mathcal{D}$ have the same radius, we call $G_{\mathcal{D}}$ a unit-disk graph. A directed version of disk graphs can be defined as follows: for $D \in \mathcal{D}$, let $c_D \in D$ denote the center of $D$. The directed transmission graph $G_{\mathcal{D}}^+$ is the directed graph with vertex set $\mathcal{D}$ and edge set

$$E(G_{\mathcal{D}}^+) = \{D_1 \rightarrow D_2 \mid D_1, D_2 \in \mathcal{D}, c_{D_2} \in D_1\}.$$  

If we ignore the direction of the edges in $G_{\mathcal{D}}^+$, we obtain a subgraph of $G_{\mathcal{D}}$.

Unit disk graphs are often used to model ad-hoc wireless communication networks and sensor networks [GG11, ZG04, HS95]. Disks of varying sizes become relevant when different sensors cover different areas. Moreover, general disk graphs may serve as a tool to approach other problems; for example, an application to the barrier resilience problem [KLA07] is discussed below.

Directed transmission graphs model ad-hoc networks where different entities have different power ranges [PR10].

Minimum $s$-$t$ cut in disk graphs. Consider a graph $G = (V, E)$ with $n$ vertices and $m$ edges, and two non-adjacent vertices $s, t \in V$. A set $X \subseteq V \setminus \{s, t\}$ of vertices is called an $s$-$t$ (vertex) cut if $G - X$ contains no path from $s$ to $t$. Two paths from $s$ to $t$ are (interior-)vertex-disjoint if their only common vertices are $s$ and $t$. By Menger’s theorem (see, for example, [KV10, Section 8.2]), the minimum size of an $s$-$t$ cut equals the maximum number of vertex-disjoint $s$-$t$ paths, both in directed and in undirected graphs. Using blocking flows, Even and Tarjan, as well as Karzanov [ET75, Kar73] showed that an $s$-$t$ minimum-cut can be computed in time $O(\sqrt{nm})$. In the worst case, if $m = \Theta(n^2)$, this is $O(n^{5/2})$. This was an improvement over the previous algorithm by Dinitz [Din70]; see [Din06] for a great historical account of the algorithms. In particular, the use of DFS did not appear in his original description [Din70], but it was developed by Shimon Even and Alon Itai and included in Even’s textbook [Eve79]. The more recent $O(m^{10/7})$-time algorithm of Mädry [Mäd13] gives a better running time for sparse graphs, i.e., for $m = o(n^{7/4})$.

The size of a minimum $s$-$t$ vertex cut in a network $G$ is a key estimator for its vulnerability. Since such networks often arise from geometric settings, it is natural to consider the case where $G$ is a disk graph. A particularly interesting scenario of this kind is the barrier resilience problem, an optimization problem introduced by Kumar, Lai, and Arora [KLA07]. We are given a vertical strip $S$ bounded by two vertical lines, $L_{l}$ and $L_{r}$, and a collection $\mathcal{D}$ of disks. Each disk $D \in \mathcal{D}$ represents a region monitored by a sensor. Let $a$ be a point in $S$ above all disks of $\mathcal{D}$, and let $b$ a point in $S$ below all disks of $\mathcal{D}$. The task is to find a curve from $a$ to $b$ that lies in the strip $S$ and that intersects as few disks of $\mathcal{D}$ as possible. This models the resilience of monitoring a boundary region with respect to (total) failures of the sensors. Kumar, Lai, and Arora show that the problem reduces to an $L_{l}$-$L_{r}$ minimum-cut problem in the intersection graph of $\mathcal{D} \cup \{L_{l}, L_{r}\}$.

A variant of the problem, called minimum shrinkage, was recently introduced by Cabello et al. [CJLM20]. Here, the task is to shrink some of the disks, potentially by different amounts, such that there is an $a$-$b$ curve that is disjoint from the interiors of all disks. The objective is to minimize the total amount of shrinkage. Cabello et al. provide an FPTAS by reducing the problem to a barrier resilience instance with $O(n^2/\varepsilon)$ disks of different radii.
Our Results. We exploit the geometric structure to provide a new algorithm to find the minimum \(s\)-\(t\) cut in disk graphs and directed transmission graphs in \(O(n^{3/2} \text{polylog } n)\) expected time. For this, we adapt the approach of Even and Tarjan [ET75], extending it with suitable geometric data structures. Our method is similar in spirit to the algorithm by Efrat, Itai, and Katz [EIK01] for maximum bipartite matching in (unit) disk graphs. However, since our graph is not bipartite, the structure of the graph is more complex and additional care is needed.

