Lines: 499

Finding a Shortest Curve that Separates Few Objects from Many

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

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 [13] 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. Here we generalize from objects that are points to objects that are interior-disjoint simple polygons in the plane, and we generalize to 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

41st International Symposium on Computational Geometry.
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

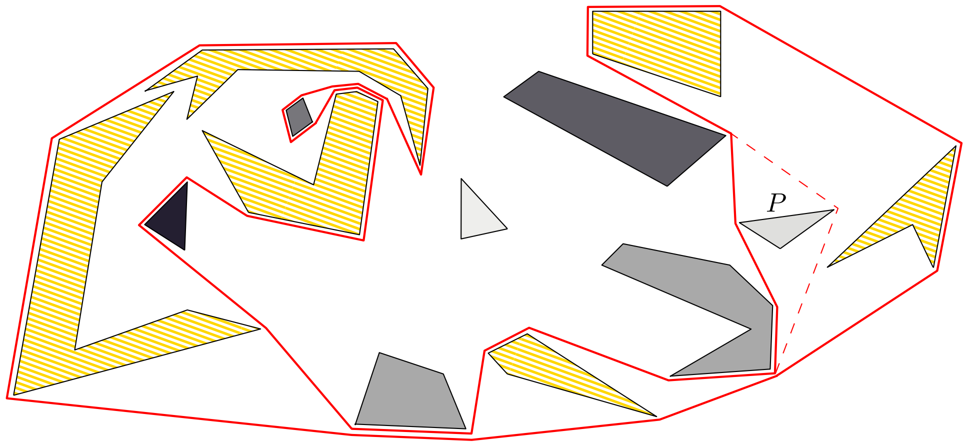


Figure 1 The GEOMETRIC-ENCLOSURE-WITH-PENALTIES problem. Objects in R are yellow (hatched) and objects in O are gray, darker for objects with larger penalties. A weakly simple solution polygon W is shown in red (bold). For visual clarity, W is drawn with an offset where it would otherwise touch the objects or itself. For example, the penalty π_P of the object $P \in O$ inside W is smaller than the detour that W would have to make in order to have P outside.

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.*

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 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 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

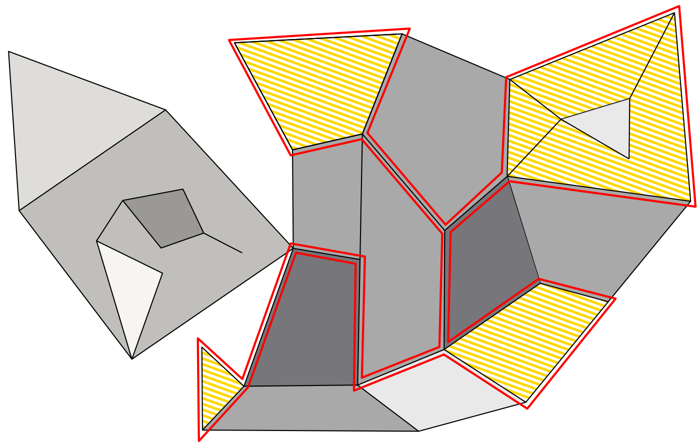


Figure 2 The Graph-Enclosure-with-Penalties problem. The colors have the same meaning as in Figure 1. A weakly simple closed walk W which is a solution for the instance is red (bold).

the 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 of the whole algorithm 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 [19, Theorem 1.2]. See 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 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 [13] (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]. 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, enumerating a set of candidate homotopy classes, the 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], but its size will be exponential in K, the number k of required objects *plus* the number of objects with nonzero penalty. The technique of *homology covers*, by Chambers, Erickson, Fox, and Nayyeri [8], is applicable, but 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 [14, 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. Separation problems using fences (which form an arbitrary plane graph, not necessarily a cycle), or by selecting a minimum subset of input shapes, have been studied recently, respectively by Abrahamsen, Giannopoulos, Löffler, and Rote [1] and by Chan, He, and Xue [9]. While we study a problem in the same spirit, a key difference is that we require a (weakly) simple cycle, which makes the techniques of these articles not applicable for us.

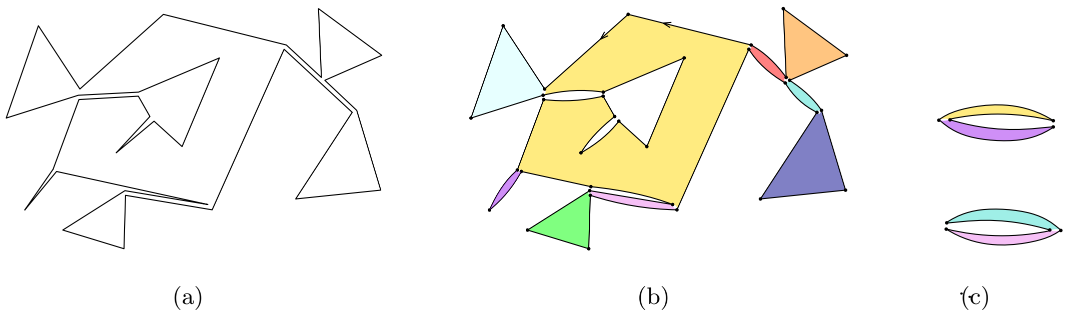


Figure 3 (a) A weakly simple polygon drawn via its ε-approximation. (b) The edges, after subdividing at interior vertices, are partitioned into interior faces. Four faces are corridors and five are chambers. The largest chamber (in yellow) is almost-simple but not simple. (c) Non-uniqueness of the faces 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.

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 (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. More formally, a ***bounded almost-simple polygon*** is the boundary walk of an interior face of a connected straight-line 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 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 we will find it useful to generalize "enclosed by" in terms of winding numbers. Let W be a polygon and let x be a point not lying on 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 Appendix B.

Note that the Enclosure-with-Penalties problem as defined above does not allow point objects (they are not almost-simple). 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 a minimal 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 programming algorithm 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 cannot afford 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 about 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 subset of required objects that are enclosed, and the integer $t \geq 0$ is an upper bound on the number of edges of the walk.

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.

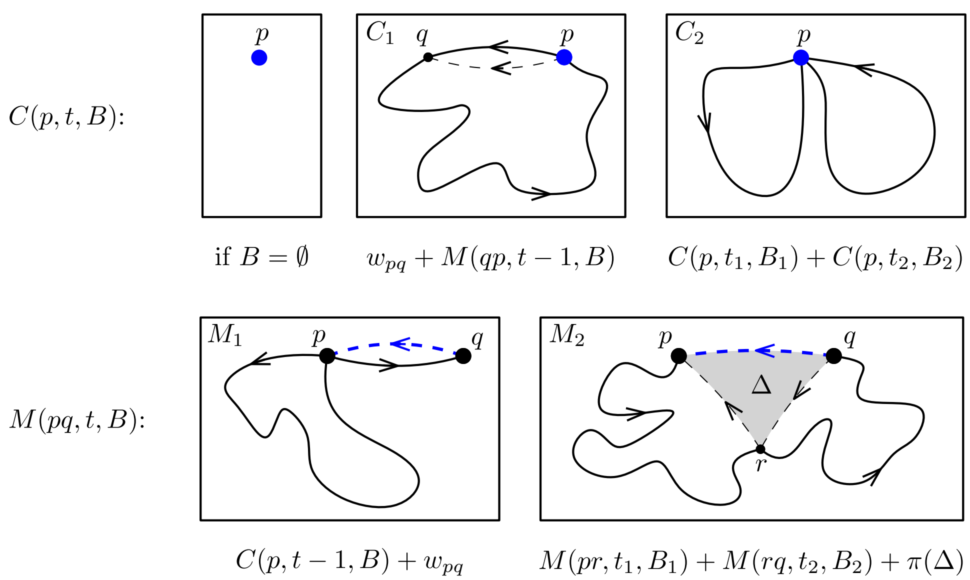


Figure 4 Cases of the recursion. Solid edges are free space edges; dashed edges are mouths.

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 one of the two closed walks.

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

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

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

In the general case, for $B\ne\emptyset$ and $t\ge 2$, we set

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

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

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

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 with reference points in Δ, where we consider Δ to be closed on pr and rq and open on pq. Define $R(\Delta)$ to be the polygons of R with reference points in Δ, and 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 $t\ge 1$:

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

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

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

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 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 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 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. (For more details, see 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).

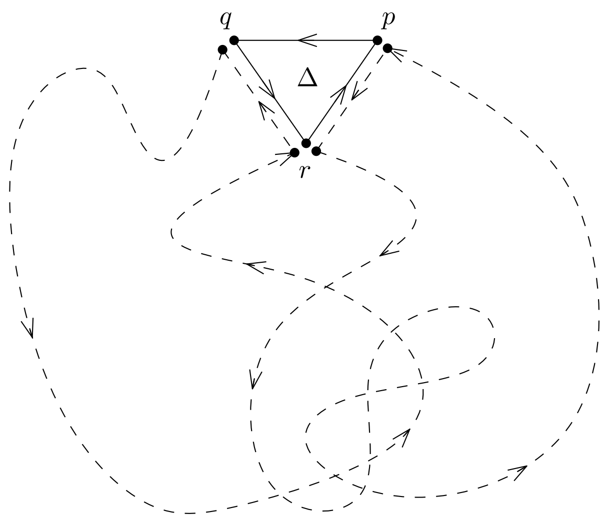


Figure 5 Gluing together closed walks, which may cross each other and are possibly self-crossing.

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*

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

When W is a counterclockwise weakly simple polygon, this matches the previous definition $c(W) = w(W) + \pi(W)$, see Equation (1). We prove the following properties of W_{DP}.

▶ **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 Appendix C.2 by induction as the dynamic program builds solutions to subproblems 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 Appendix A for more detailed definitions.) The existence of such a tour in an Eulerian plane multigraph is an easy exercise, see [21] or [23, 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 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 interior of two non-collinear edges.

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

1. *Subdivide every edge of W at every interior vertex 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 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 crossing points of W. 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.

▶ **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. Edge directions do not matter since 1 and -1 have the same parity. ◀

▶ **Definition 10.** *W_{ALG} is the output of the uncrossing algorithm on input W_{DP}, oriented in thecounterclockwise direction.*

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

Proof. Consider a polygon $P \in R$. By Lemma 6(B), $\text{wind}(W_{\text{DP}}, r_P) = 1$. 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 $\text{wind}(W_{\text{ALG}}, r_P) = 1$ and W_{ALG} encloses P.

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

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

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

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

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

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

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

Therefore $c(W_{\text{ALG}}) \leq c_{\text{DP}}$. ◀

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 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 of definition 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 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 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

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 is useful for ensuring that the quantities in dynamic programming algorithm are well-defined, and it is essential as an induction variable for the proofs. We will now show that it can be eliminated, and the recursion can be solved in the style of Dijkstra's algorithm for shortest paths. Such a generalization of Dijkstra's algorithm was proposed by Knuth [18], and it can be applied to our problem.

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. The resulting system of equations (13–19) is shown in Appendix F.1. This system involves cyclic dependencies. Nevertheless, we can show that it has a unique solution (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 Appendix F.2 (Algorithm 1).

The most numerous quantities are the $O(n^2 2^k)$ values $M(pq, B)$, and hence the space complexity is $O(n^2 2^k)$. 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 Appendix F.3. The total running time is $O(n^3 3^k)$, as claimed in Theorem 3.

8 The Inverted Problem

For the inverted problem, the approach for the original problem has to be adapted, as the region of interest is now *outside* the weakly simple polygon W. 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 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 ordered 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

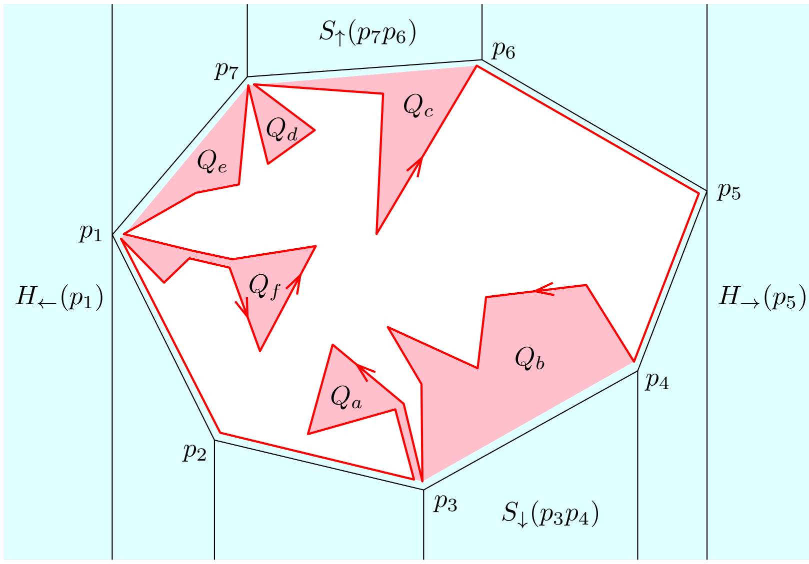


Figure 6 Partition of the outside into pockets $Q_a, \dots, Q_f$, two half-planes, and seven planks.

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 $t \le 6n$. The two unbounded rays jointly play the role of the mouth.

8.1 The dynamic programming recursion

For simplicity, we assume that distinct vertices and distinct reference points have distinct x-coordinates; this can be achieved by a rotation. We denote by $H_\leftarrow(q)$ and $H_\rightarrow(q)$ the left and right half-plane bounded by the vertical line through q. $S_\downarrow(pq)$ $S_\uparrow(pq)$ are the planks with boundary segment pq. By convention, p is always left of q.

