Finding a Shortest Curve That Separates Few Objects from Many

Therese Biedl
University of Waterloo, Canada

Éric Colin de Verdière
LIGM, CNRS, Univ Gustave Eiffel, F-77454 Marne-la-Vallée, France

Fabrizio Frati
Roma Tre University, Rome, Italy

Anna Lubiw
University of Waterloo, Canada

Günter Rote
Institut für Informatik, Freie Universität Berlin, Germany

Abstract

We present a fixed-parameter tractable (FPT) algorithm to find a shortest curve that encloses a set of k required objects in the plane while paying a penalty for enclosing unwanted objects.

The input is a set of interior-disjoint simple polygons in the plane, where k of the polygons are *required* to be enclosed and the remaining *optional* polygons have non-negative penalties. The goal is to find a closed curve that is disjoint from the polygon interiors and encloses the k required polygons, while minimizing the length of the curve plus the penalties of the enclosed optional polygons. If the penalties are high, the output is a shortest curve that separates the required polygons from the others. The problem is NP-hard if k is not fixed, even in very special cases. The runtime of our algorithm is $O(3^k n^3)$, where n is the number of vertices of the input polygons.

We extend the result to a graph version of the problem where the input is a connected plane graph with positive edge weights. There are k required faces; the remaining faces are optional and have non-negative penalties. The goal is to find a closed walk in the graph that encloses the k required faces, while minimizing the weight of the walk plus the penalties of the enclosed optional faces. We also consider an inverted version of the problem where the required objects must lie outside the curve. Our algorithms solve some other well-studied problems, such as geometric knapsack.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Enclosure, curve, separation, weakly simple polygon, Euler tour

Digital Object Identifier 10.4230/LIPIcs.SoCG.2025.18

Related Version *Full Version*: `https://arxiv.org/abs/2504.03558v1` [5]

Funding *Therese Biedl*: Natural Science and Engineering Research Council of Canada.
Fabrizio Frati: Supported by the European Union, Next Generation EU, Mission 4, Component 1, CUP J53D23007130006 PRIN proj. 2022ME9Z78 "NextGRAAL: Next-generation algorithms for constrained GRAph visuALization".
Anna Lubiw: Supported by the Natural Sciences and Engineering Research Council of Canada.

Acknowledgements This work was initiated at the Dagstuhl seminar 24072 "Triangulations in Geometry and Topology" in February 2024. We thank the organizers, Maike Buchin, Jean Cardinal, Arnaud de Mesmay, and Jonathan Spreer, as well as all participants, for the stimulating atmosphere. We also thank Alexander Wolff for significant contributions, Saul Schleimer for useful discussions, and the anonymous reviewers for their constructive feedback.

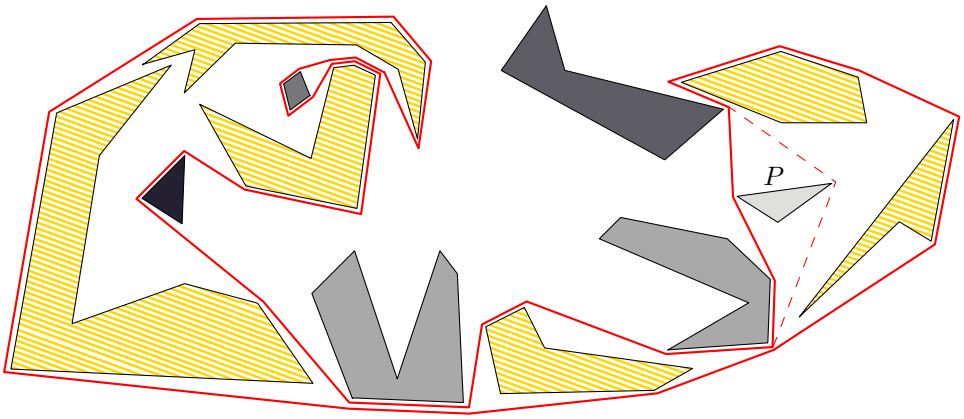


Figure 1 The Geometric-Enclosure-with-Penalties problem. The weakly simple polygon W (in red) encloses the required polygons R (yellow hatched) while paying a penalty for enclosing any of the optional polygons O (in gray, darker for larger penalties). Overlapping paths are slightly displaced for visibility. Changing W via the dashed path to put polygon P outside would be preferable if the increase in length is less than the penalty of P.

1 Introduction

We investigate the separation problem of finding a shortest curve that encloses a subset of objects while excluding other objects. A very basic setting is for points in the plane: given n points in the plane and a subset of size k, find a minimum-perimeter polygon containing the specified k points and excluding the other $n - k$ points. This problem is NP-hard when k may be large, as proved by Eades and Rappaport [11] for the case $k = n/2$ via a simple reduction from the Travelling Salesman Problem.

As a special case of our main result, we give the first algorithm for this problem that is fixed-parameter tractable (FPT) in k. Our result is far more general and applies in two settings, a geometric setting and a graph-theoretic setting.

Geometric-Enclosure-with-Penalties. We generalize from point objects to objects that are interior-disjoint simple polygons in the plane, and we consider a *weighted* form of exclusion.

Input. The input is a set of simple interior-disjoint polygons partitioned into a set R of k **required polygons** and the remaining set O of **optional polygons**. Each optional polygon $P \in O$ comes with a non-negative **penalty** π_P where we allow $\pi_P = +\infty$.

Output. The goal is to find a weakly simple polygon W that does not intersect the interior of any input polygon and encloses all polygons of R while minimizing the **cost** $c(W)$, which is defined to be the Euclidean length of W plus the penalties of the polygons of O that are inside W. See Figure 1 for an example. A polygon with penalty $+\infty$ must be excluded. A polygon with penalty 0 may be included or excluded without making a difference, so it only acts as an obstacle to the solution curve. As Figure 1 illustrates, the problem would be ill-defined if we required the solution curve W to be a simple polygon. The natural condition is that W should be a **weakly simple polygon**, whose boundary may touch or overlap itself but not cross itself. We give a precise definition in Section 2.1. An important property is that a weakly simple polygon encloses a well-defined region. Our first main result is:

▶ **Theorem 1.** Geometric-Enclosure-with-Penalties *for k required polygons can be solved in $O(3^k n^3)$ time and $O(2^k n^2)$ space, if the input polygons have n vertices in total.*

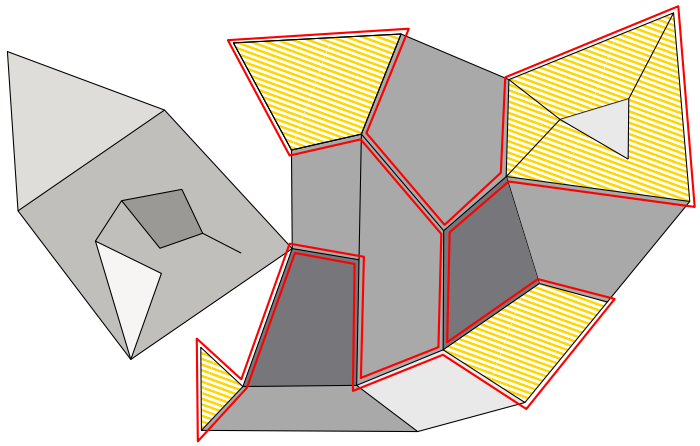


Figure 2 The Graph-Enclosure-with-Penalties problem. The colors have the same meaning as in Figure 1. A weakly simple closed walk W which is a solution for the instance is red (bold). The edge weights, which are not shown, are unrelated to the Euclidean lengths.