2 Minimum \(s\)-\(t\) Cut in Disk Graphs

Let \(\emptyset\) be a set of \(n\) disks in the plane, and let \(s, t \in \emptyset\) be two non-intersecting disks. We show how to compute the maximum number of vertex disjoint paths between \(s\) and \(t\) in \(G_\emptyset\) and in \(G_\emptyset^\prime\). This also provides a way to find a minimum \(s\)-\(t\) (vertex) cut. For this, we adapt the algorithm of Even and Tarjan [ET75] to our geometric setting. First, we suppose that certain geometric primitives are available as a black box, and we analyze the running time under this assumption. Then, we instantiate these primitives with appropriate data structures to obtain the desired result.

2.1 Generic algorithm

Let \(G\) be a graph with \(n\) vertices and \(m\) edges, and let \(s\) and \(t\) be two non-adjacent vertices of \(G\). We want to find the maximum number of paths from \(s\) to \(t\) in \(G\) that are pairwise vertex disjoint. The graph \(G\) is assumed to be directed.\(^1\)

First, we transform the graph \(G\) into another graph \(G^\prime\) in which every vertex other than \(s\) and \(t\) has in-degree or out-degree 1. More precisely, for each vertex \(v \in V(G) \setminus \{s, t\}\), we perform the following operation: we replace \(v\) with two new vertices \(v_{\text{in}}\) and \(v_{\text{out}}\), add the directed edge \(v_{\text{in}} \rightarrow v_{\text{out}}\), replace every directed edge \(u \rightarrow v\) with \(u \rightarrow v_{\text{in}}\), and replace every directed edge \(v \rightarrow w\) with \(v_{\text{out}} \rightarrow w\); see Figure 1. The vertices \(s\) and \(t\) remain untouched. The transformed graph \(G^\prime\) has \(2n - 2\) vertices and \(m + n - 2\) edges. It is bipartite, as can be seen by partitioning the vertices into the sets \(\{s\} \cup \{v_{\text{out}} \mid v \in V(G) \setminus \{s, t\}\}\) and \(\{t\} \cup \{v_{\text{in}} \mid v \in V(G) \setminus \{s, t\}\}\). Vertex-disjoint \(s\)-\(t\) paths in \(G\) directly correspond to vertex disjoint \(s\)-\(t\) paths in \(G^\prime\). Furthermore, in \(G^\prime\) we have that edge-disjoint and vertex-disjoint \(s\)-\(t\) paths are equivalent, because every vertex (other than \(s\) and \(t\)) has in-degree or out-degree 1. Thus, it suffices to find the maximum number of edge-disjoint \(s\)-\(t\) paths in \(G^\prime\).

Assume we have a family \(\Pi = \{\pi_1, \ldots, \pi_k\}\) of \(k\) edge-disjoint \(s\)-\(t\) paths in \(G^\prime\). Let \(E(\Pi) = \bigcup_{\pi \in \Pi} E(\pi)\) denote the set of all the directed edges on the paths of \(\Pi\). See Figure 2 for an illustration of the following concepts and discussion. The residual graph \(R = R(G^\prime, \Pi)\) is the directed graph with vertex set \(V(G^\prime)\) and edge set

\[
E(R) = \{u \rightarrow v \mid u \rightarrow v \in E(G^\prime) \setminus E(\Pi) \text{ or } v \rightarrow u \in E(\Pi)\}.
\]

The residual graph \(R\) is bipartite with the same bipartition as \(G^\prime\). As in \(G^\prime\), every vertex in \(V(R) \setminus \{s, t\}\) has in-degree or out-degree at most 1.