The recursion considers three cases, as illustrated in Figure 8. The easy case $(U_\leftarrow)$ is a left half-plane, which applies only for $p = q$. The other two cases are symmetric to each other; we discuss here only $U_\downarrow$. This is similar to the term M_2 for $M(pq, t, B)$ in recursion (7), except that the plank $S_\downarrow(rp)$ plays the role of the triangle $\Delta = prq$. One of the subproblems, with mouth pr, is an "ordinary" subproblem of type M, the other subproblem is of type U.

$$U(p, q, t, B) = \min\{U_\leftarrow, U_\downarrow, U_\uparrow\}, \text{ where}$$

$$U_\leftarrow = \begin{cases} \pi(H_\leftarrow(p)), & \text{if } p = q \text{ and } B = R(H_\leftarrow(p)) \\ \infty, & \text{otherwise} \end{cases} \tag{9}$$

$$U_\downarrow = \min\{\, M(pr, t_1, B_1) + U(r, q, t_2, B_2) + \pi(S_\downarrow(rp)) \mid r \text{ left of } p,\ t = t_1 + t_2,\ B = R(S_\downarrow(rp)) \sqcup B_1 \sqcup B_2 \,\} \tag{10}$$

$$U_\uparrow = \min\{\, U(p, r, t_1, B_1) + M(rq, t_2, B_2) + \pi(S_\uparrow(rq)) \mid r \text{ left of } q,\ t = t_1 + t_2,\ B = R(S_\uparrow(rq)) \sqcup B_1 \sqcup B_2 \,\} \tag{11}$$

$R(\Omega)$ and $\pi(\Omega)$ (for some region Ω) generalize the earlier notations $R(\Delta)$ and $\pi(\Delta)$ and denote the required objects and the sum of penalties of optional objects whose reference point lies inside Ω. Reference points that lie *on* pq are treated as *belonging to* $S_\downarrow(pq)$ and $S_\uparrow(pq)$. No reference points lie on other boundaries of planks and half-planes by our initial rotation.

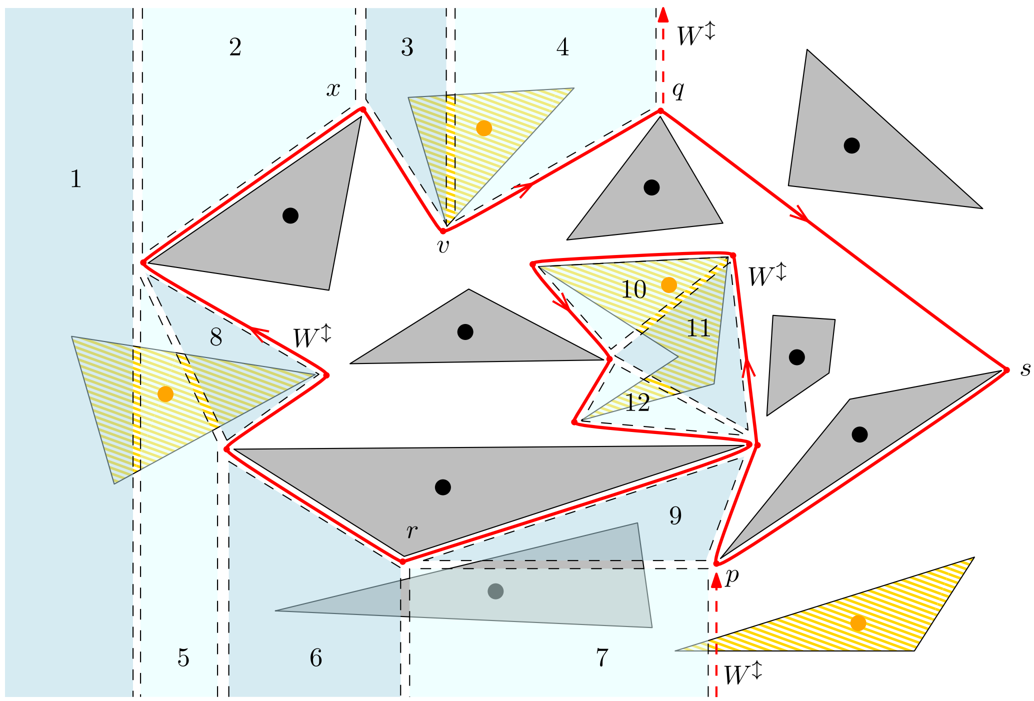


Figure 7 A weakly simple polygon W (red/bold) that has the four required objects (yellow/hatched) on the outside. The penalties of two optional (grey) objects is added to the length when the cost is computed. We grow the outer region triangle by triangle, considering also "triangles" that extend to $-\infty$ 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.

The overall solution is

$$c'_{\mathrm{DP}} := \min\{\, U(p, p, R(H_{\leftarrow}(p)), 6n) + \pi(H_{\rightarrow}(p)) \mid p \text{ is a vertex}\,\}$$

The first term, with $B = R(H_{\leftarrow}(p))$, makes sure that all required objects with the reference point to the left of p are covered. The remaining objects are then automatically covered by the right half-plane $H_{\rightarrow}(p)$. Appendix G describes how the correctness proof of Section 6 for the non-inverted problem must be adapted for the inverted problem.

9 Conclusion

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 Appendix H, and thus provides some hope for a positive answer in general, although this remains open.

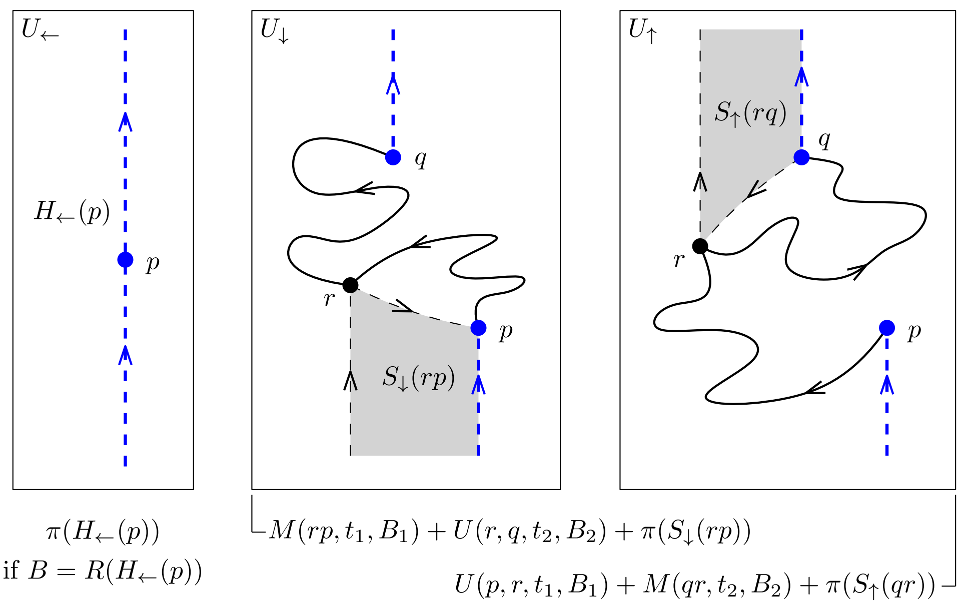


Figure 8 The recursion for the inverted problem. Free space edges are solid; mouths are dashed.

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 already handle point objects, as described in 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 Appendix L. Weakly simple polygons can be recognized in $O(n \log n)$ time [2], and self-overlapping polygons in time $O(n^3)$ [20]. Can weakly simple self-overlapping polygons be recognized efficiently?

References

1 M. Abrahamsen, P. Giannopoulos, M. Löffler, and G. 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 H. A. Akitaya, G. Aloupis, J. Erickson, and C. D. Tóth. Recognizing weakly simple polygons. *Discrete & Computational Geometry*, 58(4):785–821, 2017. doi:10.1007/S00454-017-9918-3.

3 H. A. Akitaya and C. 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 E. M. Arkin, S. Khuller, and J. S. Mitchell. Geometric knapsack problems. *Algorithmica*, 10(5):399–427, 1993. doi:10.1007/BF01769706.

5 T. Biedl. Small drawings of outerplanar graphs, series-parallel graphs, and other planar graphs. *Discret. Comput. Geom.*, 45(1):141–160, 2011. doi:10.1007/S00454-010-9310-Z.

6 D. Bulavka, É. Colin de Verdière, and N. Fuladi. Computing shortest closed curves on non-orientable surfaces. In W. Mulzer and J. 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. arXiv:2403.11749, doi:10.4230/LIPIcs.SoCG.2024.28.

7 E. W. Chambers, É. Colin de Verdière, J. Erickson, F. Lazarus, and K. Whittlesey. Splitting (complicated) surfaces is hard. *Comput. Geom.*, 41(1–2):94–110, 2008. doi:10.1016/j.comgeo.2007.10.010.

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

9 T. M. Chan, Q. He, and J. Xue. Enclosing points with geometric objects. In W. Mulzer and J. 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. arXiv:2402.17322, doi:10.4230/LIPIcs.SoCG.2024.35.

10 H.-C. Chang, J. Erickson, and C. Xu. Detecting weakly simple polygons. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1655–1670, 2015. arXiv:1407.3340, doi:10.1137/1.9781611973730.110.

11 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

12 G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. *Theor. Comput. Sci.*, 61:175–198, 1988. doi:10.1016/0304-3975(88)90123-5.

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

14 D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. Finding minimum area k-gons. *Discrete Comp. Geom.*, 7:45–58, 1992. doi:10.1007/BF02187823.

15 P. Evans and C. Wenk. Combinatorial properties of self-overlapping curves and interior boundaries. *Discrete & Computational Geometry*, 69(1):91–122, 2023. doi:10.1007/s00454-022-00416-6.

16 M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

17 S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991. doi:10.1137/0220055.

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

19 D. Marx, M. Pilipczuk, and M. 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.

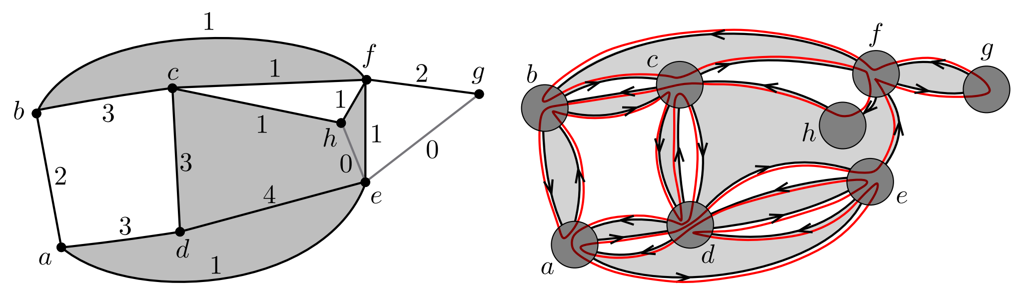
20 P. W. Shor and C. J. Van Wyk. Detecting and decomposing self-overlapping curves. *Computational Geometry: Theory and Applications*, 2(1):31–50, 1992. doi:10.1016/0925-7721(92)90019-O.

21 D. Singmaster and J. W. Grossman. Solution to problem E2897: An Eulerian circuit with no crossings. *The American Mathematical Monthly*, 90(4):287–288, 1983. URL: http://www.jstor.org/stable/2975767, doi:10.2307/2975767.

22 R. Tamassia and I. G. Tollis. A unified approach a visibility representation of planar graphs. *Discret. Comput. Geom.*, 1:321–341, 1986. doi:10.1007/BF02187705.

23 M.-T. Tsai and D. 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.

24 J. H. van Lint and R. M. Wilson. *A course in combinatorics*. Cambridge University Press, second edition, 2001.



(a) (b)

Figure 9 (a) A simple plane graph G. The edges of a closed walk $W = abcdcdaededadefgfbcfhcba$ are drawn in black and labeled with their multiplicities. (b) A non-crossing Euler tour in an expansion $M(G, W)$ of W. Vertices of $M(G, W)$ are represented as large disks. The Euler tour is shown in red and certifies that the walk W from (a) is weakly simple.

A Details for Section 2: Weakly Simple Polygons or Walks

We characterize weakly simple polygons/walks in terms of non-crossing Euler tours.

A connected plane graph or multigraph is specified via its combinatorial map (or rotation system) that specifies the counterclockwise cyclic order of edges around each vertex. If there are parallel edges, they have distinct identities and must be explicitly ordered in the rotation system. One face is designated as the outer face.

A ***non-crossing Euler tour*** of a plane multigraph is a closed walk that traverses each edge exactly once and has no ***vertex crossing***. A ***vertex crossing*** occurs when the tour visits some vertex v twice, entering once on edge e and leaving on edge f, and entering again on edge g and leaving on edge h, such that e, f and g, h interleave in the cyclic ordering of edges around v, i.e., they appear in the order e, g, f, h or e, h, f, g.

Let G be a plane multigraph with a non-crossing Euler tour T. Then the vertices of G have even degrees, so the faces of G can be 2-colored such that the two faces incident to an edge have different colors [24, Theorems 34.2 and 34.4]. Suppose the two colors are grey and white with the outer face colored white. We will traverse T so that a grey face lies to the left of the first edge of the tour. Then, because the tour is non-crossing, every edge of the tour has a grey face to the left. We call this a ***counterclockwise traversal*** of T, and we define the ***interior*** faces of T to be the grey faces. Note that the interior faces determine a partition of the edges of G.

Weakly simple walks in plane graphs. A walk W of length n in a simple plane graph G is a sequence $(v_0, v_1, \ldots, v_n)$ of vertices, such that each $v_i v_{i+1}$ is an edge of the graph. A vertex/edge of G may appear multiple times in the sequence. If $v_0 = v_n$ this is a ***closed walk***; otherwise it is an ***open walk***.

Intuitively, a closed walk W is weakly simple if multiple traversals of an edge of G can be resolved to avoid vertex crossings. We make this more formal by way of a non-crossing Euler tour that provides a certificate that W is weakly simple.

For edge e of G, define the ***multiplicity*** $m(e)$ of e in W, to be the number of times W traverses e (in either direction). An ***expansion*** of W is a plane multigraph $M(W, G)$ that replaces each edge $e = ab$ of G by a ***bundle*** of $m(e)$ parallel edges, each identified with a unique edge of W, and replaces e in the rotation systems of a and b by an ordered sequence of the edges in the bundle. Then W corresponds to an Euler tour in $M(W, G)$.

A closed walk W in a plane graph G is ***weakly simple*** if it has an expansion $M(W,G)$ in which W corresponds to a non-crossing Euler tour. We call such an $M(W,G)$ a ***certificate*** that W is weakly simple. Note that certificates are not unique; in particular they can have different rotation systems. For example, see Figure 3(c) when G is a single edge and W traverses it four times.

