

Middlesex University Research Repository

An open access repository of

Middlesex University research

<http://eprints.mdx.ac.uk>

Cabello, Sergio and Giannopoulos, Panos (2016) The complexity of separating points in the plane. *Algorithmica*, 74 (2) . pp. 643-663. ISSN 0178-4617 [Article]
(doi:10.1007/s00453-014-9965-6)

Final accepted version (with author's formatting)

This version is available at: <https://eprints.mdx.ac.uk/16041/>

Copyright:

Middlesex University Research Repository makes the University's research available electronically.

Copyright and moral rights to this work are retained by the author and/or other copyright owners unless otherwise stated. The work is supplied on the understanding that any use for commercial gain is strictly forbidden. A copy may be downloaded for personal, non-commercial, research or study without prior permission and without charge.

Works, including theses and research projects, may not be reproduced in any format or medium, or extensive quotations taken from them, or their content changed in any way, without first obtaining permission in writing from the copyright holder(s). They may not be sold or exploited commercially in any format or medium without the prior written permission of the copyright holder(s).

Full bibliographic details must be given when referring to, or quoting from full items including the author's name, the title of the work, publication details where relevant (place, publisher, date), pagination, and for theses or dissertations the awarding institution, the degree type awarded, and the date of the award.

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Middlesex University via the following email address:

eprints@mdx.ac.uk

The item will be removed from the repository while any claim is being investigated.

See also repository copyright: re-use policy: <http://eprints.mdx.ac.uk/policies.html#copy>

The Complexity of Separating Points in the Plane

Sergio Cabello · Panos Giannopoulos

Received: 16 December 2012 / Accepted: 16 December 2014
© Springer Science+Business Media New York 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@mf.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

14 1 Introduction

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

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

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

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

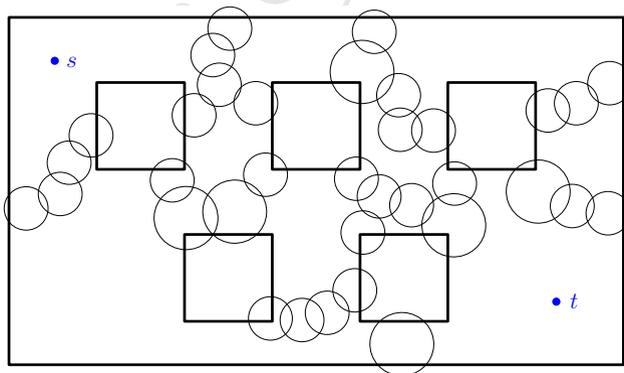


Fig. 1 A possible instance for 2-POINT-SEPARATION with weights: a polygonal domain with five rectangular holes and several disks. The task is to retain the minimum number of disks such that any path connecting s to t inside the domain intersects some retained disk

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

43 *Our results* We provide an algorithm that solves the weighted version of 2-POINT-
44 SEPARATION in $\mathcal{O}(nk + n^2 \log n)$ time, where k is the number of pairs of curves
45 that intersect. The algorithm itself is simple, but its correctness is not obvious.
46 We justify its correctness by considering an appropriate set of closed walks in the
47 intersection graph of the curves and showing that it satisfies the so-called 3-path-
48 condition [16] (see also [13, Chapter 4]). The use of the 3-path-condition for solving
49 2-POINT-SEPARATION is surprising, but it makes the connection to topology clear.
50 In fact, our approach can be interpreted alternatively as searching for a shortest
51 non-zero-homologous cycle in $\mathbb{R}^2 \setminus \{s, t\}$ (with coefficients in \mathbb{Z}_2). This approach
52 works when the optimal solution is given by at least three curves. We take care
53 for the case when the optimal solution is attained by two curves separately by
54 brute-force.

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

- 57 – horizontal and vertical segments;
- 58 – unit circles.

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

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

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

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

83 **2 Algorithm for 2-Point-Separation**

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

88 **2.1 Preliminaries**

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

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

102 *Intersection graph* The set C of input curves defines the intersection graph $\mathbb{G} =$
 103 $\mathbb{G}(C) = (C, \{cc' \mid c \cap c' \neq \emptyset\})$; see Fig. 2. Note that \mathbb{G} has k edges. To each edge
 104 cc' of \mathbb{G} we attach the weight (abstract length) $w(c) + w(c')$. Any distance in \mathbb{G} will
 105 refer to these edge weights. For any walk π in \mathbb{G} we use $\text{len}_{\mathbb{G}}(\pi)$ for its length, that
 106 is, the sum of the weights on its edges counted with multiplicity, and $C(\pi) = V(\pi)$
 107 for the set of curves that appear as vertices in the walk π .

