Algorithmica
DOI 10.1007/s00453-014-9965-6

The Complexity of Separating Points in the Plane

Sergio Cabello · Panos Giannopoulos

Received: 16 December 2012 / Accepted: 16 December 2014

Abstract We study the following separation problem: given n connected curves and two points s and t in the plane, compute the minimum number of curves one needs to retain so that any path connecting s to t intersects some of the retained curves. We give the first polynomial ($\mathcal{O}(n^3)$) time algorithm for the problem, assuming that the curves have reasonable computational properties. The algorithm is based on considering the intersection graph of the curves, defining an appropriate family of closed walks in the intersection graph that satisfies the 3-path-condition, and arguing that a shortest cycle in the family gives an optimal solution. The 3-path-condition has been used mainly in topological graph theory, and thus its use here makes the connection to topology clear. We also show that the generalized version, where several input points are to be separated, is NP-hard for natural families of curves, like segments in two directions or unit circles.

Keywords Point separation · 3-Paths property · Connected curves · NP-hardness

A preliminary version of this work appeared in the Proceedings of the 29th Annual Symposium on Computational Geometry (SoCG), pp. 379–386, 2013. Research by S. Cabello was partially supported by the Slovenian Research Agency, Program P1-0297, Projects J1-4106 and L7-5459, and within the EUROCORES Programme EUROGIGA (Project GReGAS) of the European Science Foundation. Research by P. Giannopoulos was partially supported by the German Science Foundation (DFG) under Grant Kn 591/3-1 while the author was affiliated with Universität Bayreuth, Bayreuth, Germany.

S. Cabello
Department of Mathematics, IMFM and FMF, University of Ljubljana,
Jadranska 19, 1000 Ljubljana, Slovenia
e-mail: sergio.cabello@fmf.uni-lj.si

P. Giannopoulos (✉)
School of Science and Technology, Middlesex University, The Burroughs, Hendon,
London NW4 4BT, UK
e-mail: p.giannopoulos@mdx.ac.uk

Journal: 453 Article No.: 9965 ☐ TYPESET ☐ DISK ☐ LE ☐ CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

1 Introduction

Let C be a family of n connected curves in the plane, and let s and t be two points not incident to any curve of C. In the 2-Point-Separation problem we want to compute a subset $C' \subseteq C$ of minimum cardinality that *separates* s from t, i.e., any path connecting s to t intersects some curve of C'. Its generalization where several input points are to be separated will be referred to as Points-Separation.

We will actually solve a natural weighted version of 2-Point-Separation, where we have a weight function w assigning weight $w(c) \geq 0$ to each curve $c \in C$. For any subset $C' \subseteq C$ we define its weight $w(C')$ as the sum of the weights over all curves $c \in C'$. The task is to find a minimum weight subset $C' \subseteq C$ that separates two given points s and t. Such weighted scenario is useful, for example, when we want to keep separated two points in a polygonal domain using a subset of disks. In such case, we can assign weight 0 to each edge of the domain and weight 1 to the boundary of each disk. See Fig. 1 for an example. Such problem naturally arises in so-called barrier problems when wireless sensors are modeled by disks [4,11].

In typical scenarios, C is a family of circles or segments, possibly of unit size. In our algorithms we need to assume that some primitive operations involving the input curves can be carried out efficiently. Henceforth, we will assume that the following primitive operations can be done in constant time:

1. given two curves c and c' of C, we can compute a point in $c \cap c'$ or correctly report that c and c' are disjoint;
2. given a curve c of C and two points x and y on c, we can compute the number of crossings between a path inside c that connects x to y and the segment $\overline{st}$;
3. given a curve c of C, we can decide whether c separates s and t;
4. given two curves c and c' of C, we can decide whether c and c' together separate s and t.

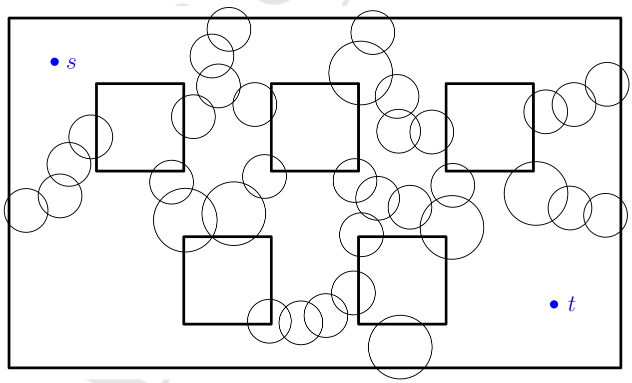


Fig. 1 A possible instance for 2-Point-Separation with weights: a polygonal domain with *five rectangular holes* and *several disks*. The task is to retain the minimum number of disks such that any path connecting s to t inside the domain intersects some retained disk

These operations take constant time for semialgebraic curves of constant description complexity.

Our results We provide an algorithm that solves the weighted version of 2-POINT-SEPARATION in $\mathcal{O}(nk + n^2 \log n)$ time, where k is the number of pairs of curves that intersect. The algorithm itself is simple, but its correctness is not obvious. We justify its correctness by considering an appropriate set of closed walks in the intersection graph of the curves and showing that it satisfies the so-called *3-path-condition* [16] (see also [13, Chapter 4]). The use of the 3-path-condition for solving 2-POINT-SEPARATION is surprising, but it makes the connection to topology clear. In fact, our approach can be interpreted alternatively as searching for a shortest non-zero-homologous cycle in $\mathbb{R}^2 \backslash \{s, t\}$ (with coefficients in $\mathbb{Z}_2$). This approach works when the optimal solution is given by at least three curves. We take care for the case when the optimal solution is attained by two curves separately by brute-force.

On the negative side, we use a reduction from PLANAR-3-SAT to show that POINTS-SEPARATION is NP-hard for two natural families of curves:

– horizontal and vertical segments;
– unit circles.

Related work Gibson et al. [7] provide a polynomial-time $\mathcal{O}(1)$-approximation algorithm for the problem POINTS-SEPARATION for disks. Their approach is based on building a solution by considering several instances of 2-POINT-SEPARATION with disks, which they solve also approximately. It should be noted that no polynomial-time algorithm that gives the exact optimum for 2-POINT-SEPARATION was previously known, even for unit disks. Using our exact solution to 2-POINT-SEPARATION for the boundaries of the disks leads to a better approximation factor in the final outcome of their algorithm.

The ideas used here for 2-POINT-SEPARATION were already included in the unpublished manuscript with Alt and Knauer [2] for segments. This work replaces and extends that part of the manuscript. In the terminology used in Wireless Sensor Networks, we are computing a minimum-size 1-barrier [4,10]. Researchers have also considered the dual problem of computing the so-called resilience: remove the minimum number of curves such that there exists a path from s to t avoiding the retained curves. Computing the resilience was shown to be NP-hard for arbitrary segments by Alt et al. [2,3], and for unit segments by Tseng and Kirkpatrick [17,18]. A constant-factor approximation algorithm for resilience in families of unit disks was given by Bereg and Kirkpatrick [4].

In an independent and simultaneous work, Penninger and Vigan [15] have shown that POINTS-SEPARATION is NP-hard for the case of unit disks. Their reduction is from the problem PLANAR-MULTITERMINAL-CUT and it is very different from ours. Note that in our reduction we need unit *circles*.

Roadmap In Sect. 2 we describe the algorithm for 2-POINT-SEPARATION. We argue its correctness in Sect. 3. In Sect. 4 we show that POINTS-SEPARATION is NP-hard.

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

2 Algorithm for 2-Point-Separation

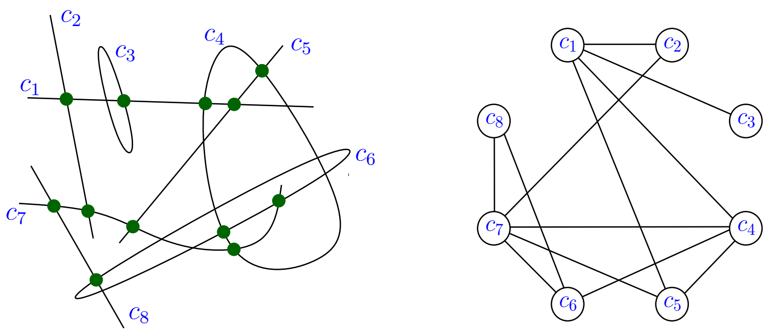
In this section we describe a polynomial-time algorithm for 2-POINT-SEPARATION. Our time bounds will be expressed as a function of n, the number of curves in C, and k, the number of pairs of curves from C with non-empty intersection. We justify the correctness of the algorithm in Sect. 3.

