Computing the Detour of Polygons

Ansgar Grüne, Rolf Klein, Elmar Langetepe

Institut für Informatik I
Universität Bonn, Germany

Outline

- Definitions
- $O(n^2)$-Algorithm Computing the Euclidean Detour
- $O(n)$-Algorithm Computing the L_1-Detour of Monotone Rectilinear Polygons
- $\Omega(n \log n)$-Lower Time Bounds

Outline

- Definitions
- $O(n^2)$-Algorithm Computing the Euclidean Detour
- $O(n)$-Algorithm Computing the L_1-Detour of Monotone Rectilinear Polygons
- $\Omega(n \log n)$-Lower Time Bounds

Outline

- Definitions
- $O(n^2)$-Algorithm Computing the Euclidean Detour
- $O(n)$-Algorithm Computing the L_1-Detour of Monotone Rectilinear Polygons
- $\Omega(n \log n)$-Lower Time Bounds

Outline

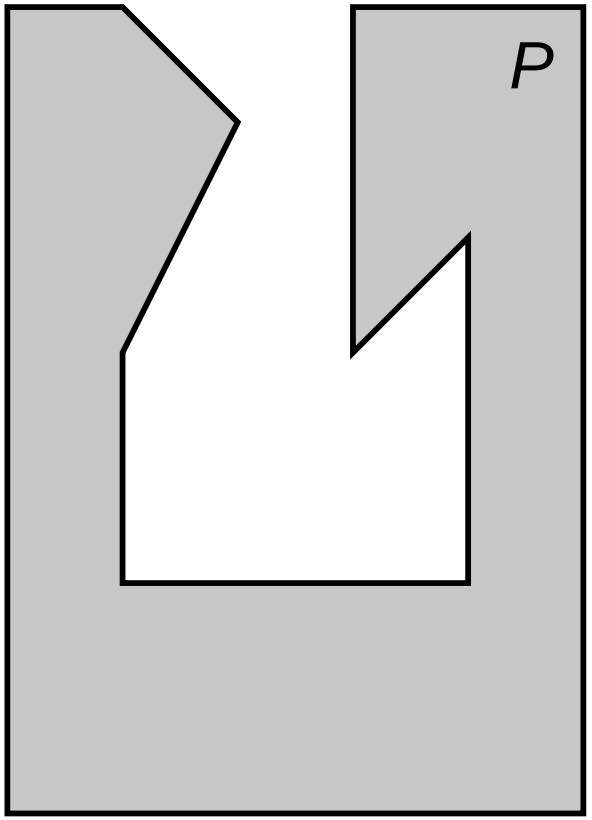
- Definitions
- $O(n^2)$-Algorithm Computing the Euclidean Detour
- $O(n)$-Algorithm Computing the L_1-Detour of Monotone Rectilinear Polygons
- $\Omega(n \log n)$-Lower Time Bounds

Outline

- Definitions
- $O(n^2)$-Algorithm Computing the Euclidean Detour
- $O(n)$-Algorithm Computing the L_1-Detour of Monotone Rectilinear Polygons
- $\Omega(n \log n)$-Lower Time Bounds

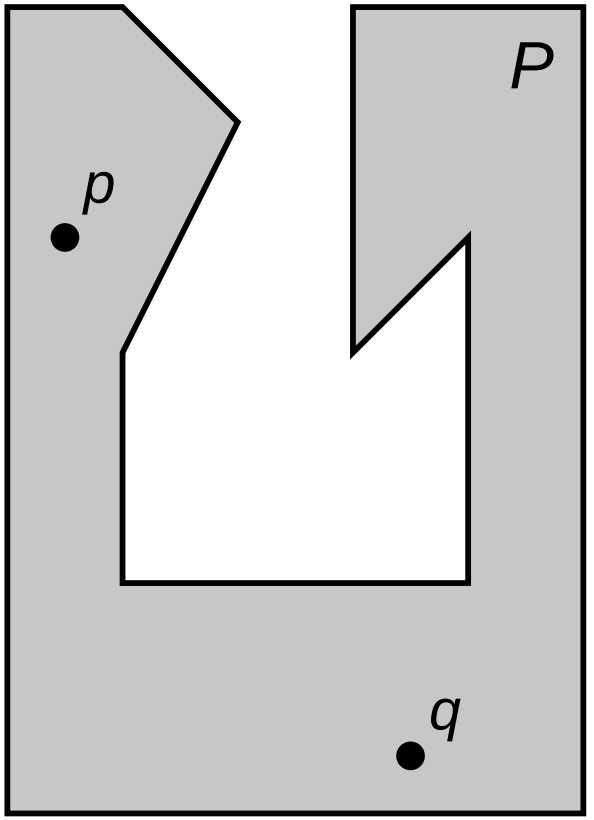
Definition of Detour

- $p, q \in P$
- $d_P(p, q) =$ length of shortest path in P w.r.t. $\|.\|$
- detour $\delta_P(p, q) := \frac{d_P(p,q)}{\|q-p\|}$
- detour $\delta(P) := \sup_{p,q \in P, p \neq q} \delta_P(p, q)$



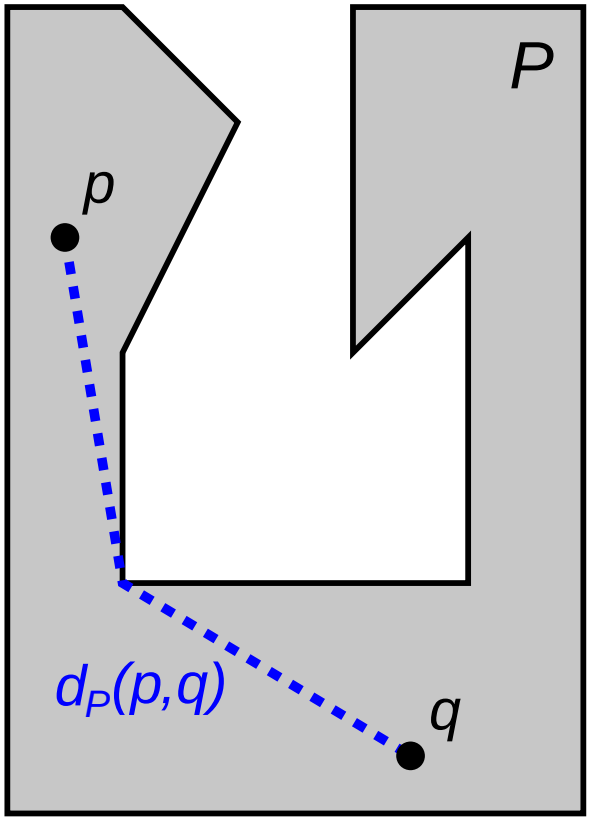
Definition of Detour

- $p, q \in P$
- $d_P(p, q) =$ length of shortest path in P w.r.t. $\|.\|$
- detour $\delta_P(p, q) := \frac{d_P(p,q)}{\|q-p\|}$
- detour $\delta(P) := \sup_{p,q \in P, p \neq q} \delta_P(p, q)$



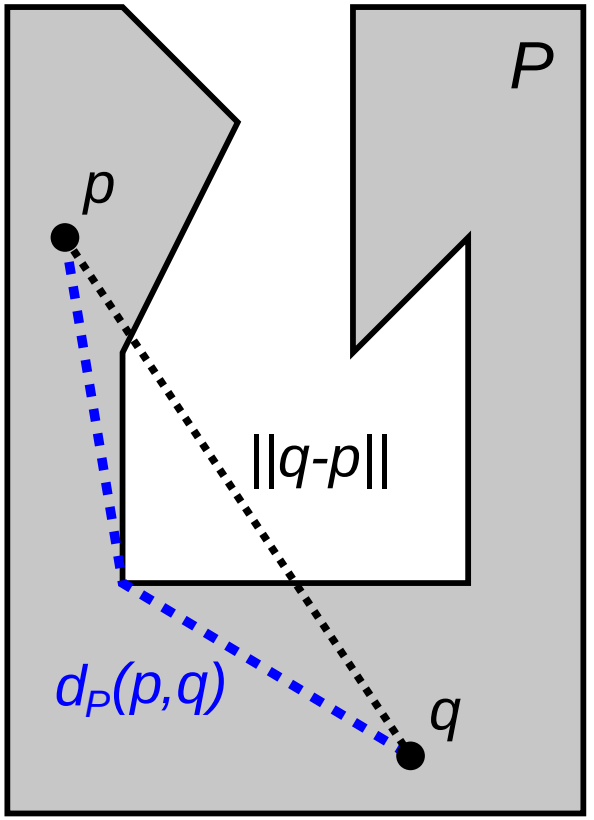
Definition of Detour

- $p, q \in P$
- $d_P(p, q) =$ length of shortest path in P w.r.t. $\|.\|$
- detour $\delta_P(p, q) := \frac{d_P(p,q)}{\|q-p\|}$
- detour $\delta(P) := \sup_{p,q \in P, p \neq q} \delta_P(p, q)$



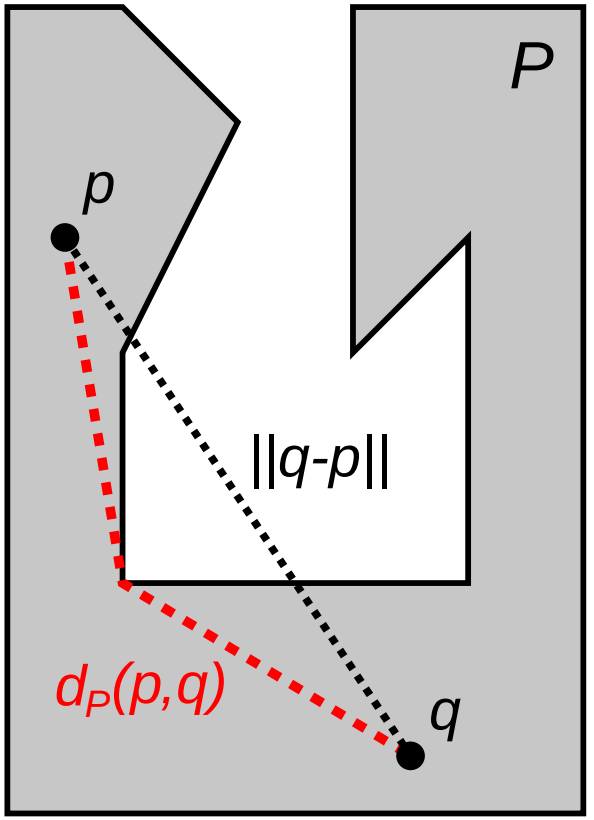
Definition of Detour

- $p, q \in P$
- $d_P(p, q) =$ length of shortest path in P w.r.t. $\|.\|$
- detour $\delta_P(p, q) := \frac{d_P(p,q)}{\|q-p\|}$
- detour $\delta(P) := \sup_{p,q \in P, p \neq q} \delta_P(p, q)$



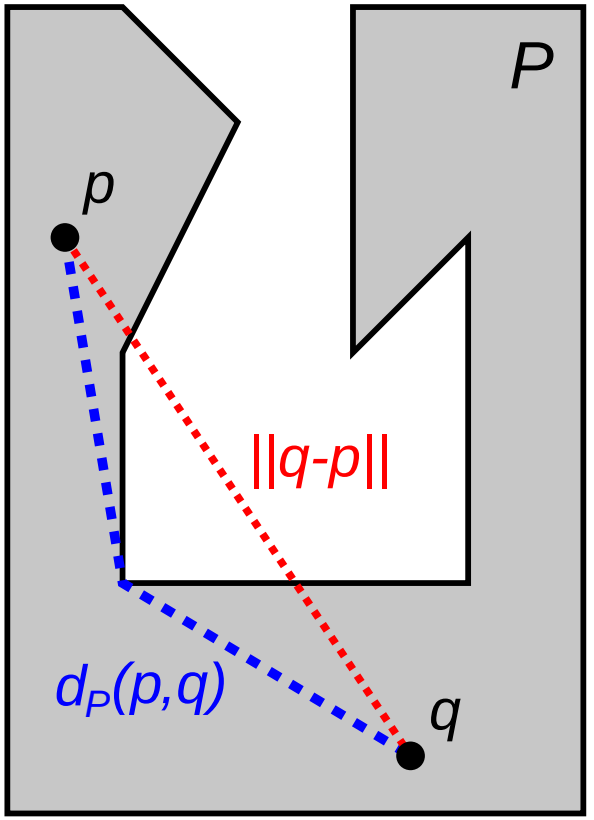
Definition of Detour

- $p, q \in P$
- $d_P(p, q) =$ length of shortest path in P w.r.t. $\|.\|$
- detour $\delta_P(p, q) := \frac{d_P(p,q)}{\|q-p\|}$
- detour $\delta(P) := \sup_{p,q \in P, p \neq q} \delta_P(p, q)$