\(^1\)Otherwise, we replace each undirected edge \(uv\) by two directed edges \(u \rightarrow v\) and \(v \rightarrow u\). An optimal solution to the directed instance directly gives an optimal solution to the undirected case.
For a vertex \( v \) of \( G' \), the *level* \( \lambda(v) \) (with respect to \( R \)) of \( v \) is the BFS-distance from \( s \) to \( v \) in \( R \), i.e., the minimum number of edges on a path from \( s \) to \( v \) in \( R \).\(^2\) If \( v \) is not reachable from \( s \) in \( R \), we set \( \lambda(v) = +\infty \). For every integer \( i \geq 0 \), the *layer* \( L[i] \) is the set of vertices at level \( i \), i.e., \( L[i] = \{ v \in V(G') \mid \lambda(v) = i \} \). The *layered residual graph* \( L(G', \Pi) \) for \( G' \) and \( \Pi \) is the subgraph of the residual graph \( R(G', \Pi) \) where only the directed edges from \( L[i-1] \) to \( L[i] \), for \( i = 1, \ldots, \lambda(t)-1 \), and the directed edges from \( L[\lambda(t)-1] \) to \( t \) are kept. More precisely, this means that \( L = L(G', \Pi) \) has vertex set \( V(G') \) and directed edge set

\[
E_t \cup \{u \rightarrow v \in E(R) \mid \lambda(u) + 1 = \lambda(v) < \lambda(t)\},
\]

where

\[
E_t = \{u \rightarrow t \in E(R) \mid \lambda(u) + 1 = \lambda(t)\}.
\]

Let \( \Gamma = \{\gamma_1, \ldots, \gamma_\ell\} \) be a family of edge-disjoint \( s \rightarrow t \) paths in the layered residual graph \( L = L(G', \Pi) \). By construction, all paths of \( \Gamma \) have exactly \( \lambda(t) \) edges. Using the \( k \) paths of \( \Pi \) in \( G' \) and the \( \ell \) paths of \( \Gamma \) in \( L \), we can obtain \( k + \ell \) edge-disjoint \( s \rightarrow t \) paths in \( G' \). For this, consider the edges

\[
E(\Pi) \oplus E(\Gamma) = \{u \rightarrow v \mid u \rightarrow v \in E(\Pi) \text{ and } v \rightarrow u \notin E(\Gamma)\} \cup \{u \rightarrow v \mid u \rightarrow v \in E(\Gamma) \text{ and } v \rightarrow u \notin E(\Pi)\}
\]

that are obtained from \( E(\Pi) \cup E(\Gamma) \) by canceling out directed edges that appear in both directions. The following observation is simple:

**Lemma 1.** The set \( E(\Pi) \oplus E(\Gamma) \) consists of \( k + \ell \) edge-disjoint \( s \rightarrow t \) paths in \( G' \). Given \( \Pi \) and \( \Gamma \), we can construct \( E(\Pi) \oplus E(\Gamma) \) and the corresponding \( k + \ell \) edge-disjoint \( s \rightarrow t \) paths in \( G' \) in \( O(|E(\Pi)| + |E(\Gamma)|) \) total time.

**Proof.** The definition of \( R \) ensures that the edges \( E(\Pi) \oplus E(\Gamma) \) all lie in \( G' \), since for \( u \rightarrow v \in E(\Gamma) \setminus E(\Pi) \), we must have \( v \rightarrow u \in E(\Pi) \). Furthermore, every vertex \( v \) of \( V(G') \setminus \{s, t\} \) has in-degree and out-degree both \( 0 \) or both \( 1 \) in \( E(\Pi) \oplus E(\Gamma) \). This is clear if \( v \) appears on at most one path in \( \Pi \cup \Gamma \). If \( v \) appears

\(^2\)Recall that \( R \) depends on both \( G' \) and \( \Pi \).
on both a path from $\Pi$ and from $\Gamma$, then one incoming edge and one outgoing edge of $v$ must cancel, since $v$ has at most one incoming or outgoing edge in $L$ and the corresponding reverse edge must have appeared on a path in $\Gamma$. The in-degree of $s$ is 0 and the out-degree of $t$ is 0. Moreover, the out-degree of $s$ is $k + \ell$, because the outgoing edges from $s$ never cancel out. This means that $E(\Pi) \oplus E(\Gamma)$ defines $k + \ell$ paths from $s$ to $t$. These paths can be found in $O(|E(\Pi)| + |E(\Gamma)|)$ time by constructing the graph $(V(\Pi \cup \Gamma), E(\Pi) \oplus E(\Gamma))$ explicitly.