If all objects are points, this can be handled by approximating each point by a small triangle. An exact solution for an arbitrary mix of point and polygon objects appears in [5, Appendix I].

Graph-Enclosure-with-Penalties. In this setting, the objects are faces of a plane graph.

Input. The input is a simple connected plane graph G and positive edge weights. The bounded faces of G are partitioned into a set R of k **required faces** and the remaining set O of **optional faces**. Each optional face F has a penalty π_F from $\mathbb{R}_{\geq 0} \cup \{+\infty\}$.

Output. The goal is to find a weakly simple closed walk W in G such that faces of R are inside W while minimizing the cost $c(W)$ which is defined to be the sum of the weights of the edges of W plus the penalties of the faces of O that are inside W. See Figure 2 for an example. Intuitively, a **weakly simple** closed walk is one without crossings; we give a more precise definition in the full version of this paper [5, Appendix A]. For a weakly simple closed walk, the notions of inside and outside are well-defined. Our second main result is:

▶ **Theorem 2.** Graph-Enclosure-with-Penalties *can be solved in* $O(3^k n^3)$ *time and* $O(2^k n^2)$ *space, where* k *is the number of required faces and* n *is the number of vertices of* G.

A common framework. Although the two settings described above seem different, we resolve them into a common geometric framework, which we call Enclosure-with-Penalties. Our algorithm applies to this general problem. The basic idea is to transform the graph problem into a geometric problem by taking a straight-line embedding of the graph. The bounded faces of the graph become polygons slightly more general than simple polygons. We also consider the outer face as an unbounded polygon. Then the "free space" between the polygons consists only of the graph edges. This gives us a geometric problem, albeit with arbitrary positive edge weights defined on edges that have a polygon on each side. In Section 3 we define the Enclosure-with-Penalties problem by generalizing the Geometric-Enclosure-with-Penalties problem to include these instances.

We remark that the resulting algorithm for Theorem 2 makes essential use of the straight-line embedding of the input graph. In particular, the subproblems that we solve depend on the embedding. This imposition of geometry seems artificial, but oddly enough, we do not know how to formulate our algorithm in a purely combinatorial setting.

Our approach. We use dynamic programming (Section 4) to build a polygon W that is locally correct – we use segments that do not intersect the interior of any object and we account for required objects and tally the penalties as we add triangles to W. We will prove that the cost computed by the algorithm is correct, but this is tricky because W itself will not necessarily be weakly simple. "Inside" is no longer well-defined. Instead, we use winding numbers to give a measure of the cost of W that matches the cost computed by the algorithm.

In Section 5 we give an algorithm to *uncross* W to a weakly simple polygon without increasing its cost, which provides our final output. Correctness is proved in Section 6.

The run-time for the Uncrossing Algorithm is dominated by the run-time for the dynamic program. To obtain our claimed run-time we speed up the dynamic program in Section 7.

Lower bounds. To complement our algorithms we prove that, under the Exponential Time Hypothesis (ETH), the Geometric- and Graph-Enclosure-with-Penalties problems cannot be solved in $2^{o(k)} \cdot n^{O(1)}$ time, implying that the linear dependence on k in the exponent of the running time of our algorithms is the best possible assuming ETH. The proof is a reduction from unweighted Planar Steiner Tree, which admits a lower bound by a result by Marx, Pilipczuk, and Pilipczuk [15, Theorem 1.2]. See [5, Appendix K].

Swapping the inside with the outside. We extend our algorithm to an **inverted** version of the Enclosure-with-Penalties problem where the required objects have to be *outside* W, and the objective is to minimize the length of W plus the penalties of the polygons of O that are *outside* W. The runtime remains the same, see Section 8. This algorithm provides a new faster solution to the geometric knapsack problem discussed below.

Negative penalties. We can allow some number ℓ of objects with negative penalties (rewards); in this case, the runtime is increased by a factor of 3^ℓ. See [5, Appendix J].

1.1 Related work

Cut problems and separator problems in graphs have a long history, and separation problems in geometric settings are a natural and well studied counterpart.

Geometric knapsack problem. Geometric separation problems were first explored by Eades and Rappaport [11] (as discussed above) and by Arkin, Khuller, and Mitchell, who introduced the *geometric knapsack problem* [4], which corresponds to the *inverted* version of the Geometric-Enclosure-with-Penalties problem in the special case where there are no required objects. (In their equivalent formulation, each object has a finite nonnegative value, and the goal is to compute a curve that maximizes the total value of the enclosed objects minus its length.) They gave an algorithm with running time $O(n^4)$ [4, Theorem 6]. In the full version [5, Appendix M], we note some issues regarding weak simplicity and interior/exterior of their output. Since there are no required objects, our algorithm for the inverted problem solves the geometric knapsack problem in time $O(n^3)$.

Relation to homotopy and homology. Our problem has a topological flavor and is therefore, in principle, amenable to homotopy and homology techniques. However, these techniques are unlikely to lead to algorithms that are FPT in k, even assuming only infinite penalties. In particular, for the Graph-Enclosure-with-Penalties problem, enumerating a set of candidate homotopy classes, i.e., the possible ways how a solution winds around the objects to

enclose the required objects and avoid the most undesirable ones, is possible using a technique by Chambers, Colin de Verdière, Erickson, Lazarus, and Whittlesey [7] (specifically, because the exchange argument [7, Proposition 4.2] is also valid for our problem), but this results in an algorithm with runtime $K^{O(K)} \cdot n \log n$, where K is the number k of required objects *plus* the number of objects with nonzero penalty. Similarly, the technique of *homology covers*, by Chambers, Erickson, Fox, and Nayyeri [8], is applicable, but yields algorithms with runtime $K^{O(K)} \cdot n \log\log n$ or $2^{O(K)} \cdot n \log n$, thus again with an exponential dependence on K. If there are many objects with nonzero penalty, our algorithm with runtime $O(3^k n^3)$ is faster.

Specifying only the number of objects to be enclosed. If we are just given a set of n points in general position and the exact *number* $k \leq n$ of points to be enclosed, a minimum-perimeter polygon enclosing at least k points is convex, contains exactly k points, and can be found in polynomial time by an algorithm of Eppstein, Overmars, Rote, and Woeginger [12, Corollary 5.3, Case 3]. This algorithm could for example be used to identify an unusual cluster in an otherwise uniformly distributed point set. However, if the input consists of polygons instead of points, we are not aware of a better method than guessing the k polygons to be enclosed and applying our main result, resulting in an algorithm of running time $O(\binom{N}{k} 3^k n^3) = O(N^k n^3)$ if there are N objects.