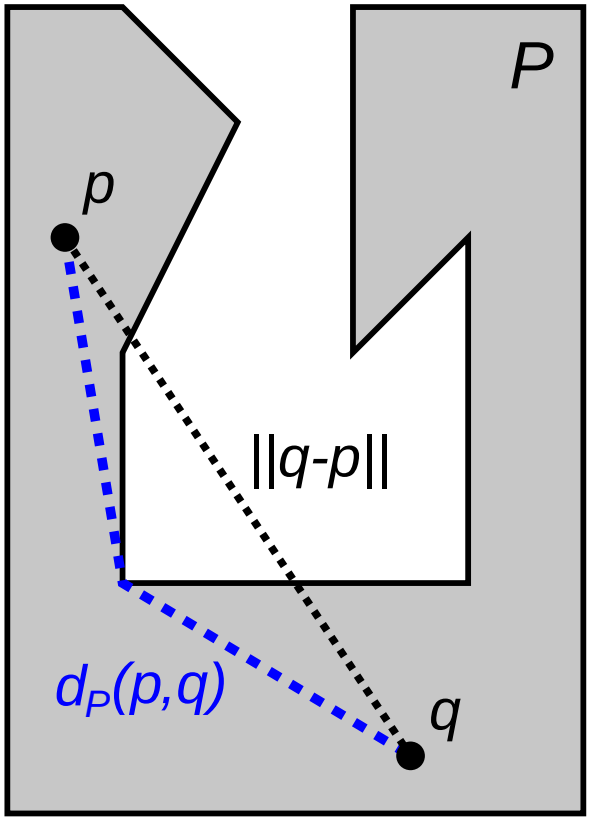
Definition of Detour

- $p, q \in P$
- $d_P(p, q) =$ length of shortest path in P w.r.t. $\|.\|$
- detour $\delta_P(p, q) := \frac{d_P(p,q)}{\|q-p\|}$
- detour $\delta(P) := \sup_{p,q \in P, p \neq q} \delta_P(p, q)$



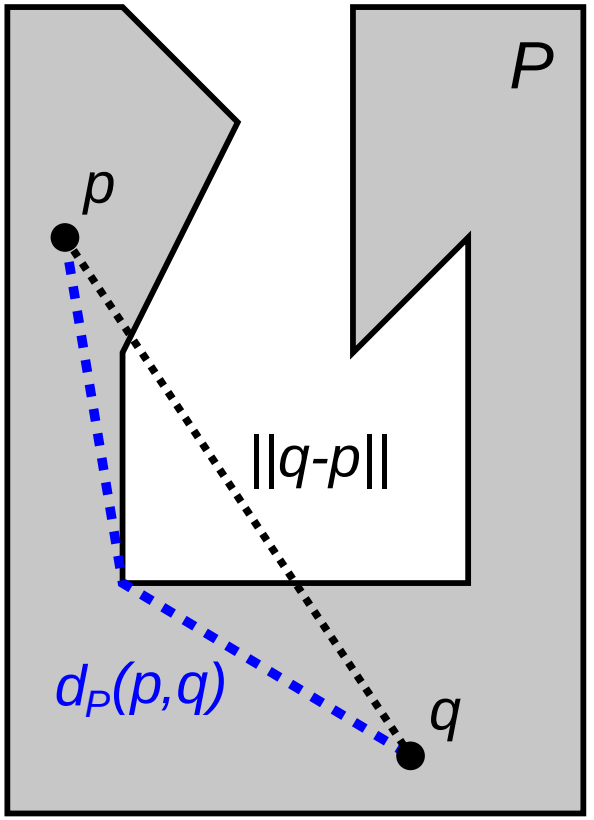
Definition of Detour

- $p, q \in P$
- $d_P(p, q) =$ length of shortest path in P w.r.t. $\|.\|$
- detour $\delta_P(p, q) := \frac{d_P(p,q)}{\|q-p\|}$
- detour $\delta(P) := \sup_{p,q \in P, p \neq q} \delta_P(p, q)$



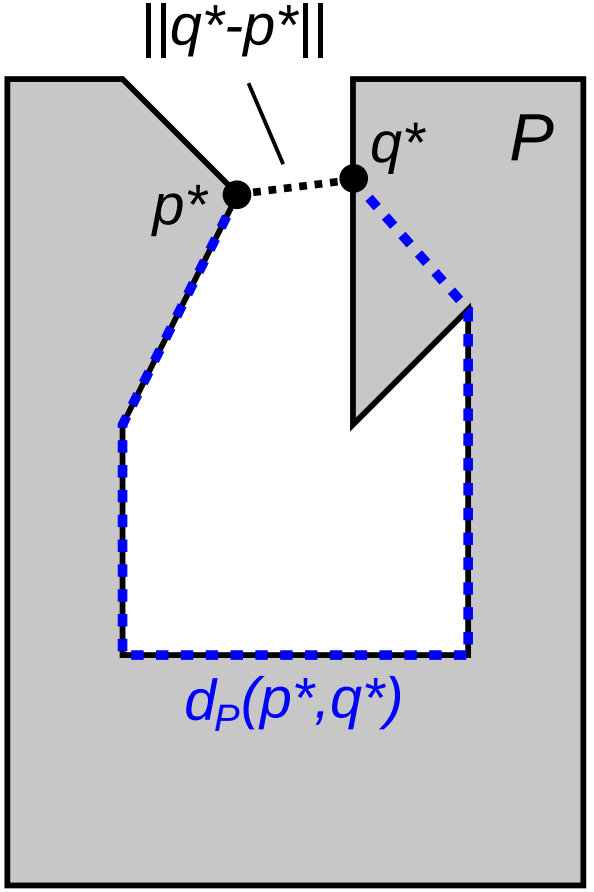
Definition of Detour

- $p, q \in P$
- $d_P(p, q) =$ length of shortest path in P w.r.t. $\|.\|$
- detour $\delta_P(p, q) := \frac{d_P(p,q)}{\|q-p\|}$
- detour $\delta(P) := \sup_{p,q \in P, p \neq q} \delta_P(p, q)$



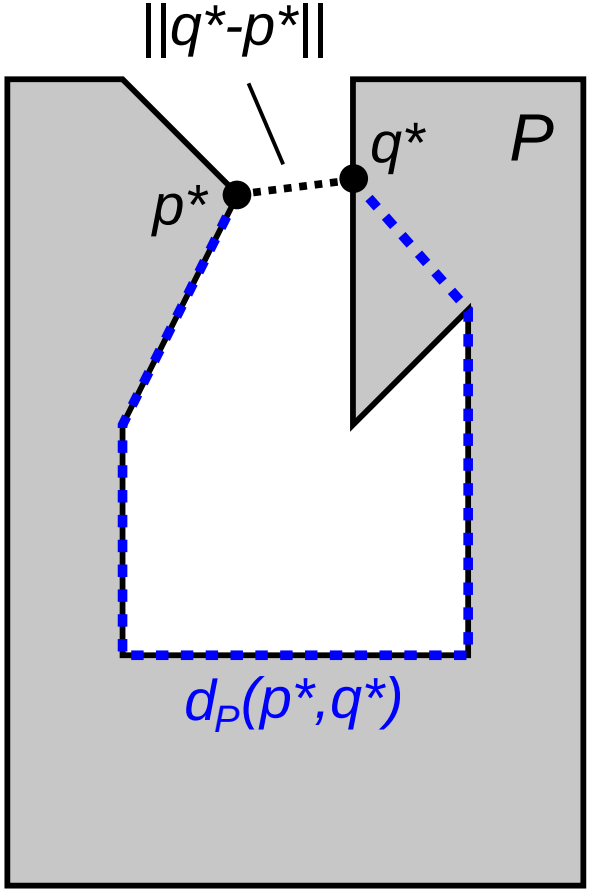
Definition of Detour

- $p, q \in P$
- $d_P(p, q) =$ length of shortest path in P w.r.t. $\|.\|$
- detour $\delta_P(p, q) := \frac{d_P(p,q)}{\|q-p\|}$
- detour $\delta(P) := \sup_{p,q \in P, p \neq q} \delta_P(p, q)$



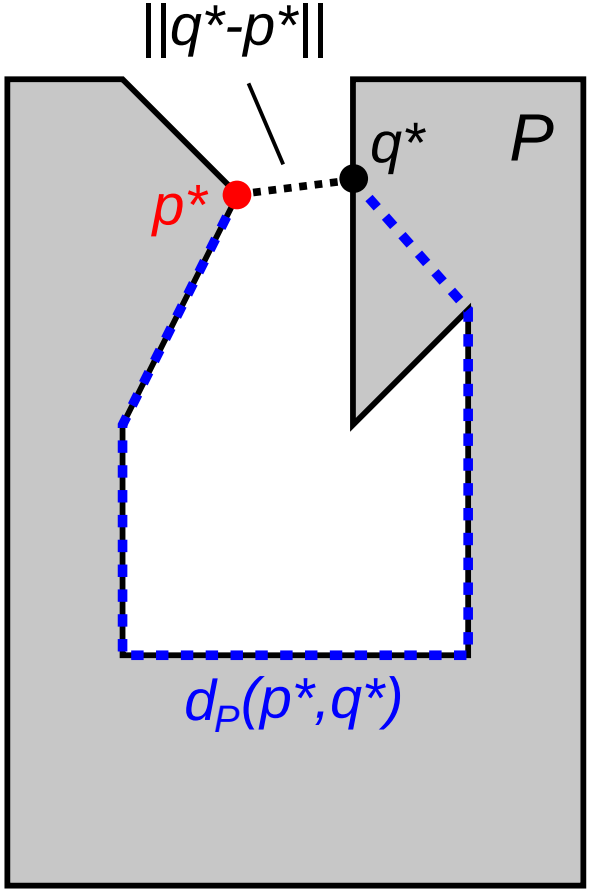
Properties of Euclidean Detour Maxima

- $\exists$ detour maximum $(p^*, q^*) \in P \times P$ such that:
  - p^* vertex
  - $q^* \in \partial P$ boundary point
  - (p^*, q^*) co-visible in P^{c}
- $\Rightarrow O(n^2)$-algorithm computing $\delta(P)$ (see abstract)



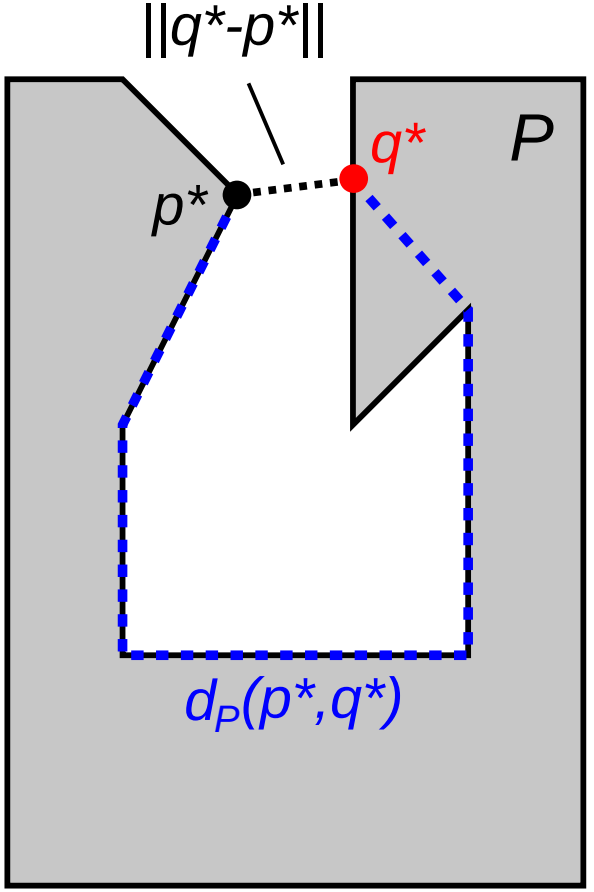
Properties of Euclidean Detour Maxima

- $\exists$ detour maximum $(p^*, q^*) \in P \times P$ such that:
  - p^* vertex
  - $q^* \in \partial P$ boundary point
  - (p^*, q^*) co-visible in P^{c}
- $\Rightarrow O(n^2)$-algorithm computing $\delta(P)$ (see abstract)



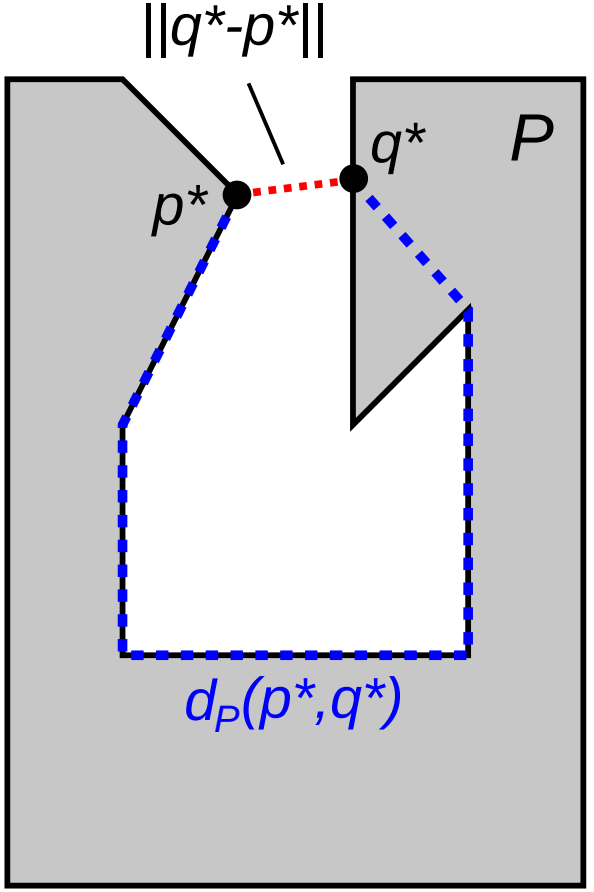
Properties of Euclidean Detour Maxima

- $\exists$ detour maximum $(p^*, q^*) \in P \times P$ such that:
  - p^* vertex
  - $q^* \in \partial P$ boundary point
  - (p^*, q^*) co-visible in P^{C}
- $\Rightarrow O(n^2)$-algorithm computing $\delta(P)$ (see abstract)



