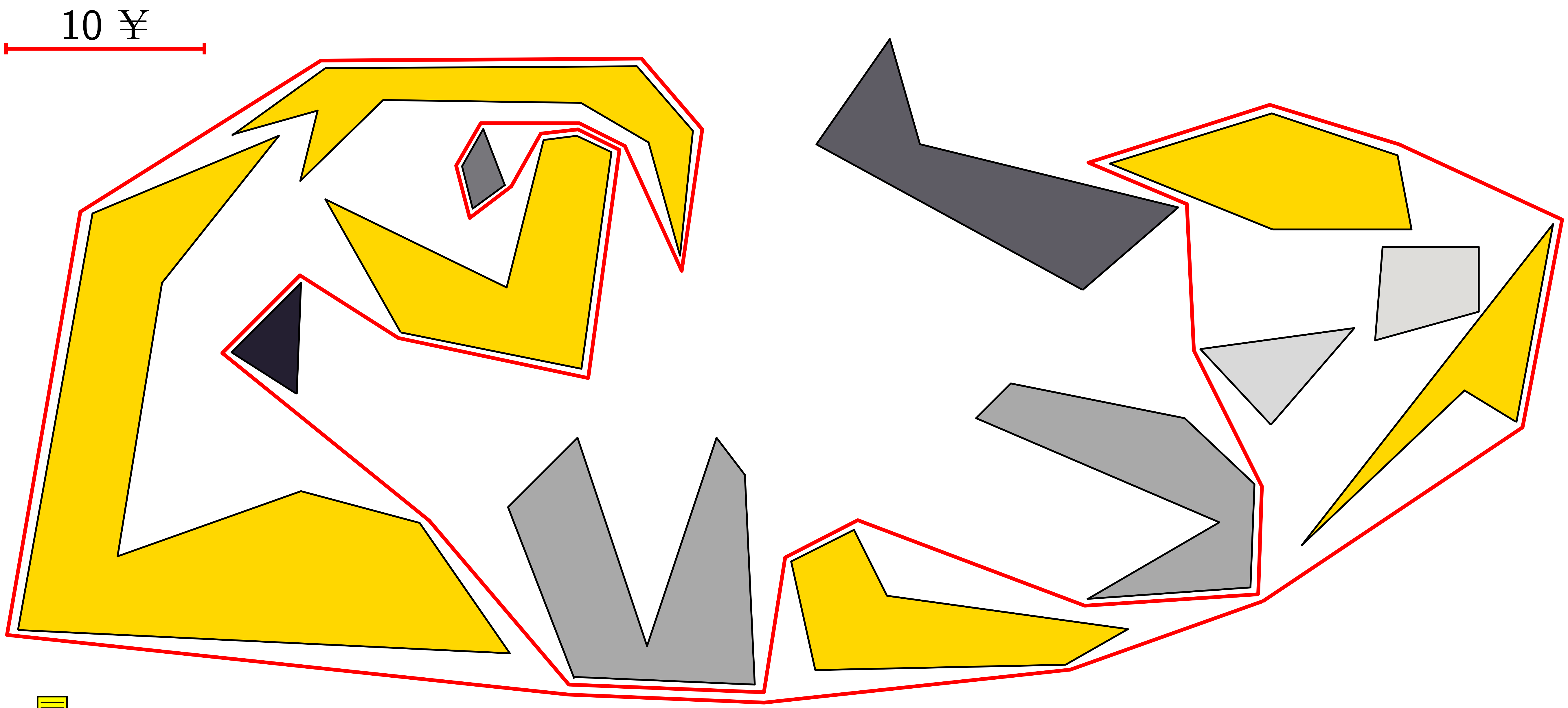
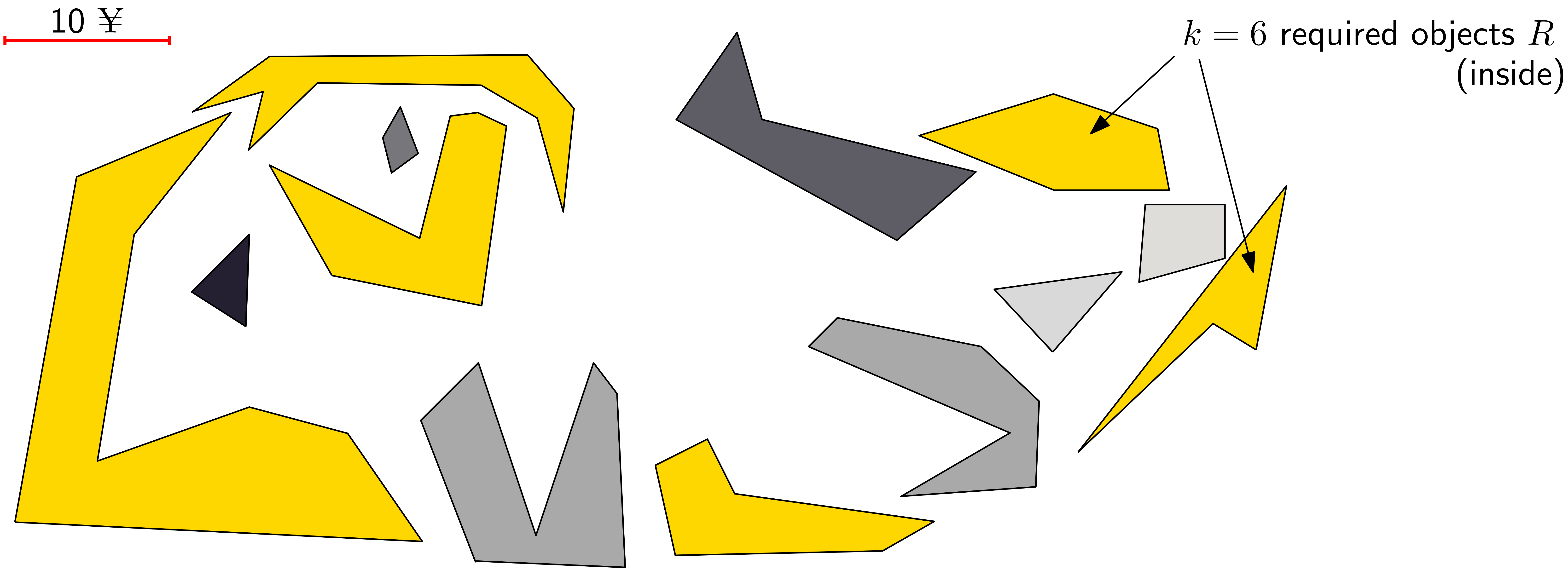
# Finding a Shortest Curve that Separates Few Objects from Many

Therese Biedl, Éric Colin de Verdière, Fabrizio Frati, Anna Lubiw, Günter Rote



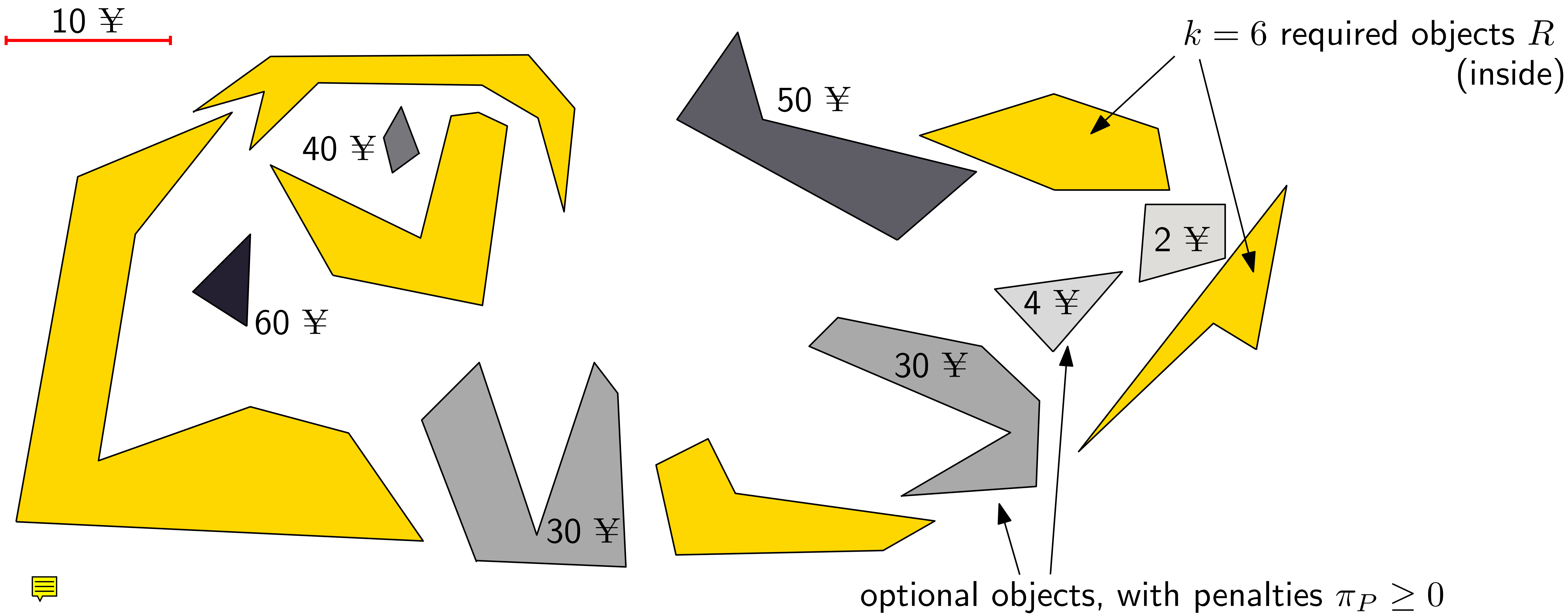
# Finding a Shortest Curve that Separates Few Objects from Many

Therese Biedl, Éric Colin de Verdière, Fabrizio Frati, Anna Lubiw, Günter Rote



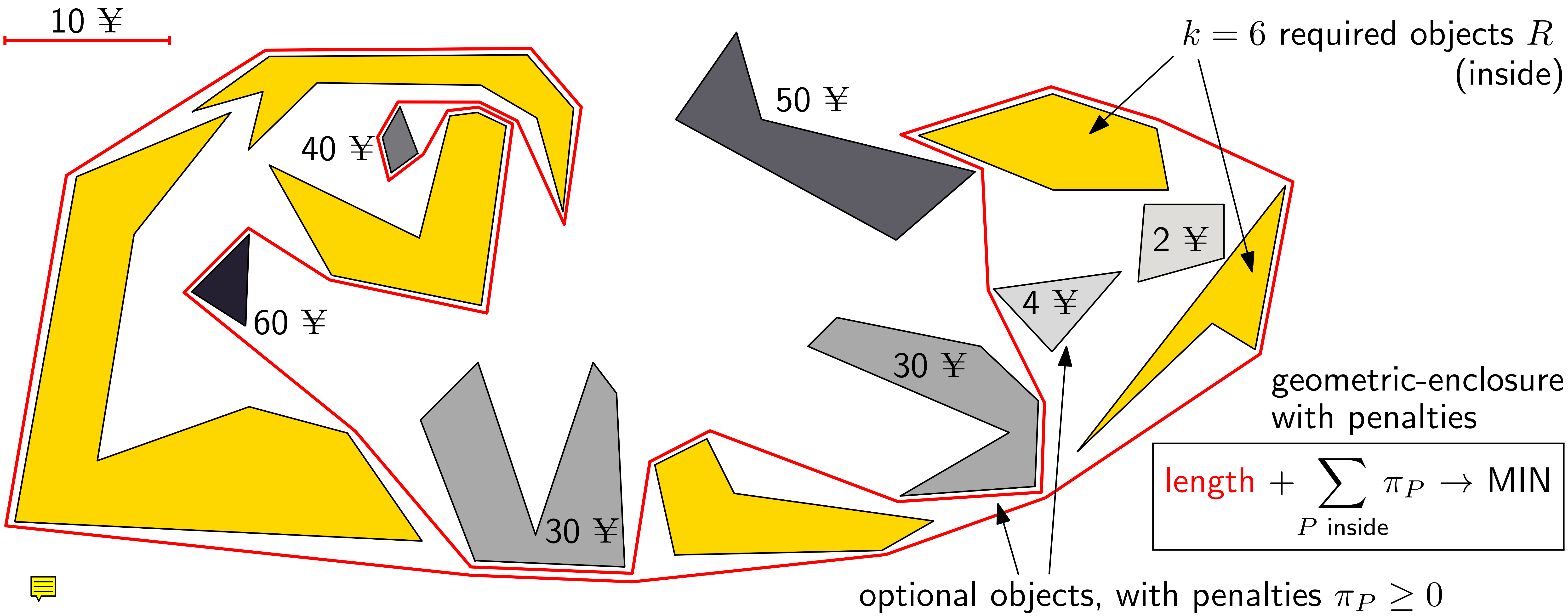
# Finding a Shortest Curve that Separates Few Objects from Many

Therese Biedl, Éric Colin de Verdière, Fabrizio Frati, Anna Lubiw, Günter Rote



# Finding a Shortest Curve that Separates Few Objects from Many

Therese Biedl, Éric Colin de Verdière, Fabrizio Frati, Anna Lubiw, Günter Rote

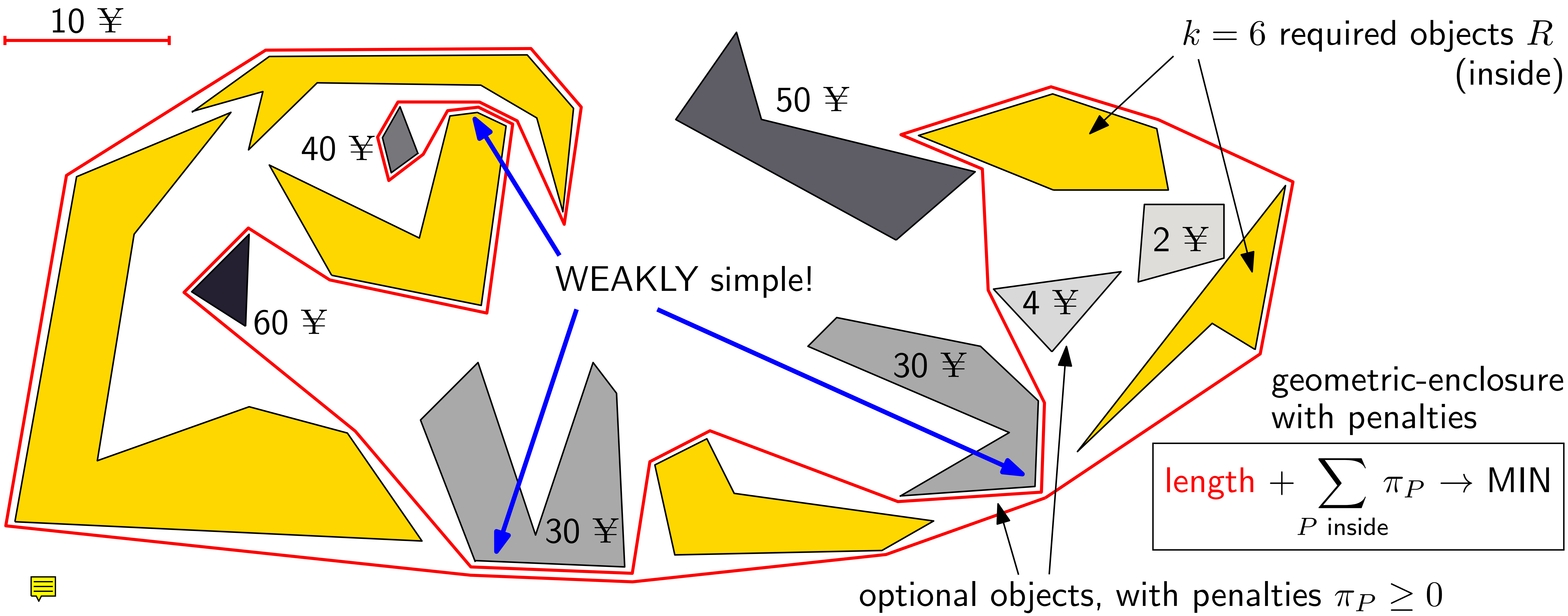



geometric-enclosure with penalties

$$\text{length} + \sum_{P \text{ inside}} \pi_P \to \text{MIN}$$

# Finding a Shortest Curve that Separates Few Objects from Many

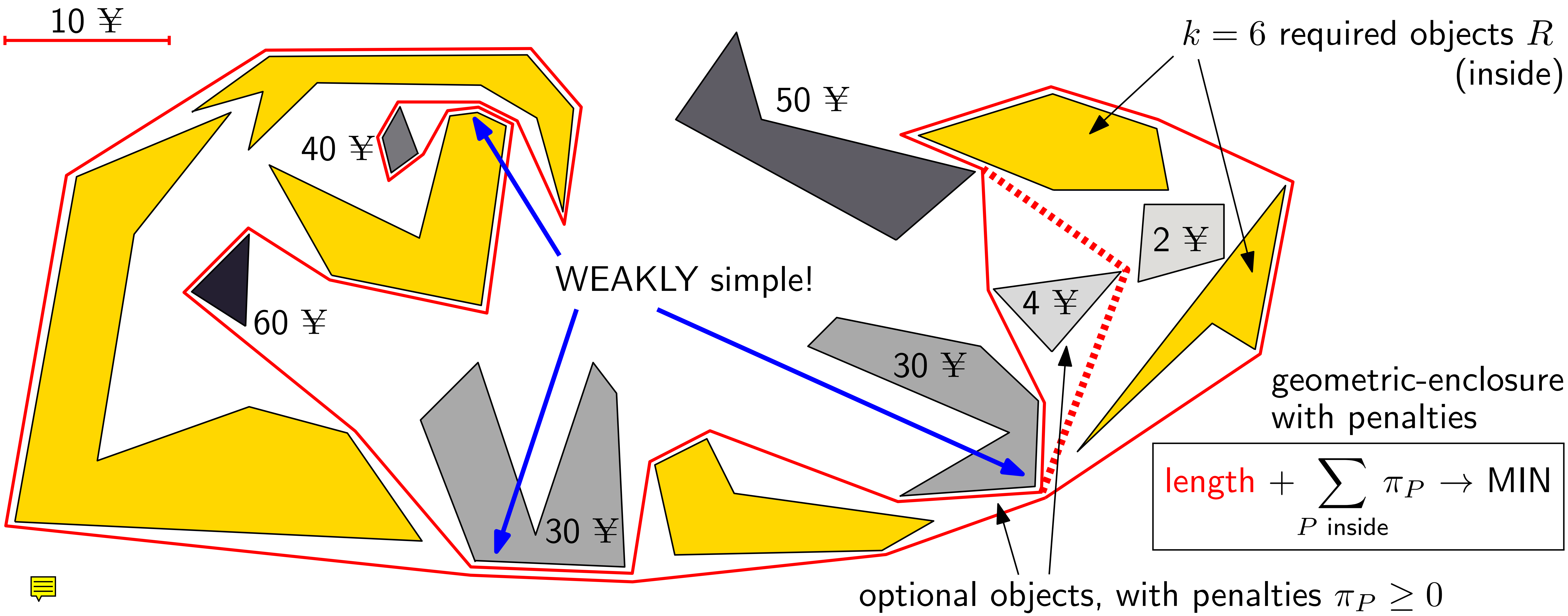
Therese Biedl, Éric Colin de Verdière, Fabrizio Frati, Anna Lubiw, Günter Rote




$$\text{length} + \sum_{P \text{ inside}} \pi_P \to \text{MIN}$$

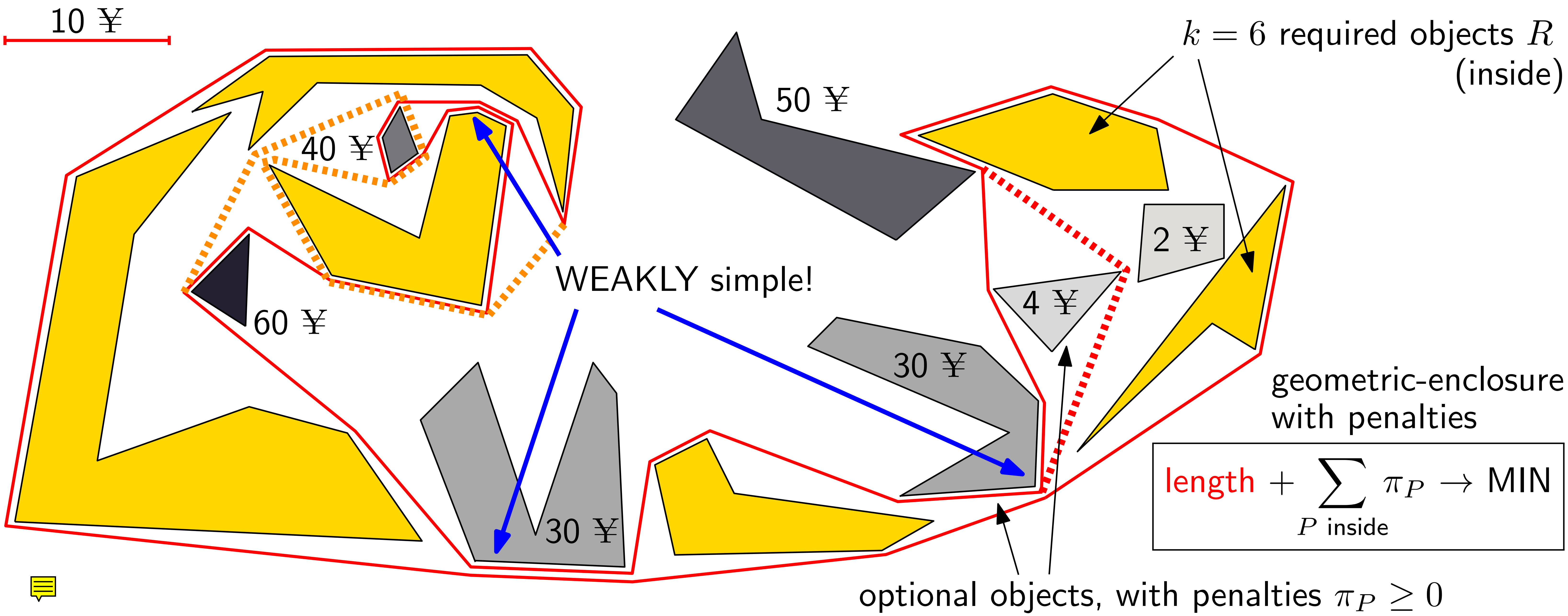
# Finding a Shortest Curve that Separates Few Objects from Many