2.1 Preliminaries

The use of the term *curve* will be restricted to elements of C. The use of the term *path* (or closed path) will be restricted to *parametric* paths constructed in our algorithm and proofs. The use of the term *walk* will be restricted to graphs. A *cycle* is a closed walk in a graph without repeated vertices.

General position We are going to count crossings between portions of the input curves and the segment $\overline{st}$. To simplify the exposition, we assume general position in the following sense: the segment $\overline{st}$ does not contain any self-intersection of a curve of C; the segment $\overline{st}$ does not contain any intersection of two curves of C; the segment $\overline{st}$ is not tangent to any curve of C, thus any intersection of $\overline{st}$ with any curve of C is a crossing; no curve of C contains a non-zero-length portion of $\overline{st}$. For reasonable curves, these assumptions can be ensured (or avoided, from the point of view of a programmer) with a small perturbation of s. Separating s and t or separating a small enough perturbation of s and t are equivalent problems.

Intersection graph The set C of input curves defines the intersection graph $\mathbb{G} = \mathbb{G}(C) = (C, \{cc' \mid c \cap c' \neq \emptyset\})$; see Fig. 2. Note that $\mathbb{G}$ has k edges. To each edge cc' of $\mathbb{G}$ we attach the weight (abstract length) $w(c) + w(c')$. Any distance in $\mathbb{G}$ will refer to these edge weights. For any walk π in $\mathbb{G}$ we use $\text{len}_{\mathbb{G}}(\pi)$ for its length, that is, the sum of the weights on its edges counted with multiplicity, and $C(\pi) = V(\pi)$ for the set of curves that appear as vertices in the walk π.



(a) **(b)**

Fig. 2 **a** A set of curves C with the fixed intersection points $x_{c,c'}$. **b** The corresponding intersection graph $\mathbb{G}$

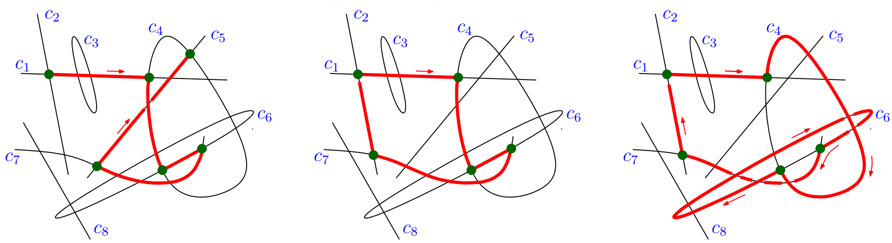
For each curve $r \in C$, let T_r be a shortest-path tree of $\mathbb{G}$ from r; if there are several, we select one of them arbitrarily and maintain this choice throughout the algorithm. For any $r \in C$ and any edge $e \in E(\mathbb{G})\setminus E(T_r)$, let $\mathsf{walk}(r, e)$ denote the closed walk obtained by concatenating the edge e with the two paths in T_r from r to the endpoints of e. When $\mathsf{walk}(r, e)$ is a cycle it is usually called a *fundamental cycle* with respect to T_r.

Fixing intersections and subpaths For each two distinct curves c and c' from C that intersect, we fix an intersection point and denote it by $x_{c,c'}$; if there are different choices, we choose $x_{c,c'}$ arbitrarily and maintain this choice throughout the algorithm. Given a curve $c \in C$ and two points x, y on C, let $c[x \to y]$ be any path contained in c connecting x to y; if there are different choices, we choose $c[x \to y]$ arbitrarily.

π-paths Consider a walk $\pi = c_0 c_1 \cdots c_t$ in $\mathbb{G}$. Let γ be a path in $\mathbb{R}^2$. We say that γ is a *π-path* if there are paths $\gamma_1, \ldots, \gamma_{t-1}$ such that: the path γ_i is contained in c_i $(i = 1, \ldots, t-1)$, the path γ_i goes from x_{c_{i-1},c_i} to $x_{c_i,c_{i+1}}$ $(i = 1, \ldots, t-1)$, and the concatenation of $\gamma_1, \ldots, \gamma_{t-1}$ gives γ. The intuition is that γ starts at x_{c_0,c_1}, follows c_1 until x_{c_1,c_2}, follows c_2 until x_{c_2,c_3}, and so on, until eventually it arrives to x_{c_{t-1},c_t} by following c_{t-1}. See Fig. 3a for an example.

If the walk $\pi = c_0 c_1 \cdots c_t$ is closed, which means that $c_t = c_0$, then a closed path γ is a *closed π-path* if there are paths $\gamma_1, \ldots, \gamma_t$ such that: the path γ_i is contained in c_i $(i = 1, \ldots, t)$, the path γ_i goes from x_{c_{i-1},c_i} to $x_{c_i,c_{i+1}}$ $(i = 1, \ldots, t$ and $c_{t+1} = c_1)$, and the concatenation of $\gamma_1, \ldots, \gamma_t$ gives γ. See Fig. 3b–c for an example. If γ is a π-path or a closed π-path, then $\gamma \subset \bigcup C(\pi)$. Even if π is a cycle, which is a closed walk without repeated vertices, a closed π-path may have self-intersections.

There may be different π-paths. Given a walk $\pi = c_0 c_1 \cdots c_t$ in $\mathbb{G}$ we can construct a π-path in linear time by concatenating $c_j[x_{c_{j-1},c_j} \to x_{c_j,c_{j+1}}]$ for $j = 1, \ldots, t-1$. If π is a closed walk with $c_0 = c_t$, we can obtain a closed π-path by closing it with $c_0[x_{c_{t-1},c_0} \to x_{c_0,c_1}]$. When the input family C is a family of pseudosegments, there is a unique π-path for each walk π and a unique closed π-path for each closed walk π.



(a) **(b)** **(c)**

Fig. 3 Some paths in the example of Fig. 2, using the fixed intersection points marked in Fig. 2. In (**a**) there is a π-path for the walk $\pi = c_2 c_1 c_4 c_6 c_7 c_5 c_4$. In (**b**) and (**c**) there are two different closed π-paths for the closed walk $\pi = c_2 c_1 c_4 c_6 c_7 c_2$

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

We will mainly use closed ($\mathsf{walk}(r, e)$)-paths, where r is a curve of C and $e \in E(\mathbb{G})\setminus E(T_r)$. Thus, we introduce the notation $\gamma(r, e)$ to denote a closed ($\mathsf{walk}(r, e)$)-path; if there are several such paths, it denotes an arbitrary one.

Counting crossings Let γ be a path contained in $\bigcup C$, possibly with self-intersections. We define $N(\gamma)$ as the number of crossings between $\overline{st}$ and γ, modulo 2. (Due to the general position assumptions, no self-intersections of γ are counted.) If $C' \subset C$ does not separate s and t, then for any closed path γ contained in $\bigcup C'$ we have $N(\gamma) = 0$.

Let π be a walk in $\mathbb{G}$ and let γ be some π-path. We define $N(\pi) = N(\gamma)$. Thus, $N(\cdot)$ is defined for paths in the plane and for walks in $\mathbb{G}$. A priori, the value $N(\pi)$ depends on the choice of the π-path γ. However, as we will see in Lemma 3, when no curve of C alone separates s and t, the value $N(\pi)$ is independent of the choice of γ. Our first step in the algorithm will be to remove from C any curve that separates s and t.

In this paper,

any arithmetic involving $N(\cdot)$ is done modulo 2.

Because of our assumptions on general position, for any walk $c_0 c_1 \cdots c_t$ and any i, $1 < i \leq t$, we have

$$N(c_0 c_1 \cdots c_t) = N(c_0 c_1 \cdots c_{i-1} c_i) + N(c_{i-1} c_i \cdots c_t).$$

2.2 The Algorithm

We now describe the algorithm. Firstly, we select the minimum-weight solution $C_{\leq 2}$ consisting of one or two curves from C. We do this by testing separately each curve and each pair of curves from C. Of course, it may be that $C_{\leq 2}$ is undefined.

We remove from C any curve that alone separates s and t. We keep using C for the remaining set of curves.