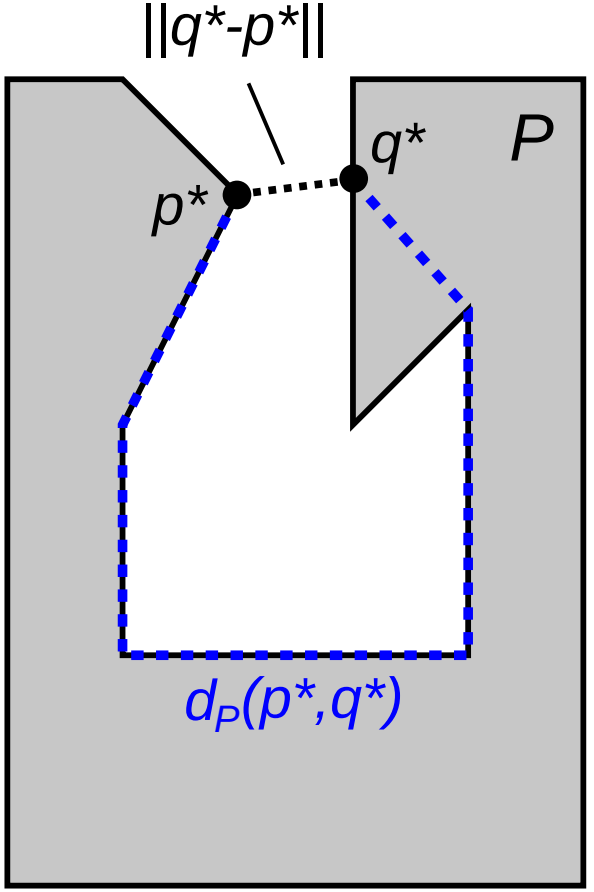
Properties of Euclidean Detour Maxima

- $\exists$ detour maximum $(p^*, q^*) \in P \times P$ such that:
  - p^* vertex
  - $q^* \in \partial P$ boundary point
  - (p^*, q^*) co-visible in P^{c}
- $\Rightarrow O(n^2)$-algorithm computing $\delta(P)$ (see abstract)



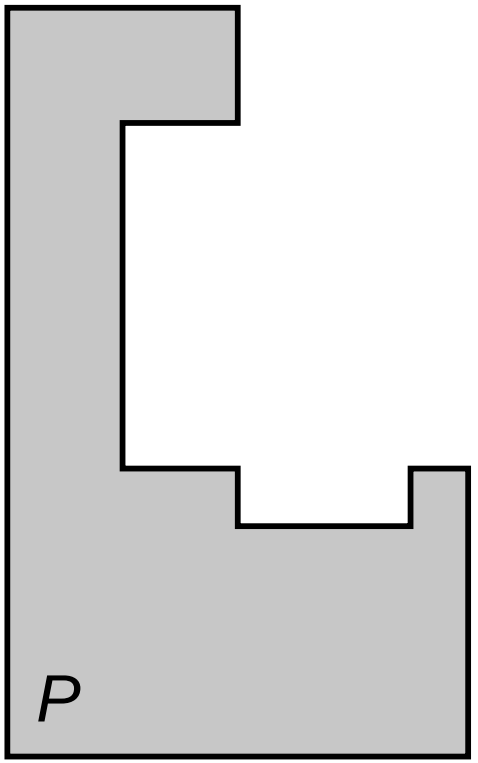
Properties of Euclidean Detour Maxima

- $\exists$ detour maximum $(p^*, q^*) \in P \times P$ such that:
  - p^* vertex
  - $q^* \in \partial P$ boundary point
  - (p^*, q^*) co-visible in P^{c}
- $\Rightarrow O(n^2)$-algorithm computing $\delta(P)$ (see abstract)



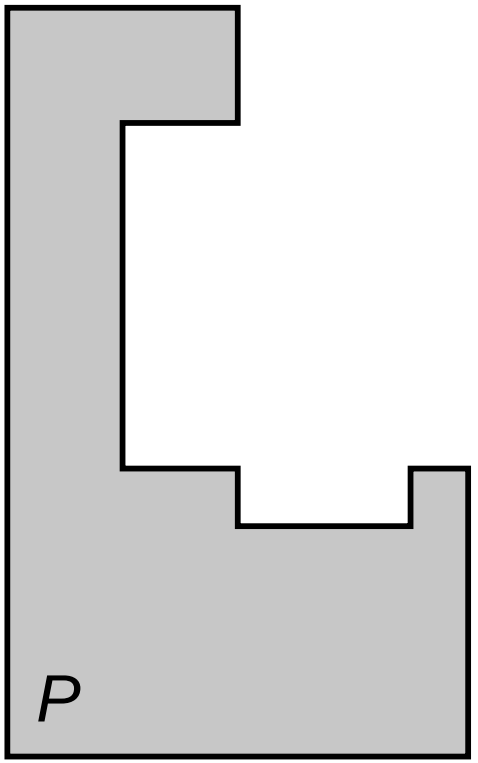
Property of L_1-Detour Maxima

- P rectilinear polygon, $\delta(P) > 1$
- (p^*, q^*) L_1-detour maximum
- bounding rectangle $R(p^*, q^*)$
- $\Rightarrow R(p^*, q^*)$ empty apart from corners



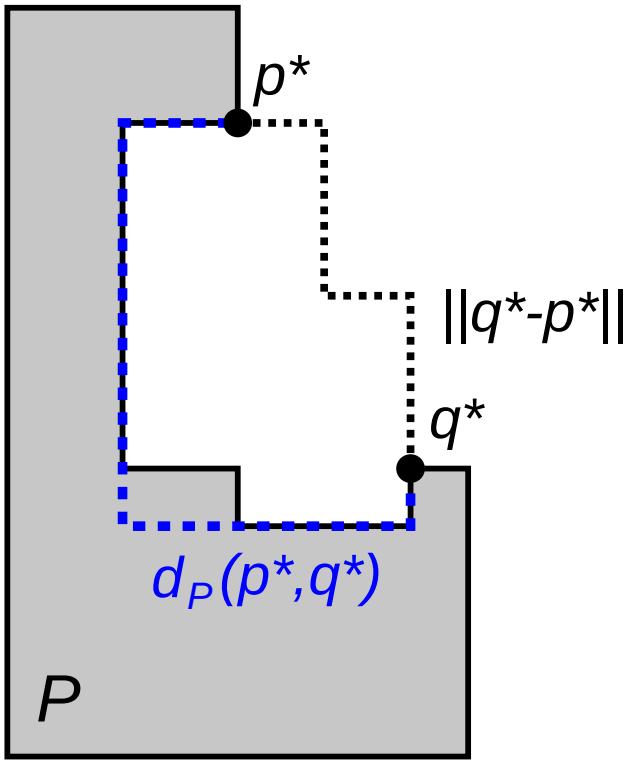
Property of L_1-Detour Maxima

- P rectilinear polygon, $\delta(P) > 1$
- (p^*, q^*) L_1-detour maximum
- bounding rectangle $R(p^*, q^*)$
- $\Rightarrow R(p^*, q^*)$ empty apart from corners



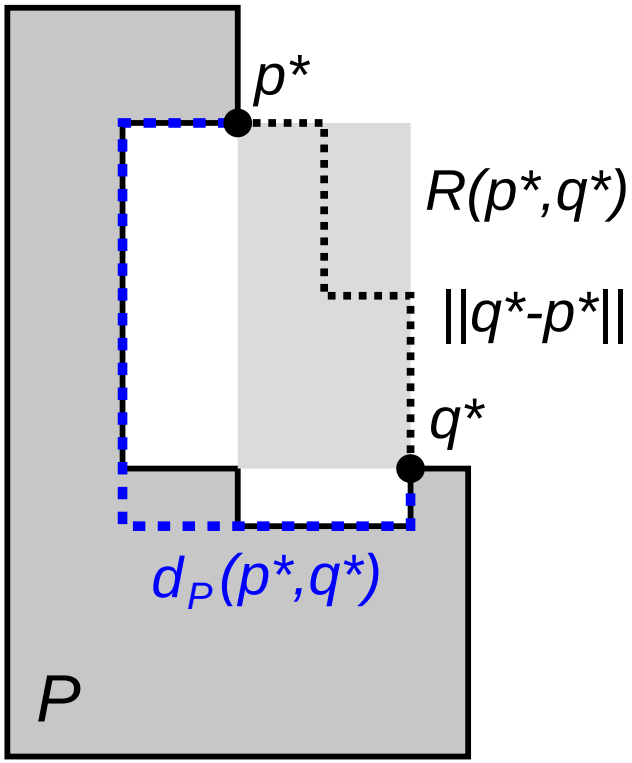
Property of L_1-Detour Maxima

- P rectilinear polygon, $\delta(P) > 1$
- (p^*, q^*) L_1-detour maximum
- bounding rectangle $R(p^*, q^*)$
- $\Rightarrow R(p^*, q^*)$ empty apart from corners



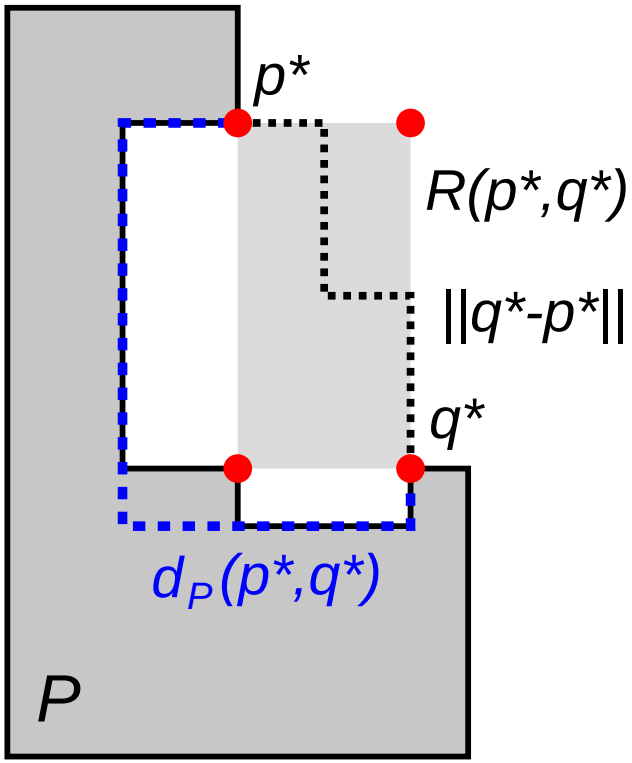
Property of L_1-Detour Maxima

- P rectilinear polygon, $\delta(P) > 1$
- (p^*, q^*) L_1-detour maximum
- bounding rectangle $R(p^*, q^*)$
- $\Rightarrow R(p^*, q^*)$ empty apart from corners



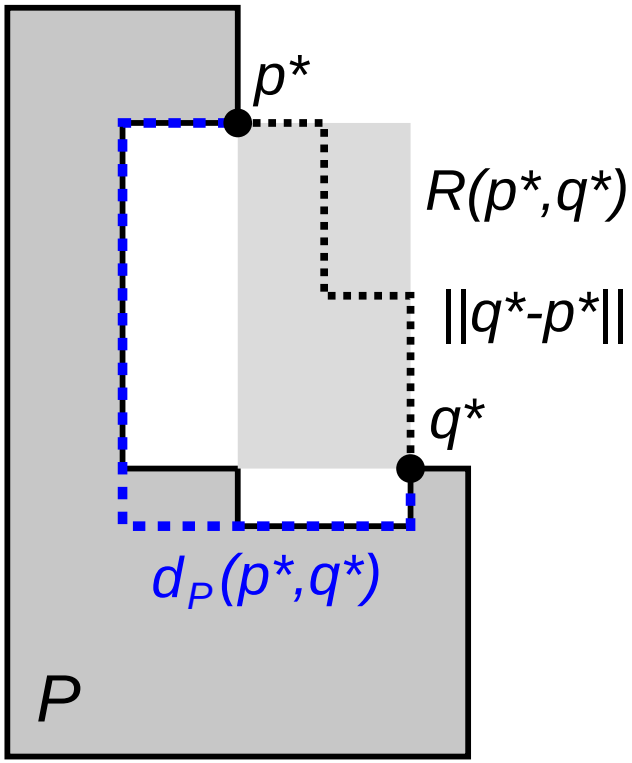
Property of L_1-Detour Maxima

- P rectilinear polygon, $\delta(P) > 1$
- (p^*, q^*) L_1-detour maximum
- bounding rectangle $R(p^*, q^*)$
- $\Rightarrow$ $R(p^*, q^*)$ empty apart from corners



Property of L_1-Detour Maxima

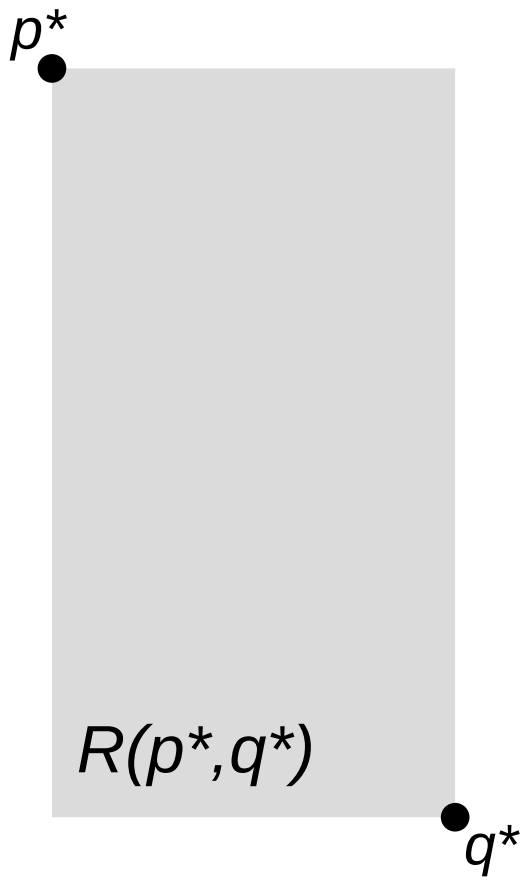
- P rectilinear polygon, $\delta(P) > 1$
- (p^*, q^*) L_1-detour maximum
- bounding rectangle $R(p^*, q^*)$
- $\Rightarrow R(p^*, q^*)$ empty apart from corners



Proof of Emptiness of $R(p^*, q^*)$

Else: axis-parallel $\overline{p'q'} \subset R(p^*, q^*) \cap P$.