Therese Biedl, Éric Colin de Verdière, Fabrizio Frati, Anna Lubiw, Günter Rote



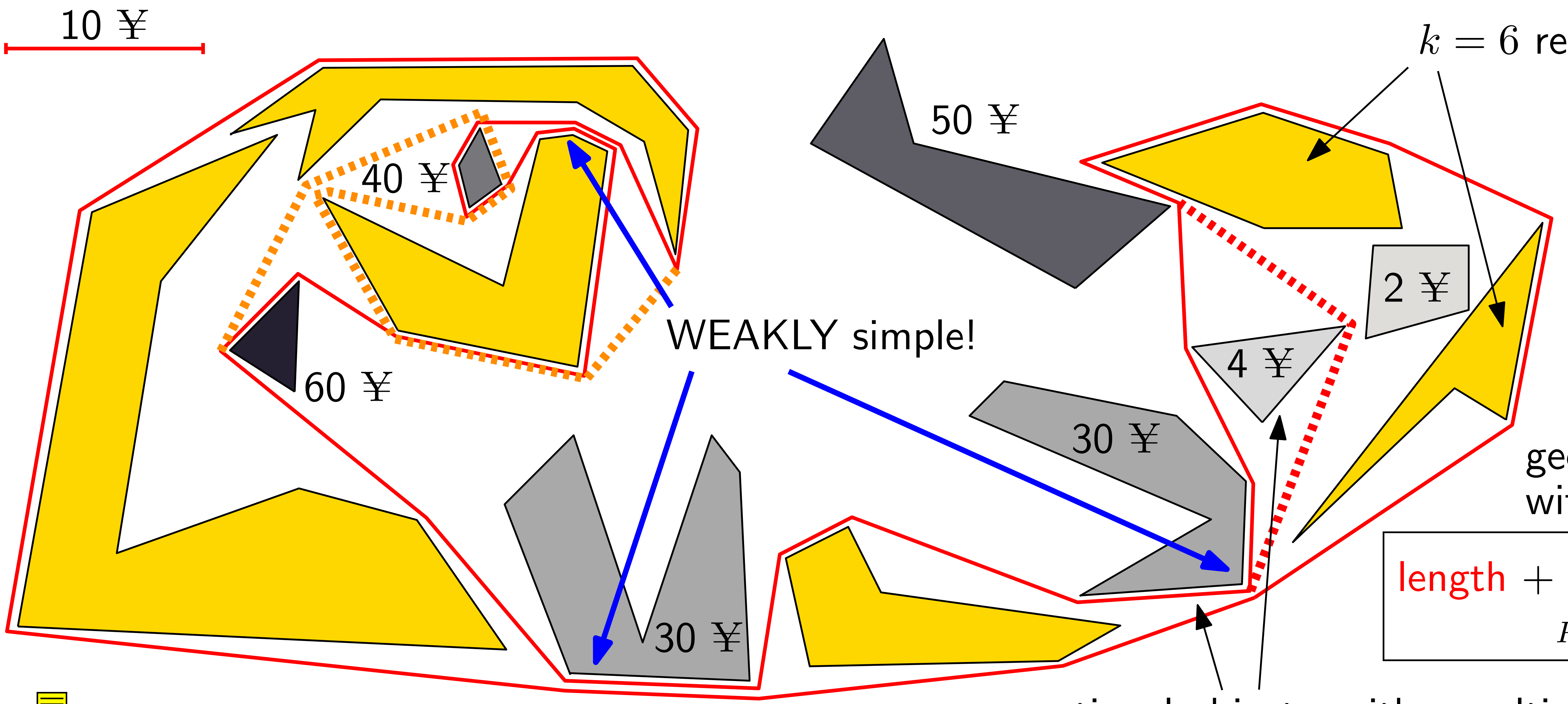
# Finding a Shortest Curve that Separates Few Objects from Many

Therese Biedl, Éric Colin de Verdière, Fabrizio Frati, Anna Lubiw, Günter Rote



# Finding a Shortest Curve that Separates Few Objects from Many

Therese Biedl, Éric Colin de Verdière, Fabrizio Frati, Anna Lubiw, Günter Rote

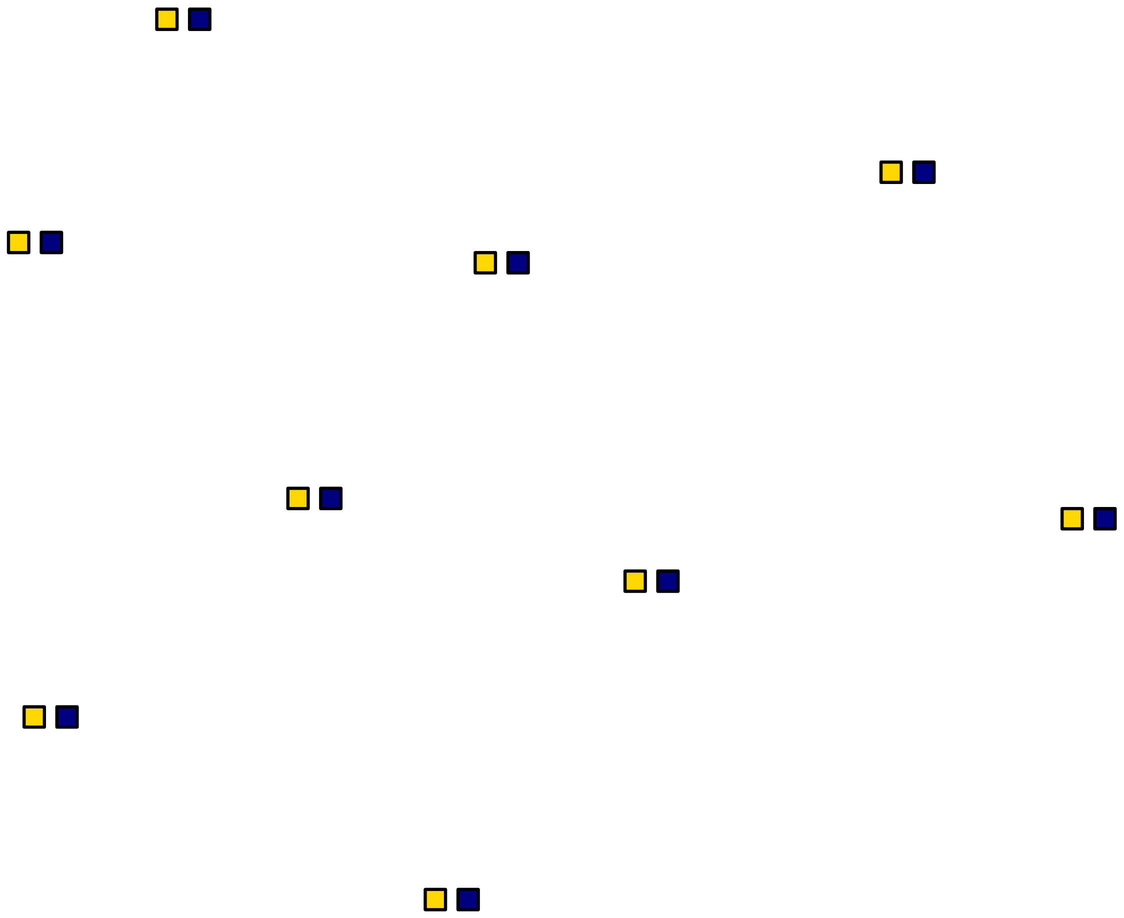



$O(3^k n^3)$ time
$O(2^k n^2)$ space

geometric-enclosure with penalties

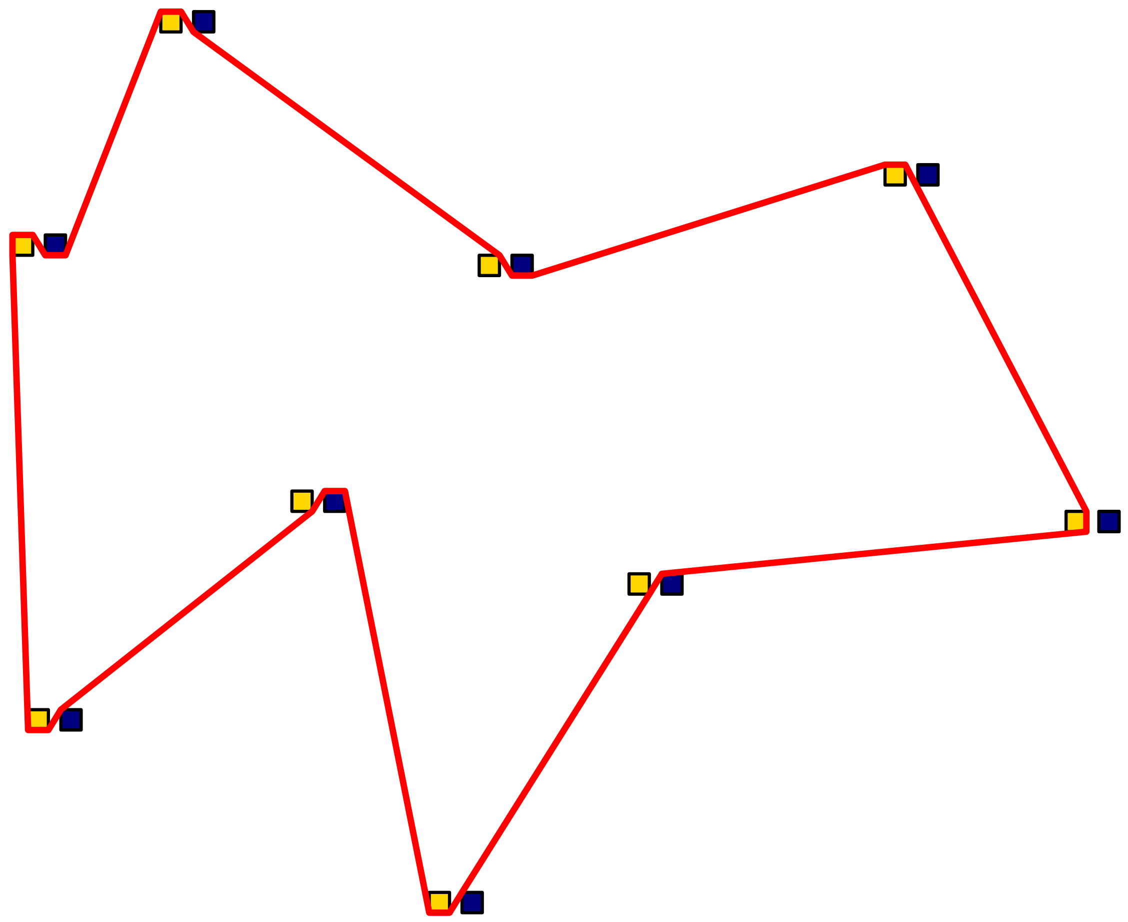
$$\text{length} + \sum_{P \text{ inside}} \pi_P \rightarrow \text{MIN}$$

# Large number $k$ of required objects



NP-hard (from the Traveling Salesperson Problem)
[Eades and Rappaport 1993]

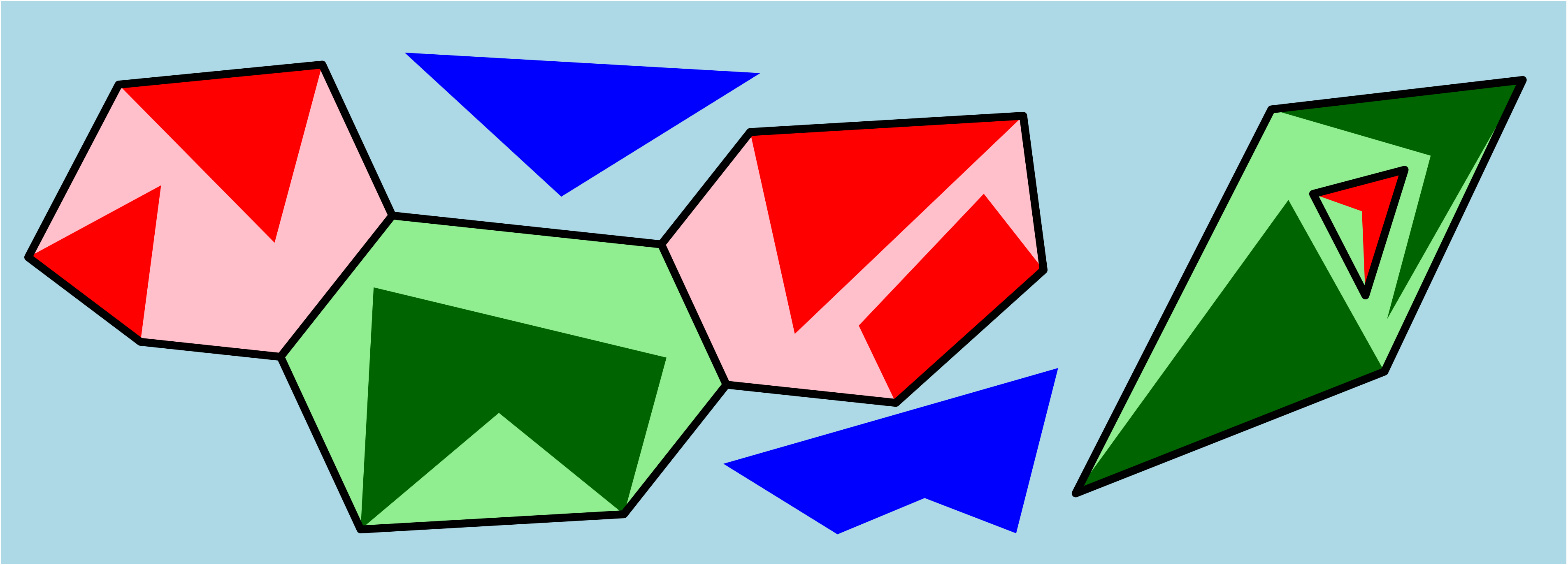
# Large number $k$ of required objects



NP-hard (from the Traveling Salesperson Problem)
[Eades and Rappaport 1993]

# Overview

- Problem definition ✓
- Related enclosure/separation problems:
  - separate $k$ *unspecified points* from the rest (OPEN for polygon objects)
  - systems of *fences* [ Abrahamsen, Giannopoulos, Löffler, Rote 2020 ]
  - geometric knapsack



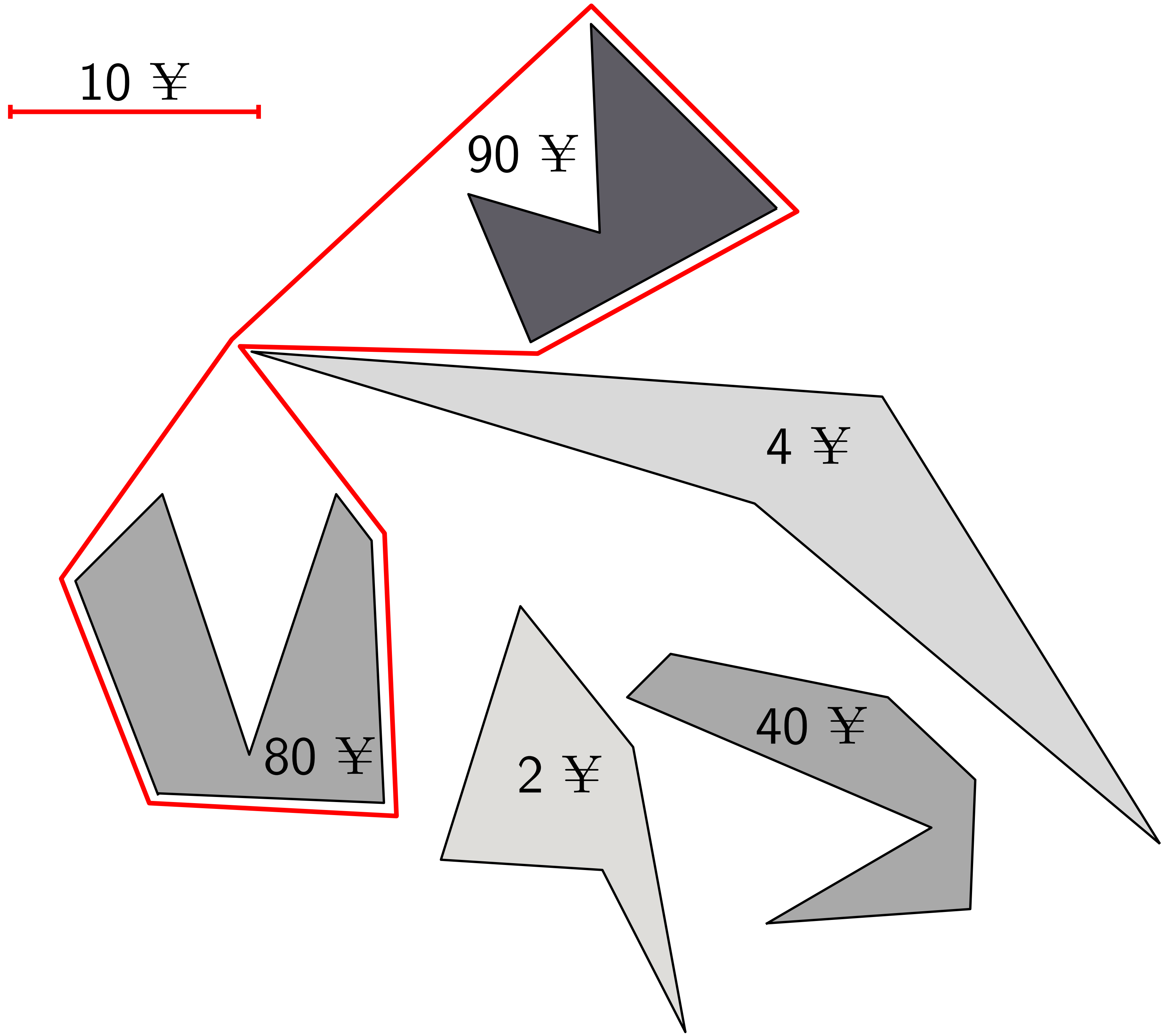
- Dynamic programming algorithm
  - definition of subproblems
  - dynamic programming recursion
  - tricky part: self-intersections
- Variations and improvements
  - the inverted problem (inside $\leftrightarrow$ outside)
  - speedup $3^k n^5 \to 3^k n^3$
  - PLANE-GRAPH-enclosure with penalties
- weakly simple immersed polygons (WSiMPs)
- ETH $\Rightarrow$ $2^k$ lower bound