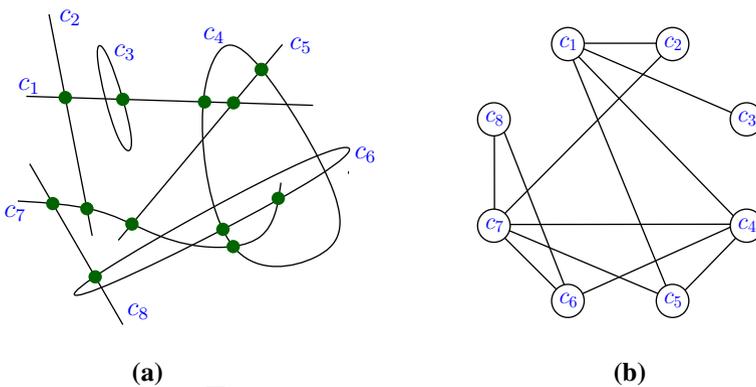


Fig. 2 a A set of curves C with the fixed intersection points $x_{c,c'}$. b The corresponding intersection graph \mathbb{G}

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

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

119 π -paths Consider a walk $\pi = c_0c_1 \cdots c_t$ in \mathbb{G} . Let γ be a path in \mathbb{R}^2 . We say that
 120 γ is a π -path if there are paths $\gamma_1, \dots, \gamma_{t-1}$ such that: the path γ_i is contained in
 121 c_i ($i = 1, \dots, t - 1$), the path γ_i goes from x_{c_{i-1},c_i} to $x_{c_i,c_{i+1}}$ ($i = 1, \dots, t - 1$),
 122 and the concatenation of $\gamma_1, \dots, \gamma_{t-1}$ gives γ . The intuition is that γ starts at x_{c_0,c_1} ,
 123 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
 124 x_{c_{t-1},c_t} by following c_{t-1} . See Fig. 3a for an example.

125 If the walk $\pi = c_0c_1 \cdots c_t$ is closed, which means that $c_t = c_0$, then a closed path
 126 γ is a *closed π -path* if there are paths $\gamma_1, \dots, \gamma_t$ such that: the path γ_i is contained in
 127 c_i ($i = 1, \dots, t$), the path γ_i goes from x_{c_{i-1},c_i} to $x_{c_i,c_{i+1}}$ ($i = 1, \dots, t$ and $c_{t+1} = c_1$),
 128 and the concatenation of $\gamma_1, \dots, \gamma_t$ gives γ . See Fig. 3b–c for an example. If γ is a
 129 π -path or a closed π -path, then $\gamma \subset \bigcup C(\pi)$. Even if π is a cycle, which is a closed
 130 walk without repeated vertices, a closed π -path may have self-intersections.

131 There may be different π -paths. Given a walk $\pi = c_0c_1 \cdots c_t$ in \mathbb{G} we can construct
 132 a π -path in linear time by concatenating $c_j[x_{c_{j-1},c_j} \rightarrow x_{c_j,c_{j+1}}]$ for $j = 1, \dots, t - 1$.
 133 If π is a closed walk with $c_0 = c_t$, we can obtain a closed π -path by closing it
 134 with $c_0[x_{c_{t-1},c_0} \rightarrow x_{c_0,c_1}]$. When the input family C is a family of pseudosegments,
 135 there is a unique π -path for each walk π and a unique closed π -path for each closed
 136 walk π .

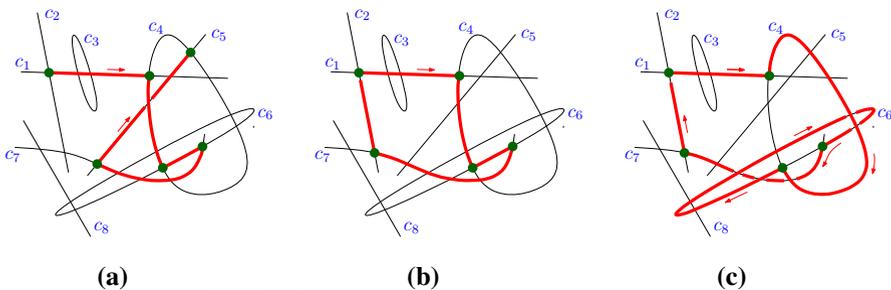


Fig. 3 Some paths in the example of Fig. 2, using the fixed intersection points marked in Fig. 2. In (a) there is a π -path for the walk $\pi = c_2c_1c_4c_6c_7c_5c_4$. In (b) and (c) there are two different closed π -paths for the closed walk $\pi = c_2c_1c_4c_6c_7c_2$

137 We will mainly use closed ($\text{walk}(r, e)$)-paths, where r is a curve of C and $e \in$
 138 $E(\mathbb{G}) \setminus E(T_r)$. Thus, we introduce the notation $\gamma(r, e)$ to denote a closed ($\text{walk}(r, e)$)-
 139 path; if there are several such paths, it denotes an arbitrary one.

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

144 Let π be a walk in \mathbb{G} and let γ be some π -path. We define $N(\pi) = N(\gamma)$. Thus,
 145 $N(\cdot)$ is defined for paths in the plane and for walks in \mathbb{G} . A priori, the value $N(\pi)$
 146 depends on the choice of the π -path γ . However, as we will see in Lemma 3, when
 147 no curve of C alone separates s and t , the value $N(\pi)$ is independent of the choice of
 148 γ . Our first step in the algorithm will be to remove from C any curve that separates s
 149 and t .