Next we compute the set

$$P = \{(r, e) \in C \times E(\mathbb{G}) \mid e \in E(\mathbb{G})\setminus E(T_r) \text{ and } N(\mathsf{walk}(r, e)) = 1\}.$$

Then we choose

$$(r^*, e^*) \in \arg \min_{(r,e)\in P} \operatorname{len}_{\mathbb{G}}(\mathsf{walk}(r, e)),$$

and compute $C_{>2} = C(\mathsf{walk}(r^*, e^*))$. It may happen that P is empty, which means that (r^*, e^*) and $C_{>2}$ are undefined.

If both $C_{\leq 2}$ and $C_{>2}$ are defined, we return the lightest of them. If only one among $C_{\leq 2}$ and $C_{>2}$ is defined, we return the only one that is defined. If both $C_{\leq 2}$ and $C_{>2}$ are undefined, we return "C does not separate s and t". This finishes the description of the algorithm. We will refer to this algorithm as ALGORITHM-2PS.

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

2.3 Time Complexity of the Algorithm

ALGORITHM-2PS, as described above, can be implemented in $\mathcal{O}(n^2k + n^2 \log n)$ time in a straightforward way. Since computing $C_{\leq 2}$ can be done trivially in $\mathcal{O}(n^2)$ time, the bottleneck of the computation is to obtain (r^*, e^*). We next describe how to obtain a better time bound.

Lemma 1 ALGORITHM-2PS *can be modified to run in* $\mathcal{O}(nk + n^2 \log n)$ *time.*

Proof The set $C_{\leq 2}$ can be computed in $\mathcal{O}(n^2)$ time by brute force. We compute (r^*, e^*) and $C_{>2} = C(\mathsf{walk}(r^*, e^*))$ as follows.

The graph $\mathbb{G}$ can be constructed explicitly in $\mathcal{O}(n^2)$ time by checking each pair of curves, whether they cross or not. Recall that $\mathbb{G}$ has k edges.

For any curve $r \in C$, let us define

$$\begin{aligned} E_r &= \{e \in E(\mathbb{G}) \mid (r, e) \in P\} \\ &= \{e \in E(\mathbb{G}) \mid e \in E(\mathbb{G}) \setminus E(T_r) \text{ and } N(\mathsf{walk}(r, e)) = 1\}. \end{aligned}$$

Note that

$$P = \bigcup_{r \in C} \{r\} \times E_r,$$

and therefore

$$\min_{(r,e) \in P} \mathrm{len}_{\mathbb{G}}(\mathsf{walk}(r, e)) = \min_{r \in C} \min_{e \in E_r} \mathrm{len}_{\mathbb{G}}(\mathsf{walk}(r, e)).$$

Thus, (r^*, e^*) can be computed by finding, for each $r \in C$, the value

$$\min_{e \in E_r} \mathrm{len}_{\mathbb{G}}(\mathsf{walk}(r, e)).$$

We shall see that, for each fixed $r \in C$, such value can be computed in $\mathcal{O}(k + n \log n)$ time. It then follows that (r^*, e^*) can be found in $|C| \times \mathcal{O}(k + n \log n) = \mathcal{O}(nk + n^2 \log n)$ time.

For the rest of the proof, let us fix a curve $r \in C$. Computing the shortest-path tree T_r takes $\mathcal{O}(|E(\mathbb{G})| + |V(\mathbb{G})| \log |V(\mathbb{G})|) = \mathcal{O}(k + n \log n)$ time. The main idea now is simple: for each edge $cc' \in E(\mathbb{G})$, we can obtain $N(\mathsf{walk}(r, cc'))$ and $\mathrm{len}_{\mathbb{G}}(\mathsf{walk}(r, cc'))$ in constant time using information stored at c and c'. (The details below become a little cumbersome.)

For any curve $c \in C$, $c \neq r$, let $T_r[c]$ denote the path in T_r from r to c, let $A_r[c]$ be the child of r in $T_r[c]$, and let $N_r[c] = N(T_r(c))$. See Fig. 4a–b.

The values $N_r[c]$, $c \in C$, can be computed in $\mathcal{O}(n)$ time using a BFS traversal of T_r, as follows. We set $N_r[r] = 0$ and, for each child c of r, we set $N_r[c] = 0$. For any other curve c, if $p_r(c)$ is the parent of c in T_r, we can compute $N_r[c]$ from $N_r[p_r(c)]$ in $\mathcal{O}(1)$ time using that

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

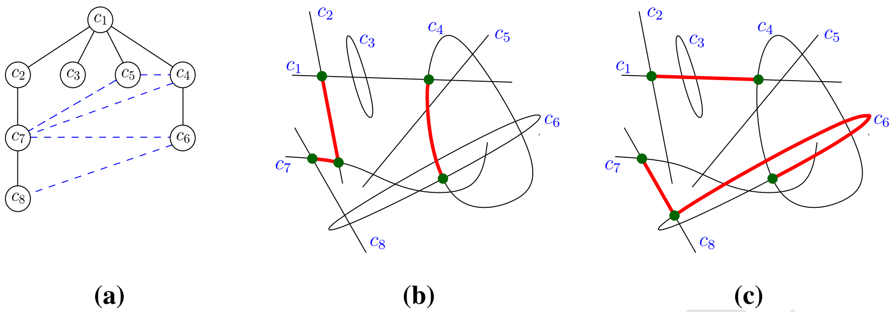


Fig. 4 **a** Tree T_{c_1} for the scenario of Fig. 2 assuming curves of unit weight. In this case $A_{c_1}[c_8] = c_2$ and $A_{c_1}[c_6] = c_4$. **b** Possible $(T_{c_1}[c_8])$-path and $(T_{c_1}[c_6])$-path used to compute $N_{c_1}[c_8]$ and $N_{c_1}[c_6]$. **c** Possible $(c_7c_8c_6c_4)$-path and $(c_4c_1c_2)$-path that are used to compute $N(\mathsf{walk}(c_1, c_6c_8))$ in Lemma 1

$$
\begin{aligned}
N_r[c] &= N_r[p_r(c)] + N\,(p_r(p_r(c))\;p_r(c)\;c)\\
&= N_r[p_r(c)] + N\left(p_r(c)[x_{p_r(p_r(c)),p_r(c)} \to x_{p_r(c),c}]\right).
\end{aligned}
$$

In this last equality we are constructing implicitly a $T_r[c]$-path from a $T_r[p_r(c)]$-path attaching to it a path contained in the curve $p_r(c)$.

We can also compute $A_r[c]$ for all $c \in C, c \neq r$, using a BFS traversal of T_r. We set $A_r[c] = c$ for each child c of r and, for any other $c \in C$, we set $A_r[c] = A_r[p_r(c)]$, where $p_r(c)$ is again the parent of c in T_r.

For $cc' \in E(\mathbb{G})\setminus E(T_r)$, we have that

$$
N(\mathsf{walk}(r, cc')) = N_r[c] + N\left(p_r(c)\,c\,c'\,p_r(c')]\right) + N_r[c'] + N\left(A_r[c']\,r\,A_r[c]]\right).
$$

See Fig. 4b–c. Therefore, each $N(\mathsf{walk}(r, cc'))$ can be computed in $\mathcal{O}(1)$ time from the values $N_r[c]$, $N_r[c']$, $A_r[c]$, $A_r[c']$. It follows that E_r can be constructed in $\mathcal{O}(|E(\mathbb{G})|) = \mathcal{O}(k)$ time.

The length of any closed walk $\mathsf{walk}(r, e)$ can be computed in $\mathcal{O}(1)$ time per pair (r, e) in a similar fashion. For each vertex c, we store at c its shortest-path distance $d_{\mathbb{G}}(r, c)$ from the root r. The length of the closed walk $\mathsf{walk}(r, cc')$ can then be recovered using

$$
\mathrm{len}_{\mathbb{G}}(\mathsf{walk}(r, cc')) = d_{\mathbb{G}}(r, c) + w(c) + w(c') + d_{\mathbb{G}}(r, c').
$$

Equipped with this, we can in $\mathcal{O}(k)$ time compute

$$
\min_{e \in E_r} \mathrm{len}_{\mathbb{G}}(\mathsf{walk}(r, e)).
$$

□

The following special case may be relevant in some applications.