# Related problem: Geometric knapsack

[Arkin, Khuller, Mitchell 1993]: $O(n^2 \cdot \#\text{visibility-edges}) = O(n^4)$

rewards

$$\text{MAX} \sum_{P \text{ inside}} \pi_P - \text{length}$$



# Related problem: Geometric knapsack

[Arkin, Khuller, Mitchell 1993]: $O(n^2 \cdot \#\text{visibility-edges}) = O(n^4)$

rewards

$$\text{MAX} \sum_{P \text{ inside}} \pi_P - \text{length}$$

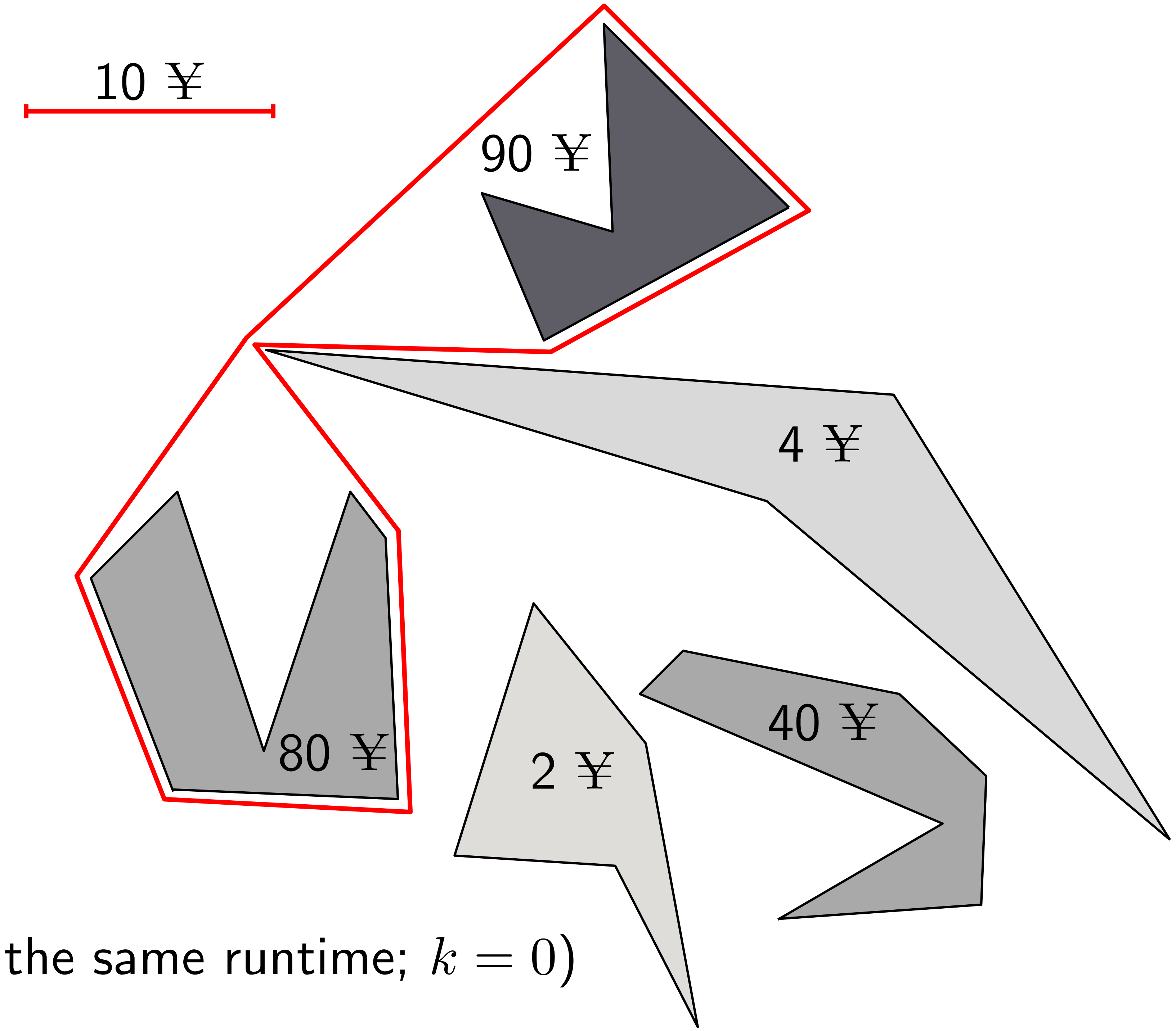
$$\text{MIN} \ \text{length} - \sum_{P \text{ inside}} \pi_P$$

$$+\sum_P \pi_P$$

$$\text{MIN} \ \text{length} + \sum_{P \text{ outside}} \pi_P$$

penalties

inverted problem




$\rightarrow$ $O(n^3)$ by our algorithm. (Inverted problem has the same runtime; $k = 0$)

# Related problem: Geometric knapsack

[Arkin, Khuller, Mitchell 1993]: $O(n^2 \cdot \#\text{visibility-edges}) = O(n^4)$

rewards

$$\text{MAX} \sum_{P \text{ inside}} \pi_P - \text{length}$$

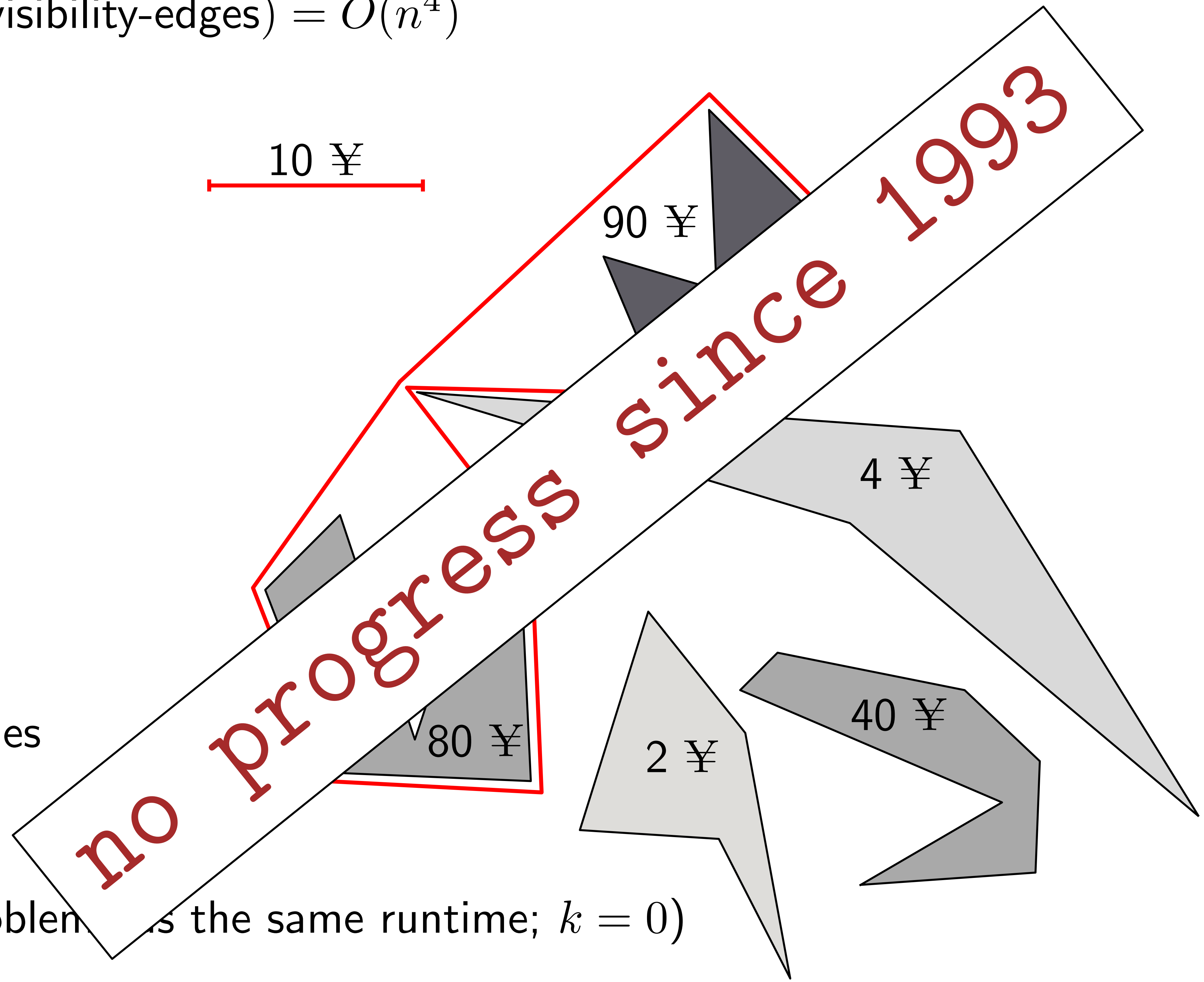
$$\text{MIN} \ \text{length} - \sum_{P \text{ inside}} \pi_P$$

$$+\sum_P \pi_P$$

$$\text{MIN} \ \text{length} + \sum_{P \text{ outside}} \pi_P$$

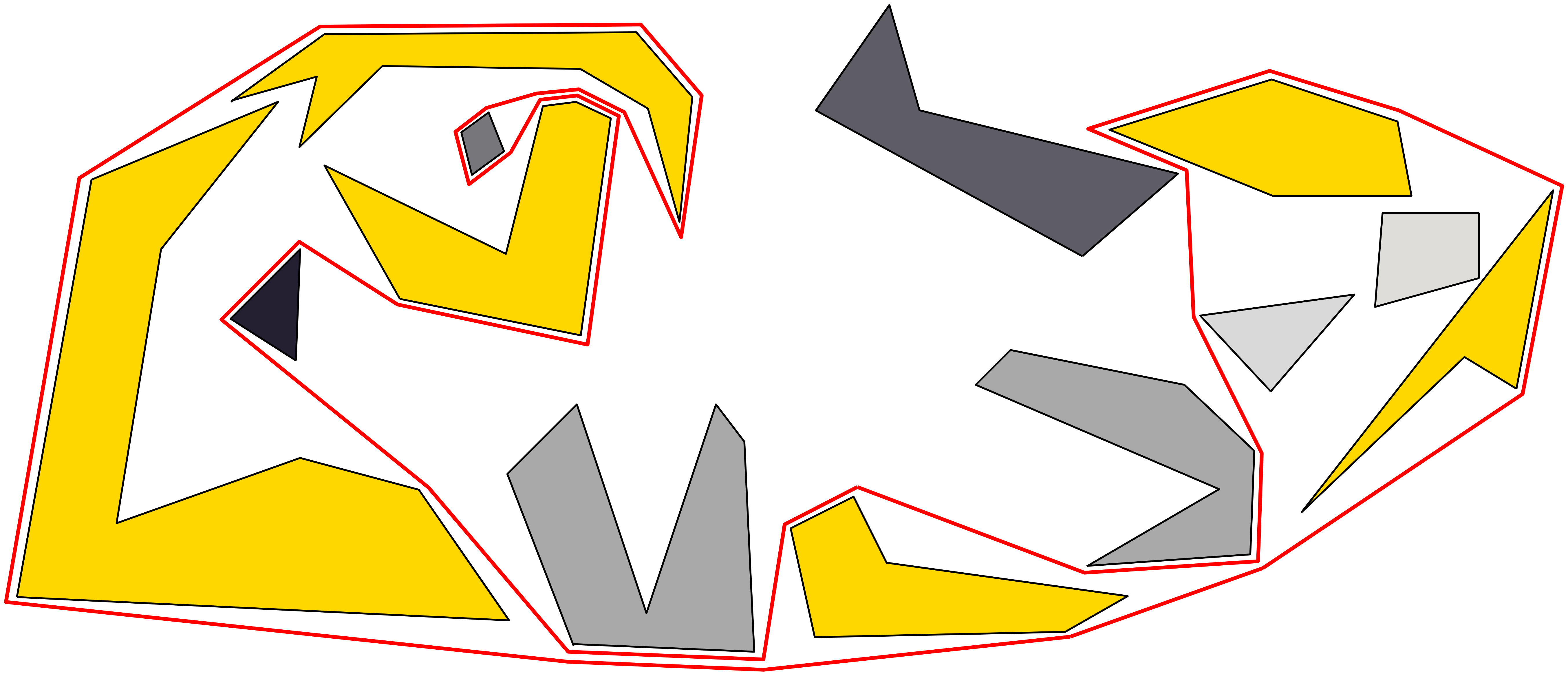
penalties

inverted problem

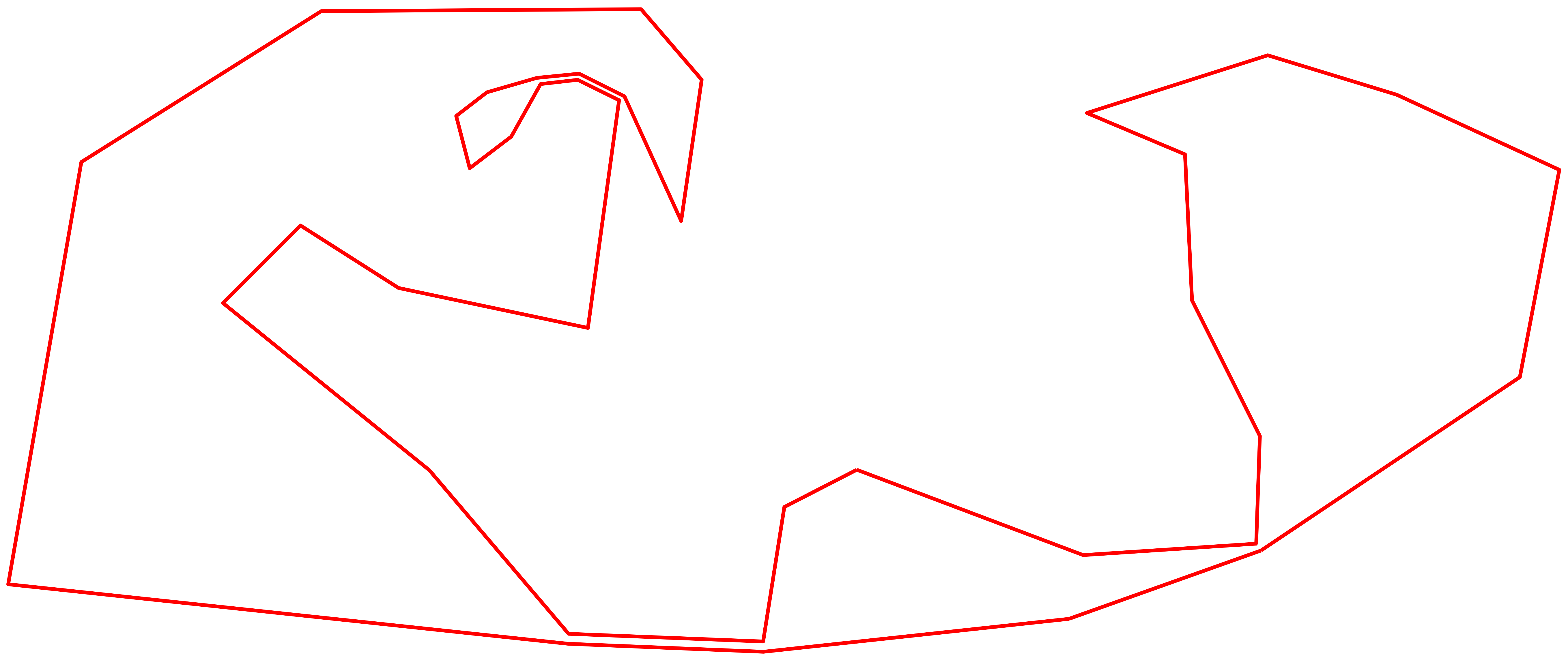



$\rightarrow$ $O(n^3)$ by our algorithm. (Inverted problem has the same runtime; $k = 0$)