A family $\Gamma$ of $s$-t paths in the layered residual graph $L$ is blocking if $L - E(\Gamma)$ contains no $s$-t path, i.e., every $s$-t path in $L$ contains at least one edge from $E(\Gamma)$.

**Lemma 2.** Let $L$ be a layered residual graph. In $O(|E(L)|)$ time, we can find a blocking family $\Gamma$ of $s$-t paths in $L$.

**Proof.** This lemma is due to Even and Tarjan. We describe the algorithm because we will adapt it to our geometric setting below. We refer to the paper of Even and Tarjan [ET75] for the running time analysis and the proof of correctness.

We start with $\Gamma = \emptyset, D_0 = L,$ and $j = 1$. The algorithm proceeds in rounds. In round $j$, we perform a DFS traversal from $s$ in $D_{j-1}$. When we reach $t$, the DFS stack contains a path $\gamma_j$ from $s$ to $t$ in $D_{j-1} \subseteq L$. We add the path $\gamma_j$ to $\Gamma$, and we obtain $D_j$ by removing from $D_{j-1}$ all the vertices (other than $s$ and $t$) that have been explored during the partial DFS traversal of $D_{j-1}$. We finish when the graph $D_{j-1}$ of the current round $j$ does not contain any $s$-t path. This is detected during the DFS traversal of $D_{j-1}$.

The algorithm to find the maximum number of edge-disjoint s-t paths in $G'$ is the following: we start with $\Pi_0 = \emptyset$. Then, for $j = 1, \ldots$, we construct the residual graph $R_j = R(G, \Pi_{j-1})$, the layered residual graph $L_j = L(G, \Pi_{j-1})$, a blocking family $\Gamma_j$ of s-t paths in $L_j$, and we set $\Pi_j$ to the set of (edge-disjoint) s-t paths defined by $E(\Pi_{j-1}) \oplus E(\Gamma_j)$. We finish when $L_j$ contains no s-t path.

The work performed for a single value of $j$ (constructing $L_j$, $R_j$, $\Gamma_j$, and $\Pi_j$), is called a phase. Let $\lambda_j(\cdot)$ denote the level of a vertex in the residual graph $R_j$. Even and Tarjan [ET75] show that $\lambda_j(t)$ increases monotonically as a function of $j$. Thus, using that the paths $\Gamma_j$ are vertex-disjoint and have length $\lambda_j(t)$ (whenever $L_j$ contains some s-t path), one obtains the following.

**Theorem 3** (Even and Tarjan [ET75]). The algorithm performs at most $O(\sqrt{n})$ phases. When the algorithm finishes, $\Pi_{j-1}$ contains the maximum possible number of vertex-disjoint s-t paths in $G'$.

### 2.2 Adaptation for neighbor queries

We want to adapt the algorithm from Section 2.1 to our geometric setting. For this, we extend the approach by Efrat, Itai, and Katz [EIK01] for finding maximum matchings in bipartite geometric intersection graphs. The idea is to avoid the explicit construction of the layered residual graphs $L_j = L(G, \Pi_{j-1})$, and to use instead an implicit representation that allows for an efficient DFS traversal of the current $L_j$. For this, we identify which vertices belong to each layer of the current $L_j$, and we use dynamic nearest-neighbor data structures to find the directed edges between the layers.

In order to encapsulate the geometric primitives, we assume that we have a certain geometric data structure to access the directed edges of $G$. Note that the assumption is on the original graph $G$, not in the transformed graph $G'$. Later, we will describe how such a data structure can be derived from known results about (semi-)dynamic nearest neighbor searching.

**Graph Encoding A.** Let $G$ be a directed graph with $n$ vertices. We assume that we have a data structure $DS = DS(U)$ that semi-dynamically maintains a subset $U \subseteq V(G)$ with the following operations:

- constructing the data structure for $m$ elements takes $T_c(m)$ time, where $T_c(\cdot)$ satisfies $T_c(m) + T_c(m') \leq T_c(m + m')$;
- a deletion of a vertex in $U$ can be done in $T_d(n)$ time; and
• for any query vertex \( v \in V(G) \), we can, in \( T_q(n) \) time, find an outgoing edge \( v \rightarrow u \) with \( u \in U \), or correctly report that \( U \) contains no such vertex.