Lemma 2 *If the weights of the curves C are 0 or 1, then* Algorithm-2PS *can be modified to run in $\mathcal{O}(nk + n^2)$ time.*

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

Proof In this case, a shortest path tree T_r can be computed in $\mathcal{O}(|E(\mathbb{G})| + |V(\mathbb{G})|) = \mathcal{O}(k + n)$ time because the edge weights of $\mathbb{G}$ are 0, 1, or 2. Using the approach described in the proof of Lemma 1 we spend $\mathcal{O}(k+n)$ per root $r \in C$, and thus spend $\mathcal{O}(nk + n^2)$ in total. □

3 Correctness of the Algorithm for 2-Point-Separation

In this section we show the correctness of ALGORITHM-2PS. Since in ALGORITHM-2PS we test each curve of C whether it separates s and t, and, if it does, then remove it from C, and since every such separating curve is tested for optimality,

we can assume henceforth that no curve in C separates s and t.

As already mentioned earlier, we first show that this assumption implies that the choice of π-paths made to define $N(\pi)$ is irrelevant.

Lemma 3 *Let π be a walk in $\mathbb{G}$ and let γ and γ' be two π-paths. Then $N(\gamma) = N(\gamma')$. Similarly, if π is a closed walk in $\mathbb{G}$ and γ and γ' are two closed π-paths, then $N(\gamma) = N(\gamma')$.*

Proof Let c be any curve of $C(\pi)$. Since c does not separate s and t, any closed path contained in c crosses $\overline{st}$ an even number of times. We can use this to make replacements that transform γ into γ' while keeping $N(\gamma)$ constant, as follows.

We consider the case where π is a closed walk and γ and γ' are closed π-paths. The other case is similar.

Let $\gamma_1, \ldots, \gamma_t$ be the pieces of γ that certify that γ is a closed π-curve. Similarly, let $\gamma'_1, \ldots, \gamma'_t$ be the pieces of γ' that certify that γ' is a closed π-curve. For $i = 1, \ldots, t$, the paths γ_i and γ'_i have the same endpoints (x_{c_{i-1},c_i} and $x_{c_i,c_{i+1}}$, where $c_0 = c_t$ and $c_1 = c_{t+1}$) and are contained in c_i. Therefore $N(\gamma_i) + N(\gamma'_i) = 0$ for $i = 1, \ldots, t$, which implies $N(\gamma_i) = N(\gamma'_i)$. We thus have

$$N(\gamma) = \sum_{i=1}^{t} N(\gamma_i) = \sum_{i=1}^{t} N(\gamma'_i) = N(\gamma').$$

□

3.1 3-Path-Condition

Consider the set of closed walks

$$\Pi(C) = \{\pi \mid \pi \text{ is a closed walk in } \mathbb{G}(C);\ N(\pi) = 1\}.$$

We will drop the dependency on C and use $\Pi = \Pi(C)$. However, towards the end we will use $\Pi(\tilde{C})$ for some $\tilde{C} \subseteq C$.

We next show the following property, known as 3-path-condition. It implies that from the 3 "natural" closed walks defined by 3 walks with common endvertices, either 2 or none belong to Π.

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

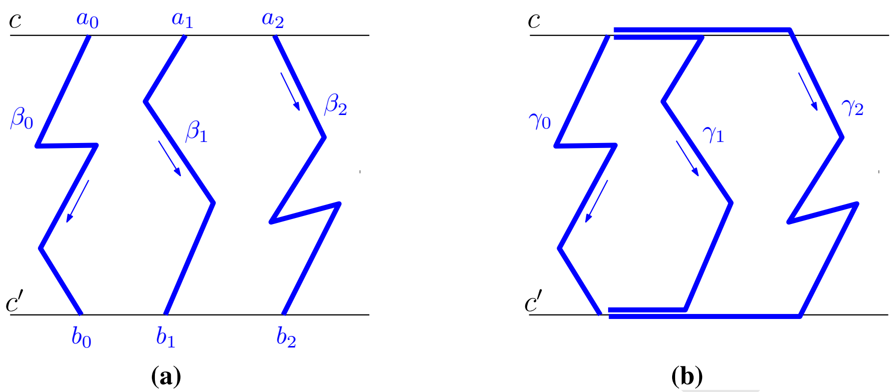


Fig. 5 Notation in the Proof of Lemma 4. (Parts of γ_1 and γ_2 lie on $c \cup c'$. We draw them outside because of the common part)

Lemma 4 *Let $\alpha_0, \alpha_1, \alpha_2$ be 3 walks in $\mathbb{G}$ from c to c'. For $i = 0, 1, 2$, let π_i be the closed walk obtained by concatenating α_{i-1} and the reverse of α_{i+1}, where indices are modulo 3. Then $N(\pi_1) + N(\pi_2) + N(\pi_3) = 0$.*

Proof This is basically a matter of parity. For $i = 0, 1, 2$, let β_i be any α_i-path, let $a_i \in c$ be its endpoint on c and let $b_i \in c'$ be its endpoint on c'. See Fig. 5a. Note that the paths $\beta_0, \beta_1, \beta_2$ start on c and finish on c', but they have different endpoints. To handle this, for $i = 0, 1, 2$, we define γ_i to be the path obtained by the concatenation of $c[a_0 \to a_i]$, β_i, and $c'[b_i \to b_0]$. Now the paths $\gamma_0, \gamma_1, \gamma_2$ start at a_0 and finish at b_0. See Fig. 5b. For $i = 0, 1, 2$, let δ_i be the closed π_i-path defined by concatenating β_{i-1}, $c'[b_{i-1} \to b_{i+1}]$, the reversal of β_{i+1}, and $c[a_{i+1} \to a_{i-1}]$, where indices are taken modulo 3. Because of Lemma 3 we have $N(\pi_i) = N(\delta_i)$ for $i = 0, 1, 2$.

A simple but tedious calculation shows that, using indices modulo 3,

$$N(\delta_i) = N(\gamma_{i-1}) + N(\gamma_{i+1}).$$

Indeed, since c does not separate s and t, any closed path contained in c crosses $\overline{st}$ an even number of times and thus

$$N(c[a_0 \to a_{i+1}]) + N(c[a_{i+1} \to a_{i-1}]) + N(c[a_{i-1} \to a_0]) = 0.$$

Since we use arithmetic modulo 2 and $N(c[a_{i-1} \to a_0]) = N(c[a_0 \to a_{i-1}])$ we obtain

$$N(c[a_{i+1} \to a_{i-1}]) = N(c[a_0 \to a_{i+1}]) + N(c[a_0 \to a_{i-1}]).$$

Similarly, for c' we have

$$N(c'[b_{i+1} \to b_{i-1}]) = N(c'[b_0 \to b_{i+1}]) + N(c'[b_0 \to b_{i-1}]).$$

Then we have

$$\begin{aligned}N(\delta_i) &= N(\beta_{i-1}) + N(c'[b_{i-1} \to b_{i+1}]) + N(\beta_{i+1}) + N(c[a_{i+1} \to a_{i-1}])\\ &= N(\beta_{i-1}) + N(c'[b_0 \to b_{i+1}]) + N(c'[b_0 \to b_{i-1}])\\ &\quad + N(\beta_{i+1}) + N(c[a_0 \to a_{i+1}]) + N(c[a_0 \to a_{i-1}])\\ &= N(c[a_0 \to a_{i-1}]) + N(\beta_{i-1}) + N(c'[b_0 \to b_{i-1}])\\ &\quad + N(c[a_0 \to a_{i+1}]) + N(\beta_{i+1}) + N(c'[b_0 \to b_{i+1}])\\ &= N(\gamma_{i-1}) + N(\gamma_{i+1}).\end{aligned}$$

It follows that, using indices modulo 3,

$$\sum_{i=0}^{2} N(\pi_i) = \sum_{i=0}^{2} N(\delta_i) = \sum_{i=0}^{2} (N(\gamma_{i-1}) + N(\gamma_{i+1})) = 0.$$

□

When a family of closed walks satisfies the 3-path-condition, there is a general method to find a shortest element in the family. The method is based on considering so-called fundamental cycles defined by shortest-path trees, which is precisely what ALGORITHM-2PS is doing specialized for the family Π. See [16] or [13, Chapter4] for the original approach, and [6] for a recent extension to weighted, directed graphs.