150 In this paper,

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

152 Because of our assumptions on general position, for any walk $c_0c_1 \cdots c_t$ and any i ,
 153 $1 < i \leq t$, we have

154
$$N(c_0c_1 \cdots c_t) = N(c_0c_1 \cdots c_{i-1}c_i) + N(c_{i-1}c_i \cdots c_t).$$

155 **2.2 The Algorithm**

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

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

161 Next we compute the set

162
$$P = \{(r, e) \in C \times E(\mathbb{G}) \mid e \in E(\mathbb{G}) \setminus E(T_r) \text{ and } N(\text{walk}(r, e)) = 1\}.$$

163 Then we choose

164
$$(r^*, e^*) \in \arg \min_{(r,e) \in P} \text{len}_{\mathbb{G}}(\text{walk}(r, e)),$$

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

167 If both $C_{\leq 2}$ and $C_{> 2}$ are defined, we return the lightest of them. If only one among
 168 $C_{\leq 2}$ and $C_{> 2}$ is defined, we return the only one that is defined. If both $C_{\leq 2}$ and $C_{> 2}$
 169 are undefined, we return “ C does not separate s and t ”. This finishes the description
 170 of the algorithm. We will refer to this algorithm as ALGORITHM-2PS.

171 2.3 Time Complexity of the Algorithm

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

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

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

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

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

$$182 \quad E_r = \{e \in E(\mathbb{G}) \mid (r, e) \in P\}$$

$$183 \quad = \{e \in E(\mathbb{G}) \mid e \in E(\mathbb{G}) \setminus E(T_r) \text{ and } N(\text{walk}(r, e)) = 1\}.$$

185 Note that

$$186 \quad P = \bigcup_{r \in C} \{r\} \times E_r,$$

187 and therefore

$$188 \quad \min_{(r,e) \in P} \text{len}_{\mathbb{G}}(\text{walk}(r, e)) = \min_{r \in C} \min_{e \in E_r} \text{len}_{\mathbb{G}}(\text{walk}(r, e)).$$

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

$$190 \quad \min_{e \in E_r} \text{len}_{\mathbb{G}}(\text{walk}(r, e)).$$

191 We shall see that, for each fixed $r \in C$, such value can be computed in $\mathcal{O}(k + n \log n)$
 192 time. It then follows that (r^*, e^*) can be found in $|C| \times \mathcal{O}(k + n \log n) = \mathcal{O}(nk +$
 193 $n^2 \log n)$ time.

194 For the rest of the proof, let us fix a curve $r \in C$. Computing the shortest-path
 195 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
 196 idea now is simple: for each edge $cc' \in E(\mathbb{G})$, we can obtain $N(\text{walk}(r, cc'))$ and
 197 $\text{len}_{\mathbb{G}}(\text{walk}(r, cc'))$ in constant time using information stored at c and c' . (The details
 198 below become a little cumbersome.)

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

201 The values $N_r[c]$, $c \in C$, can be computed in $\mathcal{O}(n)$ time using a BFS traversal of
 202 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
 203 other curve c , if $p_r(c)$ is the parent of c in T_r , we can compute $N_r[c]$ from $N_r[p_r(c)]$
 204 in $\mathcal{O}(1)$ time using that

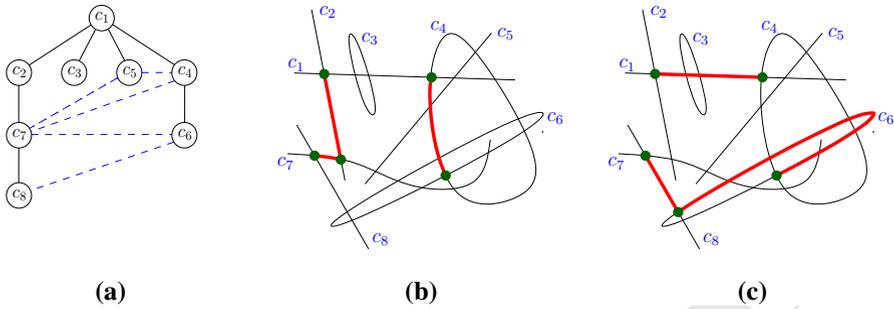


Fig. 4 **a** Tree T_{c_1} for the scenario of Fig. 2 assuming curves of unit weight. In this case $A_{c_1}[c_8] = c_2$ and $A_{c_1}[c_6] = c_4$. **b** Possible $(T_{c_1}[c_8])$ -path and $(T_{c_1}[c_6])$ -path used to compute $N_{c_1}[c_8]$ and $N_{c_1}[c_6]$. **c** Possible $(c_7c_8c_6c_4)$ -path and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

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

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

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

See Fig. 4b–c. Therefore, each $N(\text{walk}(r, cc'))$ 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 $\text{walk}(r, e)$ 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 $\text{walk}(r, cc')$ can then be recovered using

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

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

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

□

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.*

230 *Proof* In this case, a shortest path tree T_r can be computed in $\mathcal{O}(|E(\mathbb{G})| + |V(\mathbb{G})|) =$
 231 $\mathcal{O}(k + n)$ time because the edge weights of \mathbb{G} are 0, 1, or 2. Using the approach
 232 described in the proof of Lemma 1 we spend $\mathcal{O}(k + n)$ per root $r \in C$, and thus spend
 233 $\mathcal{O}(nk + n^2)$ in total. □

234 3 Correctness of the Algorithm for 2-Point-Separation

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

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

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

241 **Lemma 3** *Let π be a walk in \mathbb{G} and let γ and γ' be two π -paths. Then $N(\gamma) = N(\gamma')$.
 242 Similarly, if π is a closed walk in \mathbb{G} and γ and γ' are two closed π -paths, then
 243 $N(\gamma) = N(\gamma')$.*