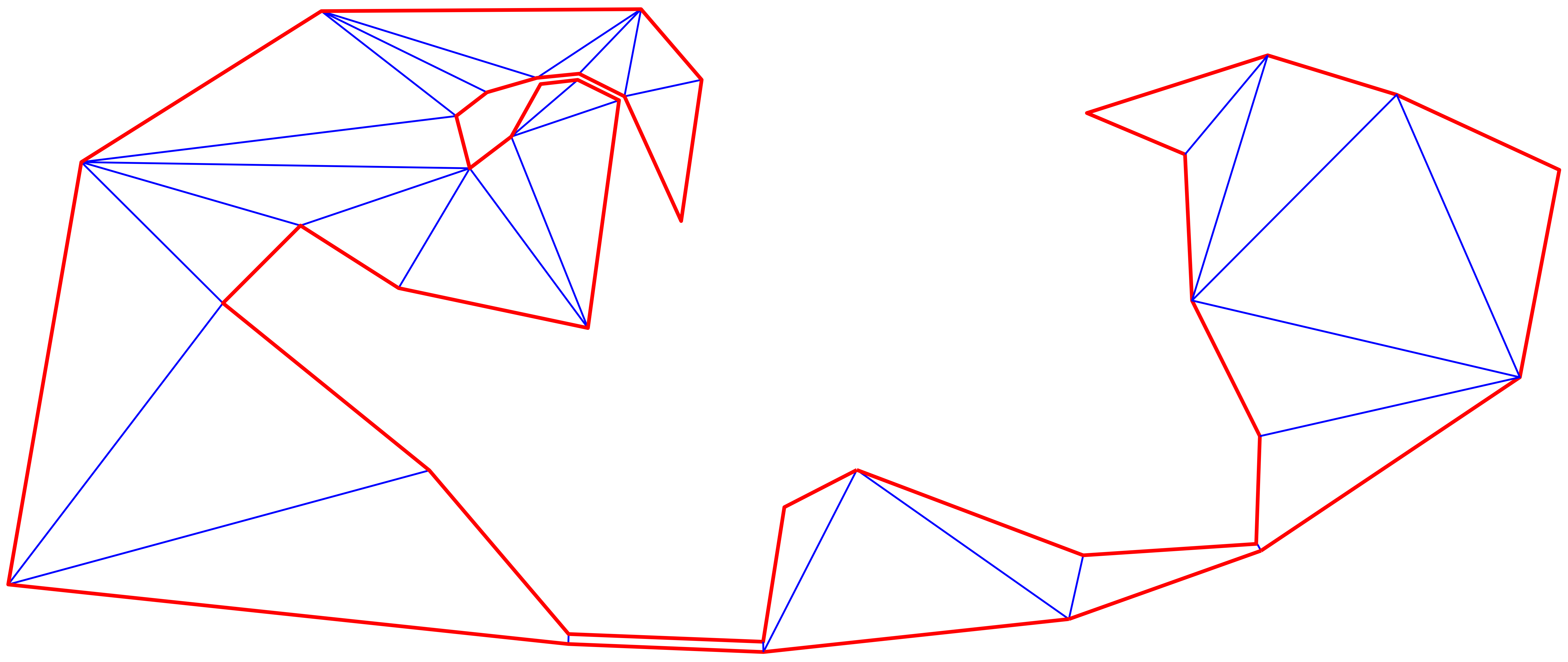
# Dynamic programming (DP) algorithm



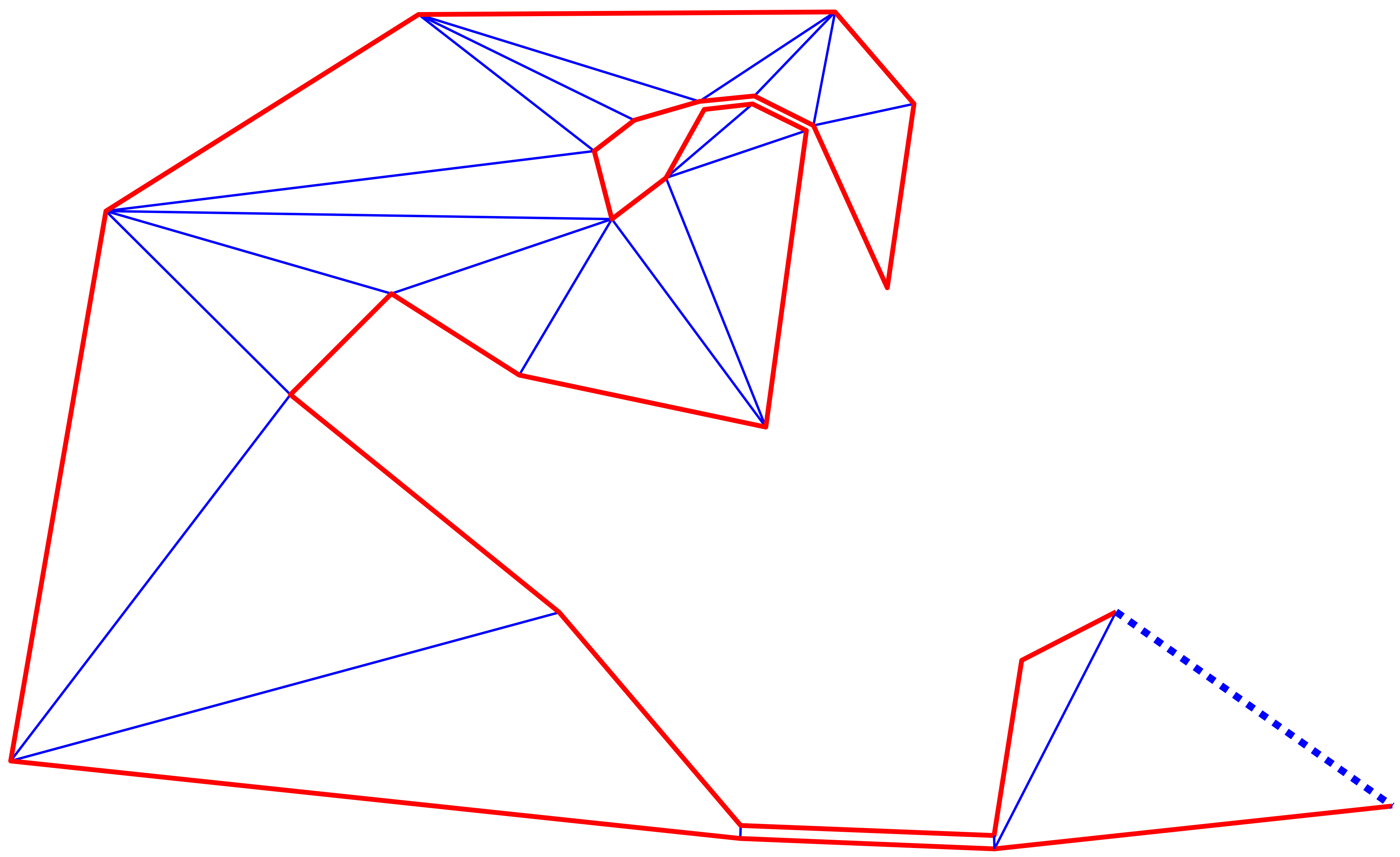
# Dynamic programming (DP) algorithm



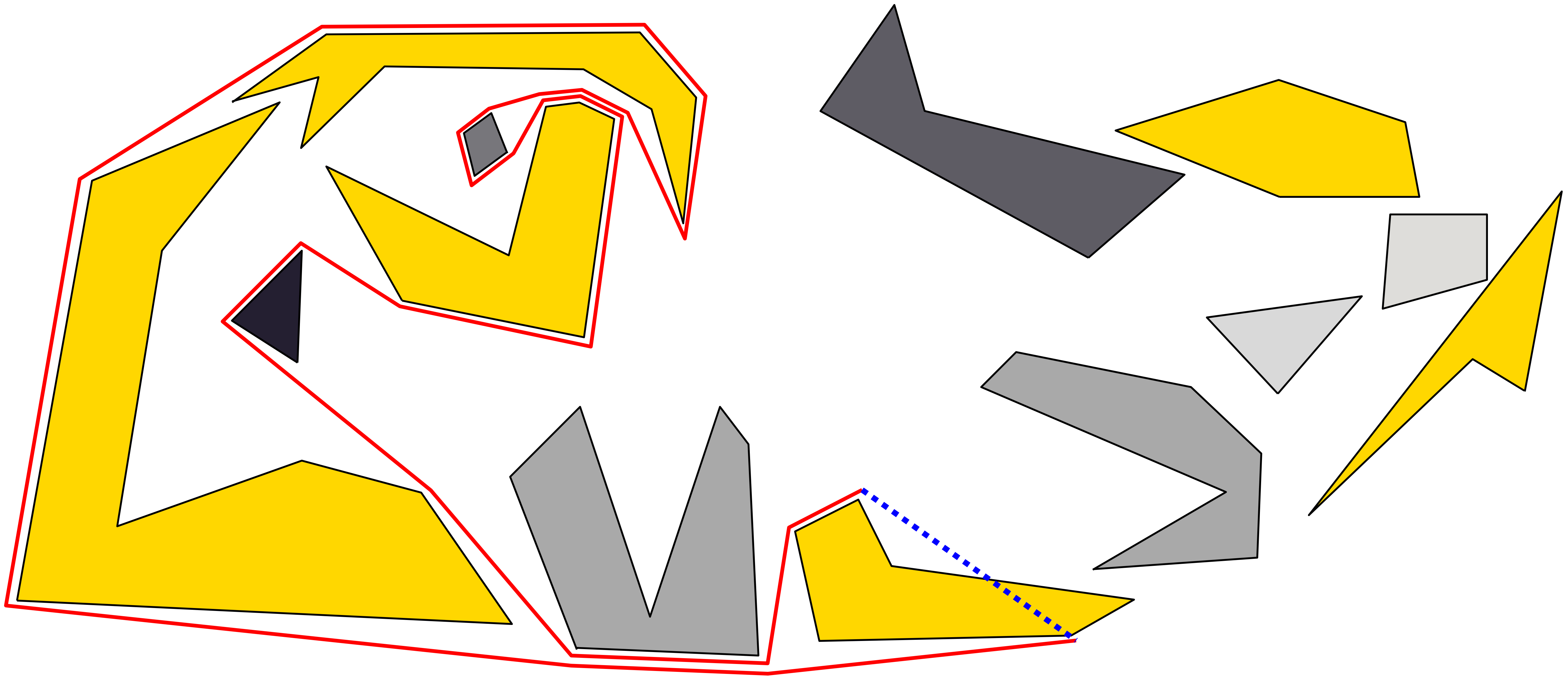
# Dynamic programming (DP) algorithm



# Dynamic programming (DP) algorithm

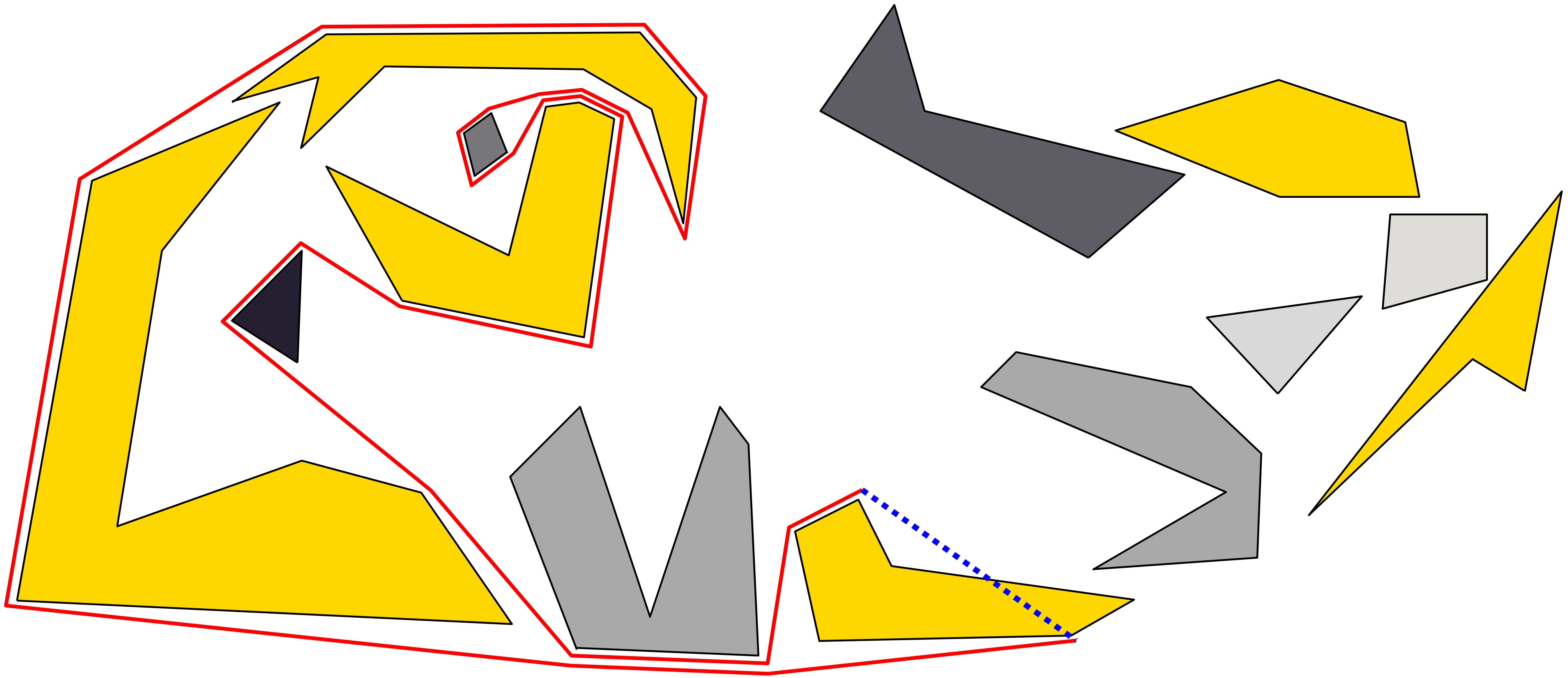


# Dynamic programming (DP) algorithm



*Mouth* of a partial solution may cut through objects.

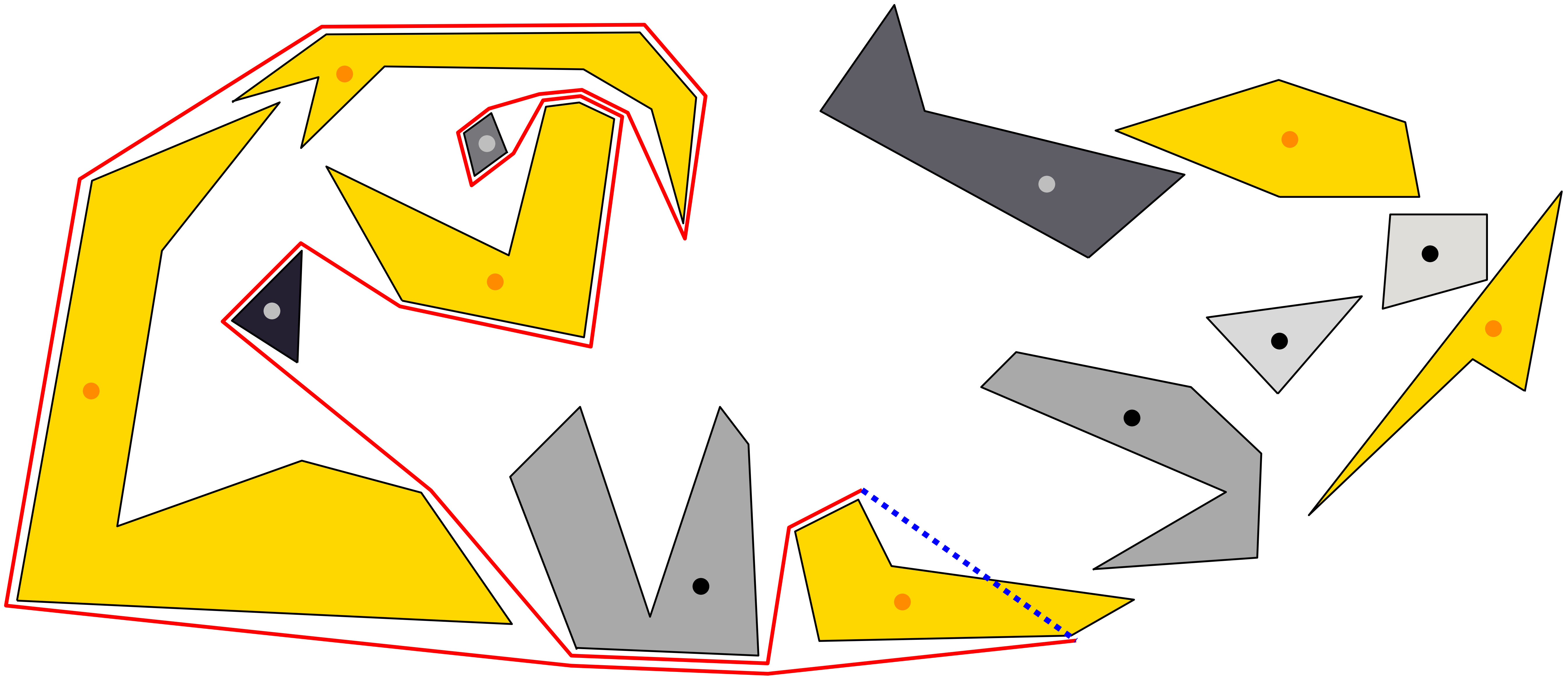
# Dynamic programming (DP) algorithm



*Mouth* of a partial solution may cut through objects.

We want to keep track of subset $B \subseteq R$ of required objects contained in a partial solution ($2^k$ choices).

# Dynamic programming (DP) algorithm



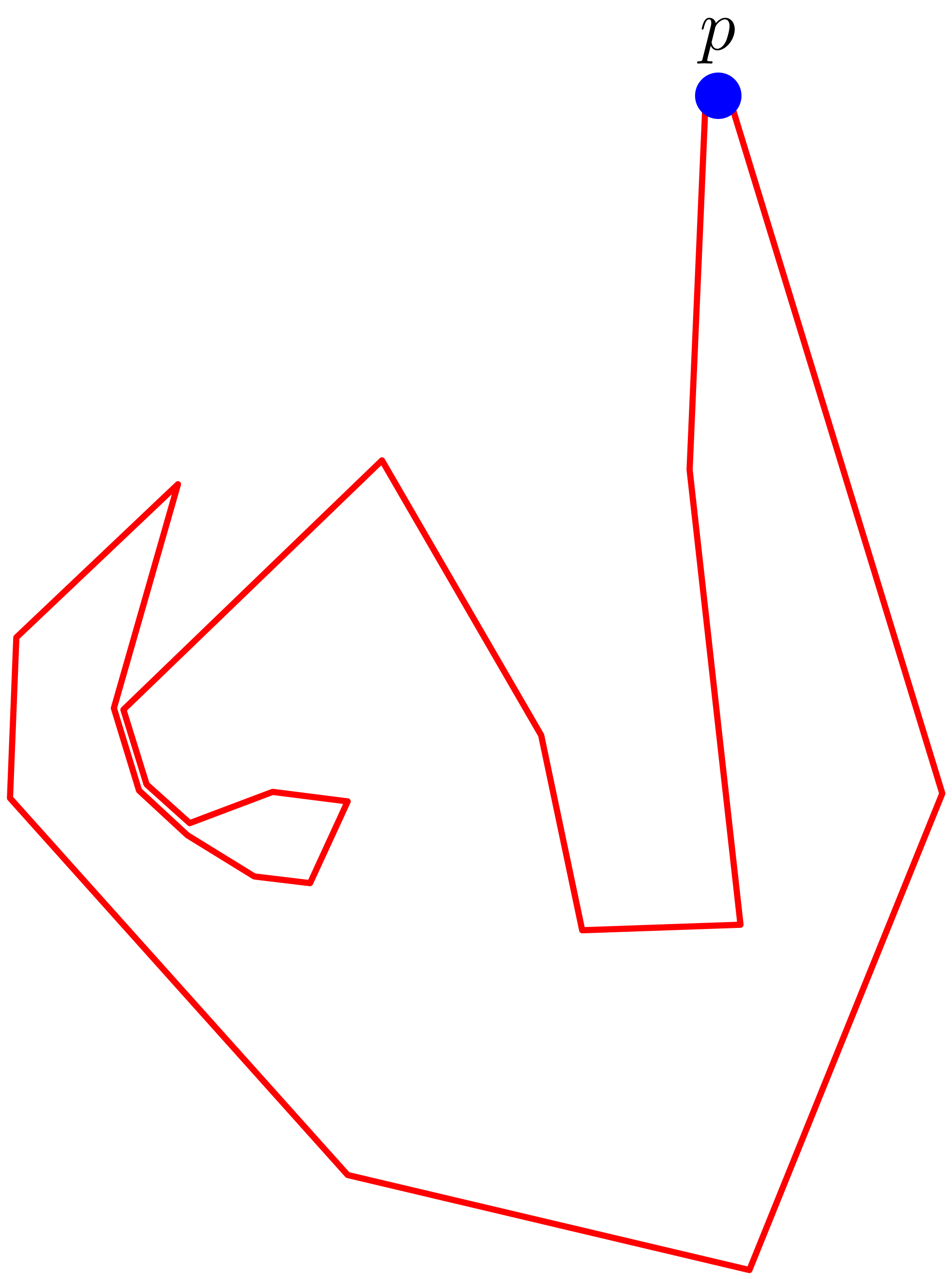
*Mouth* of a partial solution may cut through objects.

We want to keep track of subset $B \subseteq R$ of required objects contained in a partial solution ($2^k$ choices).

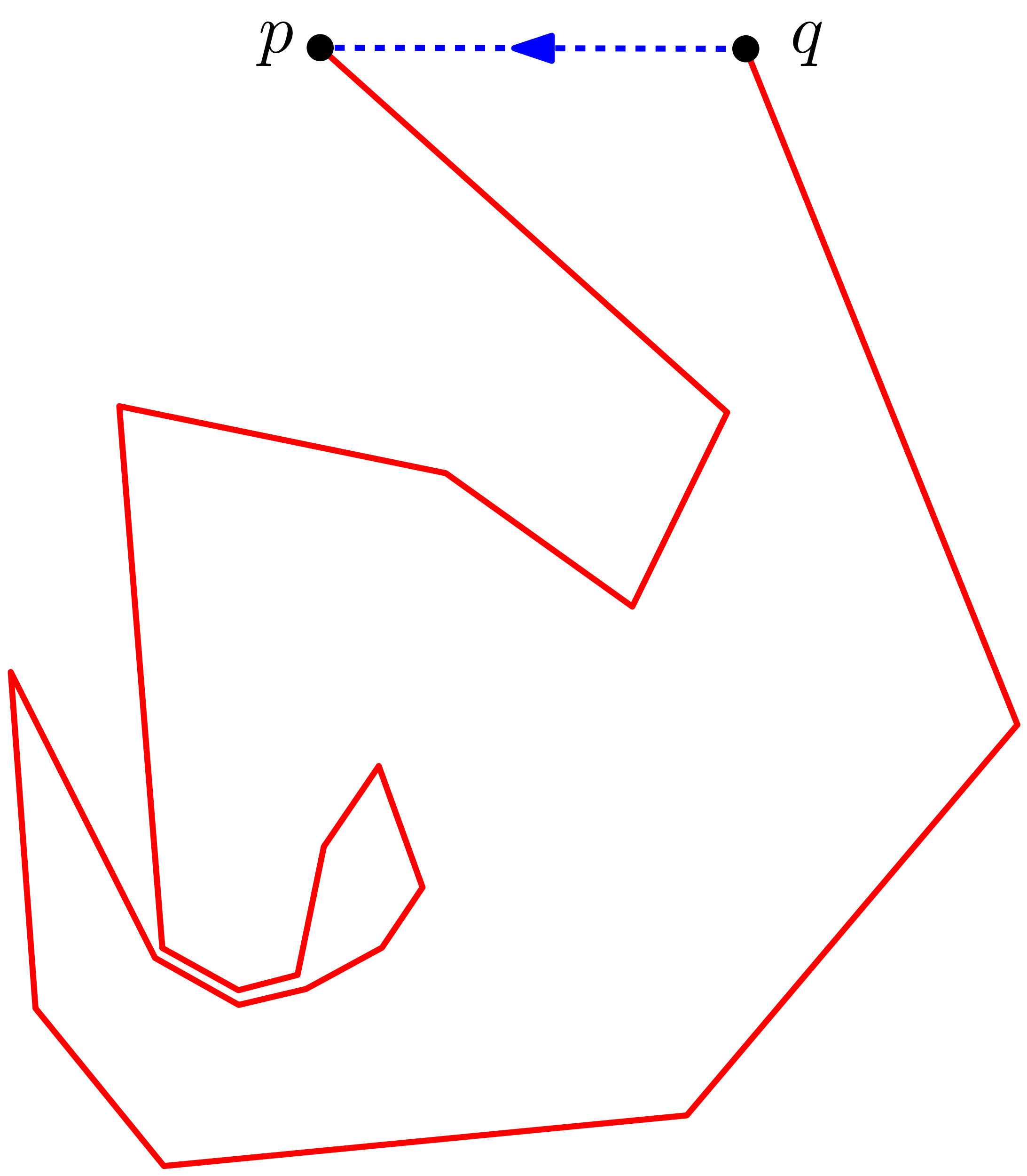
Choose a *reference point* for each object.