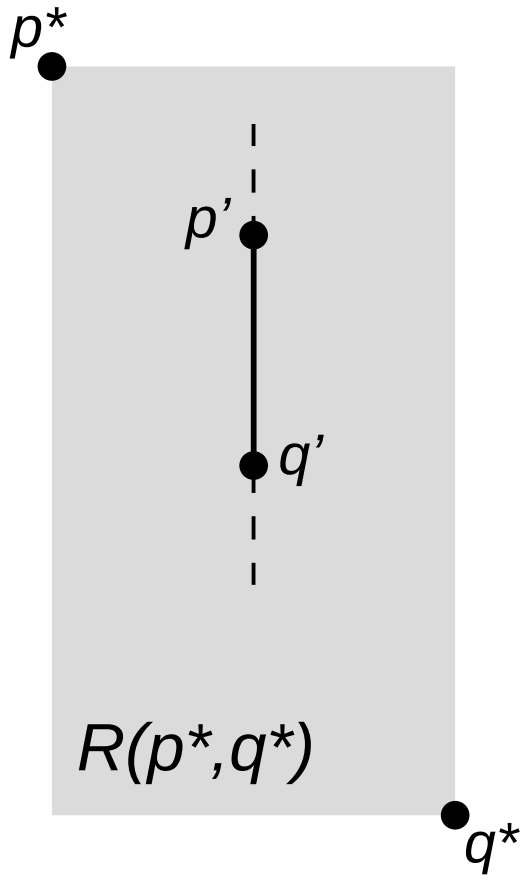
$$\begin{aligned}
\delta_P(p^*, q^*) &= \frac{d_P(p^*, q^*)}{\|q^* - p^*\|} \\
&\leq \frac{d_P(p^*, p') + d_P(p', q') + d_P(q', q^*)}{\|p' - p^*\| + \|q' - p'\| + \|q^* - q'\|} \\
&< \frac{d_P(p^*, p') + d_P(q', q^*)}{\|p' - p^*\| + \|q^* - q'\|} \\
&\leq \max\left(\delta_P(p^*, p'), \delta_P(q^*, q')\right)
\end{aligned}$$



Proof of Emptiness of $R(p^*, q^*)$

Else: axis-parallel $\overline{p'q'} \subset R(p^*, q^*) \cap P$.

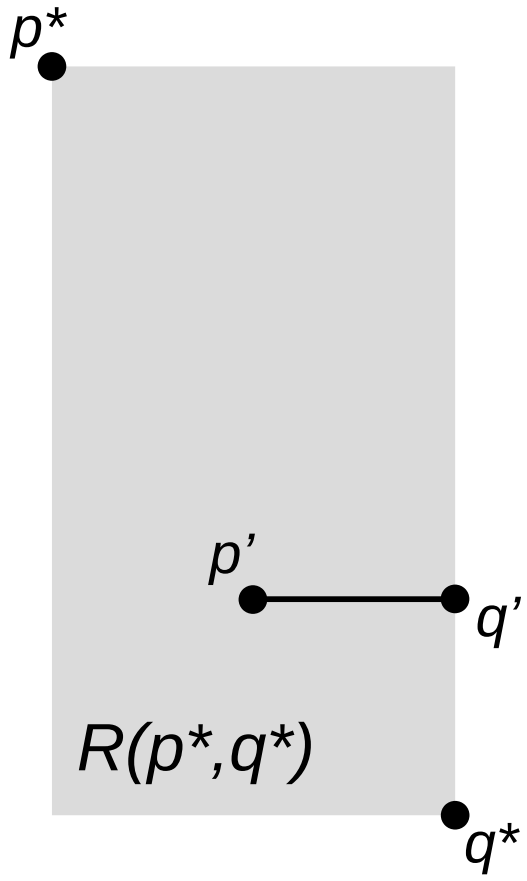
$$\begin{aligned}
\delta_P(p^*, q^*) &= \frac{d_P(p^*, q^*)}{\|q^* - p^*\|} \\
&\leq \frac{d_P(p^*, p') + d_P(p', q') + d_P(q', q^*)}{\|p' - p^*\| + \|q' - p'\| + \|q^* - q'\|} \\
&< \frac{d_P(p^*, p') + d_P(q', q^*)}{\|p' - p^*\| + \|q^* - q'\|} \\
&\leq \max\left(\delta_P(p^*, p'), \delta_P(q^*, q')\right)
\end{aligned}$$



Proof of Emptiness of $R(p^*, q^*)$

Else: axis-parallel $\overline{p'q'} \subset R(p^*, q^*) \cap P$.

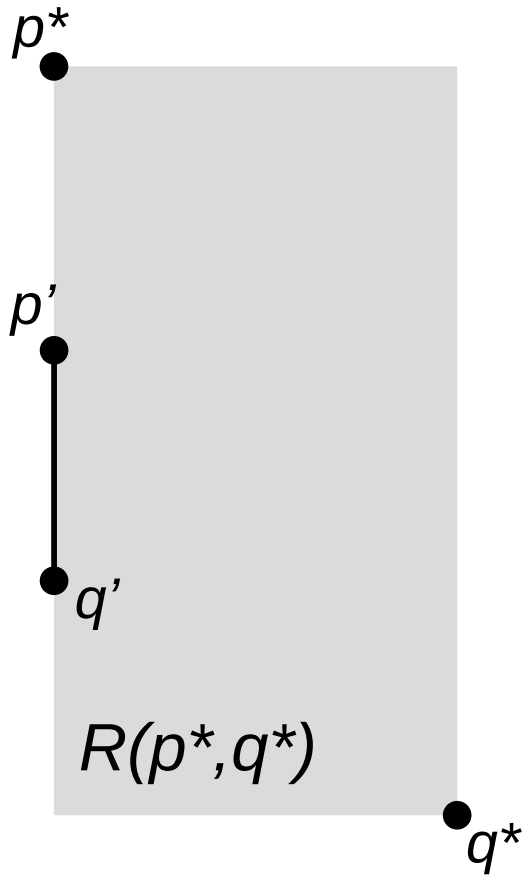
$$\begin{aligned}
\delta_P(p^*, q^*) &= \frac{d_P(p^*, q^*)}{\|q^* - p^*\|} \\
&\leq \frac{d_P(p^*, p') + d_P(p', q') + d_P(q', q^*)}{\|p' - p^*\| + \|q' - p'\| + \|q^* - q'\|} \\
&< \frac{d_P(p^*, p') + d_P(q', q^*)}{\|p' - p^*\| + \|q^* - q'\|} \\
&\leq \max\left(\delta_P(p^*, p'), \delta_P(q^*, q')\right)
\end{aligned}$$



Proof of Emptiness of $R(p^*, q^*)$

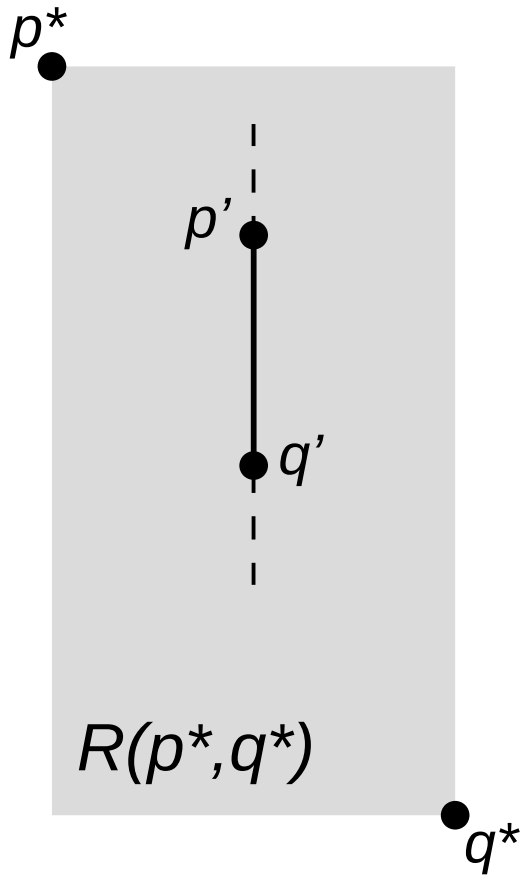
Else: axis-parallel $\overline{p'q'} \subset R(p^*, q^*) \cap P$.

$$\begin{aligned}
\delta_P(p^*, q^*) &= \frac{d_P(p^*, q^*)}{\|q^* - p^*\|} \\
&\leq \frac{d_P(p^*, p') + d_P(p', q') + d_P(q', q^*)}{\|p' - p^*\| + \|q' - p'\| + \|q^* - q'\|} \\
&< \frac{d_P(p^*, p') + d_P(q', q^*)}{\|p' - p^*\| + \|q^* - q'\|} \\
&\leq \max\left(\delta_P(p^*, p'), \delta_P(q^*, q')\right)
\end{aligned}$$



Proof of Emptiness of $R(p^*, q^*)$

$$\begin{aligned}
\delta_P(p^*, q^*) &= \frac{d_P(p^*, q^*)}{\|q^* - p^*\|} \\
&\leq \frac{d_P(p^*, p') + d_P(p', q') + d_P(q', q^*)}{\|p' - p^*\| + \|q' - p'\| + \|q^* - q'\|} \\
&< \frac{d_P(p^*, p') + d_P(q', q^*)}{\|p' - p^*\| + \|q^* - q'\|} \\
&\leq \max\left(\delta_P(p^*, p'), \delta_P(q^*, q')\right)
\end{aligned}$$



Proof of Emptiness of $R(p^*, q^*)$

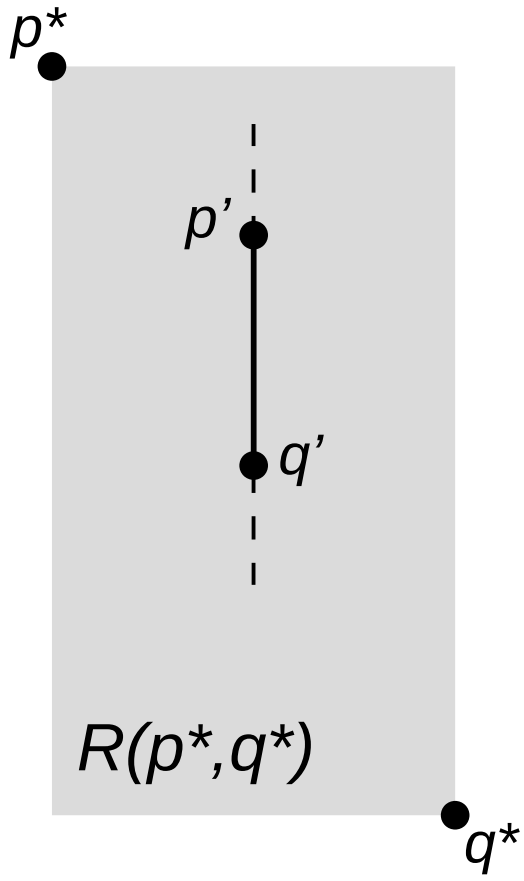
definition of $\delta_P(.,.)$

$$\delta_P(p^*, q^*) = \frac{d_P(p^*, q^*)}{\|q^* - p^*\|}$$

$$\leq \frac{d_P(p^*, p') + d_P(p', q') + d_P(q', q^*)}{\|p' - p^*\| + \|q' - p'\| + \|q^* - q'\|}$$

$$< \frac{d_P(p^*, p') + d_P(q', q^*)}{\|p' - p^*\| + \|q^* - q'\|}$$

$$\leq \max\left(\delta_P(p^*, p'), \delta_P(q^*, q')\right)$$



Proof of Emptiness of $R(p^*, q^*)$

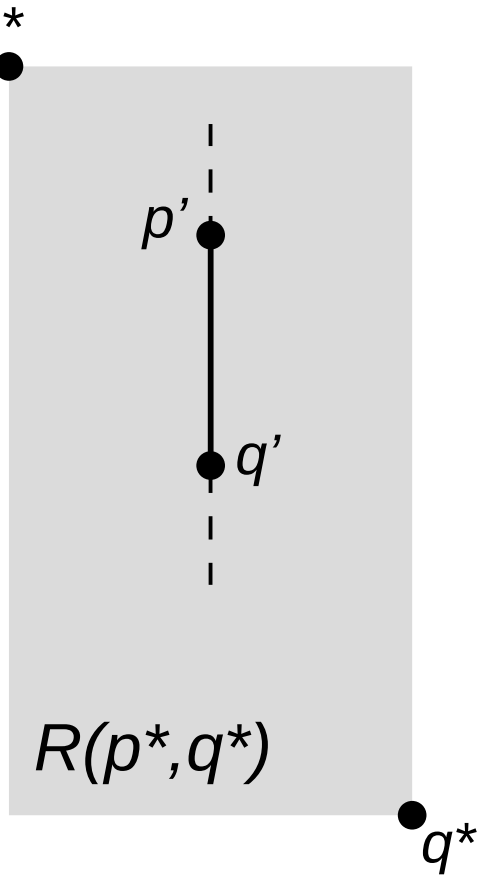
$\triangle$-inequality $d_P(.,.)$ & geometry $R(p^*, q^*)$