Henceforth, we assume our \( n \)-vertex graph \( G \) can be accessed as in Graph Encoding A. As before, we denote the corresponding transformed graph by \( G' \). First, we show how to find the levels in the layered residual graph.

**Lemma 4.** Let \( \Pi \) be a set of edge-disjoint paths in the transformed graph \( G' \). In time \( O(T_q(n) + nT_q(n) + nT_d(n)) \), we can find the level \( \lambda(v) \) of each vertex \( v \in V(G') \) in the layered residual graph \( L = L(G', \Pi) \).

**Proof.** Our goal is to perform a BFS in the residual graph \( R = R(G', \Pi) \) without explicitly constructing the edge set of \( R \). In a preprocessing phase, for every vertex \( v \) in \( V(G') \setminus \{ s, t \} \) that appears in some path of \( \Pi \), we mark \( v \) and store the unique vertices \( \text{prev}(v) \) and \( \text{next}(v) \) such that \( \text{prev}(v) \rightarrow v \) and \( v \rightarrow \text{next}(v) \) are directed edges in \( E(\Pi) \). This takes time \( O(|E(\Pi)|) = O(n) \).

Next, we set \( L[0] = \{ s \} \), construct the data structure \( DS \) of Graph Encoding A for \( V(G) \setminus \{ s \} \). Thus, the current vertex set \( U \) in \( DS \) is initially \( U = V(G) \setminus \{ s \} \). In our algorithm, we iteratively compute the layers \( L[i] \), for \( i = 1, 2, \ldots \). In the process, we maintain the invariant that, after computing \( L[i] \), the structure \( DS \) contains \( t \) and the vertices \( u \) in \( V(G) \) for which we do not yet know the level \( \lambda(u) \) in \( L(G', \Pi) \).

To find \( L[1] \), we repeatedly query \( DS \) with \( s \) and remove from \( DS \) the reported item, until \( DS \) contains no further out-neighbors of \( s \). This gives the set 

\[
U' = \{ u \in V(G) \setminus \{ s \} \mid s \rightarrow u \in E(G) \}
\]

of all out-neighbors of \( s \) in \( G \). Let \( U'_\text{in} = \{ u_{\text{in}} \mid u \in U \} \) be the set of corresponding out-neighbors of \( s \) in \( G' \). We filter \( U' \) and remove those vertices \( v \) that are in some path of \( \Pi \) and have \( \text{prev}(v) = s \). This gives a set \( U'' \) with \( L[1] = U'' \). For each vertex \( u_{\text{in}} \in U'' \), we set \( \lambda(u_{\text{in}}) = 1 \). For each vertex \( u_{\text{in}} \in U' \setminus U'' \), the level of \( u_{\text{in}} \) in \( L \) is not yet known. If \( DS \) supported insertions, we would insert the vertices \( u \) with \( u_{\text{in}} \in U' \setminus U'' \) back into \( DS \). Instead, we just construct the data structure \( DS \) anew for \( V(G) \setminus \{ s \} \cup \{ u \mid u_{\text{in}} \in U'' \} \).

Then, for \( i = 2, \ldots \), while \( L[i-1] \) is not empty and \( L[i-1] \) does not contain \( t \), we compute \( L[i] \). If \( i \) is even, we iterate over the vertices \( v_{\text{in}} \) of \( L[i-1] \); see Figure 3. The vertex \( v_{\text{in}} \) has one outgoing
edge in L: if $v_{in}$ does not lie on some path of $\Pi$, then $L$ contains only the outgoing edge $v_{in} \rightarrow v_{out}$; if $v_{in}$ lies on some path of $\Pi$, then $L$ contains only the outgoing edge $v_{in} \rightarrow \text{prev}(v_{in})$. If $v_{in}$ does not belong to any path of $\Pi$, we set $\lambda(v_{out}) = i$ and add $v_{out}$ to $L[i]$. (In this case, the only incoming edge to $v_{out}$ in the residual graph is from $v_{in}$, so we know that $\lambda(v_{out})$ was not yet determined.) If $v_{in}$ belongs to some path of $\Pi$, we set $u = \text{prev}(v_{in})$ and distinguish two cases. If $u = s$, we do not need to do anything because $\lambda(s)$ is already set. If $u \neq s$, we set $\lambda(u) = i$ and add $u$ to $L[i]$. (In this case, $u = w_{out}$ for some vertex $w \in V(G) \setminus \{s, t\}$ and $\lambda(u)$ was not yet determined because $v_{in} \rightarrow w_{out}$ is the only incoming edge to $w_{out}$ in the residual graph.)