# Types of subproblems

<u>*Closed*</u> problems $C(p)$




Open problems $M(pq)$ with a <u>*mouth*</u> $qp$




$0 \le t \le 6n$

$B \subseteq R$ ($2^k$ choices)

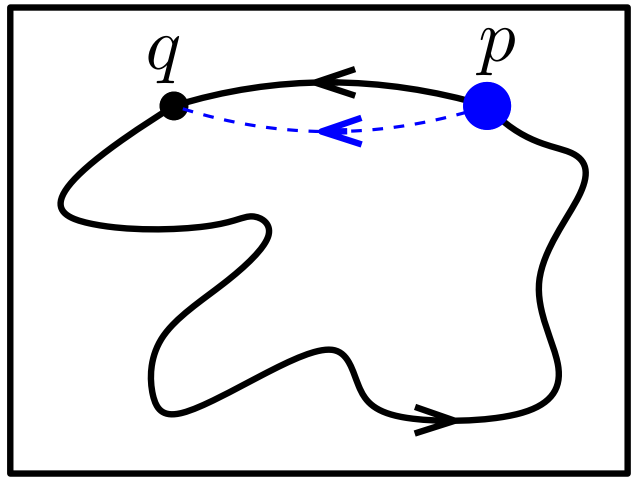
$C(p, t, B), M(pq, t, B) =$

least-cost solution with:

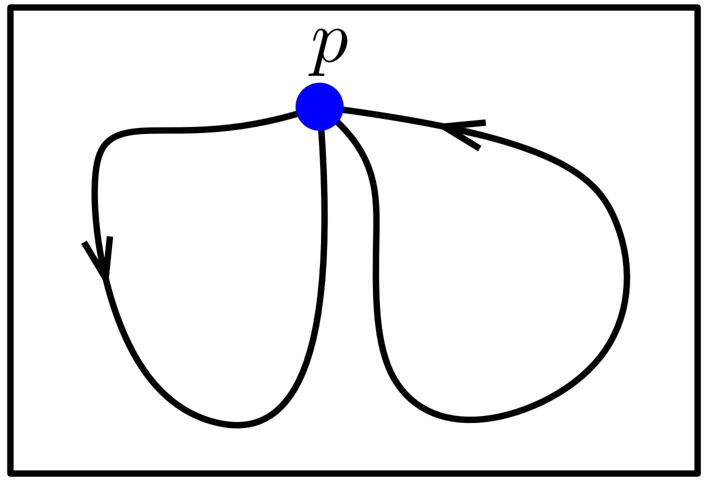
- at most $t$ edges
- enclosing exactly $B$ among the objects in $R$

# Recursion

$C(p, t, B)$:

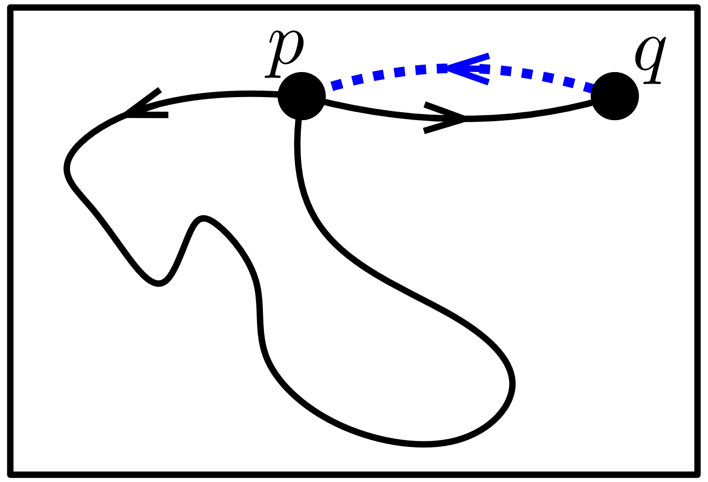



$w_{pq} + M(qp, t-1, B)$
$pq$ a free-space edge

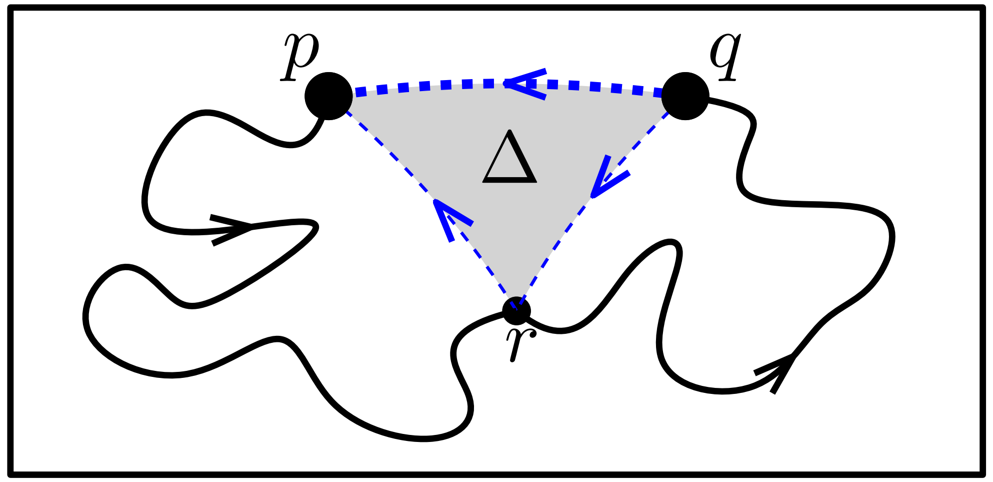



$C(p, t_1, B_1) + C(p, t_2, B_2)$
$B = B_1 \sqcup B_2,\ t = t_1 + t_2$

$M(pq, t, B)$:




$C(p, t-1, B) + w_{pq}$
$pq$ a free-space edge



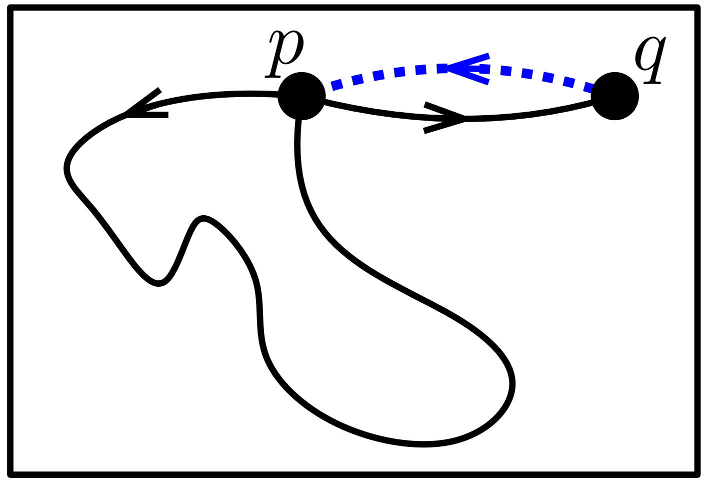

$M(pr, t_1, B_1) + M(rq, t_2, B_2) + \pi(\Delta)$
$B = B_1 \sqcup B_2 \sqcup R(\Delta),\ t = t_1 + t_2$

# Recursion

$$M(pq,t,B) := \min \begin{cases} C(p,t-1,B) + w_{pq} \text{ , if } pq \text{ is a free-space edge} \\ \min\{\, M(pr,t_1,B_1) + M(rq,t_2,B_2) + \pi(\Delta) \;\big|\; r,\ t = t_1 + t_2,\ B = B_1 \sqcup B_2 \sqcup R(\Delta) \,\} \end{cases}$$

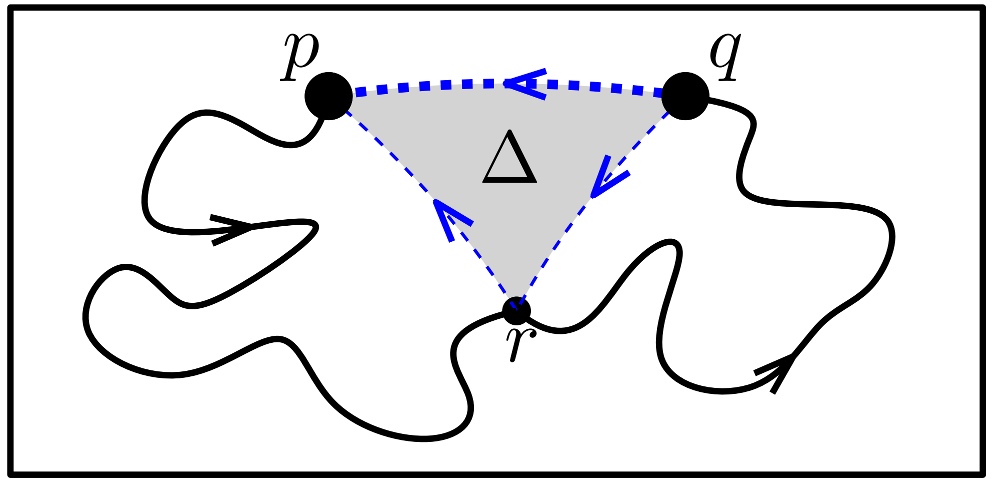
$$(\Delta = pqr)$$

$M(pq,t,B)$:




$C(p,t-1,B) + w_{pq}$

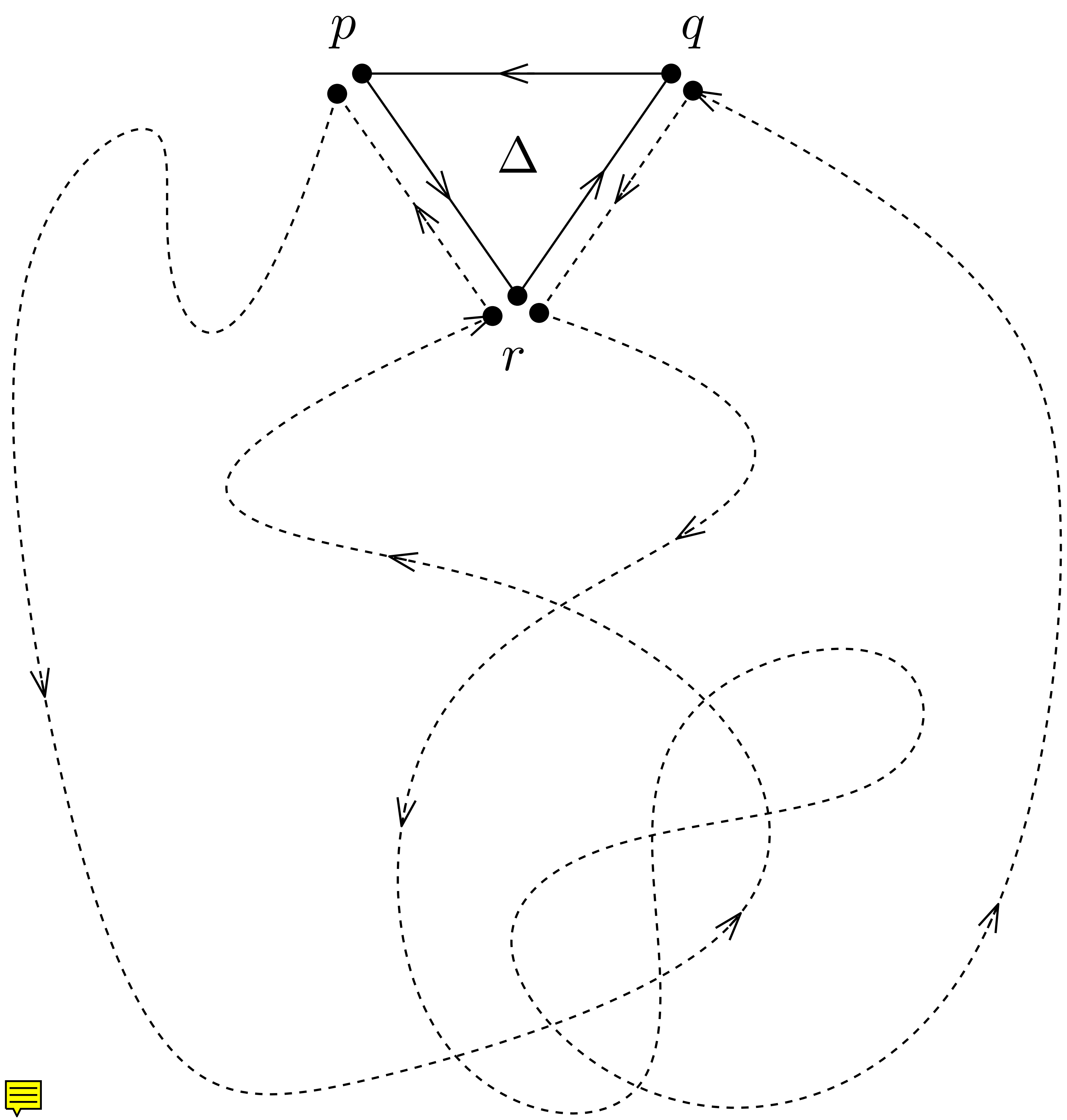
$pq$ a free-space edge




$M(pr,t_1,B_1) + M(rq,t_2,B_2) + \pi(\Delta)$

$B = B_1 \sqcup B_2 \sqcup R(\Delta),\ t = t_1 + t_2$

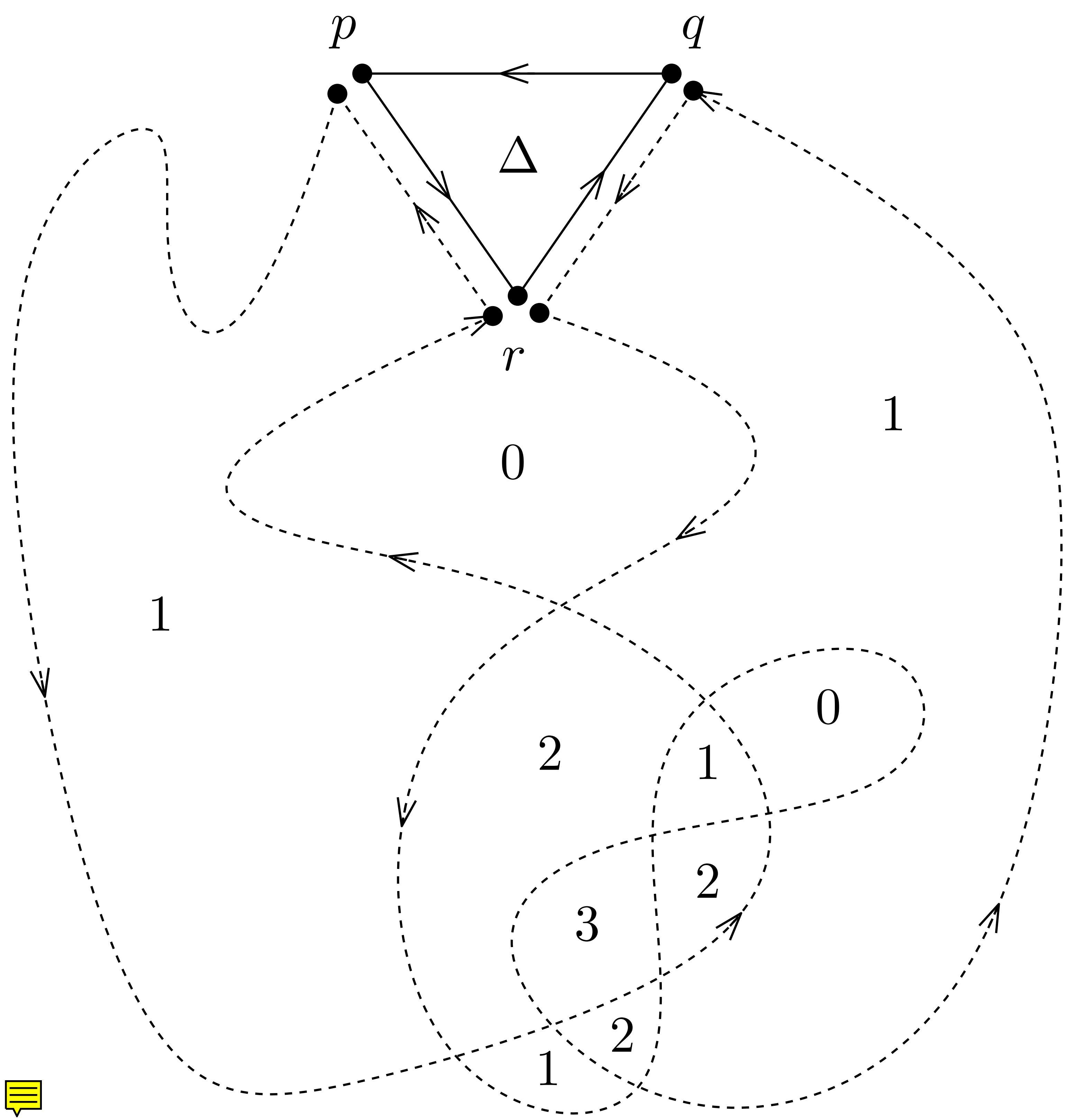
# Self-intersections




The DP algorithm cannot control self-intersections.

Winding numbers to the rescue!

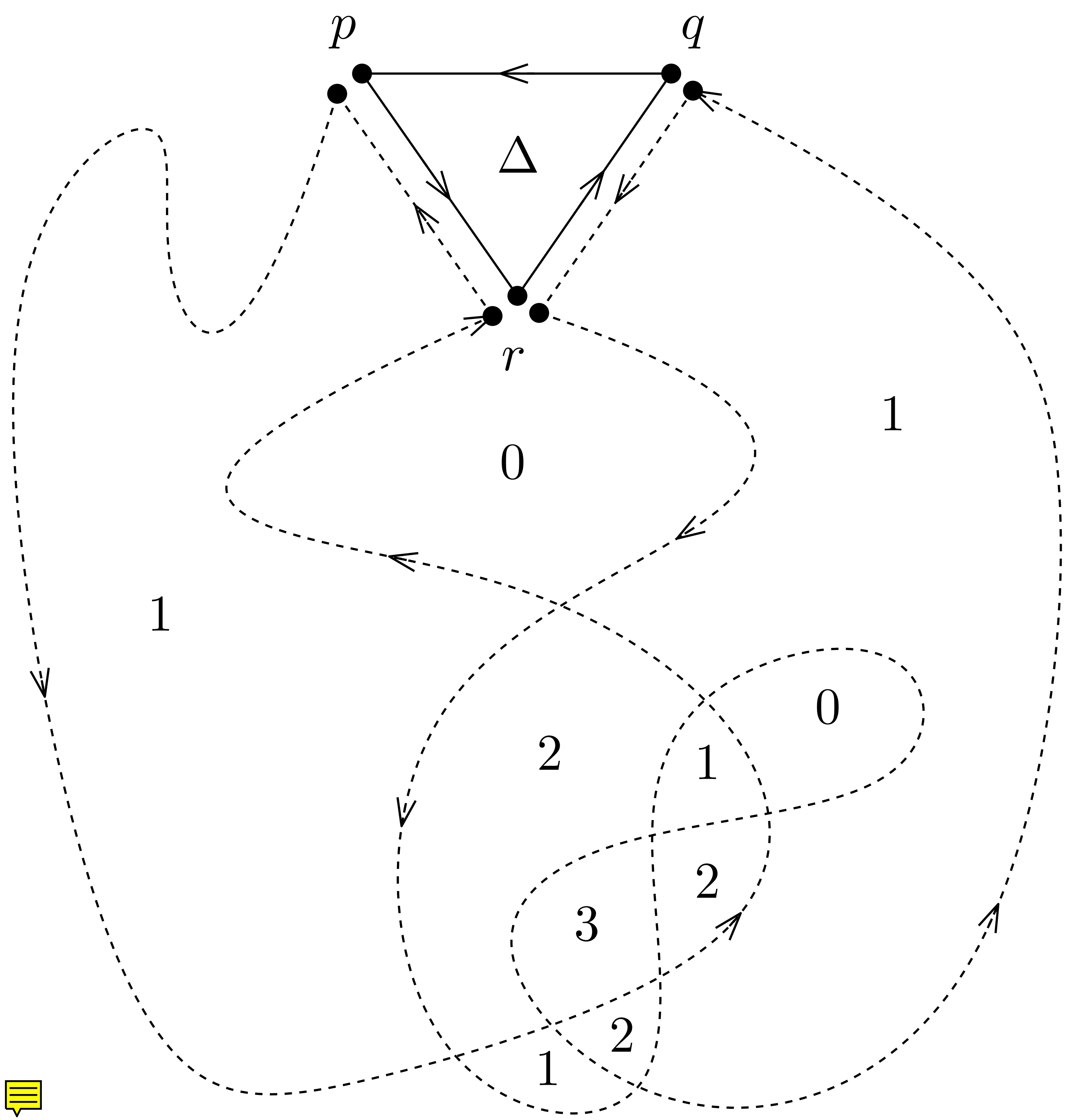
# Self-intersections




The DP algorithm cannot control self-intersections.

Winding numbers to the rescue!