More variations. Some related separation and enclosure problems have been recently studied. Abrahamsen, Giannopoulos, Löffler, and Rote [1] investigated the problem of separating two or more groups of polygonal objects by a shortest system of fences. These fences may form an arbitrary plane graph, not necessarily a cycle. For separating *two* groups of polygons by a shortest fence, there is an algorithm of running time $O(n^4 \log^3 n)$, based on minimum-cut techniques [1]. For separating more than two groups, only approximation algorithms are known. While we study a problem in the same spirit, a key difference is that we require a single (weakly) simple cycle, which makes the techniques of this article not applicable for us.

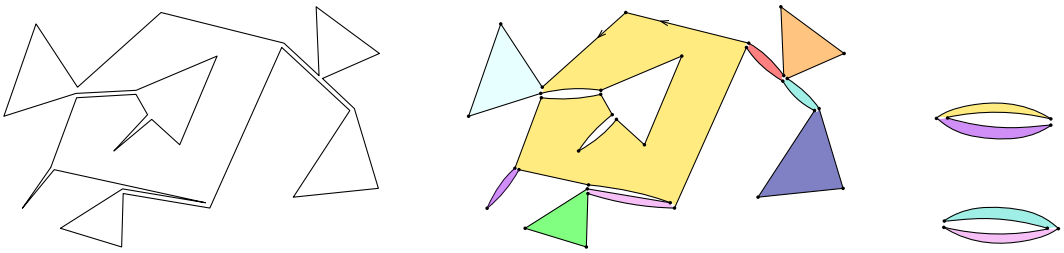
Chan, He, and Xue [9] studied the problem of choosing a smallest subset of objects from a given set whose union encloses a given point set. For this problem, there are only approximation algorithms. Klost, van Kreveld, Perz, Rote, and Tkadlec [13] have recently investigated optimum "blob-trees", where the objective is to *connect* a set of points by a combination of curves and edges.

2 Preliminaries

2.1 Weakly simple polygons

A polygon is **weakly simple** if it has fewer than three vertices, or it has at least three vertices and for any $\varepsilon > 0$, the vertices can be perturbed by at most ε to yield a simple polygon [2, 10]. We traverse a weakly simple polygon counterclockwise, i.e., with the interior to the left of each edge. Our proof uses a combinatorial characterization of a weakly simple polygon in terms of a non-crossing Euler tour in a plane multigraph [5, Lemma 16 in Appendix A]. This allows us to partition the edges of a weakly simple polygon into boundary walks of interior faces, see Figure 3. Note that we first subdivide an edge when a vertex lies in its interior.

A vertex of a weakly simple polygon with incoming edge e and outgoing edge f is a **transition vertex** if e and f belong to different interior faces. An interior face of two edges is a **corridor** and an interior face of more than two edges is a **chamber**. A chamber is not necessarily a simple polygon, but it is almost simple (see Figure 3(b)). More formally, a **bounded almost-simple polygon** is the boundary walk of an interior face of a connected straight-line



(a) (b) (c)

Figure 3 (a) A weakly simple polygon drawn via its ε-approximation. (b) The edges, after subdividing at interior vertices (forks), are partitioned into interior faces. Four faces are corridors and five are chambers. The largest chamber (in yellow) is almost-simple but not simple. (c) Non-uniqueness of the faces for a weakly simple polygon that traverses a line segment four times. In the top figure the two vertices on the left are transition vertices; this is reversed in the bottom figure.

graph drawing in the plane. We also allow an **unbounded almost-simple polygon** by traversing the boundary of the outer face clockwise. An almost-simple polygon has a connected interior and a bounded almost-simple polygon can be triangulated. Almost-simple polygons play two roles: the bounded ones arise as chambers; and our general Enclosure-with-Penalties problem allows almost-simple input polygons (including a single unbounded one).

All these concepts are made rigorous in [5, Appendix A]. We note that the partition of the edges of a weakly simple polygon into interior faces (and hence the definition of corridors and transition vertices) is not unique, see Figure 3(c). This non-uniqueness, which is inherent in the ε-approximation definition of weakly simple polygons, does not affect our proofs.

2.2 Winding number and winding parity

Our algorithm will construct intermediate polygons that are not necessarily weakly simple; so it will be useful to generalize "enclosed by" in terms of winding numbers. For a polygon W and a point x not lying on a vertex or edge of W, the **winding number** $\text{wind}(W, x)$ of x with respect to W is defined as follows. Take a ray ρ from x that avoids vertices of W. If an edge of W crosses ρ from right to left, we count this as $+1$; a crossing from left to right is counted as -1, and the total count gives the winding number. This is well-defined independent of the choice of ρ. The winding number is undefined for points x on W. Observe that, for a weakly simple polygon W traversed counterclockwise, point x lies in the interior of W if and only if $\text{wind}(W, x) = 1$. The **winding parity** of x with respect to W is $\text{wind}(W, x) \bmod 2$.

3 Our Common Framework: Enclosure-with-Penalties

In this section we formally define the Enclosure-with-Penalties problem that provides a common framework for both the geometric and graph settings.

Input:

- A set of interior-disjoint almost-simple polygons in the plane. We allow a single polygon to be unbounded. We subdivide polygon edges to ensure that no polygon vertex lies in the interior of an edge of another polygon. The **free space** is the plane minus the interiors of the polygons.
- A partition of the input polygons into a set R of k **required** polygons and the remaining set O of **optional** polygons. If there is an unbounded polygon, it must lie in O.

- For each polygon $P \in O$, a **penalty** $\pi_P \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$.
- The weight w_{ab} of a line segment ab in the free space is its Euclidean length, except for *squeezed edges*. A squeezed edge is a polygon edge that is incident to polygons on both sides. We may specify an *arbitrary* positive weight for a squeezed edge. Subsegments of a squeezed edge get proportional weight, and combinations of different squeezed or non-squeezed segments have their weights added.

Output: A weakly simple polygon W that lies in the free space and contains all polygons of R while minimizing the **cost** $c(W)$, which is defined as

$$c(W) := w(W) + \pi(W), \tag{1}$$

where $w(W)$ is the sum of the weights of the edges of W, and $\pi(W)$ is the sum of the penalties of the polygons of O that are inside W. Our main result is:

▶ **Theorem 3.** Enclosure-with-Penalties *for k required polygons can be solved in $O(3^k n^3)$ time and $O(2^k n^2)$ space, if the input polygons have n vertices in total.*

Theorem 1 is an immediate consequence of Theorem 3. Theorem 2 follows from Theorem 3 via a straight-line embedding of the graph, as outlined in Section 1 and detailed in the full version [5, Appendix B].

The Enclosure-with-Penalties problem as defined above does not allow point objects. (They are not almost-simple.) [5, Appendix I] shows how to deal with point objects.

4 Dynamic Programming Algorithm

The algorithm builds a polygon composed of free-space edges, where a **free-space edge** is an inclusionwise minimal line segment in the free space whose endpoints are vertices of the input polygons. We prove in Section 6 that this restriction to free-space edges is valid. We refer to a solution interchangeably as a polygon or as a closed walk in the graph of free-space edges.