Lemma 5 *Assume that Π is nonempty. Then the closed walk $\tau^* = \mathsf{walk}(r^*, e^*)$ computed by* ALGORITHM-2PS *is a cycle and is a shortest closed walk of Π.*

Proof We first show that each shortest closed walk of Π is a cycle. This is a consequence of Lemma 4. Assume for the sake of a contradiction that some shortest closed walk π of Π repeats a vertex c. Then we apply Lemma 4 to two non-trivial subwalks π' and π'' of π from c to c and the trivial walk with only vertex c. (Lemma 4 does not require that $c \neq c'$.) It follows that both π' and π'' are shorter than π and either $N(\pi') = 1$ or $N(\pi'') = 1$, so π could not be shortest in Π. We conclude that each shortest closed walk of Π is a cycle.

Consider the set of closed walks

$$\Pi' = \{\mathsf{walk}(r, e) \mid r \in C,\ e \in E(\mathbb{G})\backslash E(T_r),\ N(\mathsf{walk}(r, e)) = 1\} \subseteq \Pi.$$

We are going to show that some shortest closed walk of Π is in Π'.

Choose a vertex r with the property that some shortest closed walk of Π goes through r. Choose a closed walk π of Π through r that is shortest. If Π has several different shortest closed walks through r, we take π that minimizes the number of edges in $E(\mathbb{G})\backslash E(T_r)$. Since T_r is a tree, π must contain some edges from $E(\mathbb{G})\backslash E(T_r)$. We are going to show that π has exactly one edge from $E(\mathbb{G})\backslash E(T_r)$.

Assume, for the sake of contradiction, that π contains at least two edges e and e' from $E(\mathbb{G})\backslash E(T_r)$. See Fig. 6 for the following notation. Let c be a vertex between e and e' as we walk from r along π. (If e and e' have a common vertex, then c must be

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

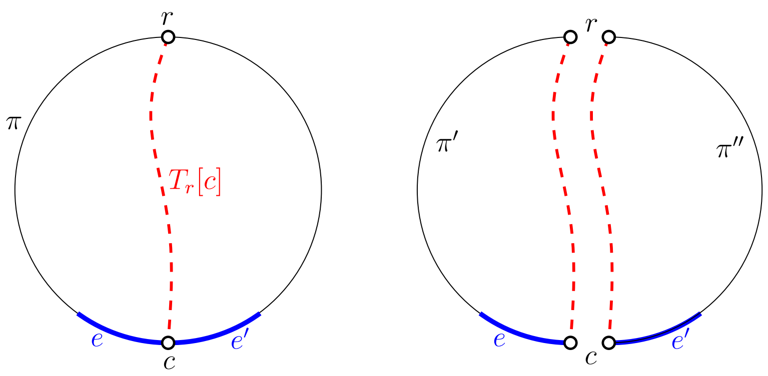


Fig. 6 Notation in the Proof of Lemma 5

that common vertex.) The closed walk π defines two walks from r to c, one in each orientation. Let π' be the closed walk obtained by concatenating one of those walks with the reversal of $T_r[c]$ and let π'' be the closed walk obtained by concatenating the other walk with the reversal of $T_r[c]$. Applying Lemma 4 to the two walks from r to c defined by π and the walk $T_r[c]$ we obtain

$$N(\pi) + N(\pi') + N(\pi'') = 0.$$

Since $N(\pi) = 1$ because $\pi \in \Pi$, then either $N(\pi') = 1$ or $N(\pi'') = 1$. Take $\tilde{\pi}$ to be the cycle among π' and π'' with $N(\tilde{\pi}) = 1$. Note that $\tilde{\pi}$ goes through r, is no longer than π (we are replacing a part of π with the shortest path $T_r[c]$), and contains at least one edge (e or e') less from $E(\mathbb{G}) \setminus E(T_r)$. Such closed walk $\tilde{\pi}$ would contradict the choice of π. We conclude that π cannot have two edges from $E(\mathbb{G}) \setminus E(T_r)$, and thus it has exactly one edge from $E(\mathbb{G}) \setminus E(T_r)$.

Since π has a single edge of $E(\mathbb{G}) \setminus E(T_r)$, then $\pi \in \Pi'$. We have seen that finding a shortest closed walk in Π amounts to finding a shortest closed walk in Π'. The closed walk $\mathsf{walk}(r^*, e^*)$, as computed by ALGORITHM-2PS, is a shortest element of Π' by construction, and thus also a shortest element of Π. □

3.2 Feasibility

The next step in our argument is showing that, when $C_{>2}$ is defined, it is a feasible solution. For this we find a closed, simple path contained in $C_{>2}$ that separates s and t.

Lemma 6 *Assume that Π is nonempty and let π be any cycle in Π. The set of curves $C(\pi)$ separates s and t.*

Proof Let γ be a closed path contained in $C(\pi)$ with $N(\gamma) = 1$ and with the minimum number of self-intersections. Such a path exists because $\pi \in \Pi$ and thus some closed π-path crosses $\overline{st}$ an odd number of times.

We can use an uncrossing argument to show that γ has no self-intersection, as follows. See Fig. 7. Assume, for the sake of contradiction, that γ has a self-intersection

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

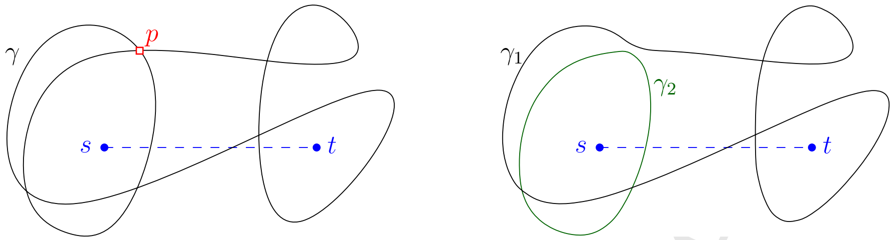


Fig. 7 Uncrossing argument in the Proof of Lemma 6

at a point p. We can uncross γ at p to obtain two closed paths γ_1 and γ_2, each of is part of γ and has fewer self-crossings than γ. Note that

$$1 = N(\gamma) = N(\gamma_1) + N(\gamma_2)$$

because the paths γ_1 and γ_2 form a disjoint partition of γ. Therefore, for $i = 1$ or $i = 2$, the path γ_i has $N(\gamma_i)$ odd, is part of γ and thus contained in $C(\pi)$, and has fewer self-crossings than γ. This would contradict the choice of γ. We conclude that γ must be simple.

Since γ is simple and $N(\gamma)$ is odd, γ separates s and t. It follows that $C(\pi)$ separates s and t because γ is contained in $C(\pi)$. □

We next argue that the algorithm computes a feasible solution, when it exists. We know that $C_{\geq 2} = C(\mathsf{walk}(r^*, e^*))$ separates s and t, when it is defined, but could it happen that Π is empty and thus (r^*, e^*) is undefined?

Lemma 7 *If C separates s and t but no two curves in C separate s and t, then Π is nonempty.*

Proof Consider the connected component of $\mathbb{R}^2 \setminus \bigcup C$ containing s. Since C separates s and t, t is in a different connected component. Let δ be a simple, closed path contained in the boundary of the connected component of s in $\mathbb{R}^2 \setminus \bigcup C$ such that δ separates s and t. We then have $N(\delta) = 1$.

Let $c_0, c_1, \ldots, c_t$ (with $c_t = c_0$) be the sequence of input curves that contain δ, in the order in which they are visited by δ. We have $t \geq 3$ because no two curves separate s and t. Note that $\pi = c_0 c_1 \cdots c_t$ is a closed walk of $\mathbb{G}$. We will see that $\pi \in \Pi$, which implies that Π is nonempty. It is not true in general that δ is a closed π-path because it does not need to pass through the fixed intersection points $x_{c_i, c_{i+1}}$. However, we can construct a closed π-path δ'' such that $N(\delta'') = N(\delta) = 1$, as follows.

Let δ_i be a path contained in c_i such that the concatenation of $\delta_0, \delta_1, \ldots, \delta_{t-1}$ is δ. For $i = 0, \ldots, t-1$, let a_i be the start point of δ_i and let δ'_i be the path obtained by the concatenation of $c_i[x_{c_{i-1}, c_i} \to a_i]$, δ_i, and $c_{i+1}[a_{i+1} \to x_{c_i, c_{i+1}}]$. Thus, for $i = 0, \ldots, t-1$, the path δ'_i starts at x_{c_{i-1}, c_i}, finishes at $x_{c_i, c_{i+1}}$, and is contained in $c_i \cup c_{i+1}$. Finally, let δ' be the concatenation of $\delta'_0, \delta'_1, \ldots, \delta'_{t-1}$. Since δ' is obtained from δ by inserting the paths $c_i[x_{c_{i-1}, c_i} \to a_i]$ twice, once in each direction, we have $N(\delta') = N(\delta) = 1$. See Fig. 8.