# Self-intersections

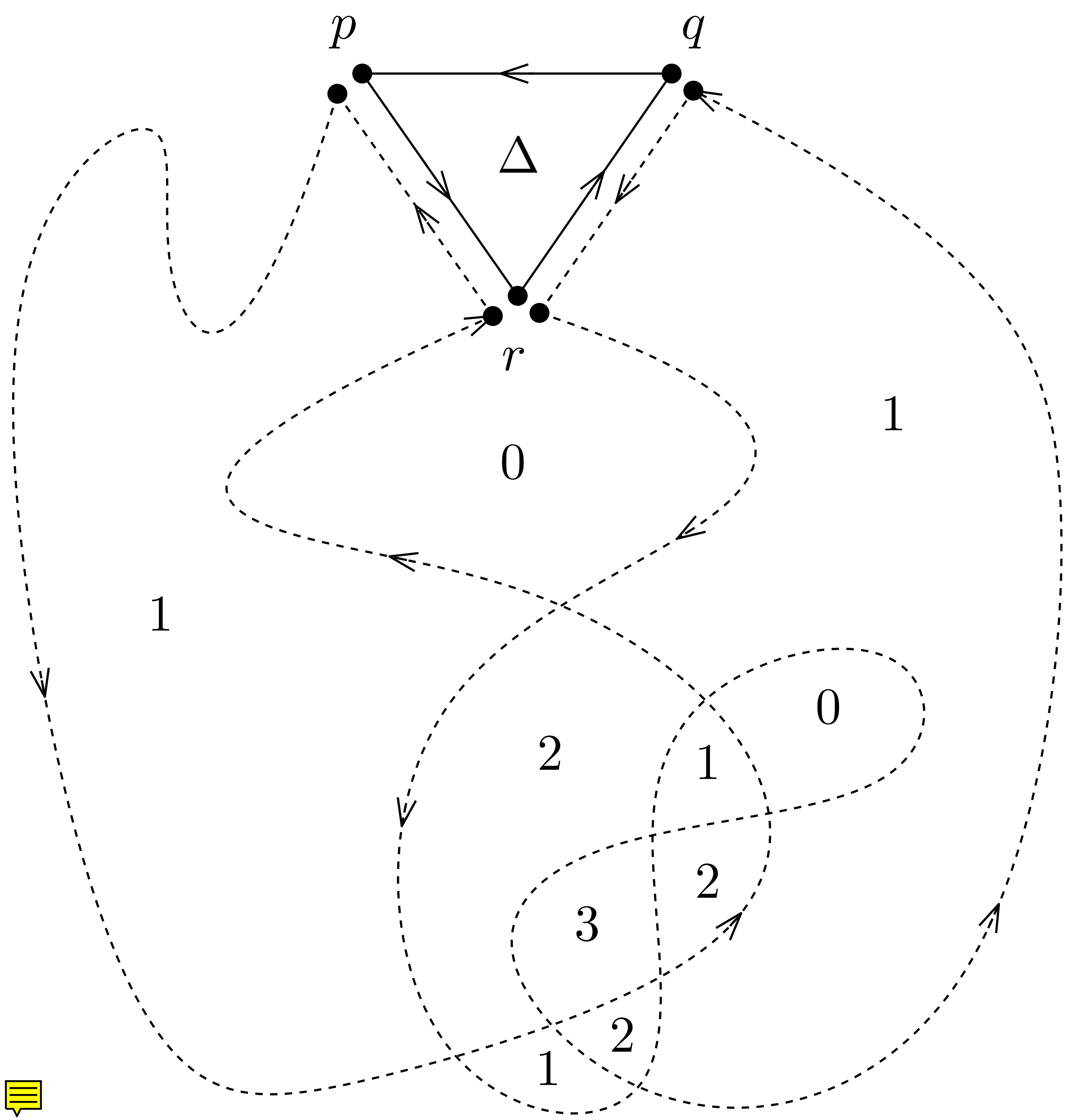



The DP algorithm cannot control self-intersections.

Winding numbers to the rescue!

- Objects in $B$ have winding number 1.
- Objects in $R \setminus B$ have winding number 0.
- Optional objects pay by winding number.
- Winding numbers $\geq 0$.

# Self-intersections




The DP algorithm cannot control self-intersections.

Winding numbers to the rescue!

- Objects in $B$ have winding number 1.
- Objects in $R \setminus B$ have winding number 0.

- Optional objects pay by winding number.

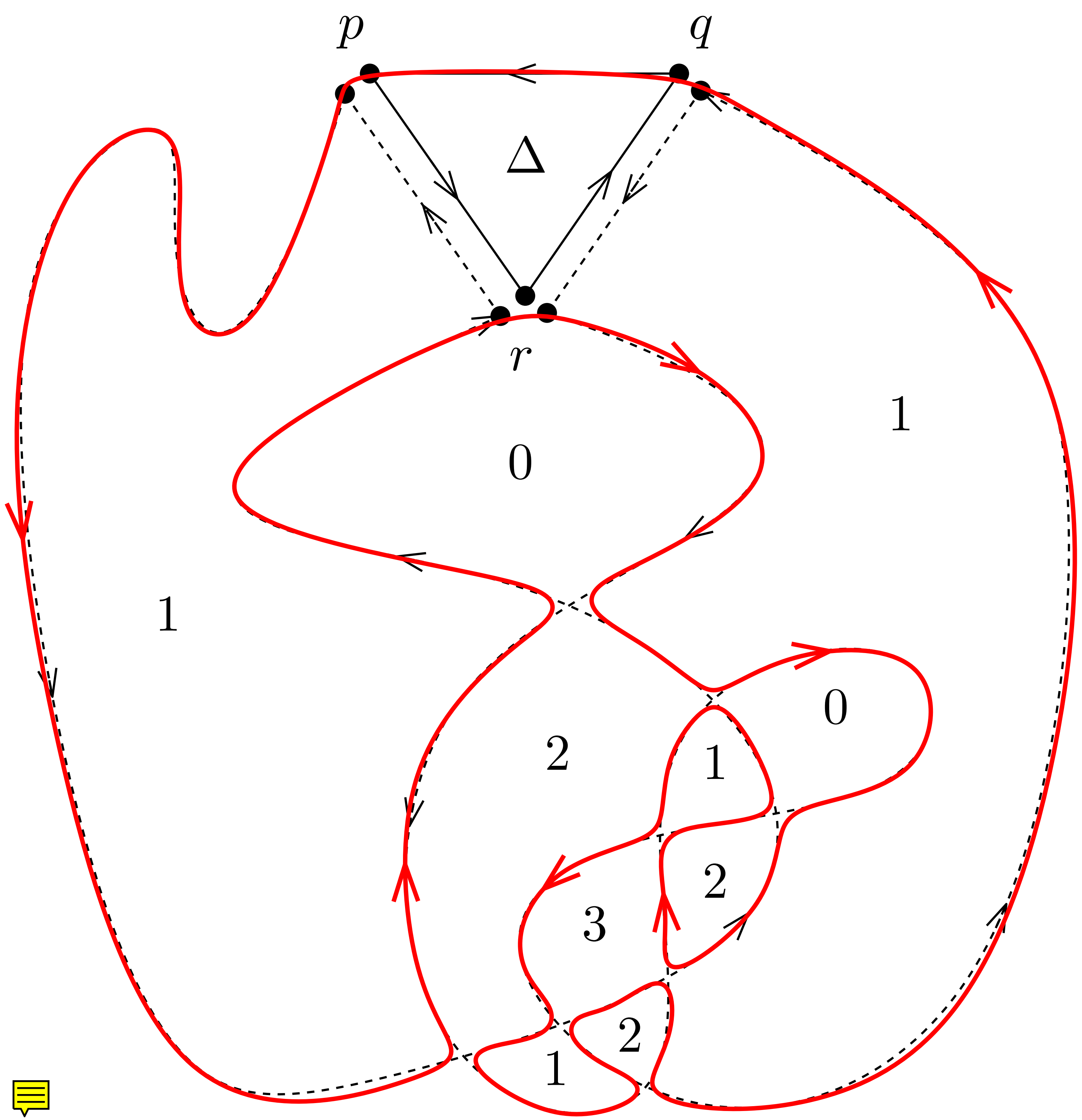
- Winding numbers $\geq 0$.

UNCROSSING ALGORITHM:
[Kotzig 1968] [Akitaya, Cs. Tóth 2018]

- ignore directions
- produce a weakly non-crossing curve (*non-crossing Euler tour*)

- Winding number *parity* is preserved.
- Winding numbers $\in \{0, 1\}$

# Self-intersections




The DP algorithm cannot control self-intersections.

Winding numbers to the rescue!

- Objects in $B$ have winding number 1.
- Objects in $R \setminus B$ have winding number 0.

- Optional objects pay by winding number.

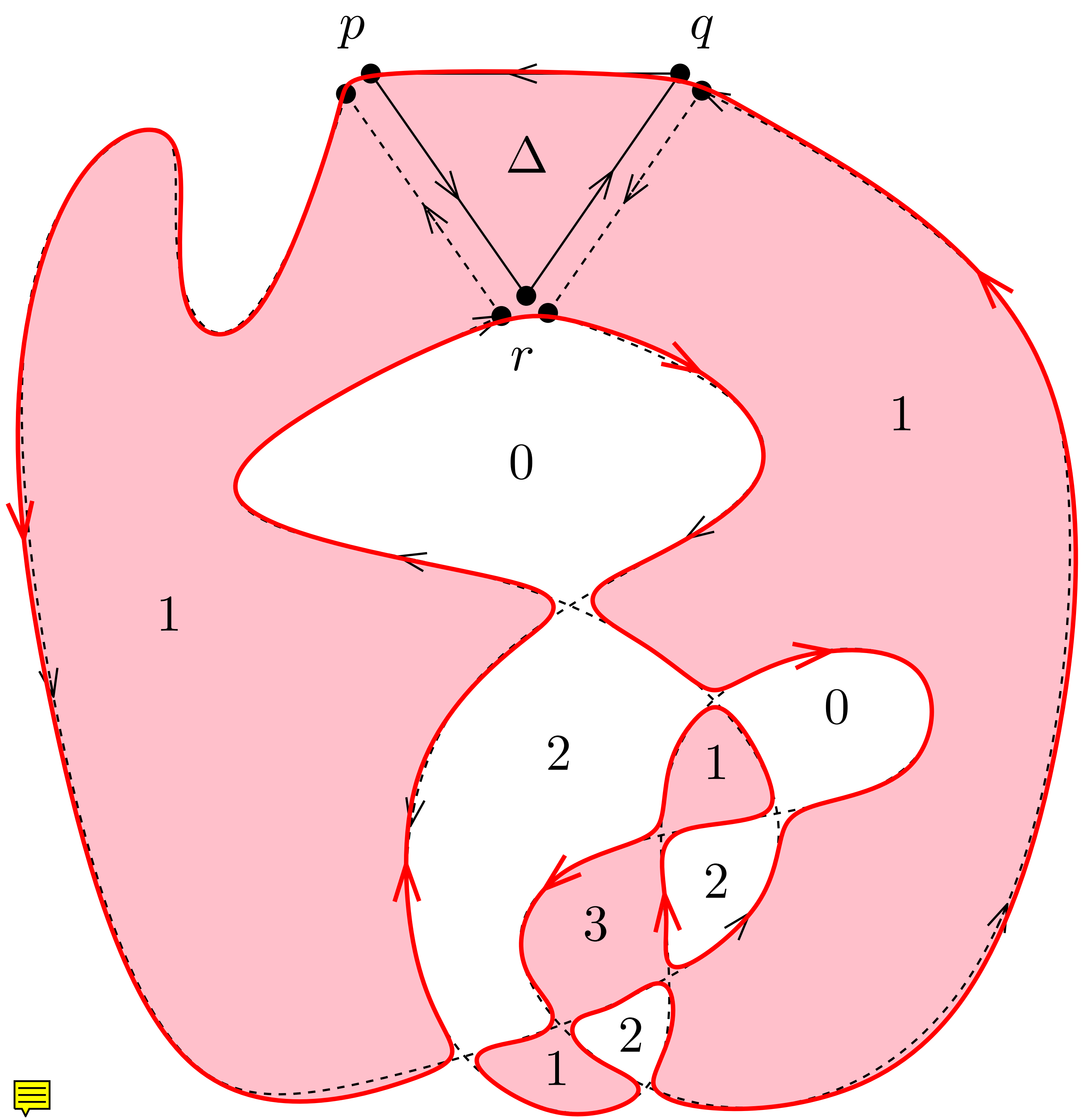
- Winding numbers $\geq 0$.

UNCROSSING ALGORITHM:
[Kotzig 1968] [Akitaya, Cs. Tóth 2018]

- ignore directions
- produce a weakly non-crossing curve (*non-crossing Euler tour*)

- Winding number *parity* is preserved.
- Winding numbers $\in \{0, 1\}$

# Self-intersections




The DP algorithm cannot control self-intersections.

Winding numbers to the rescue!

- Objects in $B$ have winding number 1.
- Objects in $R \setminus B$ have winding number 0.
- Optional objects pay by winding number.
- Winding numbers $\geq 0$.

UNCROSSING ALGORITHM:
[Kotzig 1968] [Akitaya, Cs. Tóth 2018]

- ignore directions
- produce a weakly non-crossing curve (*non-crossing Euler tour*)
- Winding number *parity* is preserved.
- Winding numbers $\in \{0, 1\}$

Stays feasible. Cost can only go down.

# Correctness



$$\mathrm{cost}_{\mathrm{OPT}} \le \mathrm{cost}_{\mathrm{ALG}} \le \mathrm{cost}_{\mathrm{DP}} \le \mathrm{cost}_{\mathrm{OPT}}$$

↑ after uncrossing

# Correctness

$$\text{cost}_{\text{OPT}} \le \text{cost}_{\text{ALG}} \overset{\checkmark}{\le} \text{cost}_{\text{DP}} \le \text{cost}_{\text{OPT}}$$

↑ after uncrossing

# Correctness



$$\mathrm{cost}_{\mathrm{OPT}} \overset{\checkmark}{\leq} \mathrm{cost}_{\mathrm{ALG}} \overset{\checkmark}{\leq} \mathrm{cost}_{\mathrm{DP}} \leq \mathrm{cost}_{\mathrm{OPT}}$$

after uncrossing

DP + UNCROSS produces a feasible solution.

# Correctness

$$\text{cost}_{\text{OPT}} \overset{\checkmark}{\leq} \text{cost}_{\text{ALG}} \overset{\checkmark}{\leq} \text{cost}_{\text{DP}} \leq \text{cost}_{\text{OPT}}$$

after uncrossing

DP + UNCROSS produces a feasible solution.

OPT can be “found” by the DP:

# Correctness

$$\text{cost}_{\text{OPT}} \overset{\checkmark}{\leq} \text{cost}_{\text{ALG}} \overset{\checkmark}{\leq} \text{cost}_{\text{DP}} \leq \text{cost}_{\text{OPT}}$$

after uncrossing

DP + UNCROSS produces a feasible solution.

OPT can be "found" by the DP:

- OPT uses only straight edges between object vertices.
- Triangulated OPT can be successively built, corresponding to the DP recursions.
- $t \leq 6n$ by planarity.

# Correctness

$$\text{cost}_{\text{OPT}} \overset{\checkmark}{\leq} \text{cost}_{\text{ALG}} \overset{\checkmark}{\leq} \text{cost}_{\text{DP}} \overset{\checkmark}{\leq} \text{cost}_{\text{OPT}}$$

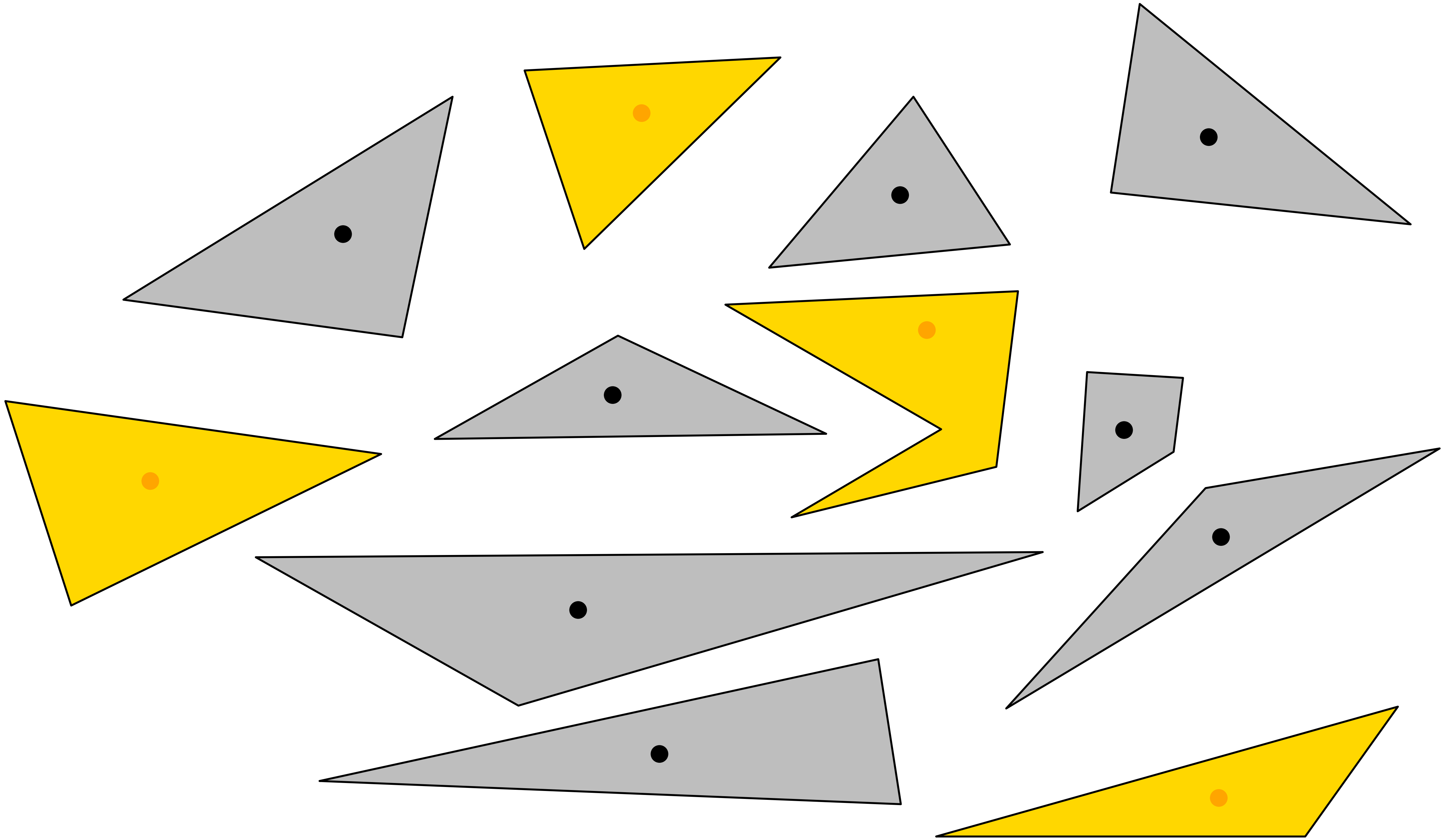
after uncrossing

DP + UNCROSS produces a feasible solution.

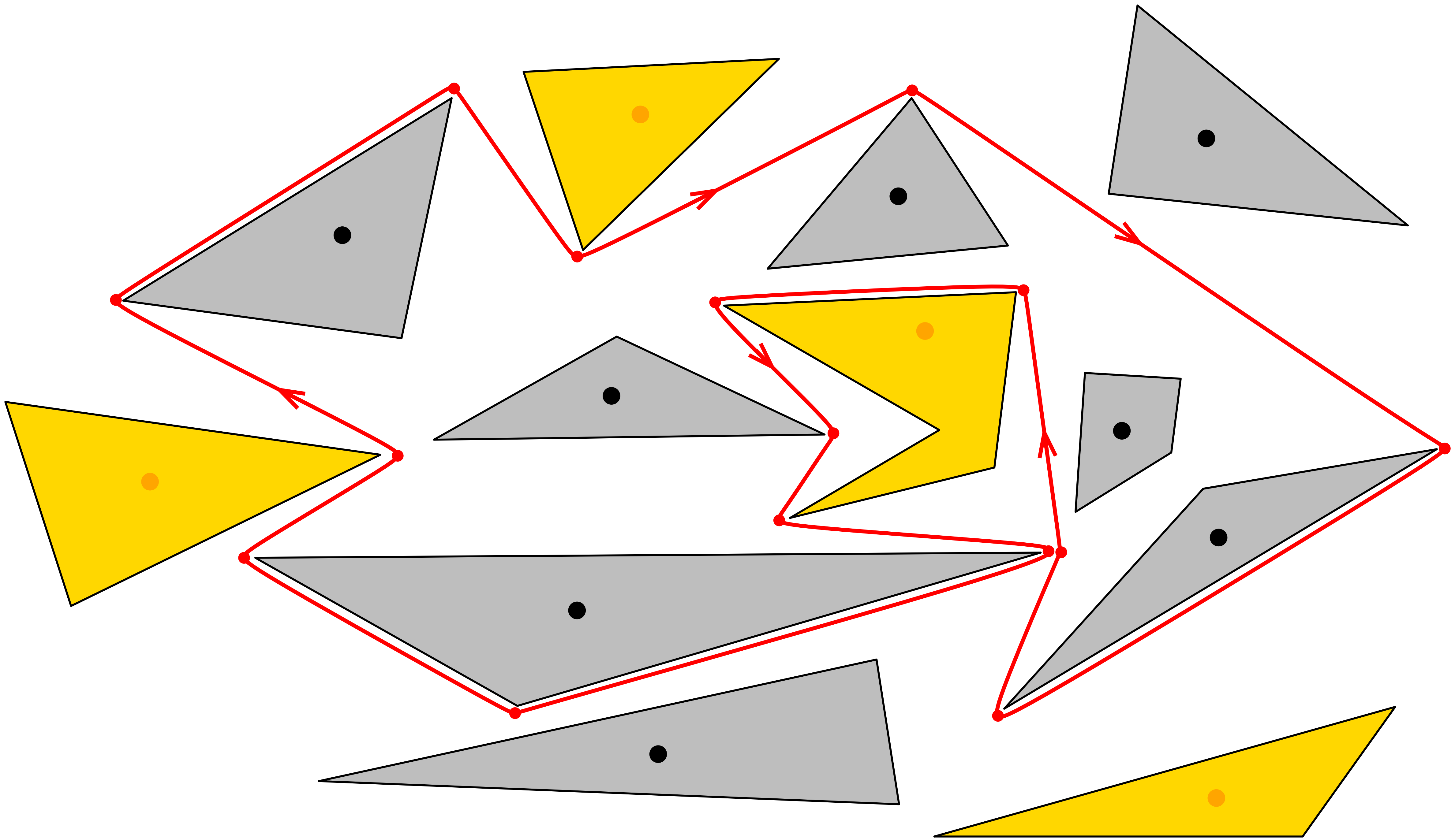
OPT can be “found” by the DP:

- OPT uses only straight edges between object vertices.
- Triangulated OPT can be successively built, corresponding to the DP recursions.
- $t \leq 6n$ by planarity.