The intuition for the algorithm is based on the decomposition of a weakly simple polygon W into corridors and chambers joined at "cutpoints", see Figure 3. A cutpoint separates W into subpolygons and partitions the set of enclosed objects. Our first type of subproblem finds polygons that enclose a specified subset of R and go through a specified vertex.

A corridor is a digon, and a chamber can be triangulated by adding chords, where a chord may cut through polygons. We therefore use digons and triangles as the basic building blocks to construct our solutions. A chord cuts off part of the solution. Our second type of subproblem finds polygons that use a walk of free-space edges between two given vertices p and q together with the chord pq (called the *mouth*) to enclose a specified subset of R.

Since a mouth may cut through polygons, we choose a **reference point** r_P in the interior of every input polygon P, and aim to enclose r_P for $P \in R$. Observe that a weakly simple polygon W in the free space encloses P if and only if r_P lies in the interior of W.

The dynamic program explicitly keeps track of the subset of required objects that are enclosed by partial solutions (2^k possibilities). However, when combining two partial solutions, the algorithm does not have enough information to check whether they cross. Thus, we allow self-crossing solutions. In particular, our use of the word "enclosing" is aspirational, and will only be made precise in terms of winding numbers, see Section 4.2. When we state the algorithm, we invite the reader to think of a weakly simple solution without crossings.

Types of subproblems. A subproblem of type C ("closed") is rooted at a vertex p, and we build a closed walk that goes through p and is composed of free-space edges. A subproblem of type M ("mouth") is rooted at a segment pq between vertices of input polygons, called the **mouth**, and we build an open walk of free-space edges from p to q; adding segment qp closes the walk. In addition to the root, each subproblem has two more parameters, B and t: The set $B \subseteq R$ specifies the precise subset of required objects that must be enclosed, and the integer $t \geq 0$ is an upper bound on the number of edges of the walk. In each case, the subproblem looks for a *minimum-cost* solution with the required properties.

4.1 Dynamic programming recursion

We now give recursive formulas for C and M, preceded in each case by an explanation of the formulas. The formulas with the respective partitions of the walk are illustrated in Figure 4.

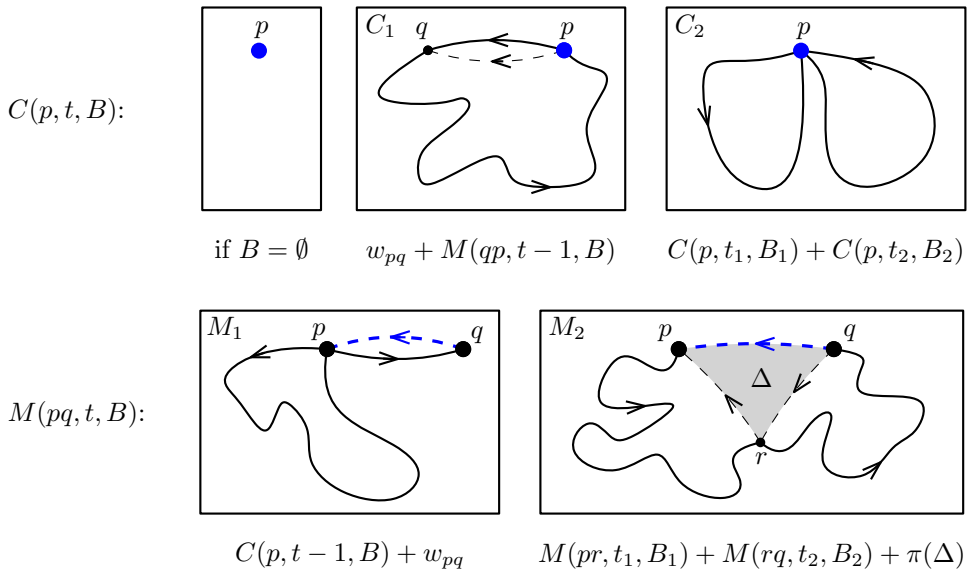


Figure 4 Cases of the recursion. Solid edges are free-space edges; dashed edges are mouths.

For $C(p,t,B)$ we have two base cases: if $B = \emptyset$, then the shortest closed walk is just the point p and its cost is 0 (Equation (2)); and if $B \neq \emptyset$ and $t \leq 1$, there is no solution, and we set the cost to ∞ (Equation (3)). Otherwise we have two general cases: the closed walk uses an edge pq of the free space (for some q) plus a solution $M(qp, t-1, B)$ (Equation (4)); or the closed walk is composed of two smaller closed walks that both go through p (Equation (5)). The notation $\sqcup$ means disjoint union: we partition the objects in B into two sets, each "enclosed" by exactly one of the two closed walks.

The base cases define $C(p,t,B)$ for $B = \emptyset$ and for $t \leq 1$:

$$C(p,t,\emptyset) := 0, \text{ for } t \geq 0 \tag{2}$$

$$C(p,t,B) := \infty, \text{ for } B \neq \emptyset \text{ and } t \leq 1 \tag{3}$$

In the general case, for $B \neq \emptyset$ and $t \geq 2$, we set

$$C(p,t,B) := \min\{C_1, C_2\}, \text{ where}$$

$$C_1 := \min\big\{ w_{pq} + M(qp, t-1, B) \;\big|\; pq \text{ is a free-space edge} \big\} \tag{4}$$

$$C_2 := \min\big\{ C(p,t_1,B_1) + C(p,t_2,B_2) \;\big|\; t = t_1 + t_2;\ B = B_1 \sqcup B_2;\ B_1, B_2 \neq \emptyset \big\} \tag{5}$$

For $M(pq, t, B)$ there are two possibilities: if pq is a free-space edge, we can use a closed walk at p plus the edge pq (Equation (6)); or we can attach a triangle $\Delta = prq$ to the mouth pq (Equation (7)). In the first case we add the weight of the edge pq. In the triangle case we take into account the polygons $R(\Delta)$ with reference points in Δ, where we consider Δ to be closed on pr and rq and open on pq, and we define $\pi(\Delta)$ to be the sum of the penalties of polygons of O with reference points in Δ.

$M(pq, t, B)$ is defined only for $t \geq 1$:

$$M(pq, t, B) := \min\{M_1, M_2\}, \text{ where}$$

$$M_1 := \begin{cases} C(p, t-1, B) + w_{pq} & \text{if } pq \text{ is a free-space edge} \\ \infty, & \text{otherwise} \end{cases} \tag{6}$$

$$M_2 := \min\Big\{ M(pr, t_1, B_1) + M(rq, t_2, B_2) + \pi(\Delta) \;\Big|\; \Delta = prq \text{ is a counterclockwise triangle}, \; t = t_1 + t_2,\ t_1 \geq 1, t_2 \geq 1,\ B = B_1 \sqcup B_2 \sqcup R(\Delta) \Big\} \tag{7}$$

As we shall see later in Lemma 13, the optimal walk has at most $6n$ edges. Thus, we define the solution to the whole problem as

$$c_{\mathrm{DP}} := \min\{\, C(p, 6n, R) \mid p \text{ a vertex} \,\}. \tag{8}$$

When we allow point objects, the algorithm needs a few refinements, see [5, Appendix I]. In the following sections we prove that c_{DP} is the correct value. Although not required by our proof, we note for completeness in [5, Appendix L] that the class of polygons over which the algorithm optimizes is the class of *immersed* or *self-overlapping* weakly simple polygons.