Let $M(W,G)$ be a certificate that W is weakly simple. Some faces of $M(W,G)$ are digons between parallel edges. Each remaining face is a union of faces of G. (If an edge of G has multiplicity 0, then the incident faces are merged in $M(W,G)$.) W corresponds to a non-crossing Euler tour of $M(W,G)$, which determines the interior faces of $M(W,G)$. We say that a face of G is ***interior*** to W if it corresponds to an interior face of $M(W,G)$. See Figure 9. (Observe that the interior faces of G are well-defined independent of choice of certificate because, in a 2-coloring of the faces of $M(W,G)$, the color of a face of G does not depend on the choice of rotation system for $M(W,G)$—the two faces incident to edge e of G have the same color if $m(e)$ is even, and opposite colors if $m(e)$ is odd.)

Weakly simple polygons. A polygon P is a sequence $(p_0, p_1, \ldots, p_{n-1})$ of points (the ***vertices*** of P) together with the line segments $p_i p_{i+1 \bmod n}$ (the ***edges*** of P). We do not allow edges of length 0. As a degenerate case, we allow a polygon with a single vertex and no edges. A polygon is ***simple*** if the vertices are distinct points and no two edges intersect except that consecutive edges intersect at their common vertex.

For a general non-simple polygon, a point of the plane may correspond to multiple polygon vertices, and polygon edges may overlap or cross. An ***interior crossing*** of P is a point that is in the relative interiors of two (or more) edges that are not collinear. A ***fork*** is a vertex that lies in the interior of an edge. Both interior crossings and forks can be eliminated by subdividing edges, albeit possibly with a quadratic blow-up in the number of vertices of the polygon.

The standard definition [2, 10] is that 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. The intuition is that a weakly simple polygon is one without crossings, but it is tricky to define crossings, since they need not be local, see the discussion by Chang, Erickson, and Xu [10].

In our proofs we find it useful to characterize weakly simple polygons in terms of the purely combinatorial notion of non-crossing Euler tours in an associated multigraph. Let P be a polygon without interior crossings. Expanding on definitions from [2, 10], we define the ***image graph*** of P to be a plane straight-line graph G formed as follows. First subdivide edges of P at interior vertices (i.e., at forks). Next replace every set of coincident vertices of P by a single vertex of G, and replace every set of equal line segments of P by a single edge of G (these are called "segments" in [2, 10]). Then P corresponds to a closed walk W_P in the plane graph G and we can apply the concept of a certificate for a weakly simple walk from above. In this context we call an expansion $M(W_P, G)$ an ***image multigraph*** of P. We use the notation $M(P)$ for an image multigraph of P, since it depends only on P.

▶ **Lemma 16.** *A polygon P is weakly simple if and only if it has no interior crossings and it has an image multigraph in which P corresponds to a non-crossing Euler tour.*

An image multigraph in which P corresponds to a non-crossing Euler tour is called a ***certificate*** that P is weakly simple. Again, note that a certificate is in general not unique.

Before turning to the proof of the lemma, we discuss equivalent notions of the interior of a weakly simple polygon. The definition of the interior of a non-crossing Euler tour gives one definition of the interior of a weakly simple polygon. This is equivalent to the definition of interior in terms of winding numbers. As seen in Figure 3, the edges of a weakly simple polygon can be partitioned into boundary walks of the interior faces.

The $O(n \log n)$ time algorithm to recognize weakly simple polygons by Akitaya, Aloupis, Erickson, and Toth [2] implicitly proves Lemma 16 (as does the earlier algorithm by Chang, Erickson, and Xu [10]). Akitaya et al. use *strip systems* as their certificates of weak simplicity. A strip system is more geometric in nature, but has the advantage of being linear size, which is important for their fast algorithm. Our image multigraphs have quadratic size, but more immediately give the properties we need.

We give a direct proof of Lemma 16 that depends only on the characterization of a weakly simple polygon as a limit of simple curves as Fréchet distance goes to zero [10, Theorem 2.1], which allows adding new vertices to the polygon.

Proof. Suppose P is weakly simple. By definition, P has no interior crossings. Let P' be the result of subdividing P at interior vertices. By definition, for any $\varepsilon > 0$ there is a simple ε-approximation of P, and this determines a simple ε-approximation of P', call it P'_ε. Any set U of coincident vertices of P' lies in a disc D of radius ε in P'_ε, and the edges incident to U leave D at distinct points. From P'_ε we construct a plane multigraph M by contracting each set U to a single vertex, and ordering the incident edges according to their order around D. Then M is a plane Eulerian multigraph that expands the image graph of P, and P corresponds to a non-crossing Euler tour of M.

For the other direction, suppose P has no interior crossings and suppose P has an image multigraph $M(P)$ in which P corresponds to a non-crossing Euler tour W. We use the result that a polygon is weakly simple if it is a limit of simple polygons (possibly with more vertices) as Fréchet distance goes to zero [10, Theorem 2.1]. For ε small enough, we construct a simple polygon P_ε within Fréchet distance ε of P. Polygon P_ε will have more vertices than P. In particular, our construction will subdivide edges of P at interior vertices, and then replace each vertex by two vertices and add vertices in the middle of edges. The coordinates of P's vertices determine a straight-line drawing of P's image graph G in the plane. We expand this to a 1-bend drawing of $M(P)$ in which the edges in each bundle are spread apart. In more detail, each edge e of G corresponds to a bundle of $m(e)$ edges in $M(P)$. We add a vertex in the middle of every edge of the bundle and space these vertices along a small line segment drawn perpendicular to e at its midpoint using the ordering of the edges in the rotation system of $M(P)$.

We complete the construction of P_ε by altering the drawing of $M(P)$ to spread apart the coincident vertices of P'. For each vertex v of $M(P)$, construct a small disc D of radius ε centered at v in the drawing. The edges that enter D are incident to v, and they cross the boundary of D in rotation system ordering. Let D_e be the point where edge e enters disc D. Suppose the Eulerian tour W visits v, entering on edge e and leaving on edge f. In the drawing and in W replace segments $D_e v$ and $v D_f$ by the chord $D_e D_f$. If two of these chords of D cross, they would correspond to a vertex crossing in W. Thus the result is a simple polygon P_ε. ◀

B Details for Section 3: Common Framework

Proof of Theorem 2, assuming Theorem 3. Consider an instance of Graph-Enclosure-with-Penalties, with simple connected plane graph G, required faces R, and optional faces O. We reduce to an instance of Enclosure-with-Penalties.

Find a straight-line plane embedding G' of G with the same combinatorial map (i.e., preserving the rotation system). The faces of G', including the outer face, become the polygons for our new instance. Observe that all these polygons are almost-simple. For the bounded faces of G' we preserve the partition into R and O and the penalties. The outer face of G' becomes an unbounded polygon. We put it in the set O with a penalty of 0 (the penalty is irrelevant, since no weakly simple polygon W can contain the unbounded polygon).

The free space of this set of polygons has no interior; it consists only of the edges of G'. Each such edge lies between two faces (polygons) so it is a squeezed edge and we assign it the weight of the corresponding edge of G.

This completes the reduction. For a graph on n vertices, the reduction produces a set of polygons with $O(n)$ vertices. The number k of required objects remains the same. The reduction takes $O(n)$ time. The runtime claim in Theorem 2 follows.

There is a one-to-one correspondence between weakly simple polygons in the free space and weakly simple closed walks in G (as defined in Appendix A), and the interior and the cost are preserved. Therefore a solution to the resulting instance of the Enclosure-with-Penalties problem provides a solution to the original graph problem. ◀

C Details for Section 4: Dynamic Programming Algorithm

C.1 Runtime of the dynamic programming algorithm

The number of subproblems of type M is $O(n^3 2^k)$: there are $O(n^2)$ choices for the mouth pq, 2^k choices for the set B, and $O(n)$ choices for the parameter t. The number of subproblems of type C is only $O(n^2 2^k)$, by the same analysis. Thus, the space requirement is $O(n^3 2^k)$.

The recursion that dominates the runtime is (7) for M_2. For fixed parameters pq, t, and B, there are at most n choices for the point r, at most $t = O(n)$ possibilities for t_1 and t_2, and at most $2^{|B|}$ choices for B_1 and B_2. We run through the possible choices of r in an outer loop. Then, for each triangle $\Delta = prq$, we can determine the reference points that lie in Δ in a straightforward way in $O(n)$ time, and this runtime will be dominated by the inner loop. This leads to an overall runtime of

$$O(n^3) \times \sum_{B \subseteq R} n \times \left(O(n) + O(n) \times 2^{|B|}\right) = O(n^5) \sum_{B \subseteq R} 2^{|B|} = O(n^5) \sum_{i=0}^{k} \binom{k}{i} 2^i = O(n^5 3^k).$$

We assume that we can access each of the $O(n^3 2^k)$ entries of the dynamic programming table in constant time. In particular, a memory word contains at least k bits, and hence set operations on subsets of R take constant time.

The above computation assumes that we can determine in $O(1)$ time, given two input vertices p and q, whether pq is a free-space edge. For this purpose, we precompute a Boolean array of size $n \times n$, with rows and columns indexed by the vertices, storing this information. The array can be determined in $O(n^2)$ time from the visibility graph of the input polygons, which can be computed in $O(n \log n + e) = O(n^2)$ time for a visibility graph with e edges [17].

C.2 Details for Section 4.2: Extracting the solution

Defining W_{DP}. With each finite value $C(p,t,B)$ that is computed in the recursions (2)–(5), we can naturally associate a polygon $W = W(p,t,B)$, a closed walk of free-space edges that goes through p. Similarly, with each finite value $M(pq,t,B)$ computed in the recursions (6)–(7), we can associate an open walk of free-space edges $W = W(pq,t,B)$ that goes from p to q. For example, in (7), where we form the sum $M(pr,t_1,B_1) + M(rq,t_2,B_2)$, the open walk W is obtained by concatenating the open walks associated with $M(pr,t_1,B_1)$ and $M(rq,t_2,B_2)$.

▶ **Definition 17.** *For an open walk W from p to q, we define $\overline{W}$ to be the polygon $W + qp$. For a closed walk W, we define $\overline{W}$ to be the polygon W itself.*

By remembering for each recursion the values t_1, t_2, B_1, B_2, etc. from which the minimum was obtained, we can recursively reconstruct the associated open/closed walks W and the polygons $\overline{W}$ in $O(t)$ time.

This formalizes the definition of W_{DP} (Definition 4).

Proving Lemma 6 about the Properties of W_{DP}. We must extend the definition of cost to open walks:

▶ **Definition 18.** *For an open or closed polygon W in the free space,*

$$c(W) := w(W) + \sum_{P\in O} \operatorname{wind}(\overline{W}, r_P)\cdot \pi_P. \tag{12}$$

In particular, if W is an open walk, we take winding numbers with respect to its closure $\overline{W}$.

This definition agrees with the previous Definition 5 when W is closed. For a reference point r_P lying on the mouth qp of an open walk W from p to q, we compute $\operatorname{wind}(\overline{W}, r_P)$ as if r_P were slightly moved to the right of the segment qp, i.e., in the direction where the outside would normally be in case of a counterclockwise simple polygon. In case of a weakly simple polygon $\overline{W}$, this means that points on the mouth are not considered to be enclosed.

To prove Lemma 6, we need results on winding numbers as walks are glued together. We first define gluing more precisely. Two closed walks W_1 and W_2 can be glued together at a common vertex, or along a common edge that is traversed in opposite directions by W_1 and W_2.

More formally: If $W_1 = (p, q_1, q_2, \dots, q_n)$ and $W_2 = (p, p_1, p_2, \dots, p_m)$, then the result of gluing the walks along the common point p is $P = (p, q_1, q_2, \dots, q_n, p, p_1, p_2, \dots, p_m)$.

If $P_1 = (p, q, q_2, \dots, q_n)$ and $P_2 = (q, p, p_2, \dots, p_m)$ both use the edge pq, but in opposite directions, then $P = (q, q_2, \dots, q_n, p, p_2, \dots, p_m)$ is the result.

We have the following easy but key property:

▶ **Lemma 19** (Additivity of Winding Numbers)**.** *The winding number is additive with respect to the gluing operation: If P is a closed walk obtained by gluing two closed walks P_1 and P_2 along a common edge or vertex, then*

$$\operatorname{wind}(P,x) = \operatorname{wind}(P_1,x) + \operatorname{wind}(P_2,x)$$

for all points x that do not lie on P_1 or P_2.

Proof. Let ρ be any ray from x to the unbounded face that avoids the vertices of P and intersects the edges of P_1 and P_2 transversally.

Assume first that P results from gluing P_1 and P_2 at a common vertex; then the multiset of the directed edges of P is exactly the union of the directed edges of P_1 and of P_2 (counting multiplicities). Let r^+, r_1^+, and r_2^+ be the number of times an edge of P, P_1, and P_2, respectively, crosses ρ from right to left; we have $r^+ = r_1^+ + r_2^+$. Similarly, with the analogous notations r^-, r_1^-, and r_2^- counting the number of crossings from left to right, we have $r^- = r_1^- + r_2^-$. Summing up, we obtain the result.

If P results from gluing P_1 and P_2 at a common edge pq, the effects of the two oppositely oriented edges pq and qp cancel out when the winding number is computed. (They contribute to r_1^+ and r_2^-, or to r_1^- and r_2^+, or not at all.) The proof for the first case carries over. ◀

We now restate and prove Lemma 6.

▶ **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}, $\mathrm{wind}(W_{\mathrm{DP}}, x) \geq 0$.

Proof. We prove by induction that the properties hold more generally for all subproblems solved in the dynamic programming algorithm. To be precise, consider a finite value $C(p,t,B)$ or $M(pq,t,B)$ computed in the recursions (2)–(7). Let $W = W(p,t,B)$ or $W = W(pq,t,B)$ be the closed or open walk associated with the solution and let $\overline{W}$ be the associated polygon as in Definition 17. We prove by induction on t that:

(i) $C(p,t,B) = c(W(p,t,B))$, $M(pq,t,B) = c(W(pq,t,B))$;
(ii) for all $P \in B$, $\mathrm{wind}(\overline{W}, r_P) = 1$ and for all $P \in R \setminus B$, $\mathrm{wind}(\overline{W}, r_P) = 0$;
(iii) for all points x that do not lie on $\overline{W}$, $\mathrm{wind}(\overline{W}, x) \geq 0$.