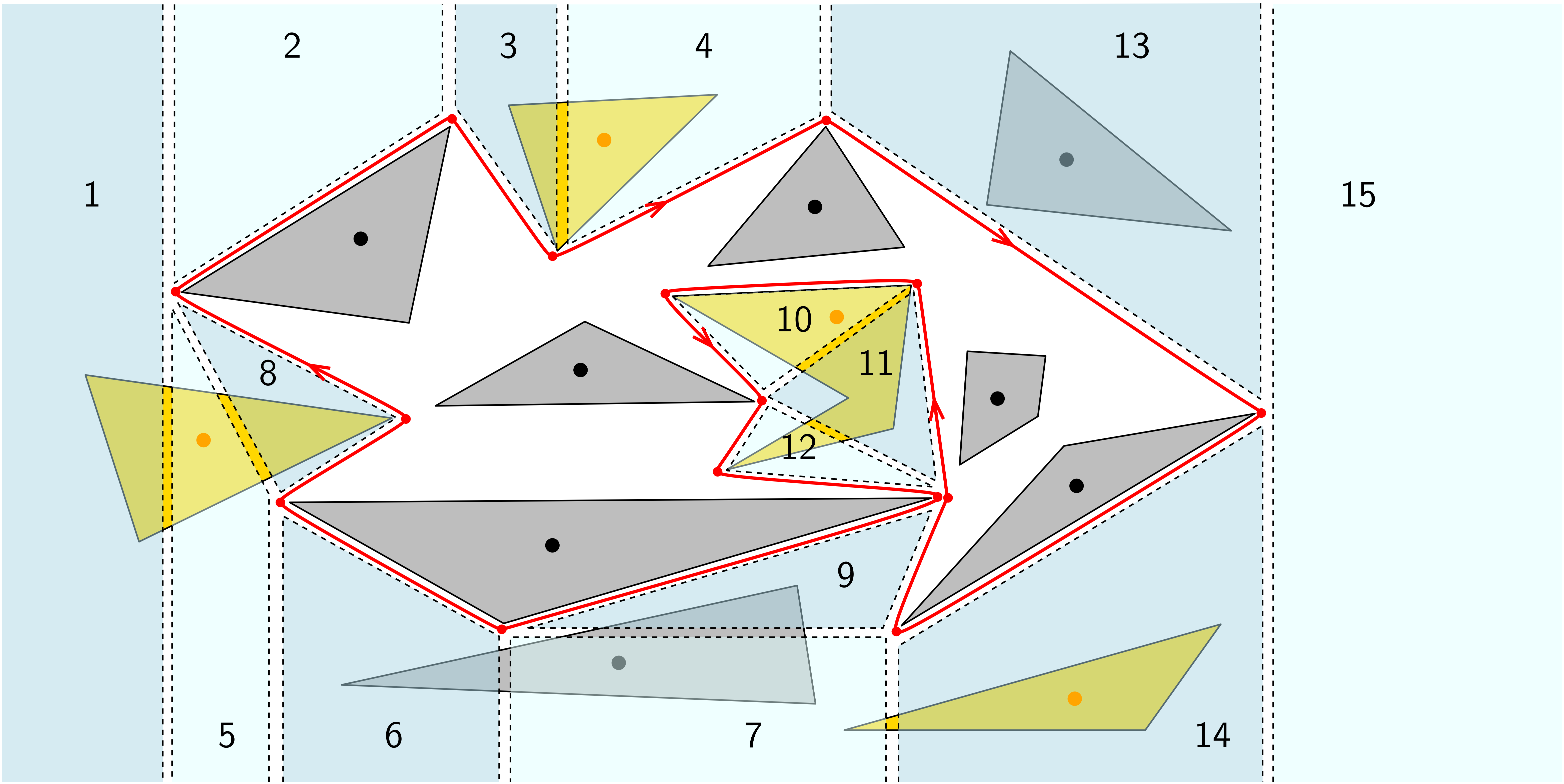
# The inverted problem



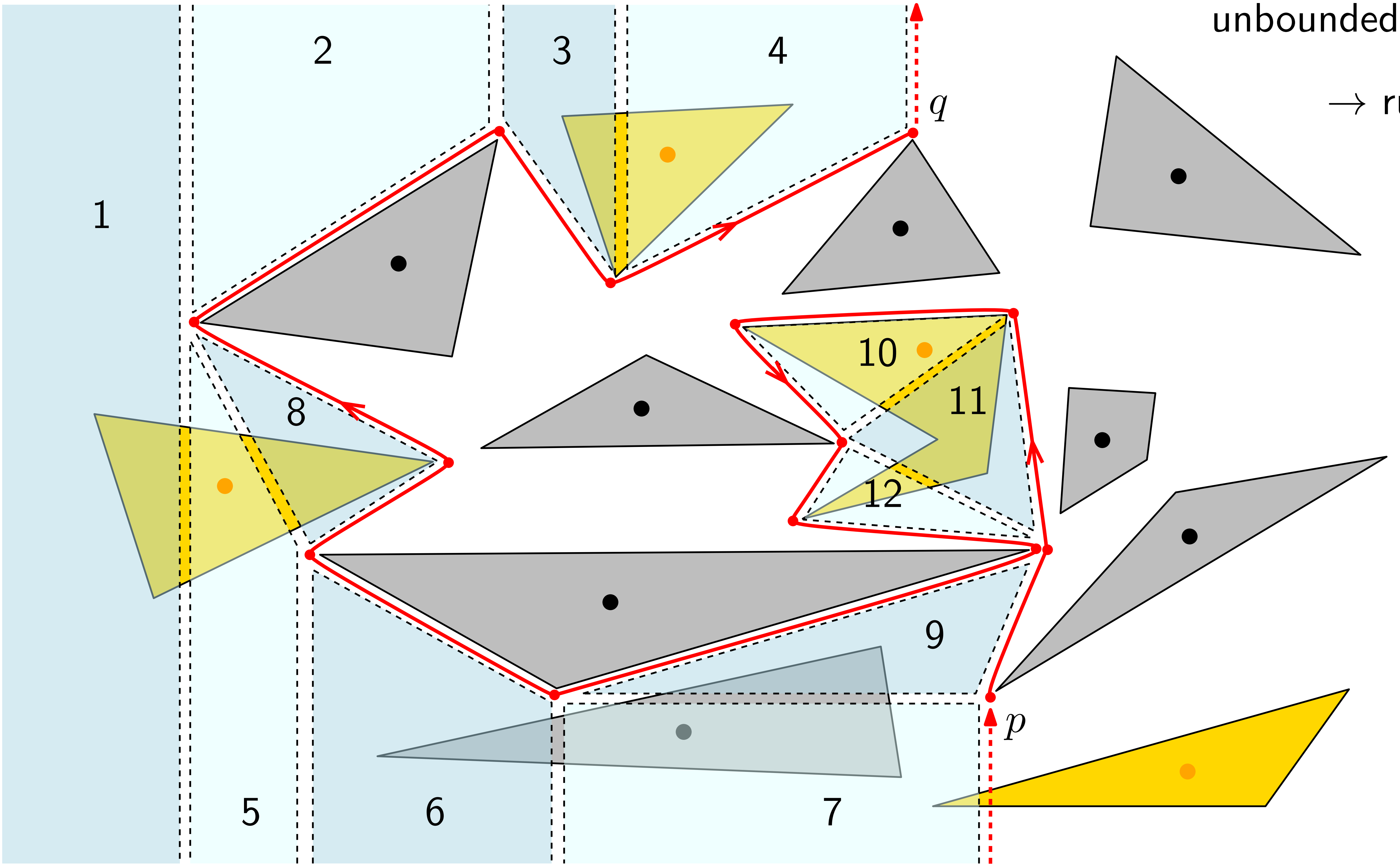
# The inverted problem



# The inverted problem



# The inverted problem




unbounded subproblems $U(p, q)$

$\rightarrow$ runtime is unchanged.

# Speedup by Dijkstra-style algorithm



From $O(3^k n^5)$ to $O(3^k n^3)$:

$$M(pq, \xcancel{t,} B) := \min \begin{cases} C(p, \xcancel{t-1,} B) + w_{pq}\ , & \text{if } pq \text{ is a free-space edge} \\ \min\{\, M(pr, \xcancel{t_1,} B_1) + M(rq, \xcancel{t_2,} B_2) + \pi(\Delta) \mid r, \xcancel{t = t_1 + t_2,}\ B = B_1 \sqcup B_2 \sqcup R(\Delta) \,\} \end{cases}$$

$$C(p, \xcancel{t,} B) := \dots$$

- Maintain *tentative* values $M(pq, B)$ and $C(p, B)$
- The smallest of the tentative values is made *permanent*.
- For all equations where this value appears on the *right-hand side*, the tentative values on the left-hand side are updated.

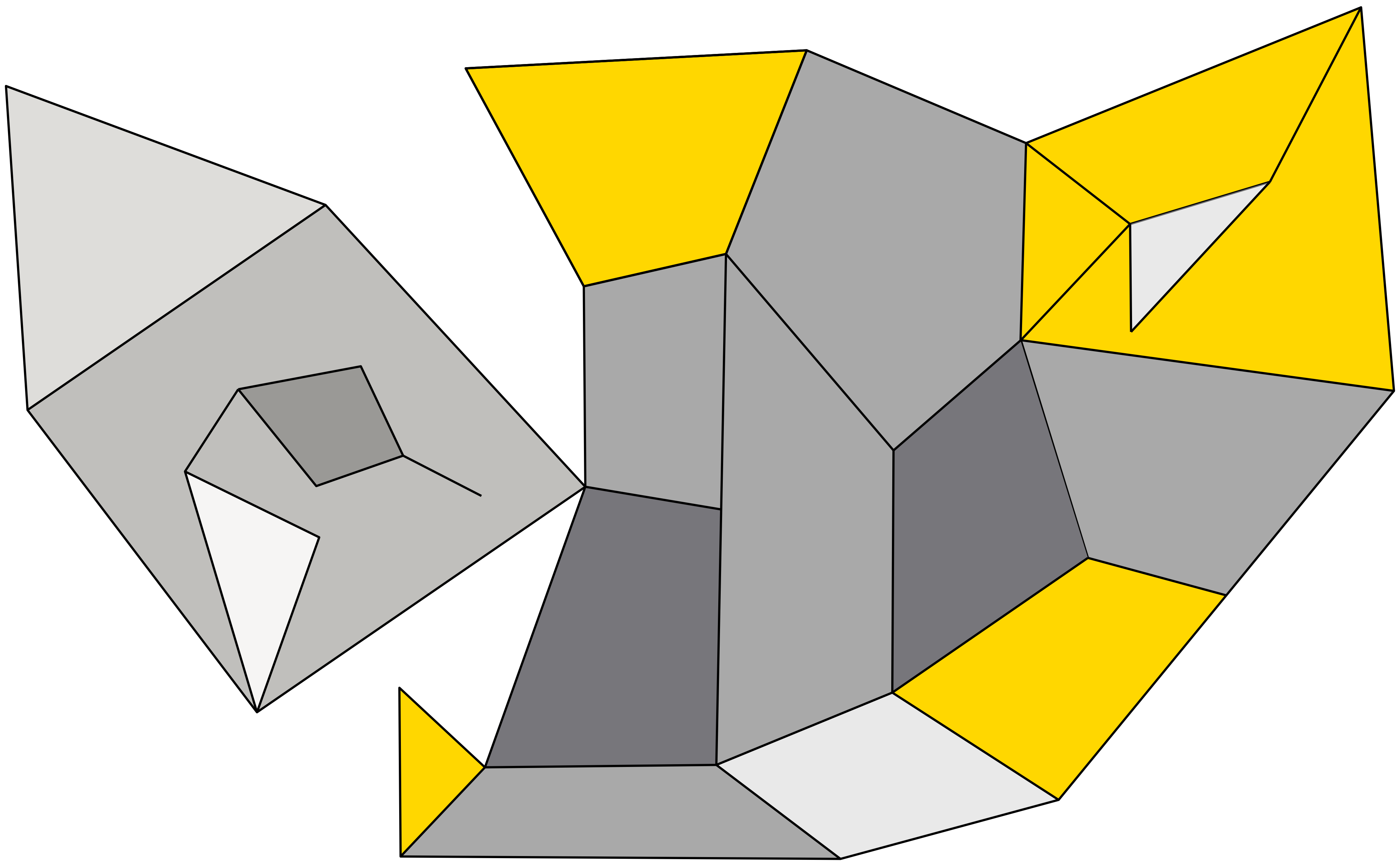
Cyclic dependencies? Why does this work:
The left-hand side is larger than the values appearing on the right-hand side.

[Knuth 1977]: "superior context-free grammar"

# GRAPH-Enclosure with penalties

INPUT: a plane graph

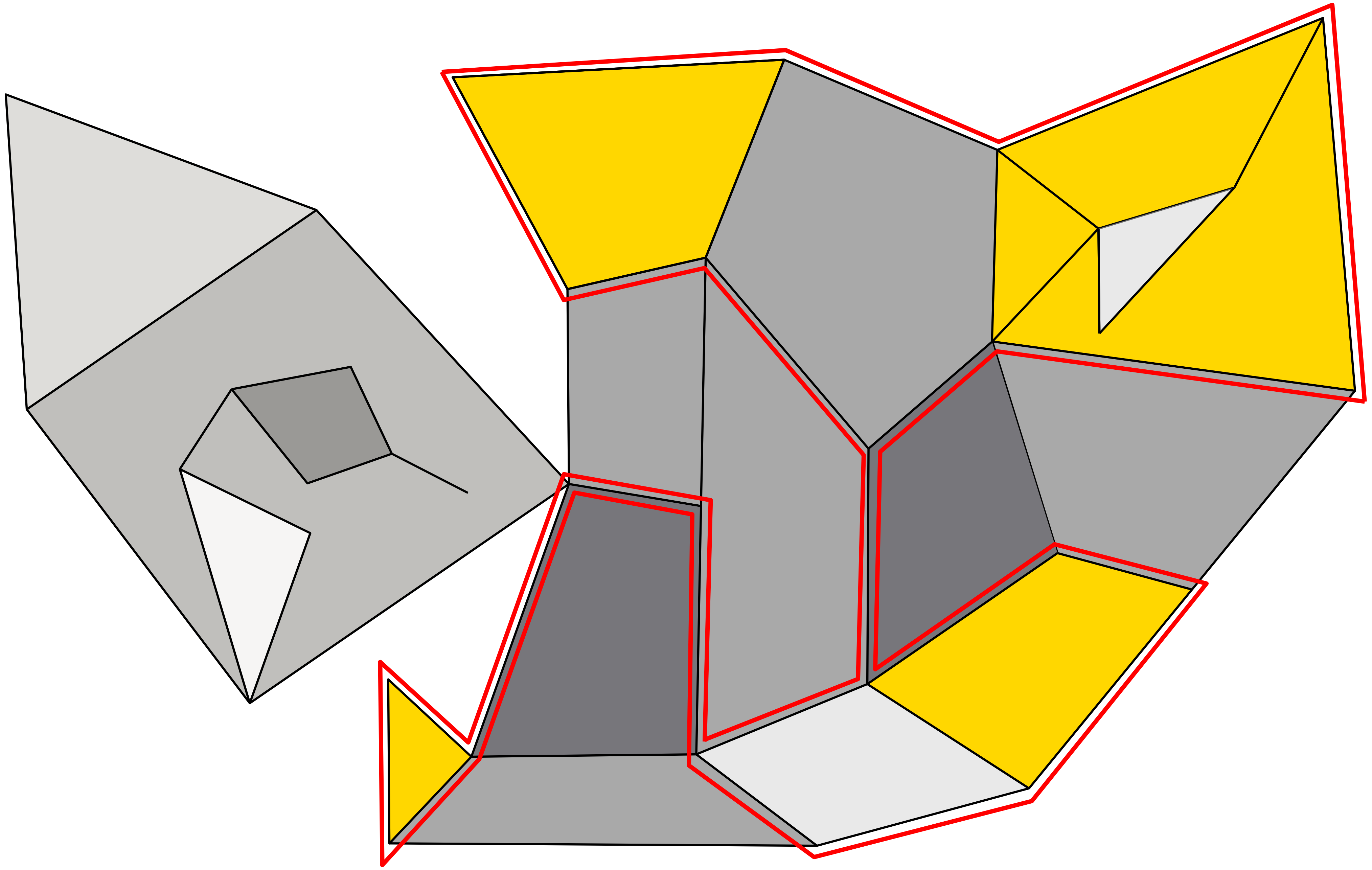
- required and optional *faces*
- *arbitrary* edge costs $> 0$



# GRAPH-Enclosure with penalties

INPUT: a plane graph

- required and optional *faces*
- *arbitrary* edge costs $> 0$

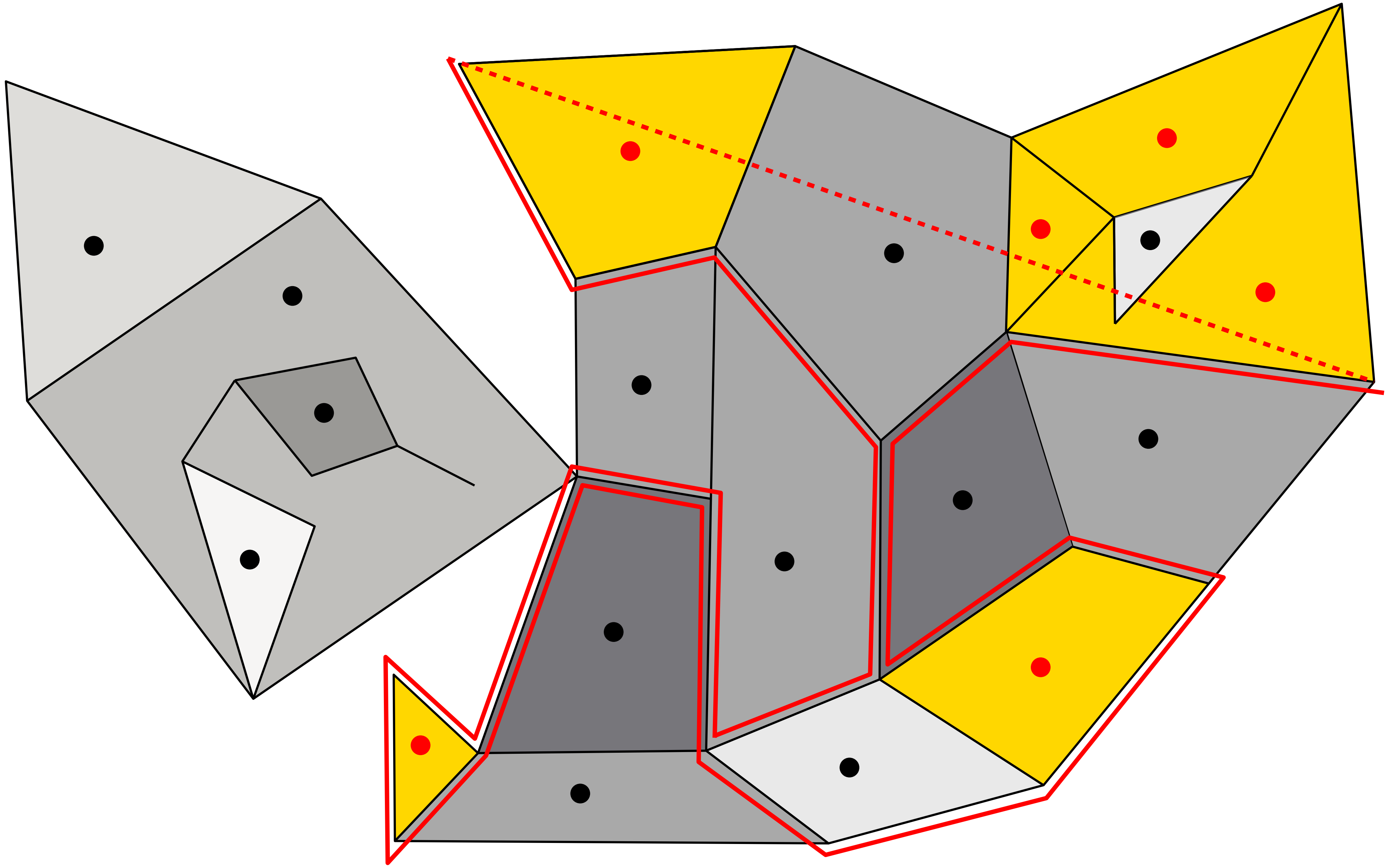


Reduction to (slighly generalized) GEOMETRIC-enclosure by an (arbitrary) straight-line embedding