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

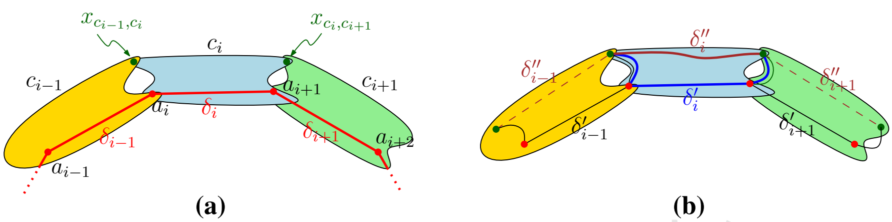


Fig. 8 **a** Notation and **b** the paths δ'_i, δ''_i constructed in the Proof of Lemma 7

For $i = 0, \ldots, t-1$, let $\delta''_i = c_i[x_{c_{i-1},c_i} \to x_{c_i,c_{i+1}}]$. Define δ'' as the concatenation of $\delta''_0, \ldots, \delta''_{t-1}$. Note that δ'' is a π-path by construction. Note that, for $i = 0, \ldots, t-1$, the paths δ'_i and δ''_i are contained in $c_i \cup c_{i+1}$ and have the same endpoints. See Fig. 8. Since $c_i \cup c_{i+1}$ does not separate s and t, it holds $N(\delta'_i) = N(\delta''_i)$. It follows that

$$N(\delta'') = \sum_i N(\delta''_i) = \sum_i N(\delta'_i) = N(\delta') = 1.$$

Since δ'' is a closed π-path and $N(\pi) = N(\delta'') = 1$, we have $\pi \in \Pi$. □

3.3 Main Result

We can now prove that ALGORITHM-2PS correctly solves the problem 2-POINTS-SEPARATION.

Theorem 1 *The weighted version of* 2-POINTS-SEPARATION *can be solved in* $\mathcal{O}(nk + n^2 \log n)$ *time, where n is the number of input curves and k is the number of pairs of curves that intersect.*

Proof We use ALGORITHM-2PS. The running time follows from Lemma 1. If C does not separate s and t, then Π is empty because of Lemma 6, both $C_{>2}$ and $C_{\leq 2}$ are undefined, and the algorithm will return the correct answer.

It remains to see the feasibility and optimality of the solution returned by ALGORITHM-2PS when C separates s and t. If there is an optimal solution consisting of at most two curves, then it is clear that the algorithm is correct because $C_{>2}$ is always a feasible solution, if defined. Let us consider the case when each optimal solution has at least three curves. Let $\tilde{C} \subseteq C$ be one such optimal solution. Because of Lemma 7 applied to $\tilde{C}$, we know that $\Pi(\tilde{C})$ is non-empty. Let $\tilde{\tau}$ be a shortest cycle in $\Pi(\tilde{C})$. Since $C(\tilde{\tau}) \subset \tilde{C}$ is a feasible solution, because of Lemma 6 applied to $\Pi(\tilde{C})$, and $\tilde{C}$ is an optimal solution, it must be $\tilde{C} = C(\tilde{\tau})$.

Now note that $\Pi(\tilde{C}) \subseteq \Pi(C)$ because $\tilde{C} \subseteq C$, which implies that $\tilde{\tau}$ is a cycle of $\Pi(C)$. Since τ^* is a shortest cycle in $\Pi(C)$ due to Lemma 5, we have $\text{len}_{\mathbb{G}}(\tau^*) \leq \text{len}_{\mathbb{G}}(\tilde{\tau})$. For any cycle π of $\mathbb{G}$ we have $\text{len}_{\mathbb{G}}(\pi) = 2w(C(\pi))$ because of the choice of the edge-weights in $\mathbb{G}$. This implies that

$$w(C_{>2}) = \tfrac{1}{2}\,\text{len}_{\mathbb{G}}(\tau^*) \leq \tfrac{1}{2}\,\text{len}_{\mathbb{G}}(\tilde{\tau}) = w(C(\tilde{\tau})) = w(\tilde{C}).$$

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

It follows that $C_{>2}$ is a feasible solution whose weight is not larger than $w(\tilde{C})$, and therefore $C_{>2}$ is optimal. □

Corollary 1 *The weighted version of* 2-POINT-SEPARATION *in which the curves have weights 0 or 1 can be solved in* $\mathcal{O}(n^2 + nk)$ *time, where* n *is the number of input curves and* k *is the number of pairs of curves that intersect.*

Proof In the proof of the previous theorem we use Lemma 2 instead of Lemma 1. □

4 Hardness of Point-Separation

In this section we show that POINTS-SEPARATION is NP-hard for two families of curves: (1) horizontal and vertical segments, and (2) unit circles. We reduce from PLANAR-3-SAT.

Consider a 3-CNF formula with a set $\mathscr{C}$ of clauses over a set X of boolean variables. Its *formula graph* is defined as the bipartite graph on $\mathscr{C} \cup X$ that has an edge connecting $x \in X$ to $C \in \mathscr{C}$ if and only if C contains literal x or $\neg x$. A *3-legged representation* of the formula graph is a plane, rectilinear drawing where the variables and clauses are drawn as axis-aligned rectangles, the variables are aligned horizontally, and the edges are vertical segments; see the example in Fig. 9. PLANAR-3-SAT is the restriction of 3-SAT to formulae whose formula graph is planar and has a 3-legged representation. PLANAR-3-SAT is NP-complete [12], and it remains so when the 3-legged representation is given as part of the input. Several NP-hardness proofs of geometric problems have used PLANAR-3-SAT; see for example [1,5,8,9], and [14].

The reductions for segments and circles are based on the same ideas. Given an instance of PLANAR-3-SAT consisting of a formula Φ, with n variables and m clauses, and a 3-legged representation L, we transform it into an instance $I(\Phi)$ of POINTS-SEPARATION by replacing the rectangles in L with gadgets, while maintaining their relative position and the planarity of the representation. In our case we do not need a gadget to represent the edges because the interaction is straightforward. We describe the reduction for segments first, and in more detail, since it is easier to visualize.

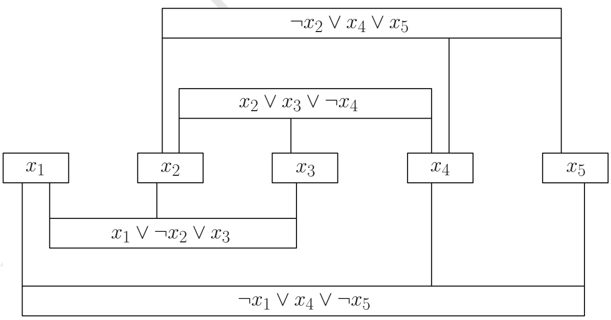


Fig. 9 Rectilinear representation of planar 3-SAT

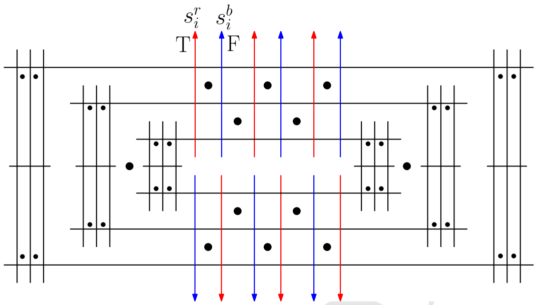


Fig. 10 Variable gadget for POINTS-SEPARATION with *horizontal/vertical* segments. The segments with *arrows* may be extended

Let $\kappa \leq m$ be the maximum number of occurrences of a variable in Φ and $\ell \leq \kappa$ be the maximum number of edge-segments connecting the top or bottom side of a variable-rectangle with a clause-rectangle in L.

4.1 Horizontal and Vertical Segments

Variables In $I(\Phi)$, a variable is now represented by three nested *frames* (drawn in black), which define two disjoint, cyclic *corridors*; see Fig. 10. (From now on, such a structure will be simply referred to as frame.) The top and bottom side of a frame consist of one horizontal segment each. The left and right side of a frame are composed of three vertical segments and one horizontal segment each. We place four points at each side in such a way that removing any one of the ten segments of a frame results in at least two points being in the same cell. Therefore, all of these segments must be present in any feasible solution. This finishes the description of a frame.