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

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

249 Let $\gamma_1, \dots, \gamma_t$ be the pieces of γ that certify that γ is a closed π -curve. Similarly, let
 250 $\gamma'_1, \dots, \gamma'_t$ be the pieces of γ' that certify that γ' is a closed π -curve. For $i = 1, \dots, t$,
 251 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
 252 $c_1 = c_{t+1}$) and are contained in c_i . Therefore $N(\gamma_i) + N(\gamma'_i) = 0$ for $i = 1, \dots, t$,
 253 which implies $N(\gamma_i) = N(\gamma'_i)$. We thus have

$$254 \quad N(\gamma) = \sum_{i=1}^t N(\gamma_i) = \sum_{i=1}^t N(\gamma'_i) = N(\gamma').$$

255 □

256 3.1 3-Path-Condition

257 Consider the set of closed walks

$$258 \quad \Pi(C) = \{\pi \mid \pi \text{ is a closed walk in } \mathbb{G}(C); N(\pi) = 1\}.$$

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

261 We next show the following property, known as 3-path-condition. It implies that
 262 from the 3 “natural” closed walks defined by 3 walks with common endvertices, either
 263 2 or none belong to Π .

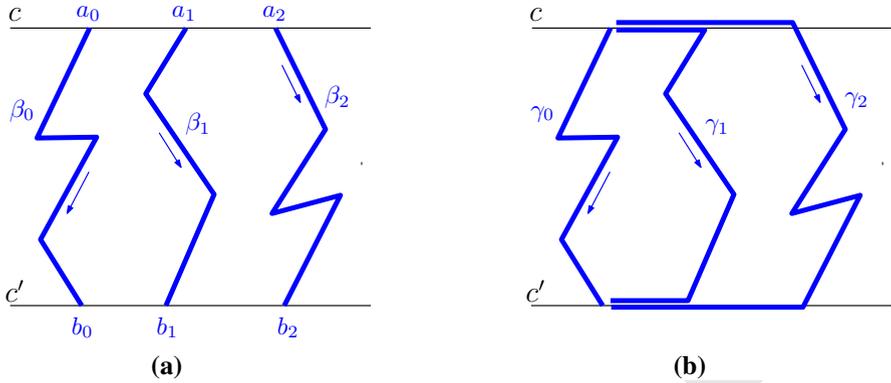


Fig. 5 Notation in the Proof of Lemma 4. (Parts of γ_1 and γ_2 lie on $c \cup c'$. We draw them outside because of the common part)

264 **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
 265 closed walk obtained by concatenating α_{i-1} and the reverse of α_{i+1} , where indices
 266 are modulo 3. Then $N(\pi_1) + N(\pi_2) + N(\pi_3) = 0$.

267 *Proof* This is basically a matter of parity. For $i = 0, 1, 2$, let β_i be any α_i -path, let
 268 $a_i \in c$ be its endpoint on c and let $b_i \in c'$ be its endpoint on c' . See Fig. 5a. Note that
 269 the paths $\beta_0, \beta_1, \beta_2$ start on c and finish on c' , but they have different endpoints. To
 270 handle this, for $i = 0, 1, 2$, we define γ_i to be the path obtained by the concatenation
 271 of $c[a_0 \rightarrow a_i]$, β_i , and $c'[b_i \rightarrow b_0]$. Now the paths $\gamma_0, \gamma_1, \gamma_2$ start at a_0 and finish at
 272 b_0 . See Fig. 5b. For $i = 0, 1, 2$, let δ_i be the closed π_i -path defined by concatenating
 273 β_{i-1} , $c'[b_{i-1} \rightarrow b_{i+1}]$, the reversal of β_{i+1} , and $c[a_{i+1} \rightarrow a_{i-1}]$, where indices are
 274 taken modulo 3. Because of Lemma 3 we have $N(\pi_i) = N(\delta_i)$ for $i = 0, 1, 2$.

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

276
$$N(\delta_i) = N(\gamma_{i-1}) + N(\gamma_{i+1}).$$

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

280
$$N(c[a_0 \rightarrow a_{i+1}]) + N(c[a_{i+1} \rightarrow a_{i-1}]) + N(c[a_{i-1} \rightarrow a_0]) = 0.$$

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

283
$$N(c[a_{i+1} \rightarrow a_{i-1}]) = N(c[a_0 \rightarrow a_{i+1}]) + N(c[a_0 \rightarrow a_{i-1}]).$$

284 Similarly, for c' we have

285
$$N(c'[b_{i+1} \rightarrow b_{i-1}]) = N(c'[b_0 \rightarrow b_{i+1}]) + N(c'[b_0 \rightarrow b_{i-1}]).$$

286 Then we have

$$\begin{aligned}
 287 \quad N(\delta_i) &= N(\beta_{i-1}) + N(c'[b_{i-1} \rightarrow b_{i+1}]) + N(\beta_{i+1}) + N(c[a_{i+1} \rightarrow a_{i-1}]) \\
 288 \quad &= N(\beta_{i-1}) + N(c'[b_0 \rightarrow b_{i+1}]) + N(c'[b_0 \rightarrow b_{i-1}]) \\
 289 \quad &\quad + N(\beta_{i+1}) + N(c[a_0 \rightarrow a_{i+1}]) + N(c[a_0 \rightarrow a_{i-1}]) \\
 290 \quad &= N(c[a_0 \rightarrow a_{i-1}]) + N(\beta_{i-1}) + N(c'[b_0 \rightarrow b_{i-1}]) \\
 291 \quad &\quad + N(c[a_0 \rightarrow a_{i+1}]) + N(\beta_{i+1}) + N(c'[b_0 \rightarrow b_{i+1}]) \\
 292 \quad &= N(\gamma_{i-1}) + N(\gamma_{i+1}). \\
 293
 \end{aligned}$$