These properties hold in the base case (2), where $C(p,t,\emptyset) = 0$ and W is the single point p. For the general formulas we heavily rely on the additivity of the winding number with respect to gluing, Lemma 19. The cases are as follows, numbered by the equation numbers; it may help to refer to Figure 4.

(4) $C = C_1 = w_{pq} + M(qp, t-1, B)$ where pq is a free space edge.
By induction, the properties hold for the open walk $W_0 = W(qp, t-1, B)$. Let W be the polygon associated with C, i.e., $W = pq + W_0$. Observe that W is the same polygon as $\overline{W}_0$. This takes care of properties (ii) and (iii). For property (i), note that $c(W) = w_{pq} + c(W_0)$. By induction, $c(W_0) = M(qp, t-1, B)$. Thus $c(W) = w_{pq} + M(qp, t-1, B) = C$, which proves property (i).

(5) $C = C_2 = C(p, t_1, B_1) + C(p, t_2, B_2)$ where $t = t_1 + t_2$, $B = B_1 \sqcup B_2$, $B_1, B_2 \neq \emptyset$.
By induction, the properties hold for the polygons $W_1 = W(p, t_1, B_1)$ and $W_2 = W(p, t_2, B_2)$. The polygon W associated with C is formed by gluing W_1 and W_2 at the common point p. The weights are additive by definition: $w(W) = w(W_1) + w(W_2)$, and by additivity of winding numbers, $\mathrm{wind}(W, x) = \mathrm{wind}(W_1, x) + \mathrm{wind}(W_2, x)$ for all points x not on W. Property (iii) follows immediately, and property (i) follows by the definition of the cost, $c(W) = w(W) + \sum_{P \in O} \mathrm{wind}(W, r_P) \cdot \pi_P$.
Property (ii) propagates from B_1 and B_2 to their disjoint union B by the additivity of winding numbers. More precisely, consider first some $P \in B$. Since $B = B_1 \sqcup B_2$, the polygon P is in exactly one of these sets. Suppose without loss of generality that $P \in B_1$. By induction, $\mathrm{wind}(W_1, r_P) = 1$ and $\mathrm{wind}(W_2, r_P) = 0$. Thus $\mathrm{wind}(W, r_P) = 1$. Finally, if $P \in R \setminus B$, then by induction $\mathrm{wind}(W_1, r_P) = 0$ and $\mathrm{wind}(W_2, r_P) = 0$, so $\mathrm{wind}(W, r_P) = 0$, as required.

(6) $M = M_1 = C(p, t-1, B) + w_{pq}$ where pq is a free space edge.
By induction, the properties hold for the polygon $W_0 = W(p, t-1, B)$. The open walk W that is associated with M starts at p, traverses the polygon W_0 and then the edge pq, ending at q. $\overline{W}$ is formed by gluing the doubled edge qp to the polygon W_0. Thus, winding numbers with respect to $\overline{W}$ are the same as for W_0, except that they become undefined for points x on pq. This proves properties (ii) and (iii), and also that the penalty term in the cost (12) for W is the same as for W_0. Since $w(W) = w(W_0) + w_{pq}$, property (i) follows.

(7) $M = M_2 = M(pr, t_1, B_1) + M(rq, t_2, B_2) + \pi(\Delta)$ where $\Delta = prq$ is a counterclockwise triangle, $t = t_1 + t_2, t_1 \geq 1, t_2 \geq 1$, $B = B_1 \sqcup B_2 \sqcup R(\Delta)$.
By induction, the properties hold for the open walks $W_1 = W(pr, t_1, B_1)$ and $W_2 = W(rq, t_2, B_2)$. Let W be the open walk associated with M. Then $w(W) = w(W_1) + w(W_2)$. $\overline{W}$ is formed by gluing $\overline{W}_1$ and $\overline{W}_2$ to Δ on the common edges pr and rq, respectively. The argument is analogous to the treatment of (5) above, except that we form the combination of *three* areas, and two gluings are performed, along common edges instead of common vertices.
An important point is therefore the treatment of reference points that lie *on* these edges: By the convention established in connection with Definition 18, the points on the mouths pr and rq are not considered to be enclosed by $\overline{W}_1$ and $\overline{W}_2$, both for determining $R(\overline{W}_i)$ and for computing $\pi(\overline{W}_i)$. However, when determining $R(\Delta)$ and $\pi(\Delta)$, these edges are considered to be part of Δ, by our conventions of Section 4.1 (in the paragraph before (6)). Thus the points on the mouth are neither overcounted nor undercounted.
The edge pq is *not* considered as part of Δ. This is in line with the convention of Definition 18 that the mouth pq should not be counted as enclosed by $\overline{W}$. ◀

D Details for Section 5: Uncrossing Algorithm

We first give more details about the following proposition (restated).

▶ **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.*

As noted in the main text, a linear-time algorithm for constructing such an Euler tour was given by Akitaya and Tóth [3, Corollary 1]. Their algorithm makes the unstated assumption that the combinatorial map is given. Their terminology differs from ours, e.g., their input is geometric, and their output is a simple polygon that ε-approximates a non-crossing Euler tour. We outline the idea of the algorithm using our terminology. For the more general setting of graphs on arbitrary surfaces, a linear time algorithm was recently described by Bulavka, Colin de Verdière, and Fuladi [6, Lemma 4.2 of the full version on arXiv], expressed in the framework of cross-metric surfaces.

Idea of the proof of Proposition 7. Take a 2-coloring (white and grey) of the faces of H, with the outer face colored white. Traverse every grey face counterclockwise to obtain a set of edge-disjoint cycles without vertex crossings. The plan is to stitch together these cycles to form a non-crossing Euler tour. Initialize T to one of the cycles. While there are other cycles, find a vertex v where an edge of T and an edge of another cycle C appear consecutively in the cyclic order of edges around v, and merge C into T at this point. This does not create vertex crossings in T. The algorithm can be implemented to run in $O(m)$ time. ◀

We next give more details of our uncrossing algorithm, restated here.

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

1. *Subdivide every edge of W at every interior vertex 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'.*

The definitions and results of Appendix A allow us to clarify this. For Step 1 we generalize the notion of an image graph to a polygon that may have interior crossings: first subdivide edges at interior crossings and then apply the previous definition of an image graph. In Step 1 we compute this image graph together with the multiplicity function. Step 2 simply modifies the multiplicities. In Step 3, the claim that a non-crossing Euler tour corresponds to a weakly simple polygon W' is justified by Lemma 16.

▶ **Lemma 20.** *The Uncrossing Algorithm can be implemented to run in time $O(t \log t + s)$ where t is the number of edges of W and $s \in O(t^2)$ is the number of interior crossing points of W. For input W_{DP} the runtime is $O(n \log n)$.*

Proof. We first show that the image graph G and multiplicities $m(e)$ can be computed in time $O(t \log t + s)$. We must be careful to avoid the quadratic blow-up that results if we construct G in the obvious way by first subdividing edges of W at forks.

One approach is to perform a plane sweep and represent overlapping segments in terms of multiplicities to avoid explicitly subdividing all edges in an overlapping bundle when one of those edges ends at a vertex.

Another approach (following ideas in [2, 10]) is to compute multiplicities before running a plane sweep. Sort by slope to partition the edges into collinear groups. If ℓ is a line that contains m edges, we can sort their endpoints along ℓ in time $O(m \log m)$ and output a corresponding set of $O(m)$ interior-disjoint edges with multiplicities. We then run plane sweep on the new edges to compute G and its multiplicities in time $O(t \log t + s)$.

Note that the image graph G (which is a simple plane graph) has at most $t + s$ vertices, hence $O(t + s)$ edges. The plane connected Eulerian multigraph M created in Step 2 (with edge multiplicities 1 or 2) has $O(t + s)$ edges, and by Proposition 7, a weakly simple Euler tour of M can be found in time $O(t + s)$.

Finally, consider running the algorithm on W_{DP}. W_{DP} has at most $6n$ edges which gives an immediate runtime bound of $O(n^2)$. In fact, the runtime is less (though note that the runtime of the dynamic program dominates in any case). As a consequence of the optimality of W_{ALG} we prove (in Corollary 21) that W_{DP} has no interior crossing points. Thus the algorithm to uncross W_{DP} runs in time $O(n \log n)$. ◀

We note that Akitaya and Tóth [3, Theorem 4] prove a related uncrossing result. Their input polygon has t edges and no interior crossings and they "uncross" to a weakly simple polygon with the same multiplicities as W and with $O(t)$ edges, rather than the obvious quadratic number. They do not give a runtime. By contrast, we allow interior crossings (at a quadratic cost), and we escape the quadratic blow-up due to forks in a simpler way because we only care about parity.

The example of Figure 10 shows that self-crossings are not just a theoretical possibility; they actually can occur in an optimal solution to a type-M subproblem. The solutions in this example are *weakly simple immersed polygons*, see Appendix L.

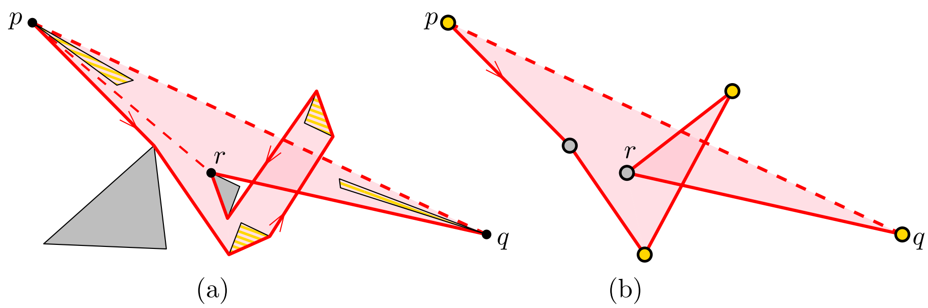


Figure 10 (a) The optimum solution with given mouth pq can indeed self-overlap. The yellow objects are required, and the grey objects have high penalties. The solution with mouth pr is a simple polygon. Attaching the triangle prq to it yields a self-overlapping polygon. (b) The same example with point objects. Observe that the solution covers more than $360°$ around r, although this is not apparent locally from looking at the boundary near r.

E Details for Section 6: Correctness Proof

We restate and prove Lemma 13.

▶ **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. Recall that $\mathcal{S}$ is the discrete set of feasible solutions that consist of free-space edges each traversed at most twice. There are two parts of the proof that warrant more detail than was given in the main text: (Part 1) $\mathcal{S}$ contains a feasible solution that encloses R and excludes O; and (Part 2) if W is a finite cost feasible solution outside $\mathcal{S}$, then there is a solution W' in $\mathcal{S}$ of no greater cost.

Part 1. We begin with the boundaries of the polygons in R traversed counterclockwise. These form a collection of k weakly simple polygons in the free space such that each free-space edge is used at most twice. As long as there is more than one polygon, combine polygons as follows. If two polygons share an edge, join them by removing that edge. Otherwise, if there are polygons that share a vertex, find a vertex v where two polygons appear consecutively in the cyclic order of edges around v, and merge the polygons at v. Otherwise, find a shortest path among all paths in the free space that connect two vertices of different polygons, and combine the two polygons into a single polygon by traversing the path once in each direction. After $k-1$ steps, the process stops with a single weakly simple polygon, and this polygon has the desired properties.

Part 2. We now prove that if W is a finite cost feasible solution outside $\mathcal{S}$, then there is a solution W' in $\mathcal{S}$ of no greater cost. The idea is to construct a weakly simple polygon W' in $\mathcal{S}$ that encloses the same objects as W and does not increase the sum of edge weights.

W may have vertices that are not object vertices. Let W_h be the result of homotopically shortening (i.e., in the free space) every subpath of W that goes from one object vertex to another. (If W contains no object vertex, we homotopically shorten all of W.) Then W_h is composed of free-space edges and $w(W_h) \leq w(W)$—this holds for edges with Euclidean weights and also for squeezed edges. Although W_h need not be weakly simple, every object has the same winding number (1 or 0) in W and W_h.

Let W' be the result of applying the Uncrossing Algorithm 8 to W_h. Then W' is a weakly simple polygon composed of free-space edges each used at most twice, so W' lies in $\mathcal{S}$. By Lemma 9, W' preserves the winding numbers so W' encloses the same objects as W. Finally, $w(W') \leq w(W_h)$. ◀

We note the following consequence of the above proof. It is used to analyze the runtime of the uncrossing algorithm but nowhere else.

▶ **Corollary 21.** *W_{DP} has no interior crossing.*

Proof. If W_{DP} had an interior crossing point, then the uncrossing algorithm (Algorithm 8) would produce a walk W_{ALG} with a vertex at that crossing point, which is not a vertex of an input polygon. The homotopic shortening step of Part 2 above would then strictly decrease the weight, and thus the cost, of the solution, a contradiction to the optimality of W_{ALG}. ◀

Finally we restate and prove Lemma 15. Recall the concepts of an open walk W_0, its closure $\overline{W}_0$ and cost $c(W_0)$ from Definitions 17 and 18.

▶ **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 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 $t \geq \ell$, $M(pq, t, B) \leq c(W_0)$.*

Proof. Let $M(W)$ be a certificate that W is weakly simple, i.e., $M(W)$ is an image multigraph in which W corresponds to a non-crossing Euler tour, see Appendix A. Via this correspondence, each edge of W has an interior face of $M(W)$ to its left, which provides a partition of the edges of W into faces of $M(W)$. We use the cyclic order of edges around faces in the proof. The reader may find it helpful to refer to Figure 3.

As in the proof of Lemma 6, we go through each case of the dynamic program recursion. The difference is that in Lemma 6 we analyze, in terms of winding numbers, the cost of any polygon constructed by the dynamic program (potentially not weakly simple), whereas here we will deconstruct any weakly simple polygon into smaller pieces as defined by the appropriate recursion formula, and winding numbers do not come into play.