# GRAPH-Enclosure with penalties

INPUT: a plane graph

- required and optional *faces*
- *arbitrary* edge costs $> 0$

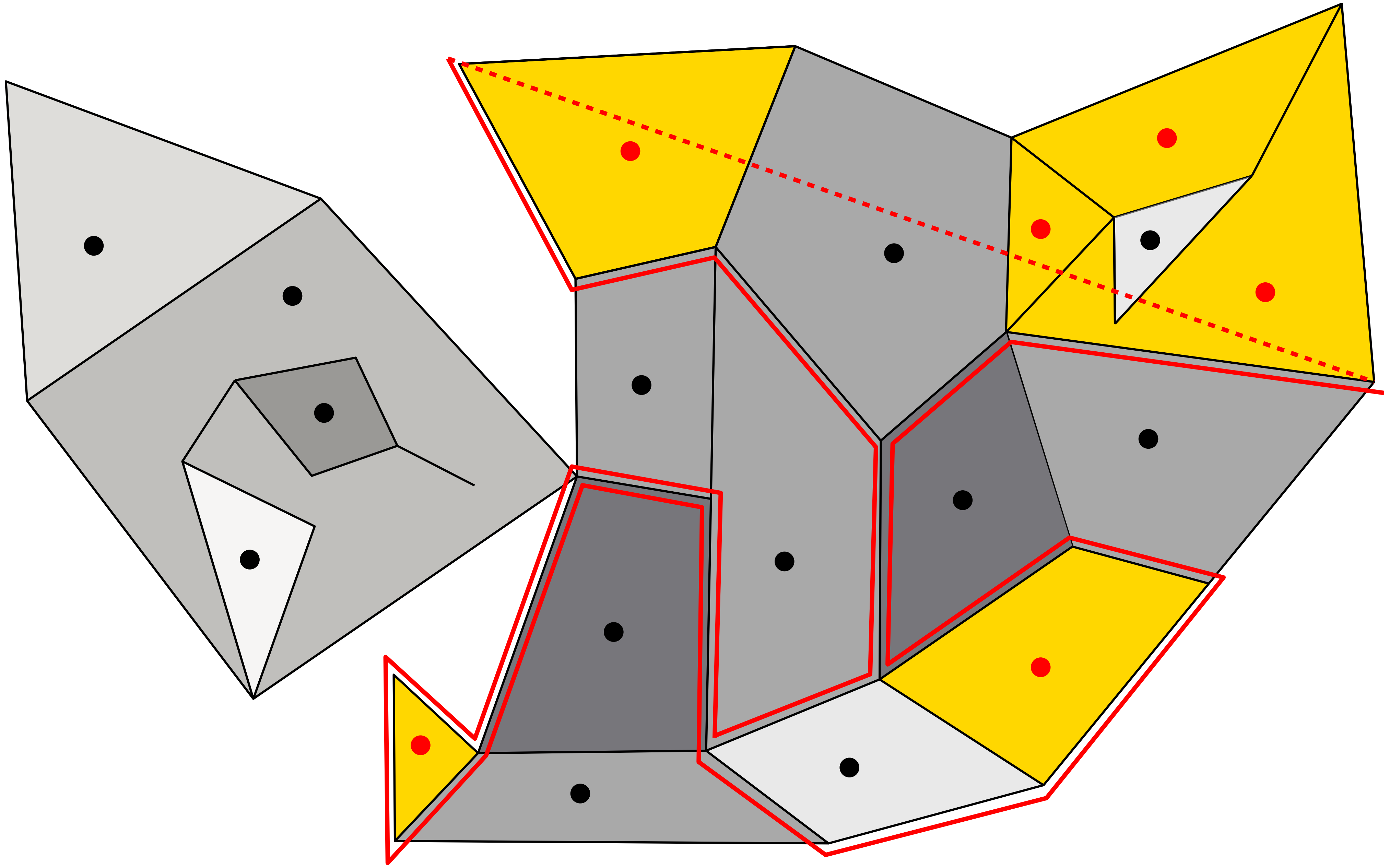


Reduction to (slighly generalized) GEOMETRIC-enclosure by an (arbitrary) straight-line embedding

# GRAPH-Enclosure with penalties

INPUT: a plane graph

- required and optional *faces*
- *arbitrary* edge costs $> 0$

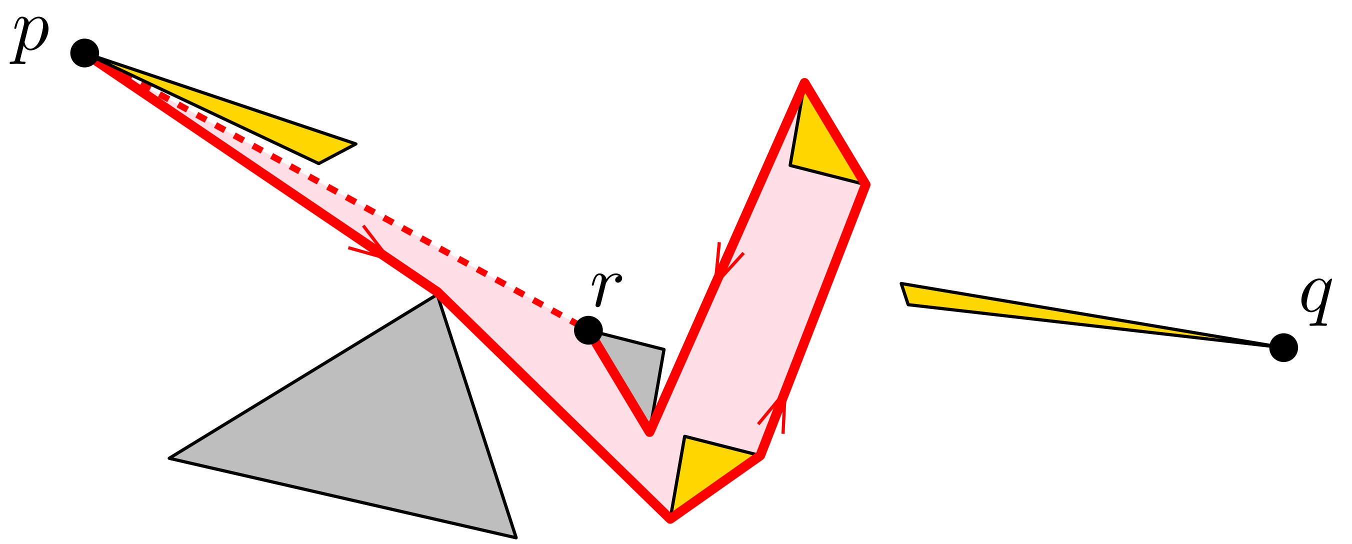


Reduction to (slighly generalized) GEOMETRIC-enclosure by an (arbitrary) straight-line embedding

Point objects can be handled. Extension to weakly simple objects is open.

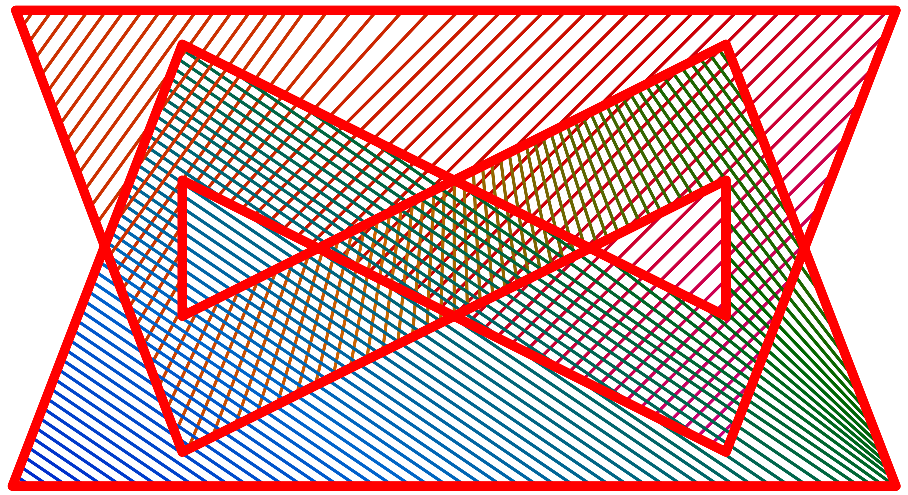
# What does $C(p, t, B)$ and $M(pq, t, B)$ REALLY represent?

- For subproblems $M(qp)$, self-intersections *do* occur:

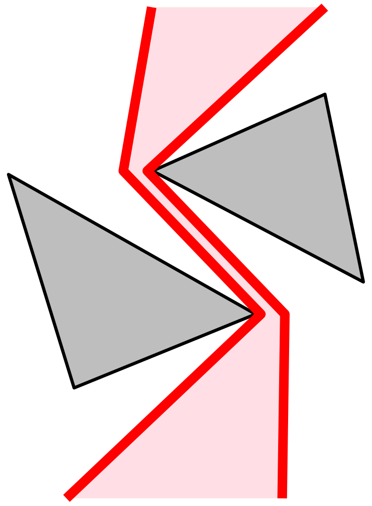



- weakly simple immersed polygons

self-overlapping + weakly simple



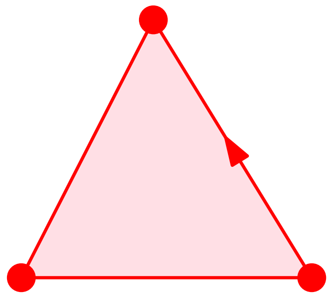
Milnor's doodle



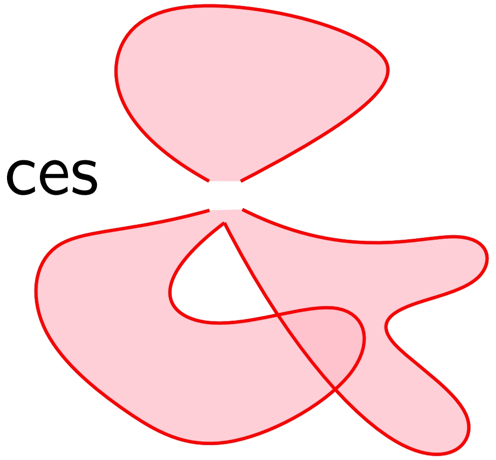
- Recognition algorithm?

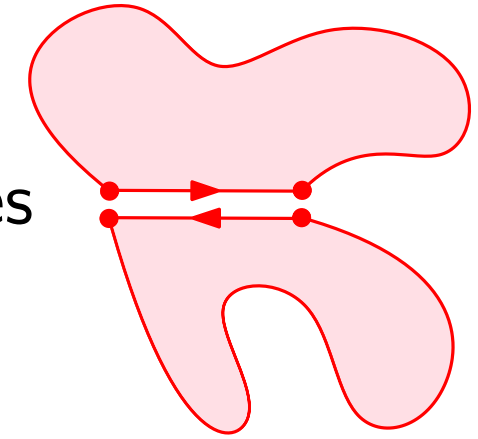
- basic building blocks

digons 

triangles 

- glue together at

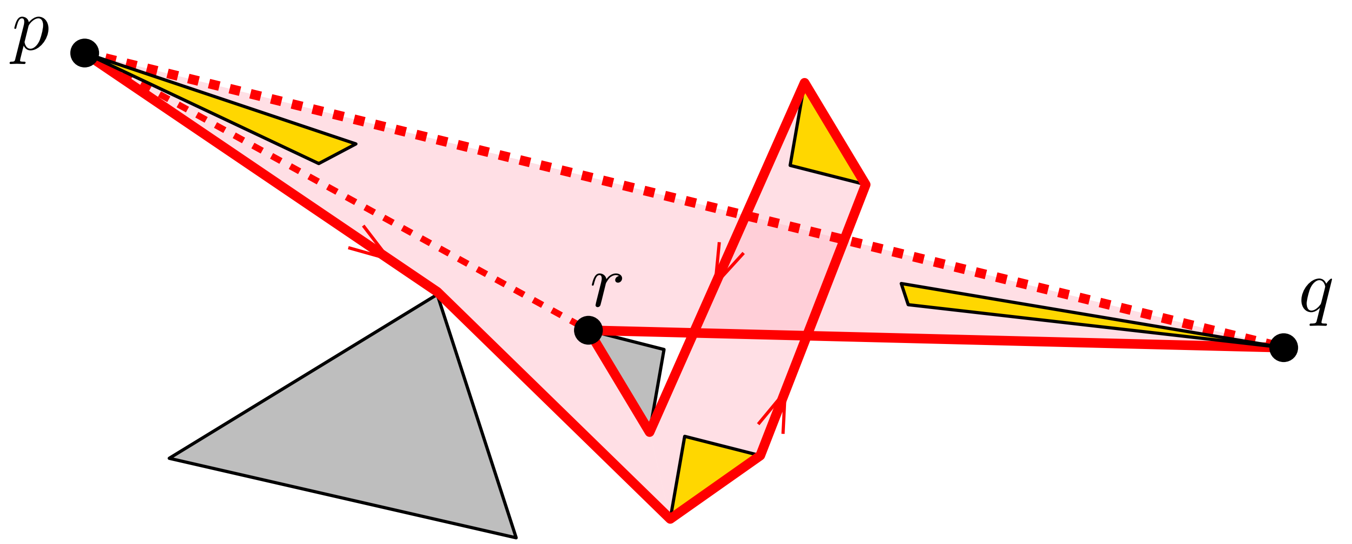
vertices 

edges 

# What does $C(p, t, B)$ and $M(pq, t, B)$ REALLY represent?

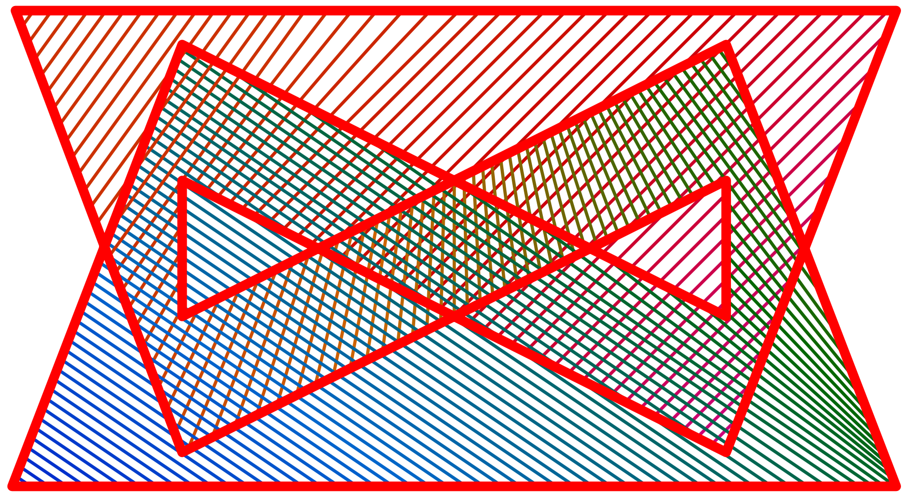


- For subproblems $M(qp)$, self-intersections *do* occur:

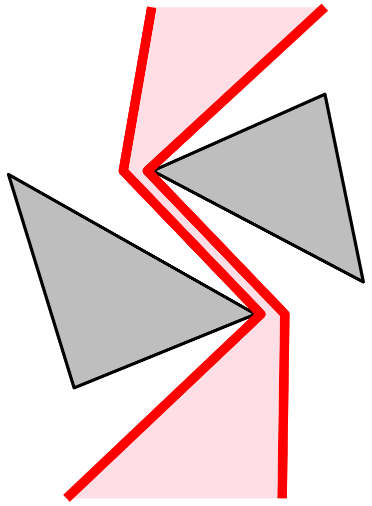



- weakly simple immersed polygons

self-overlapping + weakly simple



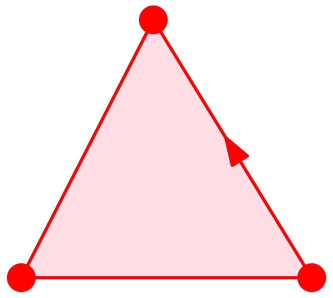
Milnor's doodle



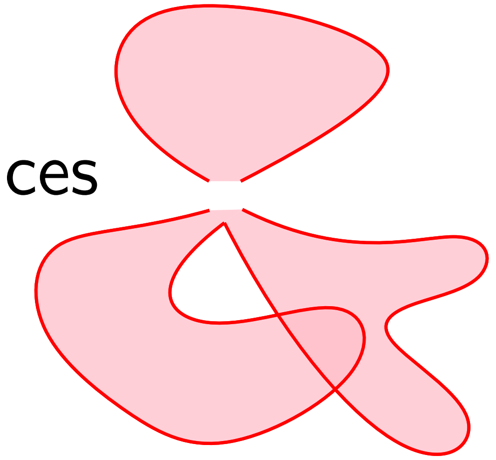
- Recognition algorithm?

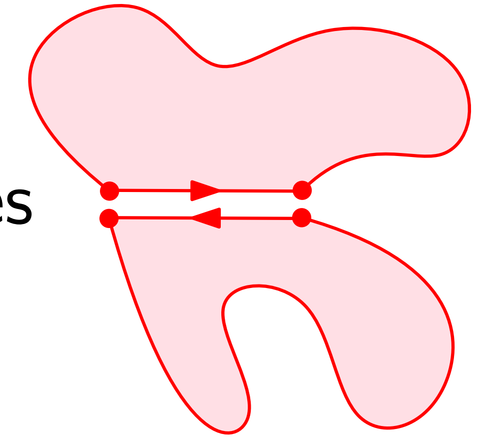
- basic building blocks

digons 

triangles 

- glue together at

vertices 

edges 

# Lower bound under ETH



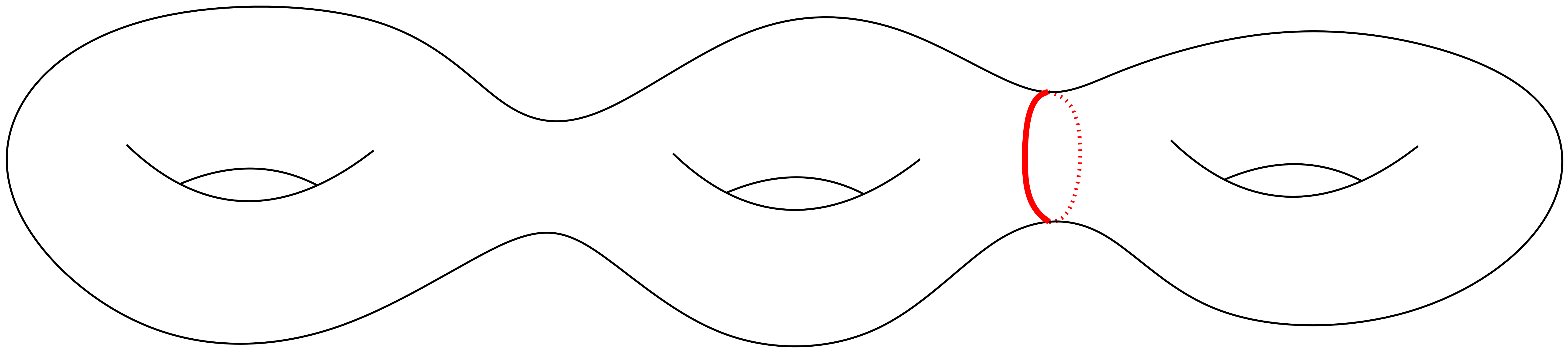
Exponential Time Hypothesis $\rightarrow$

GEOMETRIC/GRAPH-ENCLOSURE WITH PENALTIES cannot be solved in $2^{o(k)} \cdot n^{O(1)}$ time, even when all weights are $1$ and all penalties are $\infty$.

(Reduction from unweighted PLANAR STEINER TREE)

# What we originally set out to find

Find a shortest cycle that cuts off a *single handle* on a triangulated surface of high genus $g$.
(or a part of specified genus $g' < g$)



- FPT in $g$. [ Chambers, Colin de Verdière, Erickson, Lazarus, Whittlesey 2008 ]
- FPT in $g'$ ?