294 It follows that, using indices modulo 3,

$$\begin{aligned}
 295 \quad \sum_{i=0}^2 N(\pi_i) &= \sum_{i=0}^2 N(\delta_i) = \sum_{i=0}^2 (N(\gamma_{i-1}) + N(\gamma_{i+1})) = 0. \\
 296
 \end{aligned}$$

297 □

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

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

305 *Proof* We first show that each shortest closed walk of Π is a cycle. This is a conse-
 306 quence of Lemma 4. Assume for the sake of a contradiction that some shortest closed
 307 walk π of Π repeats a vertex c . Then we apply Lemma 4 to two non-trivial subwalks
 308 π' and π'' of π from c to c and the trivial walk with only vertex c . (Lemma 4 does
 309 not require that $c \neq c'$.) It follows that both π' and π'' are shorter than π and either
 310 $N(\pi') = 1$ or $N(\pi'') = 1$, so π could not be shortest in Π . We conclude that each
 311 shortest closed walk of Π is a cycle.

312 Consider the set of closed walks

$$313 \quad \Pi' = \{\text{walk}(r, e) \mid r \in C, e \in E(\mathbb{G}) \setminus E(T_r), N(\text{walk}(r, e)) = 1\} \subseteq \Pi.$$

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

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

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

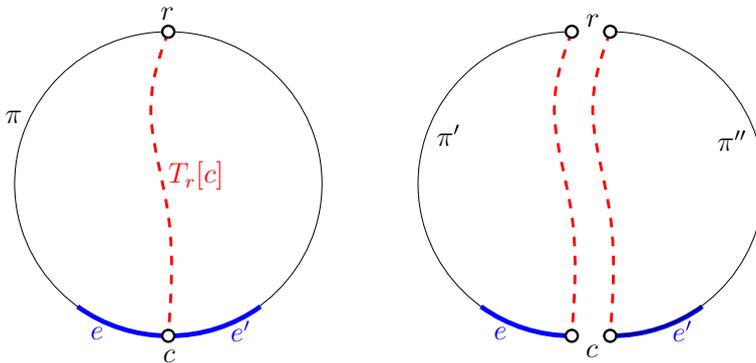


Fig. 6 Notation in the Proof of Lemma 5

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

328
$$N(\pi) + N(\pi') + N(\pi'') = 0.$$

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

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

339 **3.2 Feasibility**

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

342 **Lemma 6** Assume that Π is nonempty and let π be any cycle in Π . The set of curves
 343 $C(\pi)$ separates s and t .

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

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

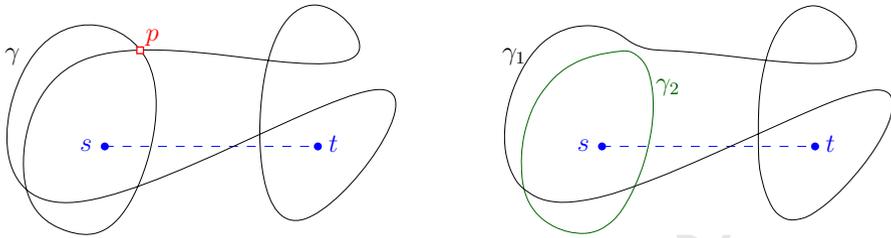


Fig. 7 Uncrossing argument in the Proof of Lemma 6

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

351
$$1 = N(\gamma) = N(\gamma_1) + N(\gamma_2)$$

352 because the paths γ_1 and γ_2 form a disjoint partition of γ . Therefore, for $i = 1$ or
 353 $i = 2$, the path γ_i has $N(\gamma_i)$ odd, is part of γ and thus contained in $C(\pi)$, and has
 354 fewer self-crossings than γ . This would contradict the choice of γ . We conclude that
 355 γ must be simple.

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

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

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

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

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

373 Let δ_i be a path contained in c_i such that the concatenation of $\delta_0, \delta_1, \dots, \delta_{t-1}$ is
 374 δ . For $i = 0, \dots, t - 1$, let a_i be the start point of δ_i and let δ'_i be the path obtained
 375 by the concatenation of $c_i[x_{c_{i-1}, c_i} \rightarrow a_i]$, δ_i , and $c_{i+1}[a_{i+1} \rightarrow x_{c_i, c_{i+1}}]$. Thus, for
 376 $i = 0, \dots, t - 1$, the path δ'_i starts at x_{c_{i-1}, c_i} , finishes at $x_{c_i, c_{i+1}}$, and is contained in
 377 $c_i \cup c_{i+1}$. Finally, let δ' be the concatenation of $\delta'_0, \delta'_1, \dots, \delta'_{t-1}$. Since δ' is obtained
 378 from δ by inserting the paths $c_i[x_{c_{i-1}, c_i} \rightarrow a_i]$ twice, once in each direction, we have
 379 $N(\delta') = N(\delta) = 1$. See Fig. 8.