If $i$ is odd, we iterate over the vertices $v_{out}$ of $L[i-1]$; see Figure 4. If the vertex $v_{out}$ does not lie on some path of $\Pi$, the outgoing edges of $v_{out}$ in $L$ correspond to the outgoing edges of $v_{in}$ in $G'$; if $v_{out}$ lies on some path of $\Pi$, then the outgoing edge $v_{out} \rightarrow \text{next}(v_{out})$ in $G'$ is replaced with the outgoing edge $v_{out} \rightarrow v_{in}$ in $R$. We proceed as follows: we query DS repeatedly with $v$ and delete the reported items. This gives the set $U'$ of vertices $u \in V(G)$ that are stored in DS and have $v \rightarrow u \in E(G)$. Due to the invariant, the set $U'$ contains exactly those out-neighbors $u$ of $v$ in $G$ such that $\lambda(u_{in})$ was not known before processing $v_{out}$. If $v_{out}$ lies on some path of $\Pi$, then we already know the level of $w_{in} = \text{next}(v_{out})$ (it is $i-2$) because $w_{in} \rightarrow v_{out}$ is the only incoming edge to $v_{out}$ in the residual graph, and therefore $w \notin U'$. For each $u \in U'$, we set $\lambda(u_{in}) = i$ and add $u_{in} \rightarrow L[i]$. If $v_{out}$ belongs to some path of $\Pi$, we check if $v_{in}$ still has no level assigned, and if so, we set $\lambda(v_{in}) = i$, add $v_{in}$ to $L[i]$, and delete $v$ from DS.

We finish when $t \in L[i]$ or when $L[i]$ is empty. In the latter case, $t$ cannot be reached from $s$ in $R$, and therefore $\Pi$ already contains a maximum number of vertex-disjoint $s$-$t$ paths. In the former case, we remove all elements from $L[\lambda(t)]$ except for $t$.

To bound the running time, we note first that it takes $O(T_{in}(n))$ time to construct the data structure DS, and this is done twice. Next, we observe that every node $u$ of $G$ is deleted at most once from DS. Additionally, each query with a vertex of $G$ in DS leads either to a deletion in DS or does not yield an out-neighbor of the vertex, but the latter happens at most once per vertex of $G$. Thus, in total we are making $O(n)$ queries and deletions in the data structure DS. The time bound follows.

The next lemma shows how to find an actual blocking family in $L$.

**Lemma 5.** Consider a set $\Pi$ of edge-disjoint paths in $G'$. In $O(T_{in}(n) + nT_{q}(n) + nT_{d}(n))$ time, we can find a blocking family of $s$-$t$ paths in the layered residual graph $L$.

**Proof.** Using Lemma 4, we compute the level $\lambda(v)$ of each vertex $v$ of $G'$. Recall the notation $\text{prev}(v)$ and $\text{next}(v)$ from the proof of Lemma 4 to denote the predecessor and successor of a vertex $v$ on a path of $\Pi$. We adapt the algorithm in the proof of Lemma 2, which is based on a DFS traversal of $L$.

For each odd $i$ with $1 \leq i \leq \lambda(t)$, we build a data structure DS[i] as in Graph Encoding A for the set $V[i] = \{v \in V(G) \mid v_{in} \in L[i]\}$. This takes $\sum_{i} |V[i]| \leq O(T_{in}(n))$ time because the sets $V[i]$ are pairwise disjoint. During the algorithm, the data structure DS[i] will contain the vertices $v \in V[i]$ such that $v_{in}$ has not yet been explored by the DFS traversal. Thus, in contrast to the approach in Lemma 2, we delete vertices as we explore them with the DFS traversal.

When we explore a vertex $v_{in}$ (at odd level $i$), there are two options; see Figure 3. If $v_{in}$ lies on some path of $\Pi$, we look at $u = \text{prev}(v_{in})$. If $u$ has been explored already, we return³. Otherwise, we continue the DFS traversal at $u$. If $v_{in}$ does not belong to any path of $\Pi$, then $v_{out}$ has not been explored yet, as $v_{in} \rightarrow v_{out}$ is the only incoming edge of $v_{out}$, so we continue the DFS at $v_{out}$. For each such vertex, we spend $O(1)$ time plus the time for the recursive calls, if they occur.