$$\delta_P(p^*, q^*) = \frac{d_P(p^*, q^*)}{\|q^* - p^*\|}$$

$$\leq \frac{d_P(p^*, p') + d_P(p', q') + d_P(q', q^*)}{\|p' - p^*\| + \|q' - p'\| + \|q^* - q'\|}$$

$$< \frac{d_P(p^*, p') + d_P(q', q^*)}{\|p' - p^*\| + \|q^* - q'\|}$$

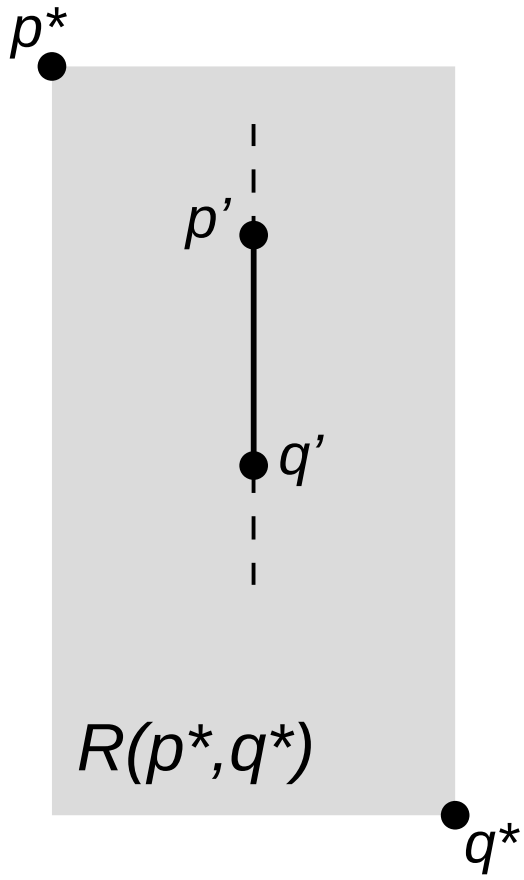
$$\leq \max\left(\delta_P(p^*, p'), \delta_P(q^*, q')\right)$$



Proof of Emptiness of $R(p^*, q^*)$

$$d_P(p', q') = \|q' - p'\| \quad \& \quad \delta_P(p^*, q^*) > 1$$

$$\begin{aligned}
\delta_P(p^*, q^*) &= \frac{d_P(p^*, q^*)}{\|q^* - p^*\|} \\
&\leq \frac{d_P(p^*, p') + d_P(p', q') + d_P(q', q^*)}{\|p' - p^*\| + \|q' - p'\| + \|q^* - q'\|} \\
&< \frac{d_P(p^*, p') + d_P(q', q^*)}{\|p' - p^*\| + \|q^* - q'\|} \\
&\leq \max\left(\delta_P(p^*, p'), \delta_P(q^*, q')\right)
\end{aligned}$$



Proof of Emptiness of $R(p^*, q^*)$

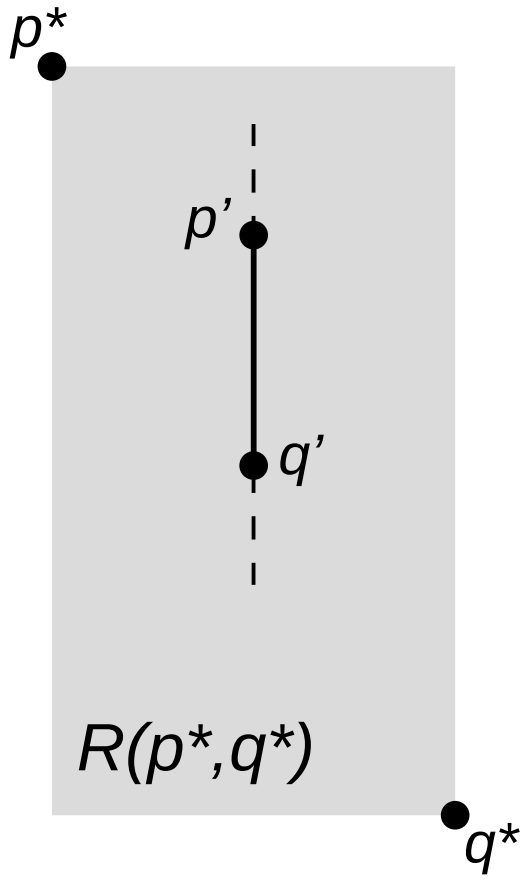
compare to Ebbers-Baumann et al. [3]

$$\delta_P(p^*, q^*) = \frac{d_P(p^*, q^*)}{\|q^* - p^*\|}$$

$$\leq \frac{d_P(p^*, p') + d_P(p', q') + d_P(q', q^*)}{\|p' - p^*\| + \|q' - p'\| + \|q^* - q'\|}$$

$$< \frac{d_P(p^*, p') + d_P(q', q^*)}{\|p' - p^*\| + \|q^* - q'\|}$$

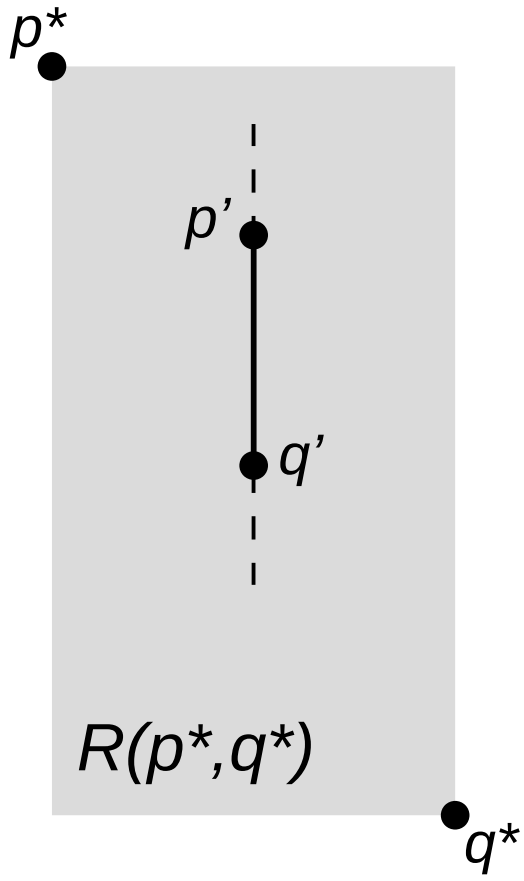
$$\leq \max\left(\delta_P(p^*, p'), \delta_P(q^*, q')\right)$$



Proof of Emptiness of $R(p^*, q^*)$

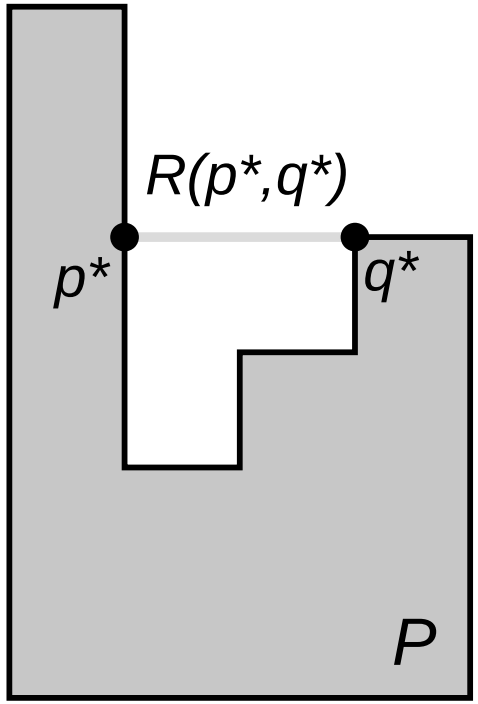
contradiction to $\delta_P(p^*, q^*)$ maximum!

$$\begin{aligned}
\delta_P(p^*, q^*) &= \frac{d_P(p^*, q^*)}{\|q^* - p^*\|} \\
&\leq \frac{d_P(p^*, p') + d_P(p', q') + d_P(q', q^*)}{\|p' - p^*\| + \|q' - p'\| + \|q^* - q'\|} \\
&< \frac{d_P(p^*, p') + d_P(q', q^*)}{\|p' - p^*\| + \|q^* - q'\|} \\
&\leq \max\left(\delta_P(p^*, p'), \delta_P(q^*, q')\right)
\end{aligned}$$



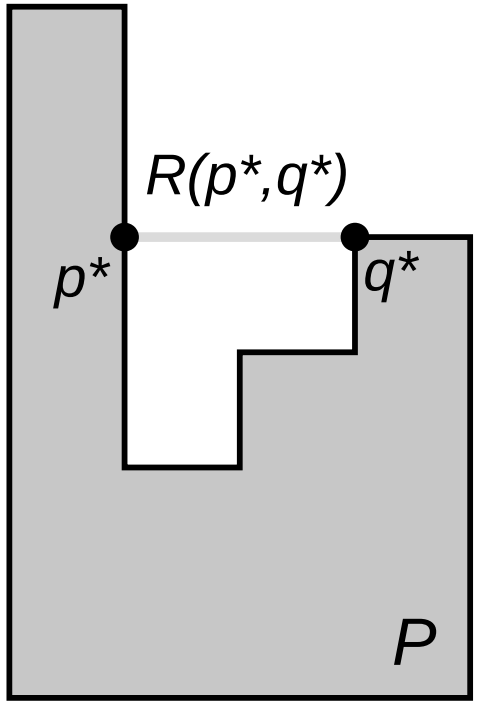
L_1-Maxima of x-Monotone Rectilinear Polygons

- co-visible in P^{C}
- $\overline{p^*q^*}$ horizontal (otherwise $R(p^*, q^*)$ intersects with P)
- p^* or q^* vertex (proof omitted)



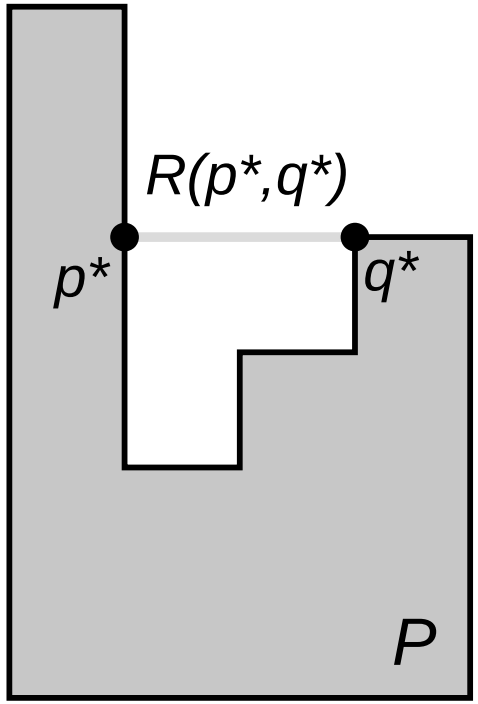
L_1-Maxima of x-Monotone Rectilinear Polygons

- co-visible in P^c
- $\overline{p^*q^*}$ horizontal (otherwise $R(p^*, q^*)$ intersects with P)
- p^* or q^* vertex (proof omitted)



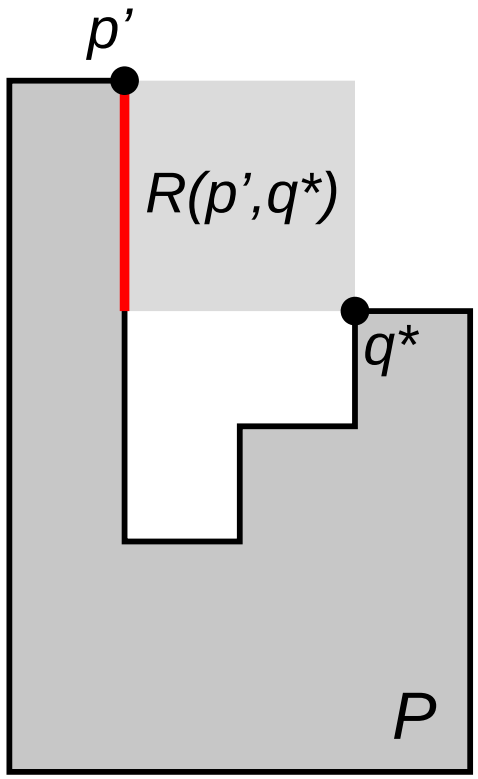
L_1-Maxima of x-Monotone Rectilinear Polygons

- co-visible in P^{c}
- $\overline{p^*q^*}$ horizontal (otherwise $R(p^*, q^*)$ intersects with P)
- p^* or q^* vertex (proof omitted)