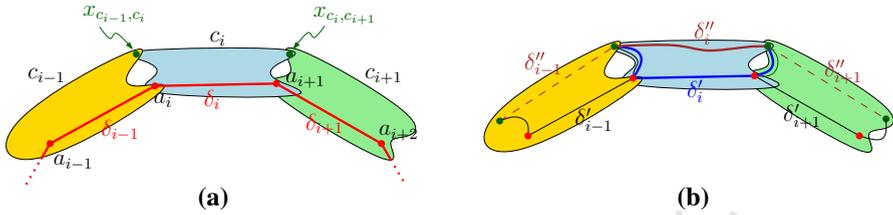


Fig. 8 **a** Notation and **b** the paths δ'_i, δ''_i constructed in the Proof of Lemma 7

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

384
$$N(\delta'') = \sum_i N(\delta''_i) = \sum_i N(\delta'_i) = N(\delta') = 1.$$

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

386 **3.3 Main Result**

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

389 **Theorem 1** *The weighted version of 2-POINTS-SEPARATION can be solved in $\mathcal{O}(nk +$
 390 $n^2 \log n)$ time, where n is the number of input curves and k is the number of pairs of
 391 curves that intersect.*

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

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

403 Now note that $\Pi(\tilde{C}) \subseteq \Pi(C)$ because $\tilde{C} \subseteq C$, which implies that $\tilde{\tau}$ is a cycle of
 404 $\Pi(C)$. Since τ^* is a shortest cycle in $\Pi(C)$ due to Lemma 5, we have $\text{len}_{\mathbb{G}}(\tau^*) \leq$
 405 $\text{len}_{\mathbb{G}}(\tilde{\tau})$. For any cycle π of \mathbb{G} we have $\text{len}_{\mathbb{G}}(\pi) = 2w(C(\pi))$ because of the choice
 406 of the edge-weights in \mathbb{G} . This implies that

407
$$w(C_{>2}) = \frac{1}{2} \text{len}_{\mathbb{G}}(\tau^*) \leq \frac{1}{2} \text{len}_{\mathbb{G}}(\tilde{\tau}) = w(C(\tilde{\tau})) = w(\tilde{C}).$$

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

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

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

414 4 Hardness of Point-Separation

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

418 Consider a 3-CNF formula with a set \mathcal{C} of clauses over a set X of boolean variables.
 419 Its *formula graph* is defined as the bipartite graph on $\mathcal{C} \cup X$ that has an edge connecting
 420 $x \in X$ to $C \in \mathcal{C}$ if and only if C contains literal x or $\neg x$. A *3-legged representation* of
 421 the formula graph is a plane, rectilinear drawing where the variables and clauses are
 422 drawn as axis-aligned rectangles, the variables are aligned horizontally, and the edges
 423 are vertical segments; see the example in Fig. 9. PLANAR-3-SAT is the restriction of
 424 3-SAT to formulae whose formula graph is planar and has a 3-legged representation.
 425 PLANAR-3-SAT is NP-complete [12], and it remains so when the 3-legged representation
 426 is given as part of the input. Several NP-hardness proofs of geometric problems
 427 have used PLANAR-3-SAT; see for example [1, 5, 8, 9], and [14].

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

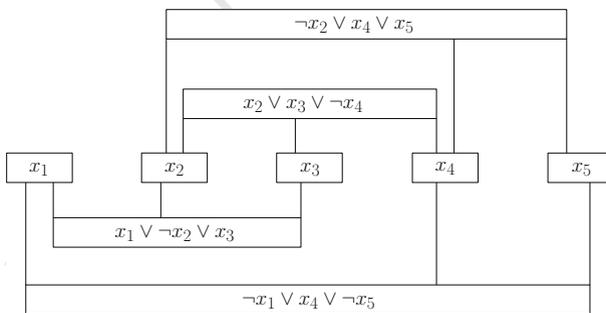
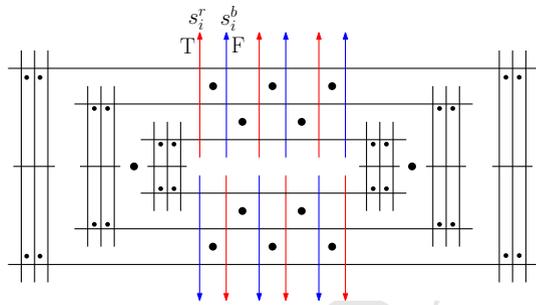


Fig. 9 Rectilinear representation of planar 3-SAT

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



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

438 4.1 Horizontal and Vertical Segments

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

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

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

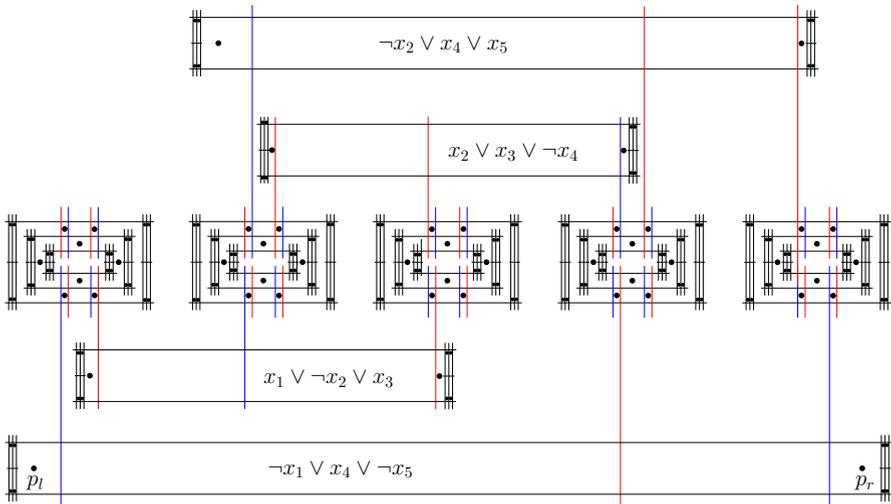


Fig. 11 The construction with segments for the example of Fig. 9

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

484 **Theorem 2** POINTS-SEPARATION is NP-hard for families of vertical and horizontal
 485 segments.