Runtime and Space. A routine analysis shows that the runtime of the dynamic program is $O(3^k n^5)$, see the full version [5, Appendix C.1]. A more efficient version of the dynamic program, given in Section 7, eliminates the parameter t and runs in time $O(3^k n^3)$.

4.2 Extracting the solution

With every finite value computed in the dynamic program we can naturally associate an open or closed walk of free-space edges that has this value as its cost (with "partially enclosed" objects taken into account according to their reference points): Figure 4 illustrates how each value is computed from the values of subproblems; the walks corresponding to those subproblems are composed accordingly. (For more details, see [5, Appendix C.2].) We will prove in Section 6 that c_{DP} is finite; so the associated closed walk exists:

▶ **Definition 4.** *W_{DP} is the polygon associated with the optimum solution value c_{DP} in (8).*

The polygon W_{DP} uses free-space edges, but it might not be weakly simple, and there is no notion of enclosed objects. Instead, we use winding numbers: we show that the reference point of any object in R has winding number 1 in W_{DP}, and we define a cost measure for W_{DP} in terms of winding numbers and prove equality with c_{DP}.

▶ **Definition 5.** *For any polygon W in the free space, define the **cost** to be*

$$c(W) := w(W) + \sum_{P \in O} \mathrm{wind}(W, r_P) \cdot \pi_P.$$

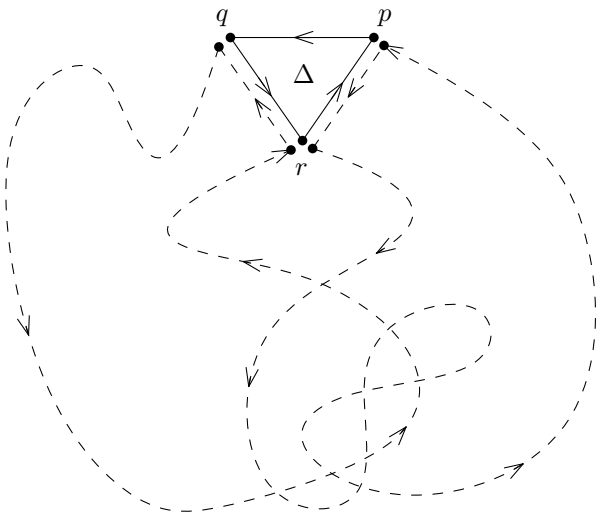


Figure 5 Gluing together closed walks, which may cross each other and are possibly self-crossing.

When W is a counterclockwise weakly simple polygon, this matches the previous definition $c(W) = w(W) + \pi(W)$, see Equation (1). The solution W_{DP} produced by the algorithm has the following properties:

▶ **Lemma 6.**
(A) $c_{\mathrm{DP}} = c(W_{\mathrm{DP}})$;
(B) *for all* $P \in R$, $\mathrm{wind}(W_{\mathrm{DP}}, r_P) = 1$;
(C) *for all points* x *that do not lie on* W_{DP}, $\mathrm{wind}(W_{\mathrm{DP}}, x) \geq 0$.

Lemma 6 is proved in [5, Appendix C.2] by induction as the dynamic program builds solutions by gluing together open/closed walks. The induction must apply also to open walks, and we use the fact that winding numbers add when gluing walks together, see Figure 5.

5 Uncrossing Algorithm and Final Output W_{ALG}

The final step of our algorithm "uncrosses" the closed walk W_{DP} produced by the dynamic program and turns it into a weakly simple polygon W_{ALG} without increasing the cost. To do so, we cut it into subpaths, eliminate some, and reorder the rest.

Our algorithm uses a known result about taking a plane multigraph (specified via its rotation system) and finding a **non-crossing Euler tour** in which successive visits to a vertex do not cross each other. (See [5, Appendix A] for more detailed definitions.) The existence of such a tour in an Eulerian plane multigraph is an easy exercise, see [17] or [18, Lemma 3.1], and a linear-time algorithm was given by Akitaya and Tóth [3]. We summarize it in the following proposition, and give a self-contained proof in [5, Appendix D].

▶ **Proposition 7** (Uncrossing Eulerian plane multigraphs)**.** *Given a plane connected Eulerian multigraph* H *with* m *edges, specified by its combinatorial map, we can, in* $O(m)$ *time, compute a non-crossing Euler tour of* H*.*

We now sketch our algorithm to uncross any polygon W to a weakly simple polygon W'. An **interior crossing** is a point that is in the relative interiors of two (or more) edges that are not collinear. A **fork** (or *interior vertex*) is a vertex in the relative interior of an edge.

▶ **Algorithm 8** (Uncrossing Algorithm).

1. *Subdivide every edge of W at every fork and interior crossing.*
2. *In the resulting multiset of edges (line segments in the plane) reduce multiplicities to* 1 *or* 2 *by repeatedly discarding pairs of equal line segments. The result is a plane connected Eulerian multigraph.*
3. *Apply Proposition 7 to find a non-crossing Euler tour. This corresponds to a weakly simple polygon W'.*

In the full version of our paper [5, Appendix D], we give further details of the algorithm and an implementation with runtime $O(t \log t + s)$ where t is the number of edges of W and $s \in O(t^2)$ is the number of interior crossings of W (with multiple crossings counted once). For input W_{DP} we show that there are no interior crossings, so the runtime is $O(n \log n)$.

We use the following important property of the Uncrossing Algorithm. Recall the definition of winding parity from Section 2.2.

▶ **Lemma 9.** *Every point x in the plane that does not lie on W has the same winding parity in W and in W'.*

Proof. For any ray r from x to infinity that avoids vertices of W, the parity of the number of edges it crosses is the same for W and W' since we have discarded pairs of equal line segments. The direction in which W' traverses an edge does not affect the winding parity. ($+1$ and -1 have the same parity.) ◀

▶ **Definition 10.** *W_{ALG} is the output of the Uncrossing Algorithm on input W_{DP}, oriented in counterclockwise direction.*

▶ **Lemma 11.** *W_{ALG} is a weakly simple polygon in the free space. W_{ALG} encloses R and $c(W_{\mathrm{ALG}}) \le c_{\mathrm{DP}}$.*

Proof. Consider a polygon $P \in R$. By Lemma 6(B), $\mathrm{wind}(W_{\mathrm{DP}}, r_P) = 1$. By Lemma 9, r_P has the same winding parity in W_{ALG}. Since W_{ALG} is weakly simple, every point has winding number 0 or 1. Thus $\mathrm{wind}(W_{\mathrm{ALG}}, r_P) = 1$ and W_{ALG} encloses P.

Next we consider costs. The definition of the costs in Equation (1) gives

$$c(W_{\mathrm{ALG}}) = w(W_{\mathrm{ALG}}) + \pi(W_{\mathrm{ALG}}),$$

where $\pi(W_{\mathrm{ALG}})$ is the sum of the penalties of objects of O enclosed by W_{ALG}.