Consider now the case that we explore a vertex $v_{out}$, at even level $i$; see Figure 4. If $i = \lambda(t) - 1$, we check whether the edge $v \rightarrow t$ belongs to $G \setminus E(\Pi)$. If so, we have found an $s$-$t$ path $\gamma$ in $L$. We add $\gamma$ to the output, and restart the DFS traversal from $s$. If not, we return from the recursive call.

³This happens only if $u = s$, as in any other case $u = w_{out}$ for some vertex $w \in V(G) \setminus \{s, t\}$ and $v_{in} \rightarrow w_{out}$ is the only incoming edge to $w_{out}$ in the residual graph and thus in the layered residual graph.
Consider the remaining case: we explore a vertex $v_{\text{out}}$ at even level $i$ and $i < \lambda(t) - 1$. If $v_{\text{out}}$ belongs to some path of $\Pi$, and $v_{\text{in}}$ has not been explored yet, we recursively explore $v_{\text{in}}$ and remove $v$ from $\text{DS}[i+1]$. If $v_{\text{out}}$ does not belong to any path of $\Pi$ or we have returned from the exploration of $v_{\text{out}}$, we explore the outgoing edges from $v_{\text{out}}$ to $L[i+1]$ by repeating the following procedure. We query $\text{DS}[i+1]$ with $v$ to obtain an edge $v \rightarrow u$ of $G$ such that $u_{\text{in}} \in L[i+1]$, we remove $u$ from $\text{DS}[i+1]$, and we continue the DFS traversal from $u_{\text{in}}$. The recursive call is correctly made along an edge of the layered residual graph because it cannot happen that $v_{\text{out}} \rightarrow u_{\text{in}}$ is an edge of $\Pi$; indeed, if $v_{\text{out}} \rightarrow u_{\text{in}}$ were an edge in $\Pi$, then in the residual graph the edge $u_{\text{in}} \rightarrow v_{\text{out}}$ would be the only edge incoming into $v_{\text{out}}$, which would mean that in the DFS traversal we arrived to $v_{\text{out}}$ from $u_{\text{in}}$, and $u$ would belong to $V[i-1]$ instead of $V[i+1]$. When the query to $\text{DS}[i+1]$ with $v$ returns an empty answer, we return from the recursive call at $v_{\text{out}}$.

Every vertex $u$ of $V[i]$, for $i$ odd, is returned and removed from $\text{DS}[i]$ at most once. Thus, each vertex of $V(G)$ is deleted exactly once from exactly one data structure $\text{DS}[i]$. Furthermore, for every vertex $v$ of $V(G)$, we make at most one query to the corresponding data structure $\text{DS}[]$ that returns an empty answer. Thus, the running time is $O(n + T_q(n) + n T_q(n) + n T_d(n))$. \hfill \square

The following lemma discusses how to find a minimum cut from a maximum family of $s$-$t$ vertex disjoint paths.

**Lemma 6.** Let $\Pi$ be a maximum family of $s$-$t$ vertex disjoint paths (in $G$ or in $G'$). Given $\Pi$, we can obtain a minimum $s$-$t$ cut in $O(T_q(n) + n T_q(n) + n T_d(n))$ time.

**Proof.** Consider the residual graph $R = R(G', \Pi)$. Let $A$ be the set of vertices in $V(G)$ that in the residual graph $R$ are reachable from $s$. A standard result from the theory of maximum flows tells that the edges from $A$ to $V(G) \setminus A$, denoted by $\delta_R(A)$, form a minimum edge $s$-$t$ cut in $G'$ and there are $|\Pi|$ edges in such a cut $\delta_R(A)$.

Let $U$ be the set of vertices $u \in V(\Pi)$ such that $u_{\text{out}} \notin A$ but $u_{\text{in}} \in A$ or such that $u_{\text{in}} \notin A$ but $\text{prev}(u_{\text{in}}) \in A$. (Here, like in previous proofs, we use $\text{prev}(u)$ to denote the vertex such that $\text{prev}(u) \rightarrow u$ belongs to $E(\Pi)$.) Each edge of the cut $\delta_R(A)$ contributes one vertex to $U$. Then $U$ is a minimum $s$-$t$ cut in $G$.