Next, we place ℓ pairs S_i, $1 \leq i \leq \ell$, of vertical segments such that both segments of each pair intersect the top side of every frame. Similarly, we place ℓ pairs S_i, $\ell < i \leq 2\ell$ that intersect the bottom sides of the frames. Some of the segments in pairs will be elongated later to cross a rectangle clause, depending on the actual formula. Each pair encodes a truth assignment for the variable and consists of a *positive* (red) segment s_i^r which corresponds to TRUE and a *negative* (blue) one s_i^b which corresponds to FALSE. The pairs are arranged in such a way that when walking around a corridor positive and negative segments alternate. In the upper corridor, we place a point between the segments of every pair, while in the inner one we place a point between every two consecutive pairs. The latter ensures that at least one segment from each pair is needed for separating the points in the inner corridor.

Clauses A clause in $I(\Phi)$ is represented by one frame (as defined in the paragraph above); see Fig. 11. For each variable that occurs in the clause, we elongate one segment from the corresponding variable gadget: a positive (red) segment is elongated for positive occurrences and a negative (blue) one for negative occurrences. Such elongated segments cross the frame for the clause. Finally, we place one point p^l at the left side of the frame and one point p^r at the right side such that at least one elongated edge-segment is needed for separating the points.

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

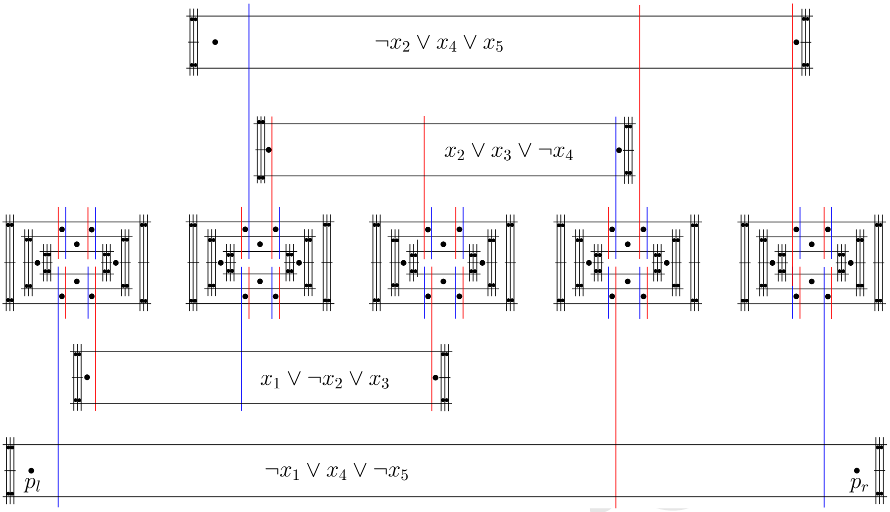


Fig. 11 The construction with segments for the example of Fig. 9

Correctness Let P and S be the set of all points and segments in $I(\Phi)$ respectively. We claim that the points in P can be separated with $30n + 10m + 2\ell \cdot n$ segments from S if and only if Φ is satisfiable. First, assume that those many segments are sufficient for separation. As argued above in the description of a frame, all its ten segments are necessary for separation, hence, we have the remaining $2\ell \cdot n$ segments at our disposal for separating the points in every corridor and points p^l and p^r in every clause gadget. From the discussion on the variable gadget we know that at least one segment from every red/blue pair S_i must be used for the points in the inner corridor to be separated. Since there are 2ℓ such pairs, exactly one segment from every pair must be used in every variable gadget. Consider an arbitrary red segment s_i^r. If s_i^r is included in the solution, then in order to separate the point between s_i^r and s_i^b from the next, in clockwise order, point in the corridor, the red segment of the adjacent pair (in the same order) must also be chosen. A similar observation holds also for an arbitrary choice of a blue segment, where now the choice propagates in counterclockwise order. Hence, in a variable gadget, either all red or all blue segments must be chosen. But since points p^l and p^r must be also separated, there must be a choice such that the frame of each clause gadget is intersected by at least one red or blue edge-segment. Such a choice corresponds to a truth assignment that satisfies Φ. The converse is obvious. We have proved the following.

Theorem 2 Points-Separation *is NP-hard for families of vertical and horizontal segments.*

4.2 Unit Circles

Variables For unit circles we use the variable gadget displayed in Fig. 12. It contains $3\ell - 1$ disjoint triples of black circles at the center, which form its *backbone*. The

Journal: 453 Article No.: 9965 TYPESET DISK LE CP Disp.:2014/12/24 Pages: 21 Layout: Small-X

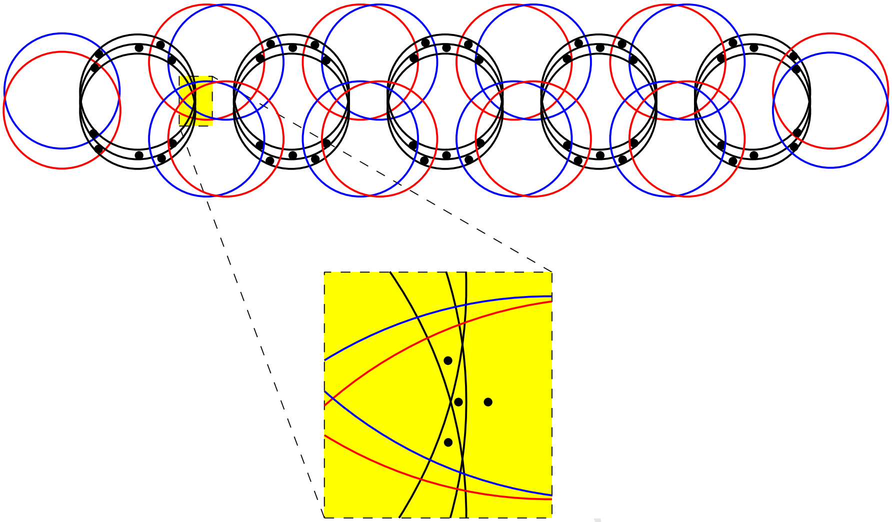


Fig. 12 Variable gadget for POINTS-SEPARATION with unit circles (*top*). The extra points that ensure that all *black circles* are part of any feasible solution are shown in the *zoomed-in area* (*bottom*)

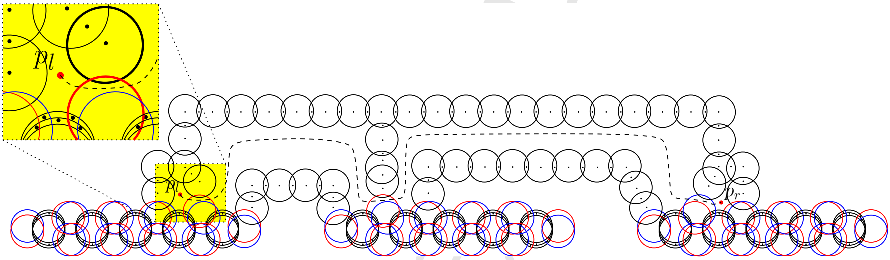


Fig. 13 The clause $(x_2 \vee x_3 \vee \neg x_4)$ with *unit circles*. The corridor is marked by a *dashed path*. The *zoomed-in area* (*top left*) shows a *red circle* intersecting a *black circle* of the corridor (both fat) and disconnecting the corridor

circles in each triple intersect pairwise and define four lunes. With four extra points per triple, as described later on, we can ensure that all these black circles are part of any feasible solution. The gadget also contains $6\ell - 2$ pairs of red/blue circles. Each pair encodes a truth assignment, where the red circle corresponds to TRUE and the blue one to FALSE. In particular, there are two pairs (a top and a bottom one) between every two consecutive triples. Each such pair intersects the lunes of both triples such that its circles cover the right-side intersection points of (the circles of) one triple and the left-side intersection points of the other one. Additionally, there is one pair intersecting the leftmost triple of the gadget and one pair intersecting the rightmost triple. The red/blue pairs are arranged in such a way that when walking along a lune red and blue circles alternate. Next, we place ten points inside the lunes of each triple, as shown in Fig. 12. Note that inside every inner-most lune there is a point that is not covered by any red or blue circle. This ensures that at least one red or blue circle from every pair must be present in any feasible solution. Finally, for every triple, we place