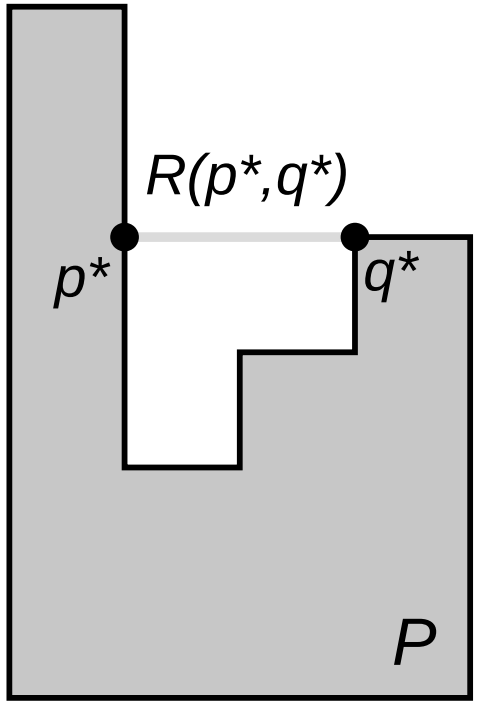
L_1-Maxima of x-Monotone Rectilinear Polygons

- co-visible in P^{c}
- $\overline{p^*q^*}$ horizontal (otherwise $R(p^*, q^*)$ intersects with P)
- p^* or q^* vertex (proof omitted)



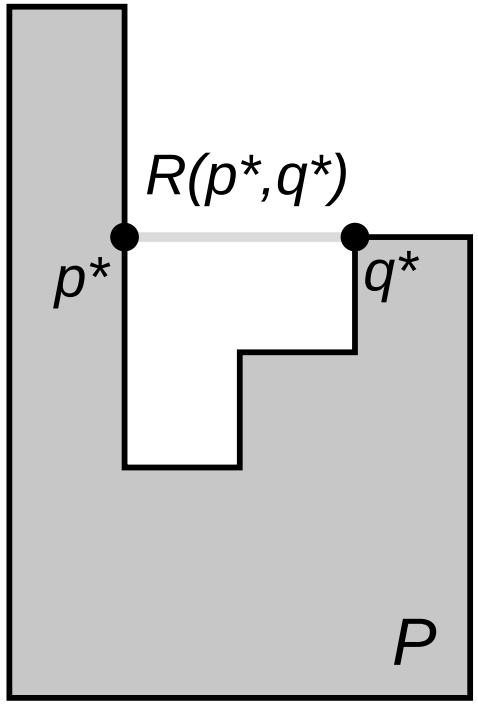
L_1-Maxima of x-Monotone Rectilinear Polygons

- co-visible in P^{c}
- $\overline{p^*q^*}$ horizontal (otherwise $R(p^*, q^*)$ intersects with P)
- p^* or q^* vertex (proof omitted)



L_1-Maxima of x-Monotone Rectilinear Polygons

- co-visible in P^{c}
- $\overline{p^*q^*}$ horizontal (otherwise $R(p^*, q^*)$ intersects with P)
- p^* or q^* vertex (proof omitted)



Algorithm for x-Monotone Rectilinear Polygons

- examine only horizontal vertex-boundary-cuts $(p, q) \in V \times \partial P$, co-visible in P^{c} ($=$ maximum candidates)
- scan upper and lower boundary separately
- store vertical segments on stack until opposite segment found

Algorithm for x-Monotone Rectilinear Polygons

- examine only horizontal vertex-boundary-cuts $(p, q) \in V \times \partial P$, co-visible in P^{c} ($=$ maximum candidates)
- scan upper and lower boundary separately
- store vertical segments on stack until opposite segment found

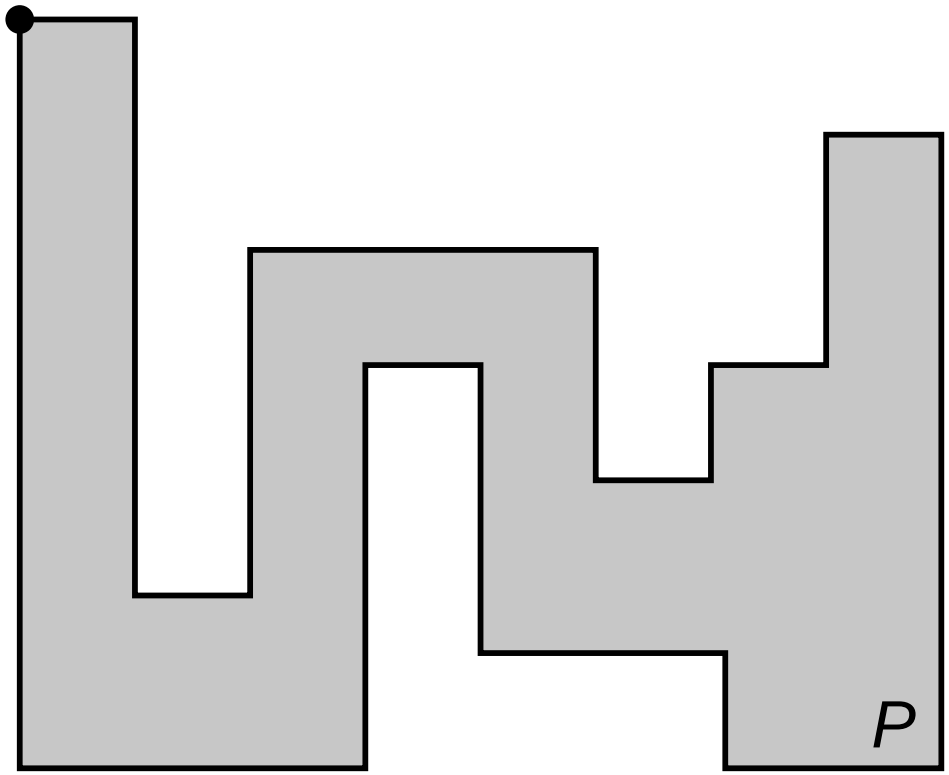
Algorithm for x-Monotone Rectilinear Polygons

- examine only horizontal vertex-boundary-cuts $(p, q) \in V \times \partial P$, co-visible in P^{c} ($=$ maximum candidates)
- scan upper and lower boundary separately
- store vertical segments on stack until opposite segment found

Algorithm for x-Monotone Rectilinear Polygons

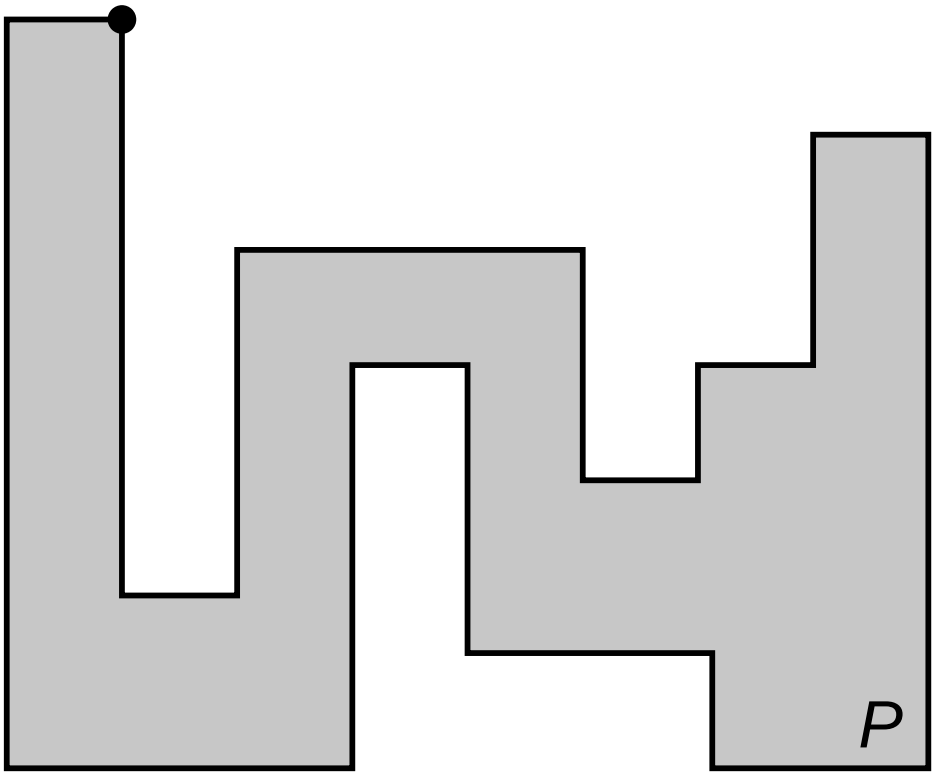
- examine only horizontal vertex-boundary-cuts $(p, q) \in V \times \partial P$, co-visible in P^{c} ($=$ maximum candidates)
- scan upper and lower boundary separately
- store vertical segments on stack until opposite segment found

Scan of Upper Boundary



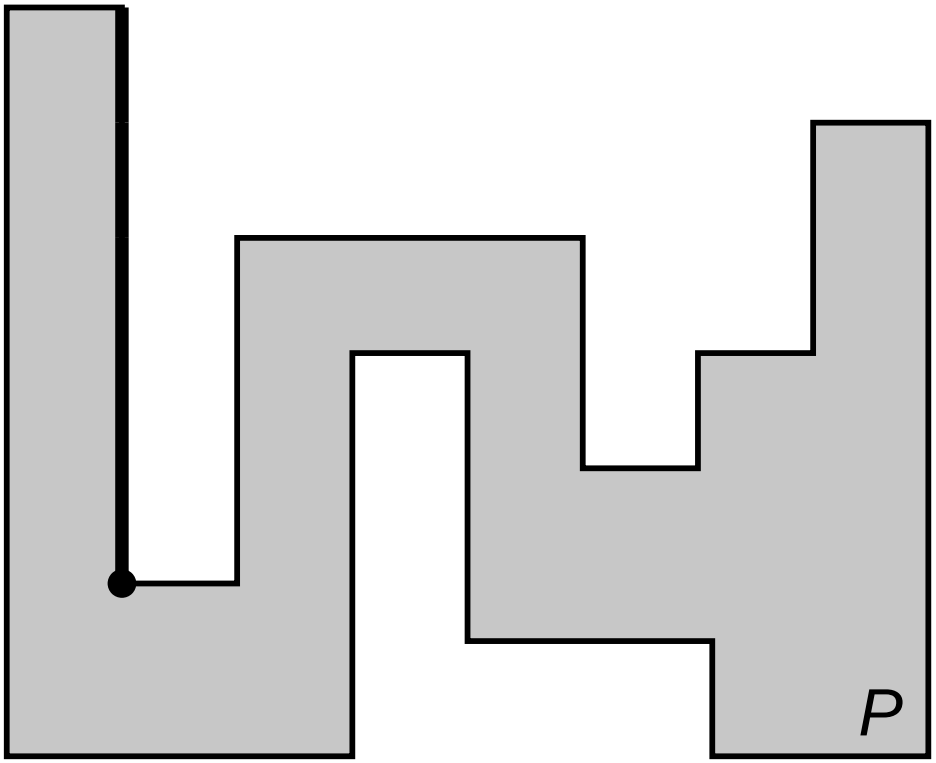
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



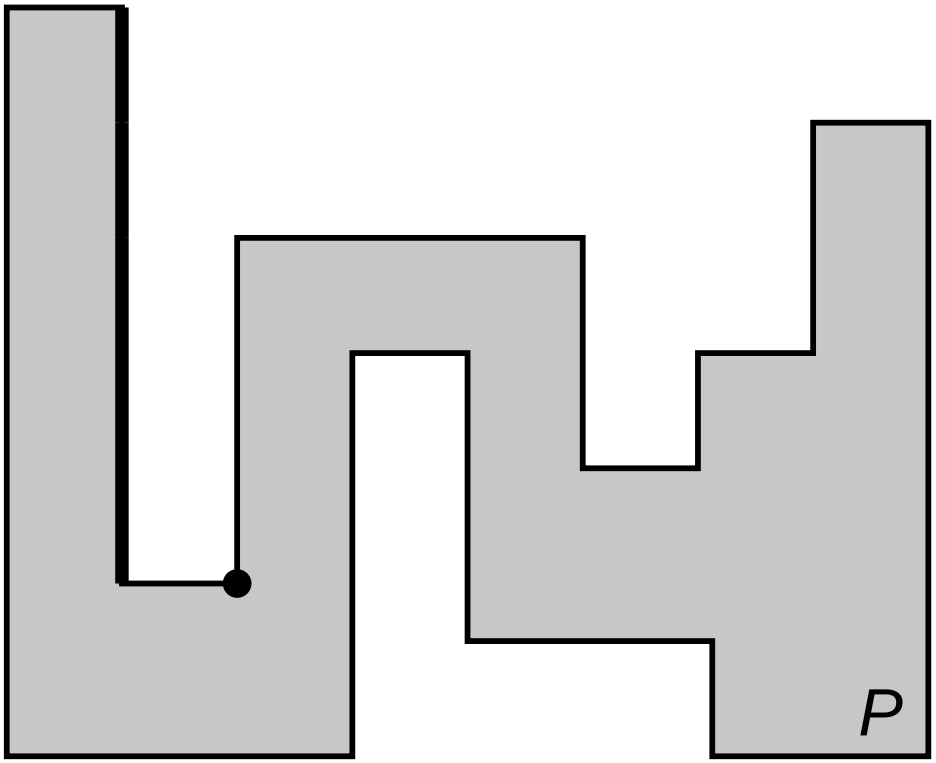
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