We prove claims (A) and (B) simultaneously by induction on ℓ. For part (A), see Figure 11.

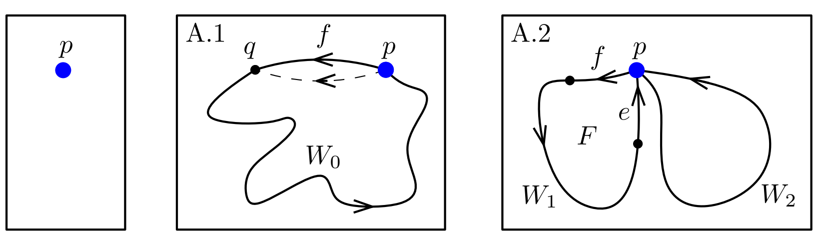


Figure 11 Cases for statement (A) of Lemma 15

In the base case, $\ell = 0$, polygon W degenerates to a single point, so only case (A) applies. For objects with interior, W cannot enclose any objects, so $B = \emptyset$. By Equation (2), $C(p, t, B) = 0 \leq c(W)$.

For part (A), the case $B = \emptyset$ was just dealt with, and for $B \neq \emptyset$ we distinguish two cases depending whether p is a transition vertex in W. Let $f = pq$ be the edge of W that follows p.

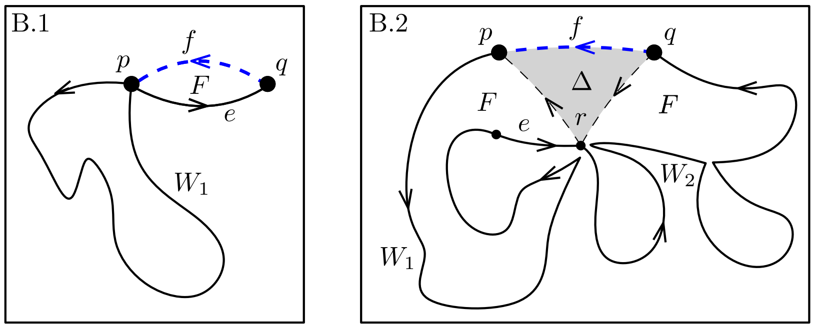


Figure 12 Cases for statement (B) of Lemma 15

Case A.1. p is not a transition vertex. Let W_0 be the open walk from q to p formed by removing the edge f from W. Then $\overline{W}_0 = W$ and W_0 has $\ell - 1 \leq t - 1$ free-space edges. By Equation (4), $C(p, t, B) \leq w_{pq} + M(qp, t-1, B)$, and by induction $M(qp, t-1, B) \leq c(W_0)$. Thus $C(p, t, B) \leq w_{pq} + c(W_0) = c(W)$, where the last equality comes from the definition of the cost of the open walk W_0.

Case A.2. p is a transition vertex. Let F be the face of $M(W)$ incident to edge $f = pq$ and let e be the edge of W that precedes f around F. Suppose edge e enters vertex r of W; then vertices r and p are coincident. Cut W into two polygons, where W_1 traverses W from p to r and W_2 traverses W from r to p. Observe that W_1 and W_2 are both weakly simple, and that no point is interior to both. For $i = 1, 2$, let ℓ_1 be the number of edges of W_i. Then $\ell_i > 0$ and $\ell_1 + \ell_2 = \ell$. Let $B_i = \{P \in O \mid r_P \text{ lies inside } W_i\}$. Then $B = B_1 \sqcup B_2$.

Suppose that neither B_1 nor B_2 is empty. Since $t = \ell_1 + (t - \ell_1)$, Equation (5) yields $C(p, t, B) \leq C(p, \ell_1, B_1) + C(p, t - \ell_1, B_2)$. By induction, $C(p, \ell_1, B_1) \leq c(W_1)$ and $C(p, t - \ell_1, B_2) \leq c(W_2)$. Thus $C(p, t, B) \leq c(W_1) + c(W_2) = c(W)$, where the last equality is because W_1 and W_2 partition the edges and the interior of W.

On the other hand, if some B_i, say B_2, is empty, then $B_1 = B$. By induction, $C(p, t, B) \leq c(W_1)$ since W_1 has $\ell_1 < t$ edges and contains B. Thus $C(p, t, B) \leq c(W_1) \leq c(W)$, where the last inequality is because W_1's edges and interior are contained in those of W.

For part (B), we distinguish two cases depending whether the interior face F of $M(W)$ incident to edge $f = qp$ of W is a corridor or a chamber, see Figure 12. Note that $B = \emptyset$ is allowed.

Case B.1. F is a corridor. Suppose q has incoming edge e and outgoing edge f. By assumption, q is not a transition vertex. Thus e is incident to face F, and forms the other side of the corridor. Since e is a free-space edge, so is f. By Equation (6), $M(pq, t, B) \leq C(p, t-1, B) + w_{pq}$. Let W_1 be the polygon formed by deleting edges e and f from W. Then W_1 is a weakly simple polygon of $\ell - 1$ free-space edges that encloses B. By induction, $C(q, t-1, B) \leq c(W_1)$. Thus $M(pq, t, B) \leq c(W_1) + w_{pq} = c(W_0)$, where the last equality is by definition of the cost of the open walk W_0.

Case B.2. F is a chamber. Take a triangulation of F (which exists because a chamber is an almost-simple polygon) and consider the triangle incident to edge $f = pq$. The triangle lies inside face F. Let v be the vertex of $M(W)$ that forms the third corner of the triangle. Vertex v may correspond to more than one polygon vertex, but we choose the "right" one as follows. Let e be the edge of face F incoming to v. In W, suppose edge e enters vertex r. Name the triangle $\Delta = pqr$.

Break W into two open walks W_1 from q to r and W_2 from r to p. Observe that $\overline{W}_1$ and $\overline{W}_2$ are weakly simple polygons and that r is not a transition vertex of $\overline{W}_1$. For $i = 1, 2$, let ℓ_i be the number of edges of W_i. Then $\ell_i > 0$ and $\ell_1 + \ell_2 = \ell$. Let B_1 be the set of polygons $P \in R$ with r_P in $\overline{W}_1$ but not on qr, and let B_2 be the set of polygons $P \in R$ with r_P in $\overline{W}_2$ but not on rp. These sets may be empty. Let $R(\Delta)$ be the set of polygons $P \in R$ with r_P inside Δ where we regard Δ as being closed on edges pr and qr and open on edge pq. Then $B = B_1 \sqcup B_2 \sqcup R(\Delta)$.

By Equation (7), $M(pq, t, B) \le M(pr, \ell_1, B_1) + M(rq, t - \ell_1, B_2) + \pi(\Delta)$. By induction, $M(pr, \ell_1, B_1) \le c(W_1)$ and $M(rq, t - \ell_1, B_2) \le c(W_2)$. Thus $M(pq, t, B) \le c(W_1) + c(W_2) + \pi(\Delta) = c(W_0)$ where the last equality is because we have partitioned the edges of W_0 and the interior of W. ◀

F Details for Section 7: Reducing the Runtime

F.1 Setting up a system of equations

Think of the recursions (2)–(7) when t is so large that it does not impose any constraint on the solution. Formally, we can define $C(p, B) := C(p, 6n, B)$ and $M(pq, B) := M(pq, 6n, B)$, where the right-hand sides of these equalities are the quantities from Section 4. Then t can be eliminated from the recursions (2)–(7), resulting in a *system of equations* between the quantities $C(p, B)$ and $M(pq, B)$, which express a mutual dependence between them:

$$C(p, \emptyset) = 0 \tag{13}$$

$$C(p, B) = \min\{C_1(p, B), C_2(p, B)\} \text{ for } B \neq \emptyset, \text{ where} \tag{14}$$

$$C_1(p, B) = \min\{\, w_{pq} + M(qp, B) \mid pq \text{ is a free space edge} \,\} \tag{15}$$

$$C_2(p, B) = \min\{\, C(p, B') + C(p, B'') \mid B = B' \sqcup B'';\ B', B'' \neq \emptyset \,\} \tag{16}$$

$$M(pq, B) = \min\{M_1(pq, B), M_2(pq, B)\}, \text{ where} \tag{17}$$

$$M_1(pq, B) = \begin{cases} w_{pq} + C(q, B), & \text{if } pq \text{ is a free space edge} \\ \infty, & \text{otherwise} \end{cases} \tag{18}$$

$$\begin{aligned} M_2(pq, B) = \min\{\, & M(pr, B') + M(rq, B'') + \pi(\Delta) \mid \\ & prq = \Delta \text{ is a counterclockwise triangle;} \\ & B = B' \sqcup B'' \sqcup \{P \in R \mid r_P \in \Delta\} \,\} \end{aligned} \tag{19}$$

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

$$c_{\mathrm{DP}} := \min\{\, C(p, R) \mid p \text{ is a vertex} \,\}. \tag{20}$$

By Lemma 13 and by the definition of $C(p, B)$ and $M(pq, B)$, the value of c_{DP} resulting from equation (20) is the same as the one resulting from equation (8).

Observe that, for the recursions (2)–(7), the absence of a cyclic dependence between the quantities $C(p, t, B)$ and $M(pq, t, B)$ is guaranteed by the second parameter t, which is always smaller on the right-hand side than on the left side. For equations (13)–(19), on the other hand, we need to argue differently. In terms of the parameter to represent the set of required objects, the parameter B, B', or B'' on the right-hand side is always a subset of the parameter B on the left-hand side. Whenever it is a strict subset, the corresponding equation cannot be part of a cyclic dependence. The recursion is more delicate when the same set B appears on the right-hand side. To separate these cases, we split M_2 into three parts M_3, M_4 and M_5 consisting of those compositions where $B' = B$, where $B'' = B$, and where both B' and B'' are strict subsets of B. Thus, equation (19) becomes:

$$M_2(pq, B) = \min\{M_3(pq, B), M_4(pq, B), M_5(pq, B)\}, \text{ where} \tag{21}$$

$$M_3(pq, B) = \min\big\{\, M(pr, B) + M(rq, \emptyset) + \pi(\Delta) \;\big|\; \Delta = prq \text{ is a counterclockwise triangle, } R(\Delta) = \emptyset \,\big\} \tag{22}$$

$$M_4(pq, B) = \min\big\{\, M(pr, \emptyset) + M(rq, B) + \pi(\Delta) \;\big|\; \Delta = prq \text{ is a counterclockwise triangle, } R(\Delta) = \emptyset \,\big\} \tag{23}$$

$$M_5(pq, B) = \min\big\{\, M(pr, B') + M(rq, B'') + \pi(\Delta) \;\big|\; \Delta = prq \text{ is a counterclockwise triangle; } B = B' \sqcup B'' \sqcup R(\Delta);\ B', B'' \subsetneq B \,\big\} \tag{24}$$

For $B = \emptyset$, the equations (22) and (23) coincide, but this redundancy is no problem.

Equations (16) and (24), defining the quantities $C_2(p, B)$ and $M_5(pq, B)$, have on the right-hand side quantities whose parameter, B' or B'', is a strict subset of B. Thus they cannot be involved in cyclic dependencies. On the other hand, equations (15), (18), (22), and (23), defining the quantities $C_1(p, B)$, $M_1(pq, B)$, $M_3(pq, B)$, and $M_4(pq, B)$, respectively, have on the right-hand side quantities whose parameter B is the same as the one on the left-hand side. By inspecting these equations, one can see that the left-hand quantity that is computed is always strictly bigger than the ingredients on the right-hand side, and hence the recurrences behave like a *superior context-free grammar* which uses strictly superior functions; see Knuth [18, Section 5]. Knuth proved that, for such a grammar, the minimum value of a string (representing a composition of functions) derived from each terminal symbol exists, is unique, and can be computed efficiently. In our problem, this translates to the fact that all the values $C(p, B)$ and $M(pq, B)$, where p and q are vertices and B is any subset of the input set of required polygons R, exist, are unique, and can be computed efficiently.

Knuth's setup does not directly apply to our problem as far as uniqueness is concerned, because our functions are not *strictly* superior functions. This is compensated by having positive additive terms in the recursion. Our uniqueness proof below (Lemma 22) is a straightforward adaptation of Knuth's proof to our situation.

We show that the system of $O(n^2 2^k)$ equations (13)–(18) and (21)–(24) has a unique solution $S = (C, M)$. (We do not consider the auxiliary quantities $C_1, C_2, M_1, M_2, M_3, M_4, M_5$ as part of the solution S, because they can be directly expressed in terms of C and M). This, together with the fact that the solution of equations (2)–(7) is a solution to the system, implies that the solution S is the same as the one coming from equations (2)–(7).

▶ **Lemma 22.** *The system of equations* (13)–(18) *and* (21)–(24) *has a unique solution* $S = (C, M)$ *with* $C(p, B) \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ *and* $M(pq, B) \in \mathbb{R}_{>0} \cup \{\infty\}$.

Proof. The existence of a solution follows by substituting the limiting solution of the equations (2)–(7) for large enough t. All quantities $M(pq, t, B)$ in those equations are positive because the quantity M_1 (see equation (6)) is at least equal to the weight w_{pq} of the mouth, which is positive, and the other term M_2 (see equation (7)) involves the addition of two quantities $M(pr, t_1, B_1)$ and $M(rq, t_2, B_2)$ whose second parameter, t_1 or t_2, is smaller than t.

We now prove uniqueness. The crucial fact that allows us to exclude a cyclic dependency is that in the equations (13)–(18) and (21)–(24), the quantities M and C on the right-hand side that could cause such a cyclic dependency (because they use the same set parameter B) must be strictly smaller than the quantities on the left side that are defined through them.

Assume, for contradiction, that there are two different solutions $S = (C, M)$ and $S' = (C', M')$. Among the quantities where the two solutions differ, select the ones for which the parameter B is minimal, and among those, consider a pair with the smallest value $T = \min\{C(p, B), C'(p, B)\}$ or $T = \min\{M(pq, B), M'(pq, B)\}$.