Journal: **453** Article No.: **9965** TYPESET DISK LE CP Disp.:**2014/12/24** Pages: **21** Layout: **Small-X**

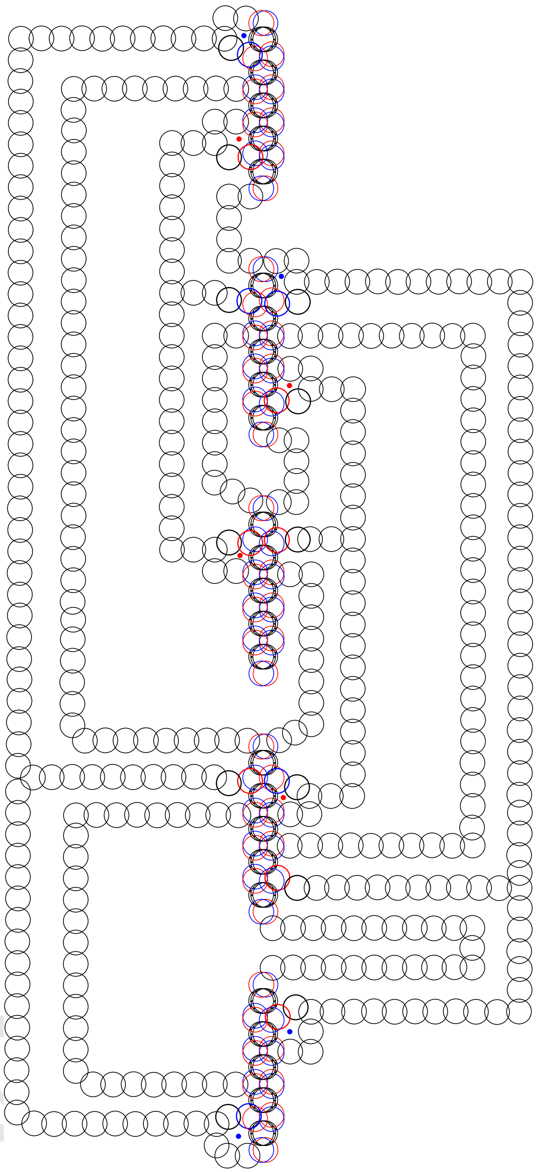


Fig. 14 The construction with unit circles for the example of Fig. 9

four extra points around the intersection points of its circles, see Fig. 12 (bottom), such that all points are covered by both circles of at least one red/blue pair, and such that removing any black circle of the triple results in two of these points being in the same cell. The latter ensures that all circles of a triple must be present in any feasible solution, while the former ensures that all extra points are pairwise separated from all other points inside the lunes, and thus, they do not influence the choice of a red or blue circle in a feasible solution.

Clauses The rectangle representing a clause above the line of variables in the 3-legged representation L is deformed into an M-shaped corridor whose boundary contains black unit circles attached to variable gadgets, see Fig. 13. For this, we use three consecutive red/blue pairs: one black circle intersects both circles of the first pair, another one intersects both circles of the third pair, and one more intersects only the red or the blue circle of the middle pair. Again, using extra points, i.e., one point per cell that is covered only by black circles, we enforce all black circles of a corridor to be part of any feasible solution. We also place two points, p^l and p^r, at the left and right end of the corridor. The corridor is traversed by three red or blue circles from the variables: each circle comes from some red/blue pair of the gadget of a variable that belongs to the clause and splits the corridor into two disconnected parts, thus cutting every path between the two points at the ends of the corridor.

The complete construction with unit circles for the example of Fig. 9 is shown in Fig. 14. To avoid a cluttered figure, some of the extra points are not shown.

Correctness Every variable gadget has $3(3\ell - 1)$ black circles, $6\ell - 2$ red circles, and $6\ell - 2$ blue circles. It is clear that for each clause gadget the number of horizontally placed black circles is some quadratic polynomial on n and ℓ and the number of vertically placed black circles is some linear function on m. Let $b(I)$ be the total number of black circles in $I(\Phi)$.

Constructing a feasible solution to $I(\Phi)$ with $b(I) + (6\ell - 2) \cdot n$ circles from a truth assignment for Φ is immediate. An argument similar to the one used for segments shows that any feasible solution with $b(I) + (6\ell - 2) \cdot n$ circles contains all black circles and, in each variable gadget, either all red circles or all blue circles. The choice of red or blue circles made in the variable gadget corresponds to a truth assignment of the variables, and such assignment satisfies the clauses because, in each clause gadget, the points p^l and p_r are separated. Therefore a feasible solution containing exactly $b(I) + (6\ell - 2) \cdot n$ circles exists if and only if Φ is satisfiable.

Theorem 3 POINTS-SEPARATION *is NP-hard for families of unit circles.*

5 Open Questions

The most prominent open questions here are whether POINTS-SEPARATION admits a PTAS and whether it is fixed-parameter tractable with respect to the solution size, i.e., the number of separating curves.

Acknowledgments We would like to thank Primož Škraba for related discussions and the referees for their careful comments.

References

1. Agarwal, P.K., Suri, S.: Surface approximation and geometric partitions. SIAM J. Comput. **27**(4), 1016–1035 (1998)
2. Alt, H., Cabello, S., Giannopoulos, P., Knauer, C.: Minimum cell connection and separation in line segment arrangements. CoRR abs/1104.4618 (2011)

Journal: **453** Article No.: **9965** TYPESET DISK LE CP Disp.:**2014/12/24** Pages: **21** Layout: **Small-X**

3. Alt, H., Cabello, S., Giannopoulos, P., Knauer, C.: On some connection problems in straight-line segment arrangements. In: Abstracts of the 27th EuroCG, pp. 27–30 (2011)
4. Bereg, S., Kirkpatrick, D.G.: Approximating barrier resilience in wireless sensor networks. In: Proceedings of 5th ALGOSENSORS, LNCS, vol. 5804, pp. 29–40. Springer, Berlin (2009)
5. Cabello, S., Demaine, E.D., Rote, G.: Planar embeddings of graphs with specified edge lengths. J. Graph Algorithms Appl. **11**(1), 259–276 (2007)
6. Cabello, S., de Verdière, É.C., Lazarus, F.: Finding shortest non-trivial cycles in directed graphs on surfaces. In: Proceedings of 26th ACM SoCG, pp. 156–165 (2010)
7. Gibson, M., Kanade, G., Varadarajan, K.: On isolating points using disks. In: Proceedings of 19th ESA, LNCS, vol. 6942, pp. 61–69. Springer, Berlin (2011)
8. King, J., Krohn, E.: Terrain guarding is NP-hard. SIAM J. Comput. **40**(5), 1316–1339 (2011)
9. Knuth, D.E., Raghunathan, A.: The problem of compatible representatives. SIAM J. Discret. Math. **5**(3), 422–427 (1992)
10. Kumar, S., Lai, T.H., Arora, A.: Barrier coverage with wireless sensors. In: Proceedings of 11th MobiCom, pp. 284–298. ACM, London (2005)
11. Kumar, S., Lai, T.H., Arora, A.: Barrier coverage with wireless sensors. Wirel. Netw. **13**(6), 817–834 (2007)
12. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. **11**(2), 329–343 (1982)
13. Mohar, B., Thomassen, C.: Graphs on Surfaces, Johns Hopkins Studies in the Mathematical Sciences. John Hopkins University Press, Baltimore (2001)
14. Mulzer, W., Rote, G.: Minimum-weight triangulation is NP-hard. J. ACM **55**(2), 11:1–11:29 (2008)
15. Penninger, R., Vigan, I.: Point set isolation using unit disks is NP-complete. CoRR abs/1303.2779 (2013)
16. Thomassen, C.: Embeddings of graphs with no short noncontractible cycles. J. Comb. Theory B **48**(2), 155–177 (1990)
17. Tseng, K.C.R.: Resilience of wireless sensor networks. Master's thesis, The University Of British Columbia (Vancouver) (2011)
18. Tseng, K.C.R., Kirkpatrick, D.: On barrier resilience of sensor networks. In: Proceedings of 7th ALGOSENSORS, LNCS, vol. 7111, pp. 130–144. Springer, Berlin (2012)

Journal: **453** Article No.: **9965** TYPESET DISK LE CP Disp.:**2014/12/24** Pages: **21** Layout: **Small-X**