By Lemma 6(A) and the definition of $c(W_{\mathrm{DP}})$,

$$c_{\mathrm{DP}} = c(W_{\mathrm{DP}}) = w(W_{\mathrm{DP}}) + \sum_{P \in O} \mathrm{wind}(W_{\mathrm{DP}}, r_P) \cdot \pi_P.$$

The Uncrossing Algorithm ensures that $w(W_{\mathrm{ALG}}) \le w(W_{\mathrm{DP}})$. It remains to compare the penalties. Let P be a polygon of O enclosed by W_{ALG}, i.e., with $\mathrm{wind}(W_{\mathrm{ALG}}, r_P) = 1$. By Lemma 9, the reference point r_P has the same winding parity in W_{DP}, and by Lemma 6(C), $\mathrm{wind}(W_{\mathrm{DP}}, r_P) \ge 0$. Thus $1 \le \mathrm{wind}(W_{\mathrm{DP}}, r_P)$ and

$$\pi(W_{\mathrm{ALG}}) = \sum_{P \in O} \mathrm{wind}(W_{\mathrm{ALG}}, r_P) \cdot \pi_P \le \sum_{P \in O} \mathrm{wind}(W_{\mathrm{DP}}, r_P) \cdot \pi_P$$

Therefore $c(W_{\mathrm{ALG}}) \le c_{\mathrm{DP}}$. (In fact, equality holds, as shown in the next section.) ◀

6 Correctness Proof

In defining W_{ALG}, we relied on the assumption that c_{DP} is finite. In this section we prove this fact, which implies that W_{ALG} exists, and we prove our main correctness result:

▶ **Theorem 12.** *W_{ALG} is an optimum solution to the* Enclosure-with-Penalties *problem.*

We defined the Enclosure-with-Penalties problem over the continuous space of all weakly simple polygons, but our algorithm only explores the discrete space of weakly simple polygons composed of at most $6n$ free-space edges. So we first prove that there is an optimum solution in this discrete space. A *feasible* solution is a weakly simple polygon that lies in the free space and encloses R.

▶ **Lemma 13.** *For the* Enclosure-with-Penalties *problem, there exists an optimum solution W_{OPT} of finite cost that consists of at most $6n$ free-space edges.*

Proof idea (Details in [5, Appendix E]). Let $\mathcal{S}$ be the discrete set of feasible solutions that consist of free-space edges each traversed at most twice. Because a planar graph on n vertices has at most $3n$ edges, any solution in $\mathcal{S}$ has at most $6n$ free-space edges.

We next prove that $\mathcal{S}$ contains a feasible solution that encloses R and excludes O, and thus has finite cost. The idea is to take the cycle boundaries of polygons in R and join them by paths traversed twice.

Since $\mathcal{S}$ is finite and nonempty, this implies that, among the solutions in $\mathcal{S}$, there is a solution W^* of minimum cost.

Finally, we prove that any feasible solution not in $\mathcal{S}$ can be homotopically shortened and then uncrossed to get a solution in $\mathcal{S}$ of no greater cost. Thus W^* is an optimum solution. ◀

We prove that the solution W_{OPT} from Lemma 13 is one of the candidate solutions over which the dynamic program optimizes. As a consequence:

▶ **Lemma 14.** $c_{\mathrm{DP}} \leq c(W_{\mathrm{OPT}})$.

Theorem 12 then follows: Lemmas 13 and 14 establish that c_{DP} is finite. Thus W_{ALG} exists. By Lemma 11, W_{ALG} is a feasible solution and $c(W_{\mathrm{ALG}}) \leq c_{\mathrm{DP}}$. Combining with Lemma 14 yields $c(W_{\mathrm{OPT}}) \leq c(W_{\mathrm{ALG}}) \leq c_{\mathrm{DP}} \leq c(W_{\mathrm{OPT}})$. Thus W_{ALG} is optimal.

We say a few words about the proof of Lemma 14. By the definition of c_{DP}, it suffices to show that $C(p, 6n, R) \leq c(W_{\mathrm{OPT}})$ for a vertex p on W_{OPT}. We give an inductive proof of the more general statement that $C(p, t, B)$ is at most the cost of any weakly simple polygon W with at most t free-space edges that encloses B and goes through p. Since W_{OPT} has at most $6n$ edges, this implies Lemma 14. The following lemma, which is proved in [5, Appendix E], includes an analogous inductive statement for $M(pq, t, B)$, with a suitable definition of the cost $c(W_0)$ of an open walk W_0. It refers to transition vertices, which were defined in Section 2.1.

▶ **Lemma 15.** (A) *Let W be a weakly simple polygon with ℓ free-space edges, going through vertex p, and let B be the objects of R that are enclosed by W. Then, for all $t \geq \ell$, $C(p, t, B) \leq c(W)$.*

(B) *Let W_0 be an open walk with ℓ free-space edges from p to q such that the polygon $W = W_0 + qp$ is weakly simple and q is not a transition vertex of W. Let B be the objects of R whose reference points lie inside W and not on pq. Then, for all $t \geq \ell$, $M(pq, t, B) \leq c(W_0)$.*

7 Reducing the Runtime with a Dijkstra-Style Algorithm

The runtime of our algorithm to solve the Enclosure-with-Penalties problem can be reduced by a $\Theta(n^2)$ factor, leading to the bound of Theorem 3.

The dynamic programming algorithm in Section 4 is guided by a parameter t, which limits the number of edges of the walk. We have proved (Lemma 13) that there is an optimal solution with at most $6n$ edges. Hence, the solution cannot be improved by allowing larger values of t; the iteration stabilizes, and the algorithm can stop when t reaches $6n$. The parameter t has been useful for ensuring that the quantities in the dynamic programming algorithm are well-defined, and it is essential as an induction variable for the proofs. We will now eliminate t and solve the recursion in the style of Dijkstra's algorithm for shortest paths. In particular, we use a generalization of Dijkstra's algorithm as proposed by Knuth [14].