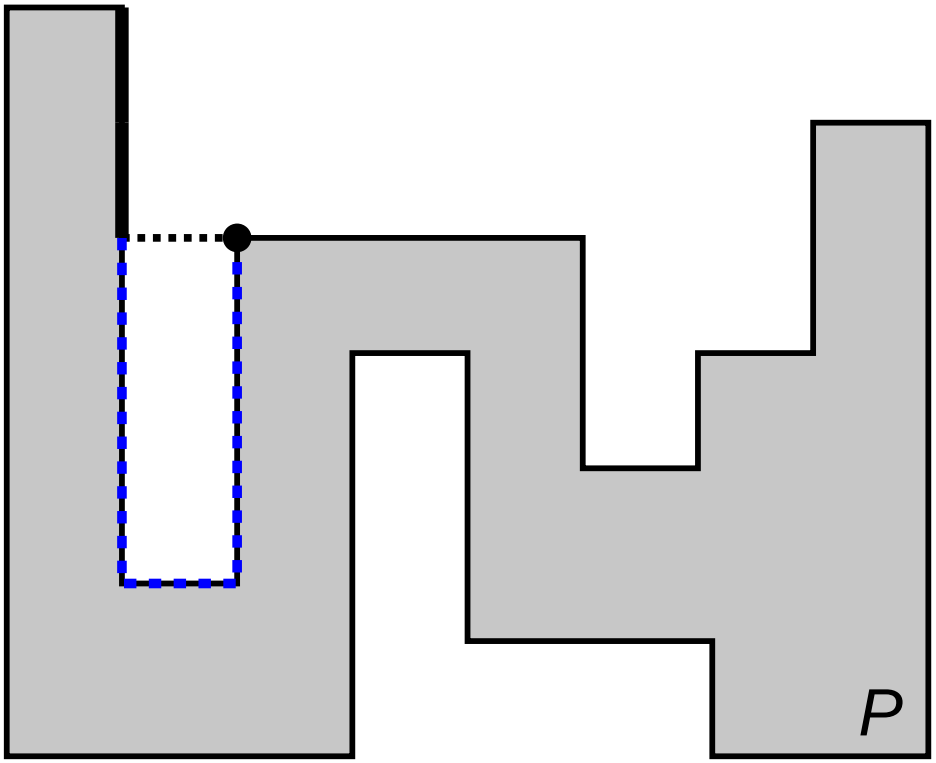
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



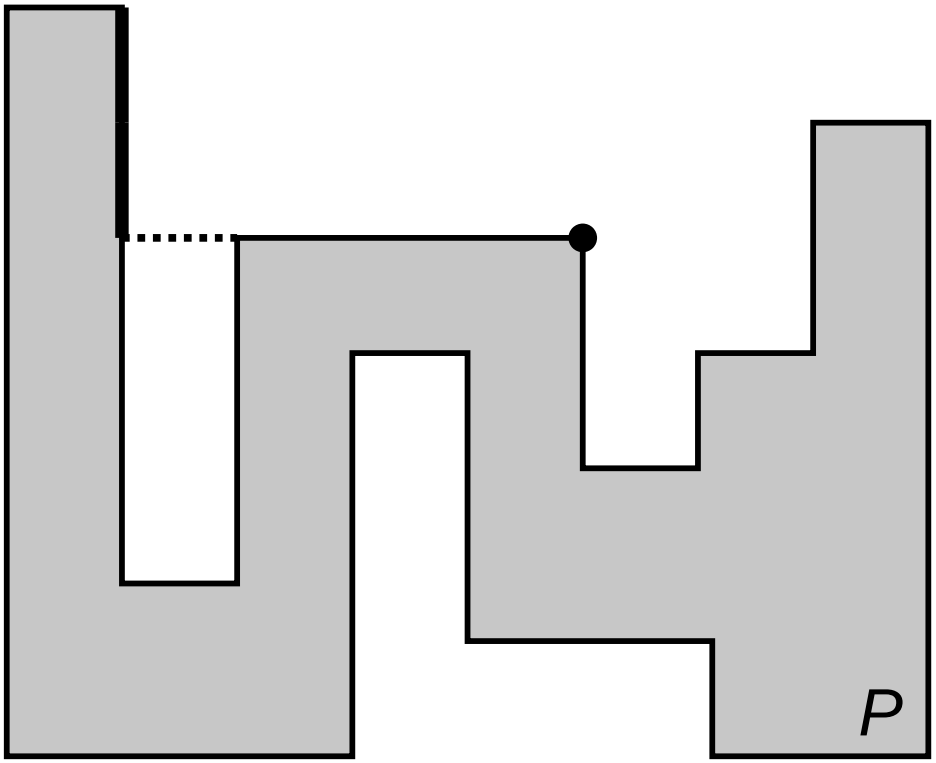
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



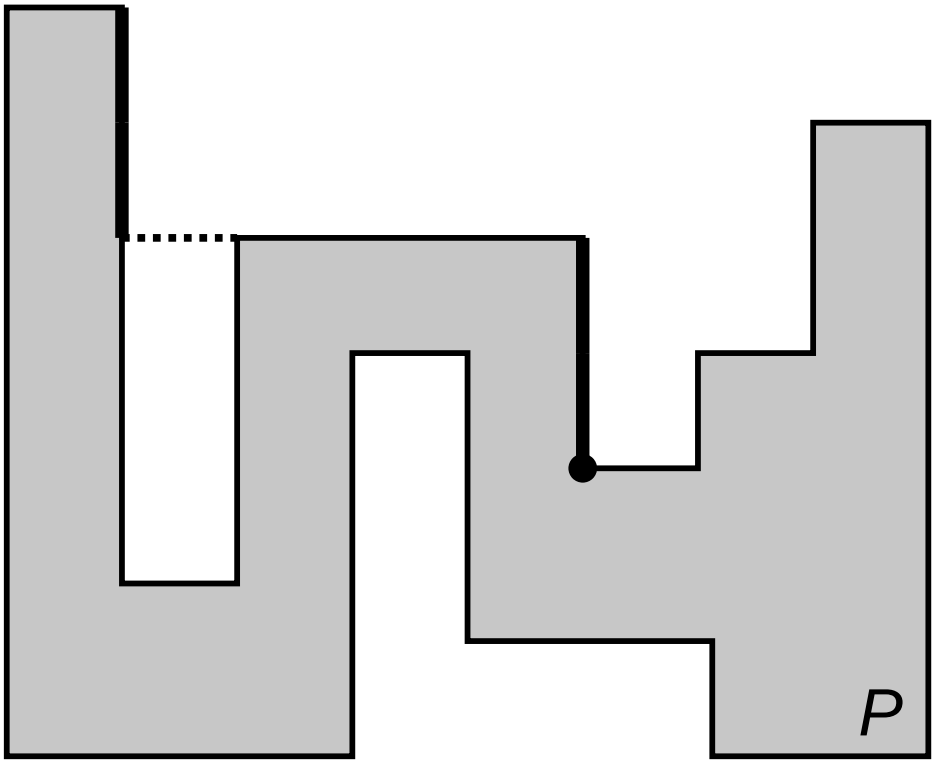
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



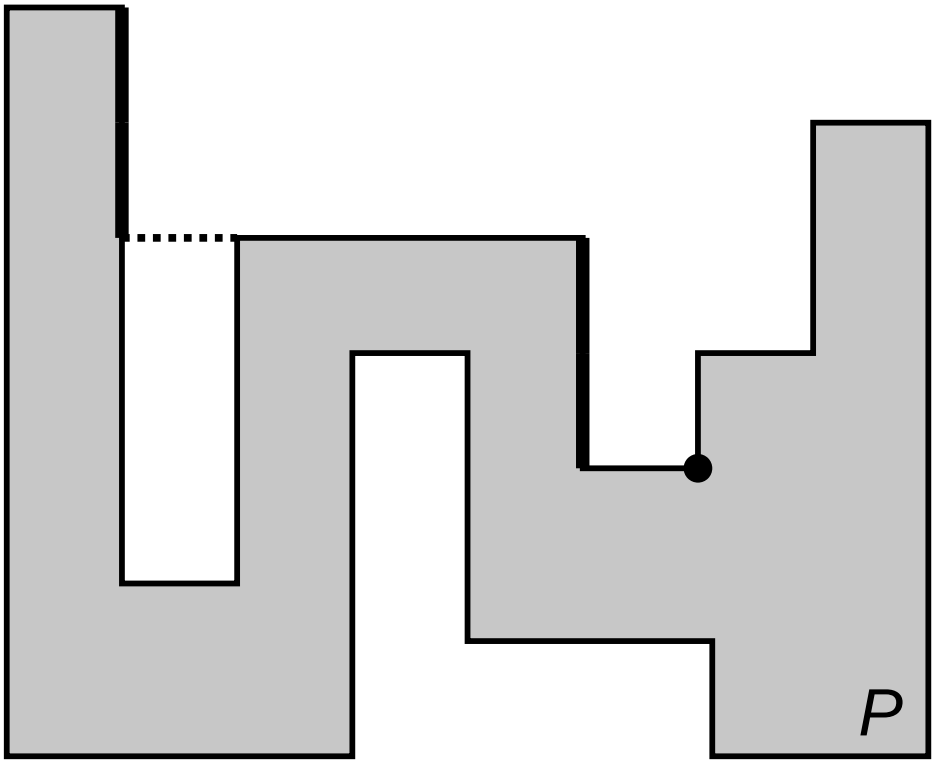
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



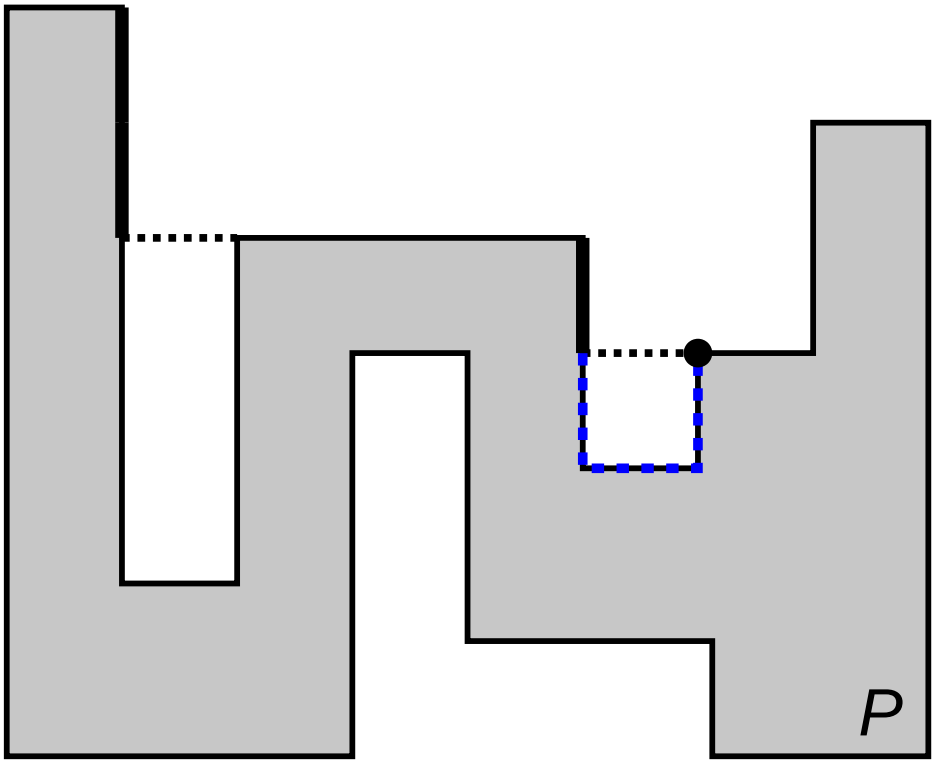
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



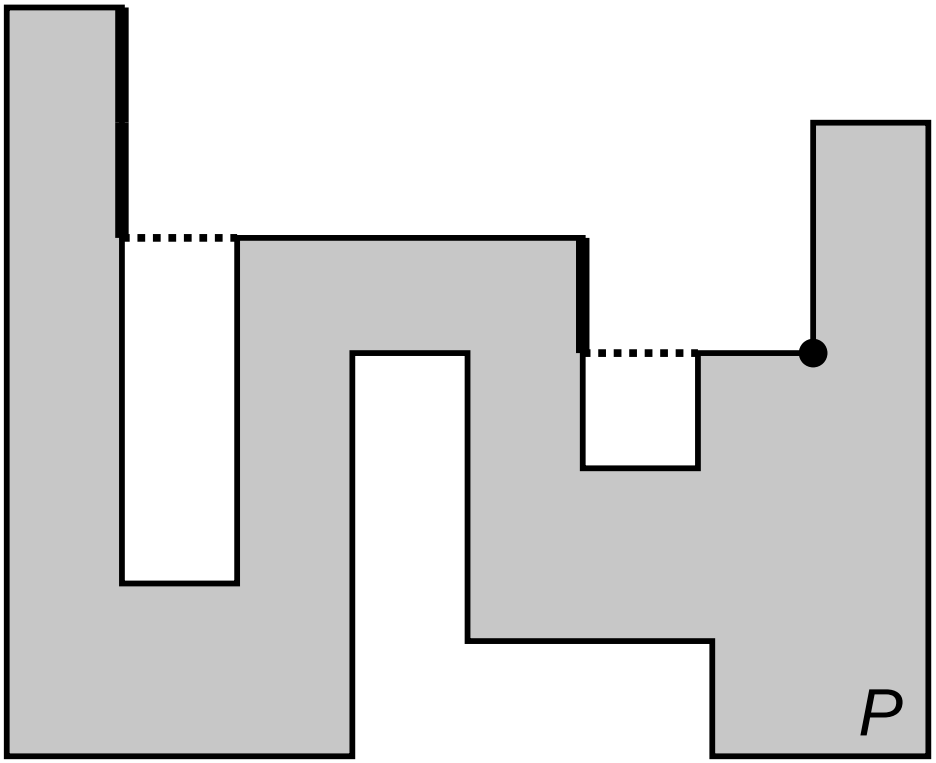
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