Let us first deal with the case that the smallest difference occurs for $C(p, B) \neq C'(p, B)$. Assume without loss of generality that $T = C(p, B) < C'(p, B)$. By the minimality of B, we have that $C(p, B') = C'(p, B')$ and $C(p, B'') = C'(p, B'')$, for any strict subsets B' and B'' of B. Hence, equation (16) gives us that $C_2(p, B) = C'_2(p, B)$ and thus $T = C_1(p, B) < C'_1(p, B)$. By equation (15), we have that $C_1(p, B)$ is equal to $w_{pq} + M(qp, B)$ for some free-space edge pq. Since the weight w_{pq} is positive, $M(qp, B)$ is strictly smaller than T, and hence, by the minimality of T, we have $M(qp, B) = M'(qp, B)$. It follows that

$$C_1(p, B) = w_{pq} + M(qp, B) = w_{pq} + M'(qp, B) \geq C'_1(p, B),$$

a contradiction.

The same argument works for the case in which $T = \min\{M(pq, B), M'(pq, B)\}$. Here it is necessary to use the fact that all values $M(pq, B)$ are positive. (Without this assumption, the identically zero solution $M(pq, B) \equiv 0$ might be an alternative solution, for example.) ◀

F.2 The algorithm

We now describe the algorithm to compute the values $C(p, B)$ and $M(pq, B)$ for all vertices p and q and all the subsets $B \subseteq R$ of required polygons. The algorithm has an outer loop that goes through all subsets $B \subseteq R$ in order of increasing size $|B|$, or in any other order that is compatible with set inclusion.

When the algorithm needs to compute the values $M(pq, B)$ and $C(p, B)$ for a certain B, the values $M(rs, B')$ and $C(r, B')$ have already been computed for all strict subsets $B' \subset B$, all vertex pairs rs and all vertices r. This allows us to compute the values $C_2(p, B)$ for all vertices p, via equation (16), and $M_5(pq, B)$ for all vertex pairs pq, via equation (24).

The algorithm maintains a set F_1 of vertices p for which the value $C(p, B)$ has been determined, and a set F_2 of vertex pairs pq for which the value $M(pq, B)$ has been determined. Initially, F_1 and F_2 are empty. The algorithm also maintains *tentative* values $M(pq, B)$ and $C(p, B)$, which are upper bounds on their final values. When they become final, the corresponding item pq or p is added to F_2 or F_1. Actually, what the algorithm maintains are tentative values for $M_1(pq, B), M_3(pq, B), M_4(pq, B)$ and $C_1(p, B)$, which are initialized to ∞. The values of $C(pq, B)$ and $M(pq, B)$ are kept up-to-date via $C(p, B) = \min\{C_1(p, B), C_2(p, B)\}$ and $M(pq, B) = \min\{M_1(pq, B), M_3(pq, B), M_4(pq, B), M_5(pq, B)\}$.

The core of the algorithm consists in making a tentative value final and adding the corresponding vertex or vertex pair to F_1 or F_2. The strategy to do so is akin to the strategy of Dijkstra's algorithm for computing shortest paths: We pick the smallest tentative value and make it final. Then we look at all equations where this value appears on the right-hand side, and update the left-hand side. The pseudo-code in Algorithm 1 implements this in a straightforward way. (The only challenge is the confusion caused by the necessary renaming of the vertices p, q, r, s.)

Correctness is established in the same way as for Dijkstra's algorithm. The smallest tentative value D that is determined at the beginning of each iteration of the main loop is simultaneously a lower bound on all tentative values and an upper bound on all permanent values. The algorithm ensures that every tentative value always fulfills its corresponding equation (14) or (17). Thus, whenever a value is finalized, the equation is fulfilled. The

Algorithm 1 Computation of the values $M(pq, B)$ and $C(p, B)$ for fixed B

Input : Set R of required polygons, set O of optional polygons, set $B \subseteq R$
Output : Values $M(pq, B)$ for every pair of vertices pq and $C(p, B)$ for every vertex p

// For every strict subset $B' \subset B$, the values $M(rs, B')$ for every pair of vertices rs and $C(r, B')$ for every vertex r have already been computed.
for each vertex p **do**
  Set $C_1(p, B) := \infty$; compute $C_2(p, B)$ by equation (16); set $C(p, B) := C_2(p, B)$
for each vertex pair pq **do**
  Set $M_1(pq, B) := M_3(pq, B) := M_4(pq, B) := \infty$, and compute $M_5(pq, B)$ by equation (24); set $M(pq, B) := M_5(pq, B)$ according to (17) and (21)
Set $F_1 := F_2 := \emptyset$ *// F_1 contains the vertices p for which $C(p, B)$ has been computed, and F_2 the vertex pairs pq for which $M(pq, B)$ has been computed.*
while there are vertices not in F_1 or vertex pairs not in F_2 **do**
  Find the smallest value D among the tentative values $C(p, B)$ with $p \notin F_1$ and the tentative values $M(pq, B)$ with $pq \notin F_2$; ties are broken arbitrarily.
  if D is $C(p, B)$ **then**
    Set $F_1 := F_1 \cup \{p\}$ *// make $C(p, B)$ permanent*
    for each free-space edge sp incident to p **do**
      Set $M_1(sp, B) := \min\{M_1(sp, B), w_{sp} + C(p, B)\}$ *// by equation (18)*
      Update $M(sp, B) = \min\{M_1(sp, B), M_3(sp, B), M_4(sp, B), M_5(sp, B)\}$
  if D is $M(pq, B)$ **then**
    Set $F_2 := F_2 \cup \{pq\}$ *// make $M(pq, B)$ permanent*
    if pq is a free-space edge **then**
      Set $C_1(q, B) := \min\{C_1(q, B), w_{qp} + M(pq, B)\}$ *// by equation (15)*
      Update $C(q, B) := \min\{C_1(q, B), C_2(q, B)\}$
    for each counterclockwise triangle $\Delta = psq$ with $R(\Delta) = \emptyset$ **do**
      Set $M_3(ps, B) := \min\{M_3(ps, B), M(pq, B) + M(qs, \emptyset) + \pi(\Delta)\}$ *// by (22)*
      Update $M(ps, B) := \min\{M_1(ps, B), M_3(ps, B), M_4(ps, B), M_5(ps, B)\}$;
      Set $M_4(sq, B) := \min\{M_4(sq, B), M(sp, \emptyset) + M(pq, B) + \pi(\Delta)\}$ *// by (23)*
      Update $M(sq, B) := \min\{M_1(sq, B), M_3(sq, B), M_4(sq, B), M_5(sq, B)\}$

values on the right-hand side on which it depends do not change any more because they are smaller than D, and hence they have already been finalized.

Thus, when the algorithm terminates, the computed values $C(p, B)$ and $M(pq, B)$ fulfill (14) and (17). The correct values $C(p, 6n, B)$ and $M(pq, 6n, B)$ also fulfill these equations, and by Lemma 22 the solution of (14) and (17) is unique, and hence the computed values agree with the correct values.

F.3 Runtime analysis

Clearly, there are $O(n^2 2^k)$ values $M(pq, B)$ and $C(p, B)$, and this defines the space complexity.

We first analyze the computations of the quantities $C_2(p, B)$ and $M_5(pq, B)$, which are computed directly by equations (16) and (24), respectively. The dominating term for the runtime comes from the computation of the $O(n^2 2^k)$ quantities $M_5(pq, B)$. For each of them, we have to run through all points r and check each counterclockwise triangle $\Delta = prq$: We have to find the set

$$R(\Delta) := \{P \in R \mid r_P \in \Delta\}, \tag{25}$$

and compute the sum $\pi(\Delta)$ of the penalties of the polygons in O whose reference point is in Δ and, in case $R(\Delta) \subseteq B$, run through all partitions of $B - R(\Delta)$ into two sets B' and B''. We describe below a preprocessing step that allows us to obtain the quantity $\pi(\Delta)$ in constant time. The set $R(\Delta)$ can be trivially computed in $O(k)$ time. Thus the total running time for computing the quantities $M_5(pq, B)$ is

$$n^2 \sum_{B \subseteq R} \Big(n \times \big(O(k) + 2^{|B|} \big) \Big) = O(n^3 2^k k) + O(n^3) \sum_{B \subseteq R} 2^{|B|} = O(n^3 3^k). \tag{26}$$

Let us now look at the running time of the core of the algorithm, in which, repeatedly, a tentative value is made final. Consider a fixed subset $B \subseteq R$ (there are 2^k such sets). We need to maintain a priority queue for the $O(n^2)$ tentative values for the quantities $C(p, B)$ and $M(pq, B)$. Each of the $O(n^3)$ expressions on the right-hand side of any of the equations (15), (16), (18), and (22)–(24) is evaluated exactly once (when the corresponding quantity becomes final) or twice (in case $B = \emptyset$, for the expression $M(pq, \emptyset) + M(pq, \emptyset)$). The evaluation potentially triggers an update to the priority queue, which takes $O(1)$ amortized time with Fibonacci heaps [11, 16]. We need to extract the minimum $O(n^2)$ times, at an amortized cost of $O(\log n)$ per operation. The overall runtime for the heap operations is then $O(n^2 \log n + n^3) = O(n^3)$. In summary, the overall runtime for this part of the algorithm is $2^k \cdot O(n^3)$, which is dominated by (26).

Thus, up to showing how $\pi(\Delta)$ can be determined in constant time, we have established Theorem 3. ◀

F.4 Preprocessing for quickly determining the penalty of a triangle

One can set up a table with $O(n^2)$ entries from which, for any triangle Δ whose vertices are vertices of the input polygons, the sum of the penalties of the polygons whose reference point is in Δ can be obtained in constant time. This is a standard technique in this area, see for example [14, Section 2]. We give some details.

A ***plank*** is a region bounded by a line segment pq and two vertical upward rays or two vertical downward rays. We consider the right boundary ray and the open line segment pq to be part of the plank, but not the left boundary ray including the point p. Figure 13 shows some downward planks ("bottomless trapezoids", so-to-speak). Upward planks (or "topless trapezoids") are used in Section 8.

We store for each vertex pair pq, the sum of the penalties in the downward plank below the segment pq. From this, the same data $\pi(\Delta)$ can then be computed for any triangle $\Delta = abc$ in constant time by addition and subtraction from three planks, see Figure 13 for an example. The table can be computed in $O(n^2)$ time and $O(n^2)$ space [14, Theorem 2.1]. Even a straightforward $O(n^3)$ preprocessing would be acceptable for us, as the running time is dominated by the cost of other parts of the algorithm.

Reference points of objects with infinite penalties are handled separately, in the same way: Instead of storing the sum of their penalties, they are merely *counted*, to determine whether the triangle in question contains at least one of them or none. Finally, recall that $\Delta = prq$ was defined to be open on segment pq. To handle this, we also calculate the sum the penalties of the reference points *on* each segment pq (as well as the number of infinite-penalty points). These have then to be added or subtracted as appropriate.

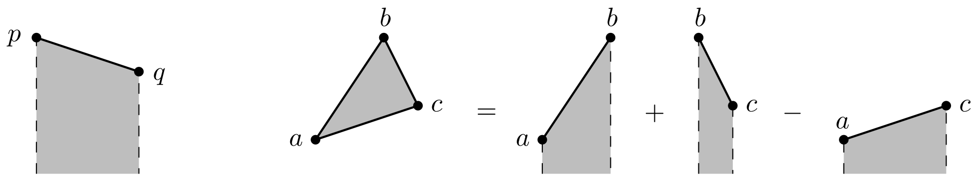


Figure 13 Left: a plank under the segment pq. Right: A triangle area abc is obtained by addition and subtraction of planks.

G Details for Section 8: Adapting the Correctness Proof for the Inverted Problem

To apply the arguments from Section 6, we have to extend the notion of winding number to regions $W^{\updownarrow}$ whose boundary includes an upward and a downward vertical ray. We pick an anchor point X_- to the left of all object vertices. We define $\text{wind}(W^{\updownarrow}, X_-) = 1$, and define the winding number for other points relative X_- by connecting them by a curve to X_- and counting signed intersections. It follows that $\text{wind}(W^{\updownarrow}, x) = 0$ for any point x that is sufficiently far to the right. The winding number of planks must also be defined appropriately: The winding number of $S_{\downarrow}(pq)$ or $S_{\uparrow}(pq)$ is 1 between the two rays, on the "correct" side of the segment pq, and 0 otherwise.

When the region is finished off by adding a right halfplane, W becomes a closed clockwise cycle. By the usual conventions, the interior has winding number -1 and the exterior has winding number 0. The winding number that we are using has an additive offset of 1, due to the stipulation that the point X_- has winding number 1. Thus, according to our convention, the winding number is 1 outside W and 0 inside W, which is precisely what we need because the objects whose presence is checked and whose penalties are added are the objects outside W.

The correctness proof in Lemma 15 must be adapted as follows. In the inductive step, we have a solution $W^{\updownarrow}$ consisting of a finite walk W from p to q and two vertical rays. The case that $p = q$ and the walk W is trivial is handled by formula (9) for U_-. Suppose that $p \neq q$ and, w.l.o.g., p is not the leftmost point of W. Then we take the edge pr on the lower convex hull of W. The formula (10) for $U_{\downarrow}$ shows that $W^{\updownarrow}$ can be reduced to smaller pieces.

There is another issue that we have to address, namely the partial solutions of type C ("closed") that are incident to the convex hull, such as the pieces Q_a, Q_d, Q_f in Figure 6, are properly handled. Every convex hull edge, such as the edge p_6p_7, appears once as a mouth in the recursion. As can be checked in Figure 8, it appears always in counterclockwise direction along the boundary. For example, the edge p_6p_7 appear in a subproblem of the form $M(p_6p_7, t, B)$, for appropriate parameters B and t. According to Lemma 15, this problem will consider partial solutions in which p_6 is not a transition vertex. However, there is no such restriction on p_7. Thus we assign all type-C pieces hanging off p_7 to this subproblem. (In the example, there is only one such piece, Q_d.)