486 4.2 Unit Circles

487 *Variables* For unit circles we use the variable gadget displayed in Fig. 12. It contains
 488 $3\ell - 1$ disjoint triples of black circles at the center, which form its backbone. The

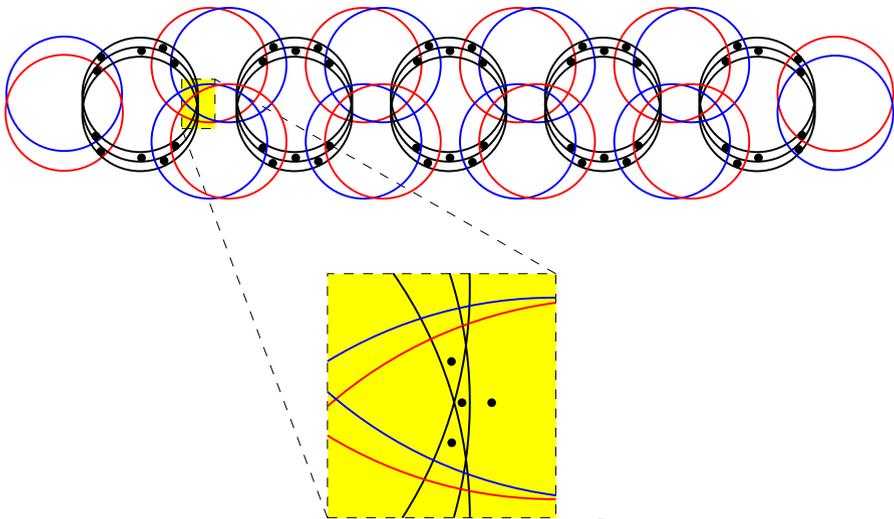


Fig. 12 Variable gadget for POINTS-SEPARATION with unit circles (top). The extra points that ensure that all black circles are part of any feasible solution are shown in the zoomed-in area (bottom)

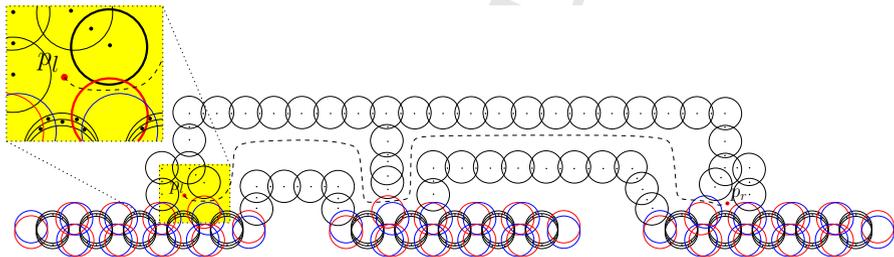
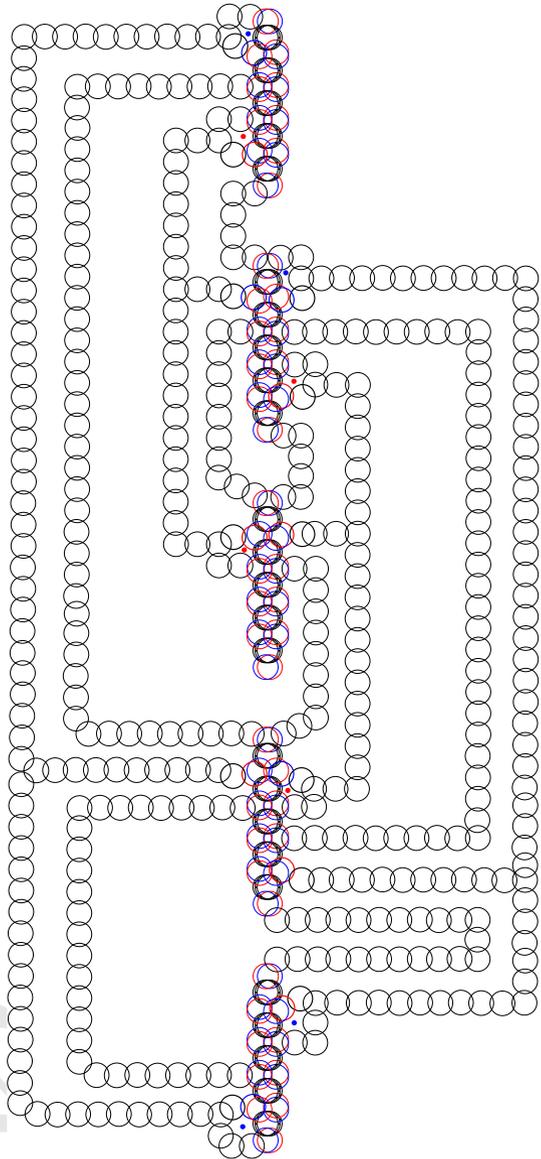