If $t$ is not reachable from $s$ in $R$, then a vertex $u$ is reachable from $s$ in the residual graph $R$ if and only if $u$ is reachable from $s$ in layered residual graph $L$. Thus, to compute $U$, we apply Lemma 4 to find the level $\lambda(v)$ of every vertex $v$ in $L$. Then, the set $U$ is

$$\{ u \in V(G) \mid \lambda(u_{\text{in}}) < +\infty, \lambda(u_{\text{out}}) = +\infty \} \cup \{ u \in V(G) \mid \lambda(\text{prev}(u_{\text{in}})) < +\infty, \lambda(u_{\text{in}}) = +\infty \},$$

as desired. \hfill \square

Now, we put everything together. By Theorem 3, we have $O(\sqrt{n})$ phases, and each phase can be implemented in $O(T_q(n) + n T_q(n) + n T_d(n))$ time because of Lemma 1 and Lemma 5.

**Theorem 7.** Let $G$ be a directed graph with $n$ vertices and assume that a representation of its edges as given in Graph Encoding $A$ is possible. Then, we can find in $O(n^{1/2} T_q(n) + n T_q(n) + n T_d(n))$ time the maximum number of vertex-disjoint $s$-$t$ paths for any given $s, t \in V(G)$. Similarly, we can find a minimum $s$-$t$ cut.

**Proof.** We use the algorithm described in Section 2.1, before Theorem 3. Because of Theorem 3, we have $O(\sqrt{n})$ phases. At phase $j$, we have a set $\Pi_{j-1}$ of vertex-disjoint paths in $G'$, and we use Lemma 5 to find a blocking family $\Gamma_j$ of $s$-$t$ paths in the layered residual graph $L_j = L(G', \Pi_{j-1})$. This takes $O(T_q(n) + n T_q(n) + n T_d(n))$ time per phase. Because of Lemma 1, we can then obtain the new family of $s$-$t$ paths $\Pi_j$ in $O(n)$ time per phase. The result for maximum number of vertex-disjoint $s$-$t$ paths follows. For the minimum $s$-$t$ cut, we use Lemma 6. \hfill \square
3 Geometric Applications

Theorem 7 leads to several consequences for geometrically defined graphs, as we can use geometric data structures to realize Graph Encoding A efficiently. For unit disk graphs, there is the semi-dynamic data structure of Efrat, Itai, and Katz [EIK01]. The construction takes $O(n \log n)$ time, while each deletion and neighbor query takes $O(\log n)$ amortized time. For arbitrary disks, we can use the structure of Kaplan et al. [KMR+17].

Corollary 8. Let $\mathcal{U}$ be a set of $n$ unit disks in the plane and let $s$ and $t$ be two of the disks. We can find in $O(n^{3/2} \log n)$ time the minimum $s$-$t$ cut in the intersection graph $G_{\mathcal{U}}$. For arbitrary disks, the running time becomes $O(n^{3/2} \log^{11} n)$ in expectation.

We can easily adapt the algorithm to the case where $s$ and $t$ are arbitrary shapes (and the other vertices are still represented as disks), by precomputing the disks that are intersected by $s$ and the disks that are intersected by $t$. We get the following consequence.

Corollary 9. The barrier resilience problem with $n$ unit disks can be solved in $O(n^{3/2} \log n)$ time. For arbitrary disks, the running time becomes $O(n^{3/2} \log^{11} n)$.

For directed transmission graphs, we can use the data structure of Chan [Cha19] to report a disk center contained in a query disk. It takes $O(\log^4 n)$ amortized time per edition and query. (See [AM95, Cha10, CT16, KMR+17, Liu20] for related bounds and for an alternative presentation of Chan’s data structure.)

Corollary 10. Let $\mathcal{U}$ be a set of $n$ disks of arbitrary radii in the plane and let $s$ and $t$ be two of the disks. We can find in $O(n^{3/2} \log^4 n)$ time the minimum $s$-$t$ cut in the directed transmission graph $G_{\mathcal{U}}$.

Similar results can be obtained for squares and rectangles using data structures for orthogonal range searching.

References