More specifically, we define $C(p,B) := C(p,6n,B)$ and $M(pq,B) := M(pq,6n,B)$ in terms of the quantities from Section 4. By the above observations, $C(p,B) = C(p,t,B)$ and $M(pq,B) = M(pq,t,B)$ for all $t \geq 6n$. Therefore, the limit quantities $C(p,B)$ and $M(pq,B)$ fulfill a variation of the recursions (2–7) where the parameter t is eliminated:

$$C(p,\emptyset) = 0 \tag{9}$$

$$C(p,B) = \min\{C_1(p,B), C_2(p,B)\} \text{ for } B \neq \emptyset, \text{ where} \tag{10}$$

$$C_1(p,B) = \min\big\{\, w_{pq} + M(qp,B) \;\big|\; pq \text{ is a free-space edge} \,\big\} \tag{11}$$

$$C_2(p,B) = \min\big\{\, C(p,B') + C(p,B'') \;\big|\; B = B' \sqcup B'';\ B', B'' \neq \emptyset \,\big\} \tag{12}$$

$$M(pq,B) = \min\{M_1(pq,B), M_2(pq,B)\}, \text{ where} \tag{13}$$

$$M_1(pq,B) = \begin{cases} w_{pq} + C(q,B), & \text{if } pq \text{ is a free-space edge} \\ \infty, & \text{otherwise} \end{cases} \tag{14}$$

$$\begin{aligned} M_2(pq,B) = \min\big\{\, & M(pr,B') + M(rq,B'') + \pi(\Delta) \;\big|\; \\ & prq = \Delta \text{ is a counterclockwise triangle;} \\ & B = B' \sqcup B'' \sqcup \{P \in R \mid r_P \in \Delta\} \,\big\} \end{aligned} \tag{15}$$

As in Equation (8), we define the solution to the whole problem as

$$c_{\mathrm{DP}} := \min\{\, C(p,R) \mid p \text{ is a vertex} \,\}. \tag{16}$$

By Lemma 13 and by the definition of $C(p,B)$ and $M(pq,B)$, the value of c_{DP} resulting from (16) is the same as the one resulting from (8).

The system (9–15) is a *system of equations* that involves cyclic dependencies. Nevertheless, we can show that it has a unique solution [5, Appendix F.1, Lemma 22]. The reason is that on the right-hand side of the equations, the result of any expression combining some quantities of the form $C(p,B)$ and $M(pq,B)$ is always *larger* than these quantities.

Similar to Dijkstra's shortest-path algorithm, our algorithm maintains tentative values $C(p,B)$ and $M(pq,B)$. The smallest of the tentative values is made permanent, and all right-hand side expressions where this value appears are evaluated and used to update the corresponding tentative left-hand side values. The algorithm that carries out this idea is shown in the full version [5, Appendix F.2, Algorithm 1].

The most numerous quantities are the $O(2^k n^2)$ values $M(pq,B)$, and hence the space complexity is $O(2^k n^2)$. Compared to the running time for the dynamic programming algorithm in Section 4, we save a factor n^2: The elimination of t reduces the number of recursions by a factor $\Theta(n)$, and we save another factor $\Theta(n)$ because we need not go through all decompositions $t = t_1 + t_2$ on the right-hand side. The analysis of the full algorithm is given in [5, Appendix F.3]. The total running time is $O(3^k n^3)$, as claimed in Theorem 3.

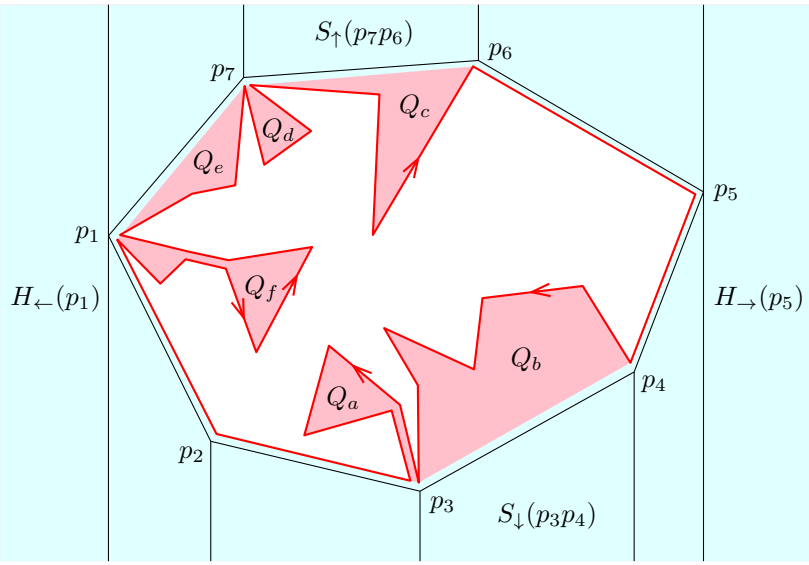


Figure 6 Partition of the outside into pockets $Q_a, \ldots, Q_f$, two half-planes, and seven planks.

8 The Inverted Problem

In the inverted problem, the required objects have to be outside the weakly simple polygon W, and the penalties of the polygons that are outside W are counted (see the introduction). The approach for the original problem has to be adapted accordingly. To derive a suitable dynamic programming formulation, we decompose the *outside* of W into elementary pieces, as shown in Figure 6: We form the convex hull of W and extend vertical rays upward and downward from the convex hull vertices. This leads to two additional types of regions:

- a left and a right half-plane, each bounded by a vertical line through an object vertex;
- vertical *planks*, that is, regions bounded by a line segment and two vertical upward rays or two vertical downward rays. We discuss such regions in [5, Appendix F.4].

We stick to the convention that the interior of the region of interest lies on the left side of W. Accordingly, the solution polygon W is now oriented clockwise.

In the algorithm, we build the region of interest outside-in, see Figure 7, starting from a left half-plane bounded by two vertical rays through an object vertex. We add planks from left to right along common rays, and, as in Section 4, we may also attach triangles along common edges and digons along common vertices. In addition to the usual bounded walks, we now also consider polygonal walks $W^{\updownarrow}$ that start from the endpoint of a vertical downward ray and end at a vertical upward ray. More precisely, for each pair of vertices p, q, we consider a subproblem of type U ("unbounded"), which considers regions bounded by a vertical ray down from p, a walk W from p to q, and a vertical ray up from q, see Figure 7 for an example; $p = q$ is allowed. Accordingly, the algorithm computes quantities $U(p, q, t, B)$ for all $B \subseteq R$ and $t \leq 6n$. The two unbounded rays jointly play the role of the mouth.

The full paper [5, Appendix G] describes how the dynamic programming recursion and the correctness proof of Section 6 for the non-inverted problem is adapted for the inverted problem.