Fig. 13 The clause $(x_2 \vee x_3 \vee \neg x_4)$ with unit circles. The corridor is marked by a dashed path. The zoomed-in area (top left) shows a red circle intersecting a black circle of the corridor (both fat) and disconnecting the corridor

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

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



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

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

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

524 *Correctness* Every variable gadget has $3(3\ell - 1)$ black circles, $6\ell - 2$ red circles, and
 525 $6\ell - 2$ blue circles. It is clear that for each clause gadget the number of horizontally
 526 placed black circles is some quadratic polynomial on n and ℓ and the number of
 527 vertically placed black circles is some linear function on m . Let $b(I)$ be the total
 528 number of black circles in $I(\Phi)$.

529 Constructing a feasible solution to $I(\Phi)$ with $b(I) + (6\ell - 2) \cdot n$ circles from a truth
 530 assignment for Φ is immediate. An argument similar to the one used for segments
 531 shows that any feasible solution with $b(I) + (6\ell - 2) \cdot n$ circles contains all black
 532 circles and, in each variable gadget, either all red circles or all blue circles. The choice
 533 of red or blue circles made in the variable gadget corresponds to a truth assignment of
 534 the variables, and such assignment satisfies the clauses because, in each clause gadget,
 535 the points p^l and p^r are separated. Therefore a feasible solution containing exactly
 536 $b(I) + (6\ell - 2) \cdot n$ circles exists if and only if Φ is satisfiable.

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

538 5 Open Questions

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

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

544 References

- 545 1. Agarwal, P.K., Suri, S.: Surface approximation and geometric partitions. *SIAM J. Comput.* **27**(4),
 546 1016–1035 (1998)
- 547 2. Alt, H., Cabello, S., Giannopoulos, P., Knauer, C.: Minimum cell connection and separation in line
 548 segment arrangements. *CoRR* [abs/1104.4618](https://arxiv.org/abs/1104.4618) (2011)

- 549 3. Alt, H., Cabello, S., Giannopoulos, P., Knauer, C.: On some connection problems in straight-line
550 segment arrangements. In: Abstracts of the 27th EuroCG, pp. 27–30 (2011)
- 551 4. Bereg, S., Kirkpatrick, D.G.: Approximating barrier resilience in wireless sensor networks. In: Pro-
552 ceedings of 5th ALGOSENSORS, LNCS, vol. 5804, pp. 29–40. Springer, Berlin (2009)
- 553 5. Cabello, S., Demaine, E.D., Rote, G.: Planar embeddings of graphs with specified edge lengths. *J.*
554 *Graph Algorithms Appl.* **11**(1), 259–276 (2007)
- 555 6. Cabello, S., de Verdière, É.C., Lazarus, F.: Finding shortest non-trivial cycles in directed graphs on
556 surfaces. In: Proceedings of 26th ACM SoCG, pp. 156–165 (2010)
- 557 7. Gibson, M., Kanade, G., Varadarajan, K.: On isolating points using disks. In: Proceedings of 19th ESA,
558 LNCS, vol. 6942, pp. 61–69. Springer, Berlin (2011)
- 559 8. King, J., Krohn, E.: Terrain guarding is NP-hard. *SIAM J. Comput.* **40**(5), 1316–1339 (2011)
- 560 9. Knuth, D.E., Raghunathan, A.: The problem of compatible representatives. *SIAM J. Discret. Math.*
561 **5**(3), 422–427 (1992)
- 562 10. Kumar, S., Lai, T.H., Arora, A.: Barrier coverage with wireless sensors. In: Proceedings of 11th
563 MobiCom, pp. 284–298. ACM, London (2005)
- 564 11. Kumar, S., Lai, T.H., Arora, A.: Barrier coverage with wireless sensors. *Wirel. Netw.* **13**(6), 817–834
565 (2007)
- 566 12. Lichtenstein, D.: Planar formulae and their uses. *SIAM J. Comput.* **11**(2), 329–343 (1982)
- 567 13. Mohar, B., Thomassen, C.: *Graphs on Surfaces*, Johns Hopkins Studies in the Mathematical Sciences.
568 John Hopkins University Press, Baltimore (2001)
- 569 14. Mulzer, W., Rote, G.: Minimum-weight triangulation is NP-hard. *J. ACM* **55**(2), 11:1–11:29 (2008)
- 570 15. Penninger, R., Vigan, I.: Point set isolation using unit disks is NP-complete. CoRR [abs/1303.2779](https://arxiv.org/abs/1303.2779)
571 (2013)
- 572 16. Thomassen, C.: Embeddings of graphs with no short noncontractible cycles. *J. Comb. Theory B* **48**(2),
573 155–177 (1990)
- 574 17. Tseng, K.C.R.: Resilience of wireless sensor networks. Master's thesis, The University Of British
575 Columbia (Vancouver) (2011)
- 576 18. Tseng, K.C.R., Kirkpatrick, D.: On barrier resilience of sensor networks. In: Proceedings of 7th ALGO-
577 SENSORS, LNCS, vol. 7111, pp. 130–144. Springer, Berlin (2012)