The general strategy is as follows. We cut the solution walk W into pieces at the convex hull vertices, and assign a piece to each convex hull edge, which acts as the mouth of the piece. Pieces of type C that start and end at a hull vertex p_i are assigned to the hull edge $p_{i-1}p_i$ clockwise from p_i. In this way, the pieces are uniquely defined, and we have ensured that for each mouth p_ip_{i-1}, p_{i-1} is never a transition vertex of the respective piece. Thus, by Lemma 15, the weight of the corresponding piece is an upper bound on $M(p_ip_{i-1}, t, B)$ with the appropriate parameters t and B. The remainder of the proof, regarding the coverage of outside the convex hull by adding halfspaces and planks, is straightforward.

The example of Figure 7 illustrates that the region covered by the planks need not actually be the outside of the convex hull of the solution polygon: Regions 3 and 4 form an indentation in the convex hull. In fact, any "x-monotone hull" of the solution can be taken.

H Illustration for Section 9: Splitting a Surface by a Curve

Figure 14 illustrates the problem of splitting off a piece of given genus from a surface, which was mentioned in the conclusion, Section 9.

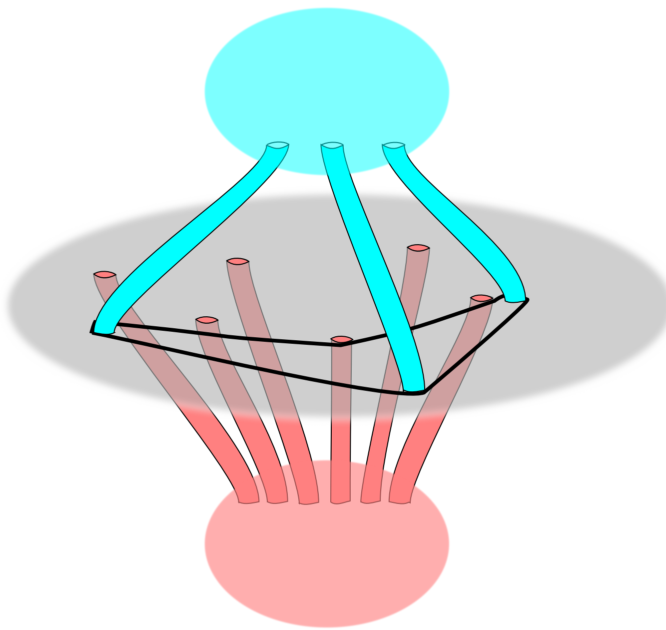


Figure 14 Any instance of Graph-Enclosure-with-Penalties with infinite penalties only, and with parameter k, can be recast as an instance of the problem of splitting off a surface of genus $k-1$. In this example, the graph G that is an instance of Graph-Enclosure-with-Penalties is embedded in the middle gray disk; each of the $k=3$ required faces is connected by a tube (in cyan) to the top sphere; each of the 6 optional faces, with infinite penalty, is connected by a tube (in red) to the bottom sphere. Finally, the gray disk is extended to a sphere (not shown), to obtain a surface S without boundary. Any weakly simple closed walk in G separating the required faces from the optional ones corresponds to a weakly simple closed walk splitting off a surface of genus $k-1=2$, and conversely. (The graph G is not cellularly embedded on S, but can be made so by adding edges of large weight.) Figure inspired by [7].

I Point Objects

In this section we modify our algorithm to handle input objects that are a mix of points and almost-simple polygons. The condition that polygons have disjoint interiors is replaced by the condition that the interior of an object may not intersect another object. So a point object is either disjoint from all polygon objects, or it lies on the boundaries of some polygon objects. We subdivide polygon edges to ensure that no point object lies in the interior of a polygon edge.

Point objects disjoint from polygons could be approximated by tiny polygons, but point objects at polygon vertices cannot be dealt with so cavalierly. Instead, we show how to modify our algorithm to deal directly with point objects.

We must clarify the output requirements in case a point object p lies on the boundary of a solution W. The only reasonable way to decide the matter in this case without making the problem ill-posed is to consider p to be enclosed or not at our discretion, since, by an arbitrarily small perturbation of W in the vicinity of p, either outcome can be achieved. This agrees with the convention used by Eades and Rappaport [13]. More precisely, we adapt the notion of a feasible solution W as follows:

For each point object $p \in R$, we require that p lies in the interior or on the boundary of W (possibly several times).

The penalty of an optional point object $p \in O$ that lies on the boundary of W is not counted towards the cost in formula (1).

We use a reference point r_P for each object P. For a point object, the reference point must be that point itself. A reference point lying on a mouth in a subproblem $M(pq, t, B)$ was already handled by the algorithm. What is new is the possibility that a vertex is a point object, either required or optional.

We make two changes to the dynamic programming algorithm:

1. If $p \in R$ is a required point object, we add an extra possibility to equation (3) for the case $B = \{p\}$ as follows:

$$C(p, t, \{p\}) := 0 \text{ for } t \geq 0 \tag{27}$$

2. For equation (7) we used a triangle $\Delta = prq$ that was defined to be open on edge pq and closed on edges pr and qr. We redefine Δ to exclude its corners p, q, r, i.e., the only boundary points of Δ that are included are the interiors of edges pr and qr. This affects both $\pi(\Delta)$ and $R(\Delta)$.

We explain the effect of these changes informally, and then outline how our proofs of correctness must be modified.

First observe that the changes to Δ mean that $\pi(\Delta)$ does not count penalties of optional point objects at the corners of Δ in equation (7), which is the correct thing to do. This is the only place in the equations where penalties are added.

Consider now a required point object p. At the top level, p lies in R, and this is passed to the disjoint sets B used in the recursions. If ever p is contained in $R(\Delta)$, then this is where p is considered to be enclosed (and it can only be enclosed once in this way). At the bottom of the recursion, rule (27) permits us to consider p as enclosed, and rule (2) permits us to consider p as not enclosed. Thus, we can “catch” the object p on the boundary if we have not done so already in a triangle. If the boundary goes through p several times, we can catch it on one occasion and pass over it on the other occasions.

To make this more formal, we adapt Lemma 15 in the following way:

▶ **Lemma 23.** (A) *Let W be a weakly simple polygon that goes through some vertex p and consists of ℓ free-space edges. Let B_{in} be the objects of R that are enclosed by W, and let B_{point} be the point objects of R that coincide with vertices of W. Then, for all $t \geq \ell$ and for all B with $B_{\text{in}} \subseteq B \subseteq B_{\text{in}} \cup B_{\text{point}}$, $C(p, t, B) \leq c(W)$.*

(B) *Let W_0 be an open walk with ℓ free-space edges from vertex p to vertex q such that the polygon $W = W_0 + qp$ is weakly simple. Let B_{in} be the objects of R whose reference points lie inside W and not on pq, and let B_{point} be the point objects of R that coincide with vertices of W,* excluding p.

In addition, assume that q is not a transition vertex of W. Then, for all $t \geq \ell$ and for all B with $B_{\text{in}} \subseteq B \subseteq B_{\text{in}} \cup B_{\text{point}}$, $M(pq, t, B) \leq c(W_0)$. ◀

Note in particular that, in the statement of part (B), we have added the condition that if p is a required point in R, it cannot be in B (in addition to requiring that p is not a transition vertex).

The induction basis, treating the trivial polygons with $t = 0$ edges, is covered by (2) and (27).

For the closed walks in statement (A), when p is a point object in B, we cannot apply the strategy for case A.1, because it would lead to the subproblem $M(qp, \dots)$ in which p is a point object, for which the extra requirement for (B) does not hold. Thus, in this case, if p is not a transition vertex anyway, we make a degenerate split as in case A.2, with an empty walk W_2 and $B_2 = \{p\}$, and consequently $W_1 = W$, see Figure 11. Equation 5 yields

$$C(p, t, B) \le C(p, t, B \setminus \{p\}) + C(p, 0, \{p\}) \le C(p, t, B \setminus \{p\}) + 0.$$

To the subproblem $C(p, t, B_1)$ with $B_1 = B \setminus \{p\}$, case A.1 applies, since p is no longer an element of B_1 for this subproblem. This leads to

$$C(p, t, B_1) \le w_{pq} + M(qp, t - 1, B_1) \le c(W)$$

and hence to $C(p, t, B) \le c(W)$.

In case B.2, where we split the set B into $B_1 \sqcup B_2 \sqcup R(\Delta)$, the splitting is clear: we have to assign any point objects on W_1 to B_1 and any point objects on W_2 to B_2. If the vertex r is a required point and belongs to B, we use the freedom of choice to put it in B_2 (the subproblem belonging to the mouth rp) and not in B_1, ensuring that the inductive hypothesis can be applied to the first subproblem $M(pr, t_1, B_1)$.

Case B.1 does not require any changes.

The other part of the correctness proof is based the properties of W_{DP} proved in Lemma 6. Lemma 6(B) claims that for $P \in R$, $\text{wind}(W_{\text{DP}}, r_P) = 1$. But for a point object $p \in R$ that lies on W_{DP}, the winding number is undefined. Thus, we have to restrict this claim to required point objects that do not lie on W_{DP} (and ditto in the claims for the subproblems in the inductive proof). For a point object $p \in R$ lying on W_{DP}, we simply observe that it will also lie on W_{ALG} because the uncrossing algorithm does not remove points from the polygon boundary. Hence p fulfills the adapted requirements of a feasible solution.

J Negative Penalties, or Rewards

We now consider the extension of Geometric-Enclosure-with-Penalties and Graph-Enclosure-with-Penalties in which we allow objects with negative penalties. In order to have a balanced and general statement, we use a greater variety of types of polygons/faces: there is a set R^- of ***required*** polygons/faces, a set R^+ of ***forbidden*** polygons/faces (the notation suggests that these sets correspond to objects with penalty $-\infty$ and $+\infty$, respectively), and a set $O^+ \sqcup O^0 \sqcup O^-$ of ***optional*** polygons/faces, whose penalties are finite and positive for the objects in O^+, zero for the objects in O^0, and finite and negative for the objects in O^-. The goal is to find a weakly simple closed curve/walk disjoint from the objects, enclosing R^-, excluding R^+, and minimizing the length of the curve plus the penalties of the objects in $O^+ \cup O^0 \cup O^-$ that are enclosed by the curve.

▶ **Theorem 24.** *We can solve these generalized problems in time $O(3^k n^3)$ time and $O(2^k n^2)$ space, where $k = \min\{|R^-| + |O^-|, |R^+| + |O^+|\}$.*

Proof. Let us first consider the case where $k = |R^-| + |O^-|$. The idea is to try all subsets of O^- that can be enclosed in an optimal solution.

We run the dynamic program with set of required objects $R := R^- \cup O^-$ and set of optional objects $O := R^+ \cup O^+ \cup O^0$, where the objects in R^+ have infinite penalties and those in $O^+ \cup O^0$ keep their original nonnegative penalties; this takes $O(3^{|R|}n^3) = O(3^k n^3)$ time. As part of the dynamic programming recursion, the algorithm determines $C(p, B)$ for all subsets $B \subseteq R$ and all vertices p. We therefore have available all quantities that enter the following formula for the optimum solution:

$$C^*_{\text{final}} := \min_{O_1^- \subseteq O^-} \left(\left(\min_{p \text{ a vertex}} C(p, R^- \cup O_1^-) \right) + \sum_{P \in O_1^-} \pi_P \right) \tag{28}$$

This formula is justified as follows: The solutions considered for $\min_p C(p, R^- \cup O_1^-)$ are those solutions that, among the objects in $R = R^- \cup O^-$, enclose precisely the objects of $R^- \cup O_1^-$ and exclude the remaining objects of O^-. In this way, we consider all solutions that enclose R plus some arbitrary subset $O_1^- \subseteq O^-$ of the negative-weight objects. The negative weights of the objects in O_1^- are explicitly added in (28) to get the correct value of the objective function.

The time $O(2^{|O^-|}n)$ for calculating C^*_{final} by (28) is dominated by the runtime $O(3^k n^3)$ of the dynamic programming algorithm.

In the other case where $k = |R^+| + |O^+|$, we invoke the inverted algorithm (Section 8) instead of the algorithm of Theorem 1 or 2. This swaps R^+ with R^- and O^+ with O^-. The rest of the argument is identical. ◀

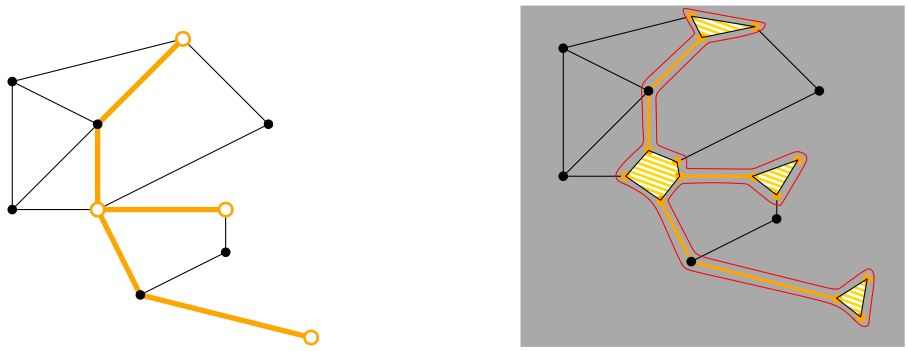
K Exponential Lower Bounds

We first prove a (conditional) exponential lower bound for the Graph-Enclosure-with-Penalties problem. The proof consists of a simple reduction to our problem from the Planar Steiner Tree problem.

▶ **Theorem 25.** *Assuming the Exponential Time Hypothesis, the* Graph-Enclosure-with-Penalties *problem cannot be solved in $2^{o(k)} \cdot n^{O(1)}$ time, even when all the weights are 1, and all penalties are ∞.*

Proof. The proof consists of a reduction from the Planar Steiner Tree problem, whose input is an edge-weighted planar graph G with n vertices and a set T of k vertices of G, usually called *terminals*. The problem asks for a minimum-weight tree in G connecting all terminals. Marx, Pilipczuk, and Pilipczuk [19, Theorem 1.2] proved that the Planar Steiner Tree problem cannot be solved in $2^{o(k)} \cdot n^{O(1)}$ time, assuming the Exponential Time Hypothesis, even if the input graph is unweighted (that is, all the edge weights are 1). We now describe the reduction.