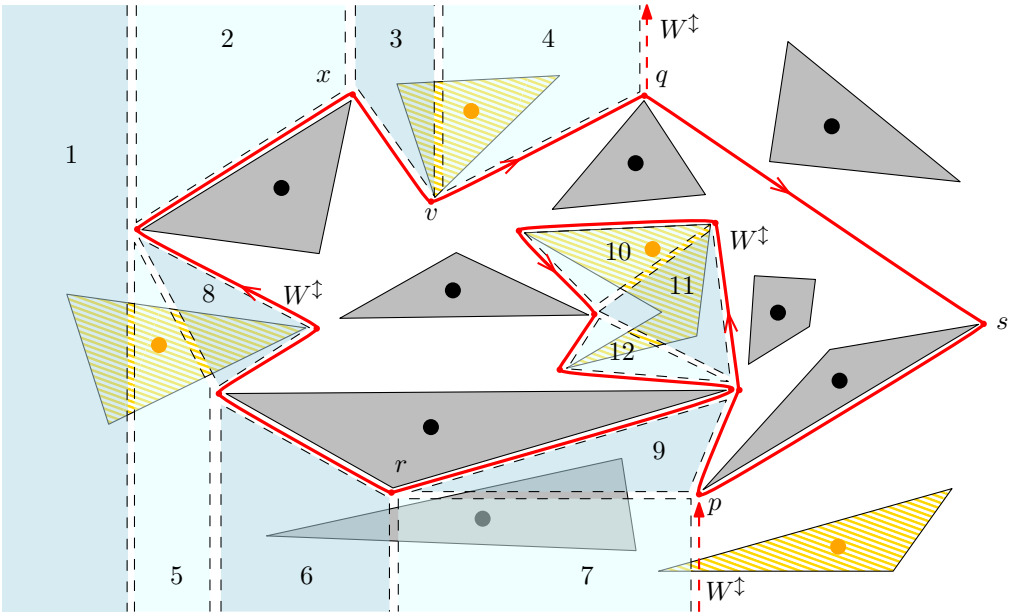


Figure 7 A weakly simple polygon W (red/bold) that has the four required objects (yellow/hatched) on the outside. The penalties of two optional (grey) objects is added to the length when the cost is computed. We grow the outer region triangle by triangle, considering also "triangles" that extend to $-\infty$ or $+\infty$ (vertical planks), or both, like the left half-plane (number 1). The union of the shaded blue regions is bounded by vertical rays from p downward and from q upward plus a weakly simple walk between p and q. The extended walk $W^{\updownarrow}$ is a candidate solution considered for the subproblem $U(p, q, B, t)$, when $t \geq 13$ and B consists of the three leftmost required objects. (The fourth required object has its reference point (thick dot) outside the region.) Two more planks, spanned by ps and qs, plus the right half-plane through s, would complete the outside region of W.

9 Applications, Extensions, and Open Problems

Splitting a surface. Our result may shed some light on the following open problem by Bulavka, Colin de Verdière, and Fuladi [6, Conclusion]: given an orientable combinatorial surface of genus g, and an integer g', $1 \leq g' < g$, is it FPT in g' to compute a shortest weakly simple closed curve that cuts off a surface of genus g'? The problem is FPT in g [7, Theorem 6.1]. Our algorithm for Graph-Enclosure-with-Penalties shows that the answer is yes when restricting to some (admittedly very special) instances, see [5, Appendix H], and thus provides some hope for a positive answer in general, although this remains open.

Curved objects and line segments. We believe that our approach carries over to more general objects. Curved objects that are sufficiently well behaved can be treated by considering all bitangents as free-space edges. We can handle point objects [5, Appendix I]; line segments without other vertices in their interior should also be doable. However, extending to weakly simple polygon objects seems difficult. Even for a path object consisting of two line segments joined at point p it is a challenge to prevent a solution from cutting through the path at p.

Recognizing weakly simple self-overlapping polygons. As mentioned, our dynamic program optimizes over the class of weakly simple self-overlapping polygons, see [5, Appendix L]. Weakly simple polygons can be recognized in $O(n \log n)$ time [2], and self-overlapping polygons in time $O(n^3)$ [16]. Can weakly simple self-overlapping polygons be recognized efficiently?

References

1 Mikkel Abrahamsen, Panos Giannopoulos, Maarten Löffler, and Günter Rote. Geometric multicut: shortest fences for separating groups of objects in the plane. *Discrete Comput. Geom.*, 64:575–607, 2020. doi:10.1007/s00454-020-00232-w.

2 Hugo A. Akitaya, Greg Aloupis, Jeff Erickson, and Csaba D. Tóth. Recognizing weakly simple polygons. *Discrete & Computational Geometry*, 58(4):785–821, 2017. doi:10.1007/S00454-017-9918-3.

3 Hugo A. Akitaya and Csaba D. Tóth. Reconstruction of weakly simple polygons from their edges. *International Journal of Computational Geometry & Applications*, 28(02):161–180, 2018. doi:10.1142/S021819591860004X.

4 Esther M. Arkin, Samir Khuller, and Joseph S. B. Mitchell. Geometric knapsack problems. *Algorithmica*, 10(5):399–427, 1993. doi:10.1007/BF01769706.

5 Therese Biedl, Éric Colin de Verdière, Fabrizio Frati, Anna Lubiw, and Günter Rote. Finding a shortest curve that separates few objects from many, April 2025. arXiv:2504.03558v1.

6 Denys Bulavka, Éric Colin de Verdière, and Niloufar Fuladi. Computing shortest closed curves on non-orientable surfaces. In Wolfgang Mulzer and Jeff M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SoCG.2024.28.

7 Erin W. Chambers, Éric Colin de Verdière, Jeff Erickson, Francis Lazarus, and Kim Whittlesey. Splitting (complicated) surfaces is hard. *Comput. Geom.*, 41(1–2):94–110, 2008. doi:10.1016/j.comgeo.2007.10.010.

8 Erin W. Chambers, Jeff Erickson, Kyle Fox, and Amir Nayyeri. Minimum cuts in surface graphs. *SIAM J. Comput.*, 52(1):156–195, 2023. doi:10.1137/19M1291820.

9 Timothy M. Chan, Qizheng He, and Jie Xue. Enclosing points with geometric objects. In Wolfgang Mulzer and Jeff M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2024.35.

10 Hsien-Chih Chang, Jeff Erickson, and Chao Xu. Detecting weakly simple polygons. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1655–1670, 2015. doi:10.1137/1.9781611973730.110.

11 Peter Eades and David Rappaport. The complexity of computing minimum separating polygons. *Patt. Recog. Lett.*, 14(9):715–718, 1993. doi:10.1016/0167-8655(93)90140-9.

12 David Eppstein, Mark Overmars, Günter Rote, and Gerhard Woeginger. Finding minimum area k-gons. *Discrete Comp. Geom.*, 7:45–58, 1992. doi:10.1007/BF02187823.

13 Katharina Klost, Marc van Kreveld, Daniel Perz, Günter Rote, and Josef Tkadlec. Minimum spanning blob-trees. In Jan Kratochvíl and Giuseppe Liotta, editors, *Abstracts of the 41st European Workshop on Computational Geometry (EuroCG 2025), April 9–11, Liblice, Czech Republic*, pages 23:1–23:8, 2025. doi:10.48550/arXiv.2503.02439.

14 Donald E. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5, 1977. doi:10.1016/0020-0190(77)90002-3.

15 Dániel Marx, Marcin Pilipczuk, and Michał Pilipczuk. On subexponential parameterized algorithms for Steiner tree and directed subset TSP on planar graphs. In *Proc. 59th Ann. IEEE Symp. Foundat. Comput. Sci. (FOCS)*, pages 474–484, 2018. doi:10.1109/FOCS.2018.00052.

16 Peter W. Shor and Christopher J. Van Wyk. Detecting and decomposing self-overlapping curves. *Comput. Geom.: Theory and Applications*, 2(1):31–50, 1992. doi:10.1016/0925-7721(92)90019-O.

17 David Singmaster and Jerrold W. Grossman. Solution to problem E2897: An Eulerian circuit with no crossings. *The American Mathematical Monthly*, 90(4):287–288, 1983. doi:10.2307/2975767.

18 Mu-Tsun Tsai and Douglas B. West. A new proof of 3-colorability of Eulerian triangulations. *Ars Mathematica Contemporanea*, 4:73–77, 2011. doi:10.26493/1855-3974.193.8e7.