Given an unweighted planar graph G and a set $T \subseteq V(G)$, as the one in Figure 15(a), we replace each terminal v in T with a corresponding *terminal cycle* C_v, whose number of edges is equal to $\max\{3, d_G(v)\}$, where $d_G(v)$ denotes the degree of v in G; each vertex of C_v is connected to a different neighbor of v, and the interior of C_v is a face, see Figure 15(b). Denote by H the obtained plane graph and by γ the total number of *terminal edges*, i.e., edges in terminal cycles. Every edge of H has weight 1. Let R be the set of faces inside terminal cycles, and let O be the set of all the other faces. The faces in O have penalty ∞. This completes the reduction. We now prove that G contains a tree with weight $\leq w$ connecting the terminals in T if and only if H has a weakly simple closed walk W with weight $\leq 2w + \gamma$ that has the faces in R inside (and the faces in O outside).



(a) (b)

Figure 15 (a) An instance of the Planar Steiner Tree problem. Terminals are large empty disks. Edges of a tree S connecting the terminals are thick and yellow. (b) The corresponding instance of the Graph-Enclosure-with-Penalties problem. Faces in R are yellow/hatched, faces in O (including the unbounded face) are gray. The weakly simple closed walk W constructed from S is represented by a red curve.

For the forward implication, given any tree S in G with weight at most w connecting the terminals in T, one can construct the desired walk as follows. The walk traverses each edge in S twice (once in each direction), and traverses each terminal cycle once, in counter-clockwise direction.

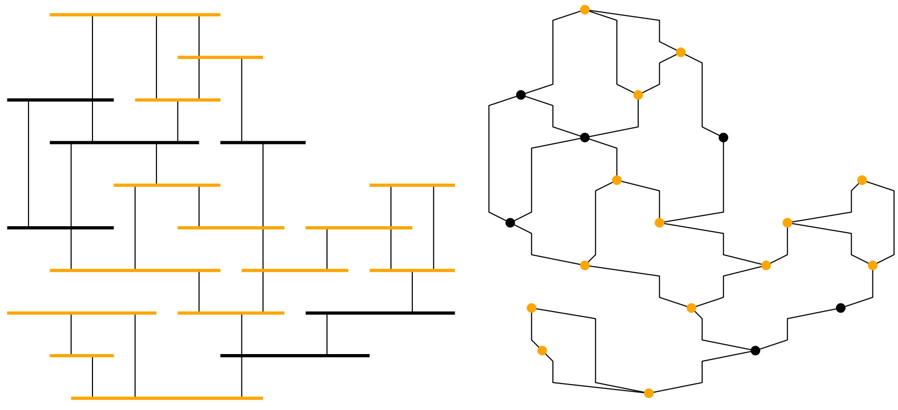
For the backward implication, let W be a weakly simple closed walk in H with weight at most $2w + \gamma$ that has the faces in R inside and the faces in O outside. W contains each terminal edge at least once, because W must separate R from O. Moreover, W uses each non-terminal edge of H an even number of times, as otherwise one of its incident faces, both of which have penalty ∞, would be inside W. Since W is connected, it contains at least twice each edge of a connected subgraph S_H spanning the terminal cycles. The simple graph S_G in G corresponding to S_H connects all the terminals. The weight of S_G is at most $((2w + \gamma) - \gamma))/2 = w$, which it is obtained from the weight $2w + \gamma$ of W by subtracting the total weight of the terminal edges, which is at least γ, and by then dividing by two as each edge of S_G is used at least twice in W. We conclude the proof by observing that S_G contains a tree that spans all terminals and has weight at most w. ◀

We now present a reduction similar to, and slightly more technical than, the one of Theorem 25 for the geometric version of our problem.

▶ **Theorem 26.** *Assuming the Exponential Time Hypothesis, the* Geometric-Enclosure-with-Penalties *problem cannot be solved in* $2^{o(k)} \cdot n^{O(1)}$ *time, even when all penalties are* ∞.

Proof. Consider an instance (G, T) of the Planar Steiner Tree problem in which G has n vertices and edges with weight 1. We start by constructing the $O(n)$-vertex planar graph H as in the proof of Theorem 25. We now construct a sequence of representations of H, and eventually get the desired instance of Geometric-Enclosure-with-Penalties.

First, we construct a *visibility representation* Γ of H on an $O(n) \times O(n)$ grid [22], see Figure 16(a). In Γ, vertices are represented by disjoint horizontal segments lying on grid rows and edges are represented by disjoint vertical segments lying on grid columns. Each vertical segment representing an edge has its endpoints on the horizontal segments representing the end-vertices of the edge and otherwise does not cross any horizontal segment representing a vertex. We scale all the coordinates in the drawing up by a factor of 5.



(a) (b)

Figure 16 (a) A visibility representation Γ of the graph H from Figure 15(b). We stress that the relative interior of a vertical segment representing an edge of H does not intersect any horizontal segment representing a vertex of H; vertical lines may consist of several vertical segments. (b) A poly-line drawing Γ' of H constructed from Γ. Both representations lie on an $O(n) \times O(n)$ grid.

Second, we turn Γ into a poly-line drawing Γ'; this can be done by modifying Γ only "close" to its vertices, see [5, 12] and Figure 16(b). Specifically, each horizontal segment s_v representing a vertex v is replaced by a grid point p_v on s_v. Also, we shorten each vertical segment representing an edge uv by one unit at the top and at the bottom and connect the endpoints to p_u and p_v.

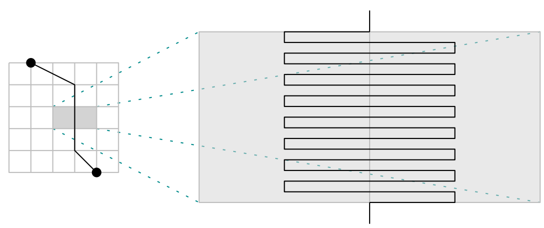


Figure 17 The left part of the figure shows an edge e in the drawing Γ'. The right part shows an enlarged central section of e in which the drawing of e is modified in order to transform Γ' into a poly-line drawing Γ'' of H in which e has length between n^2 and n^2+1. The edge e is only modified in its portion σ_e inside the two gray grid cells.

Third, we turn Γ' into a poly-line drawing Γ'' in which all edges have "almost" the same length. Intuitively, we are going to modify the representation of each edge by "orthogonally zig-zagging" in an intermediate part of the edge, so that the edge has length between n^2 and n^2+1, see Figure 17. As a consequence of the scaling of Γ, the representation of each edge e in Γ' contains a vertical segment σ_e between two grid points (i,j) and $(i,j+1)$ such that Γ' has no intersection with the two grid cells incident to σ_e, other than at σ_e itself. Let ℓ_e be the length of the polygonal chain representing e in Γ' and let $a_e = n^2 - \lfloor \ell_e \rfloor$ be the increase of length that we want for e. Then we can replace σ_e by the orthogonal line passing through points $(i,j),(i+\frac{1}{2},j),(i+\frac{1}{2},j+\frac{1}{a_e}),(i-\frac{1}{2},j+\frac{1}{a_e}),(i-\frac{1}{2},j+\frac{2}{a_e}),(i+\frac{1}{2},j+\frac{2}{a_e}),\ldots,(i,j+1)$. The vertical segments of this line have total length 1, while the horizontal segments have total length a_e (two of them have length $\frac{1}{2}$, while the other a_e-1 have length 1). Hence, the length of e has increased by $n^2 - \lfloor \ell_e \rfloor$ and it is now between n^2 and n^2+1.

In order to get the instance of Geometric-Enclosure-with-Penalties, we interpret the faces of Γ'' as polygons: those inside the terminal cycles are in R and those corresponding to faces of G are in O and have penalty ∞. Since all the edges have approximately the same length, between n^2 and n^2+1, the same proof as in Theorem 25 shows that G contains a tree with weight $\leq w$ connecting the terminals in T if and only if there exists a weakly simple closed walk W with weight $\leq (2w+\gamma)\cdot(n^2+1)$ in the instance of Geometric-Enclosure-with-Penalties. Indeed, the "only if" part is easy, and the proof for the "if" part uses the following argument. From the weakly simple closed walk W, one can extract a simple graph S_G in G spanning all the terminals whose weight is at most $\frac{(2w+\gamma)\cdot(n^2+1)-\gamma\cdot n^2}{2n^2} = w + \frac{(2w+\gamma)}{2n^2}$. Since $2w+\gamma$ is in $O(n)$, we have that $\frac{(2w+\gamma)}{2n^2}$ is in $o(1)$, hence S_G has at most w edges provided n is large enough, which we can obviously assume. The described reduction takes polynomial time, given that all the vertex coordinates in Γ'' are rational numbers whose numerators and denominators are polynomially bounded. ◀

The above proof essentially contains a polynomial, parameter-preserving reduction from Graph-Enclosure-with-Penalties to Geometric-Enclosure-with-Penalties. In passing, we mention that there is also a polynomial-time, parameter-preserving reduction in the other direction: Given an instance of Geometric-Enclosure-with-Penalties, we know that the output will consist of free-space edges, so one can compute the graph that is the overlay of all free-space edges (equivalently, of the visibility graph of the input vertices), assign each subdivided edge a weight that is its Euclidean length, and assign penalty zero to each face of this arrangement that does not come from an input polygon. This results in an equivalent instance of Graph-Enclosure-with-Penalties.

L Weakly Simple Immersed Polygons

We can define the precise class of polygons over which the dynamic program optimizes. They are more general than the weakly simple polygons that we want as a solution, because they can self-cross, but they are not arbitrary polygons.

It turns out that $M(pq,t,B)$ and $C(p,t,B)$ is the minimum cost (with the extended meaning of Definition 5) of a *weakly simple immersed polygon* that satisfies appropriately modified constraints that correspond to the intended constraints regarding the number of edges, the set B of objects whose reference points are enclosed, and the mouth pq or startpoint p, respectively.

As in Appendix A and Appendix C.2, we describe such a polygon as a sequence of vertices forming its boundary cycle, in the form $P = (p_1, p_2, \ldots, p_n)$. The polygon runs counterclockwise around its "enclosed region", with the interior to its left.

Weakly simple immersed polygons (WSImP). A ***weakly simple immersed polygon*** (WSImP) is obtained by gluing together triangles and digons in a tree-like fashion.

There are two base cases:

- a counterclockwise nondegenerate triangle (p,q,r)
- a digon (p,q)

The two ways of inductively combining two WSImPs into a larger WSImP are the same combinations that we introduced in Appendix C.2 for arbitrary polygons:

- Two WSImPs can be glued together along a common *edge*: If $P_1 = (p,q,q_2,\ldots,q_n)$ and $P_2 = (q,p,p_2,\ldots,p_m)$ both use the edge pq, but in opposite directions, then we can form the WSImP $P = (q,q_2,\ldots,q_n,p,p_2,\ldots,p_m)$. (This is the same as gluing together two polygons in the plane along a common edge if they lie on different sides of that edge, except that we do not care whether they overlap.)

- Two WSImPs $P_1 = (p, q_1, q_2, \ldots, q_n)$ and $P_2 = (p, q_1, q_2, \ldots, q_n)$ can be glued together at a shared *vertex* p, forming a new WSImP $P = (p, q_1, q_2, \ldots, q_n, p, p_1, p_2, \ldots, p_m)$. (The vertex p becomes a transition vertex.)

By construction, our dynamic program computes a WSImP W that has the correct winding number for all objects in R. Since WSImPs can be triangulated, the same proof as that of Lemma 15 shows that the cost of W is optimal. More precisely, $M(pq, t, B)$ and $C(p, t, B)$ is the minimum cost of a weakly simple immersed polygon W under the following constraints:

1. For $M(pq, t, B)$, the walk connects the endpoints p and q and is closed by the mouth qp; for $C(p, t, B)$, it goes through the startpoint p.
2. The number of free-space edges is at most t, not counting the mouth in case of $M(pq, t, B)$.
3. For each object $P \in B$, $\text{wind}(W, r_P) = 1$, and for each object $P \in R \setminus B$, $\text{wind}(W, r_P) = 0$.

The cost is interpreted with the extended meaning of Definition 5, and the weight w_{pq} is subtracted in case of $M(pq, t, B)$.

If we restrict the base case to triangles and only allow gluings along edges, we arrive at the subclass of ***(simple) immersed polygons*** (SImPs). Here, the construction defines a simply connected surface, which is obtained by starting with the triangles and performing the gluing as an identification of common points. The boundary walk of a SImP is known as a ***self-overlapping polygon***, see for example Evans and Wenk [15] for a recent discussion. The boundary walks of WSImPs are related to self-overlapping polygons in the same way as *weakly simple* polygons are related to *simple* polygons.

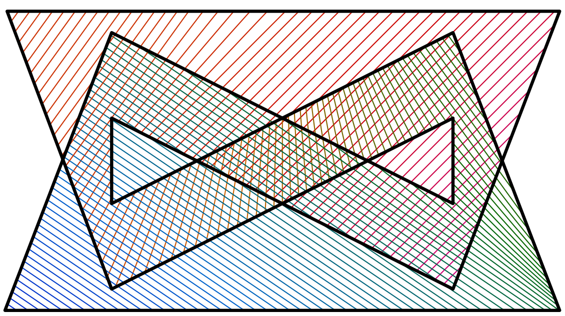


Figure 18 Milnor's doodle. The hatching indicates one of two symmetric ways of viewing this self-overlapping polygon as the boundary of an immersed surface.

The relation between a WSImP and its boundary walk is delicate, just as for self-overlapping polygons: It is not the case that the SImP as a boundary determines this surface uniquely. Figure 18 shows the simplest counterexample, which is known under the name *Milnor's doodle*. By construction, a SImP comes with a triangulation, but the triangulation is obviously not unique. Shor and Van Wyk [20] define a SImP as an equivalence class of triangulations of a self-overlapping polygon, and they give an algorithm for counting the number of SImPs for a given self-overlapping polygon [20, Section 6].

Thus, to specify a WSImP, the boundary walk is not sufficient: We would have to specify the sequence of gluings describing how the WSImP was built. For our purposes, however, the precise SImP or WSImP is irrelevant, and the boundary walk is all we need: By Lemma 19, we can find out how often a point of the plane is covered by P by calculating the winding number.

Table of Contents