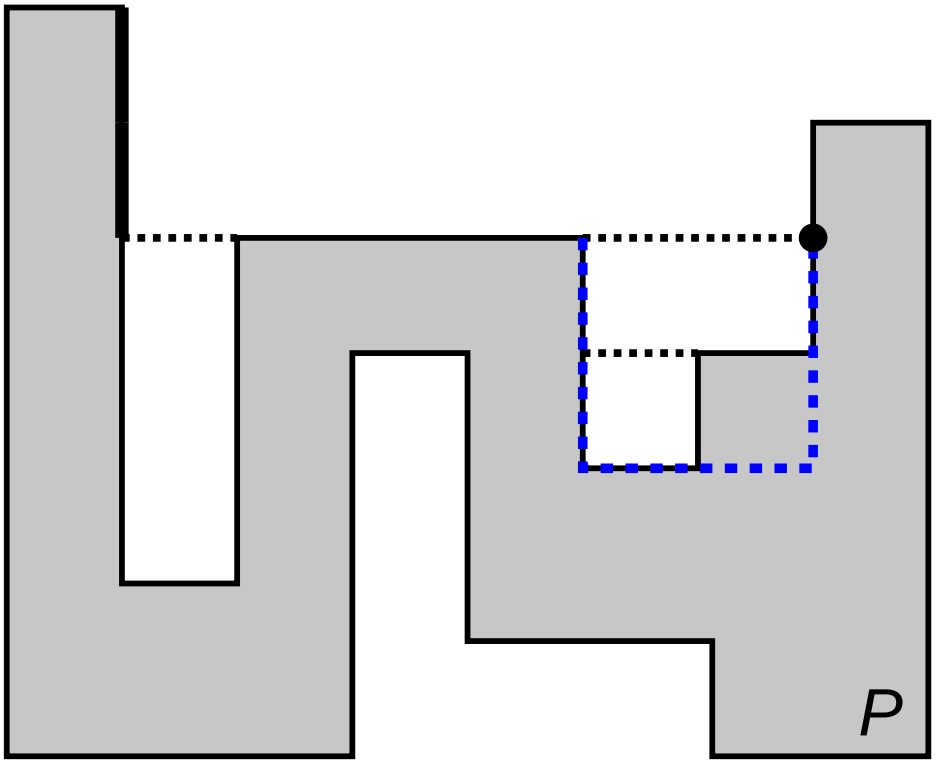
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



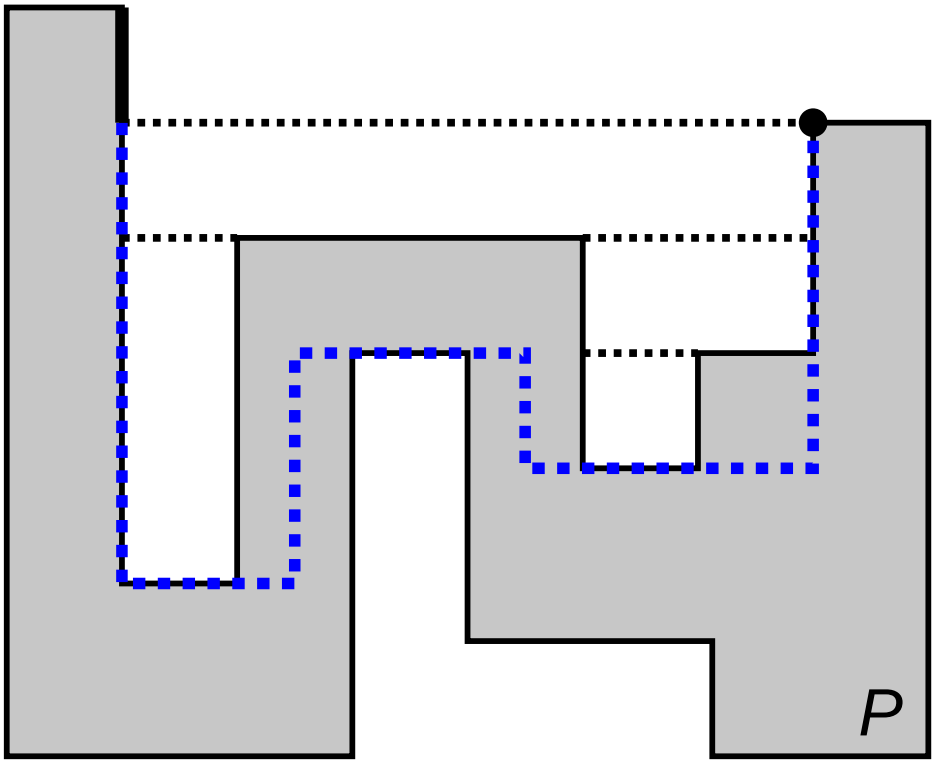
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



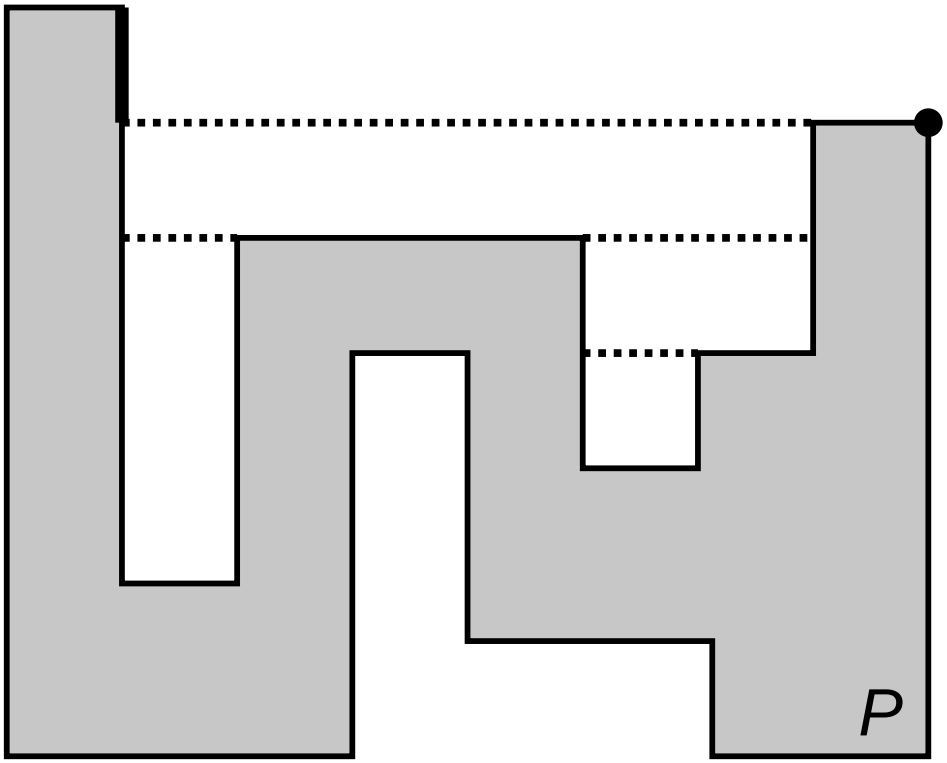
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



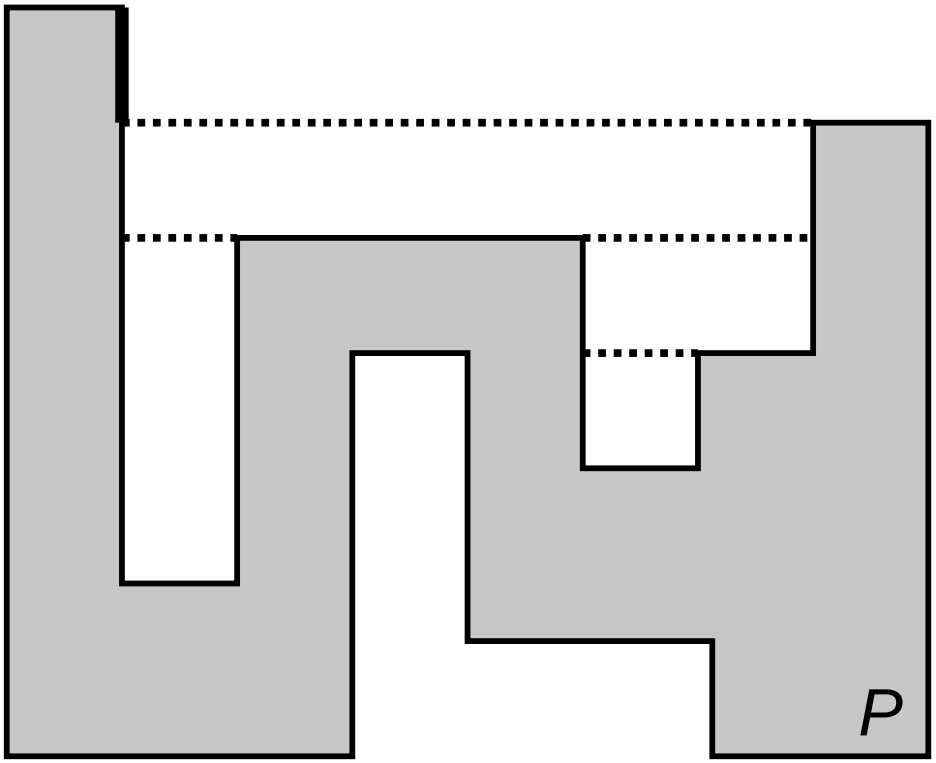
- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Scan of Upper Boundary



- current boundary point
- segments on stack
- found candidates
- $d_P(.,.)$

Analysis of Algorithm

- x-component of $d_P(p, q)$ is $|q_x - p_x|$
- algorithm has to compute y-component
- additional path information can be stored on stack
- $\Rightarrow$ time $O(n)$

Analysis of Algorithm

- x-component of $d_P(p, q)$ is $|q_x - p_x|$
- algorithm has to compute y-component
- additional path information can be stored on stack
- $\Rightarrow$ time $O(n)$

Analysis of Algorithm

- x-component of $d_P(p, q)$ is $|q_x - p_x|$
- algorithm has to compute y-component
- additional path information can be stored on stack
- $\Rightarrow$ time $O(n)$

Analysis of Algorithm

- x-component of $d_P(p, q)$ is $|q_x - p_x|$
- algorithm has to compute y-component
- additional path information can be stored on stack
- $\Rightarrow$ time $O(n)$

Analysis of Algorithm

- x-component of $d_P(p, q)$ is $|q_x - p_x|$
- algorithm has to compute y-component
- additional path information can be stored on stack
- $\Rightarrow$ time $O(n)$

Other Detour Results

- "Approximating the Stretch Factor of Euclidean Graphs", G. Narasimhan and M. Smid, 2000
- "A Fast Algorithm for Approximating the Detour of a Polygonal Chain", A. Ebbers-Baumann, R. Klein, E. Langetepe and A. Lingas, 2001
- "Computing the Detour of Polygonal Curves" (Technical Report), P. K. Agarwal, R. Klein, C. Knauer, M. Sharir, 2002
- "Computing the Maximum Detour and Spanning Ratio of Planar Chains, Trees and Cycles", S. Langerman, P. Morin, M. Soss, 2002

Lower Time Bound of Euclidean Vertex Detour

- computing detour maximum of vertices $\delta_V(P) := \max_{p,q \in V} \delta(p,q)$
- reduction to Element-Uniqueness of integers (Yao, 89): $\Omega(n \log n)$
  - build special polygon P from given integers $y_1, \ldots, y_n$ in O(n)
  - compute vertex detour $\delta_V(P)$
  - Element-Uniqueness $\Leftrightarrow \delta_V(P) \leq C$

Lower Time Bound of Euclidean Vertex Detour

- computing detour maximum of vertices $\delta_V(P) := \max_{p,q \in V} \delta(p,q)$
- reduction to Element-Uniqueness of integers (Yao, 89): $\Omega(n \log n)$
  - build special polygon P from given integers $y_1, \ldots, y_n$ in O(n)
  - compute vertex detour $\delta_V(P)$
  - Element-Uniqueness $\Leftrightarrow \delta_V(P) \leq C$

Lower Time Bound of Euclidean Vertex Detour

- computing detour maximum of vertices $\delta_V(P) := \max_{p,q \in V} \delta(p,q)$
- reduction to Element-Uniqueness of integers (Yao, 89): $\Omega(n \log n)$
  - build special polygon P from given integers $y_1, \ldots, y_n$ in O(n)
  - compute vertex detour $\delta_V(P)$
  - Element-Uniqueness $\Leftrightarrow \delta_V(P) \leq C$

Lower Time Bound of Euclidean Vertex Detour

- computing detour maximum of vertices $\delta_V(P) := \max_{p,q \in V} \delta(p,q)$
- reduction to Element-Uniqueness of integers (Yao, 89): $\Omega(n \log n)$
  - build special polygon P from given integers $y_1, \ldots, y_n$ in O(n)
  - compute vertex detour $\delta_V(P)$
  - Element-Uniqueness $\Leftrightarrow \delta_V(P) \leq C$

Lower Time Bound of Euclidean Vertex Detour

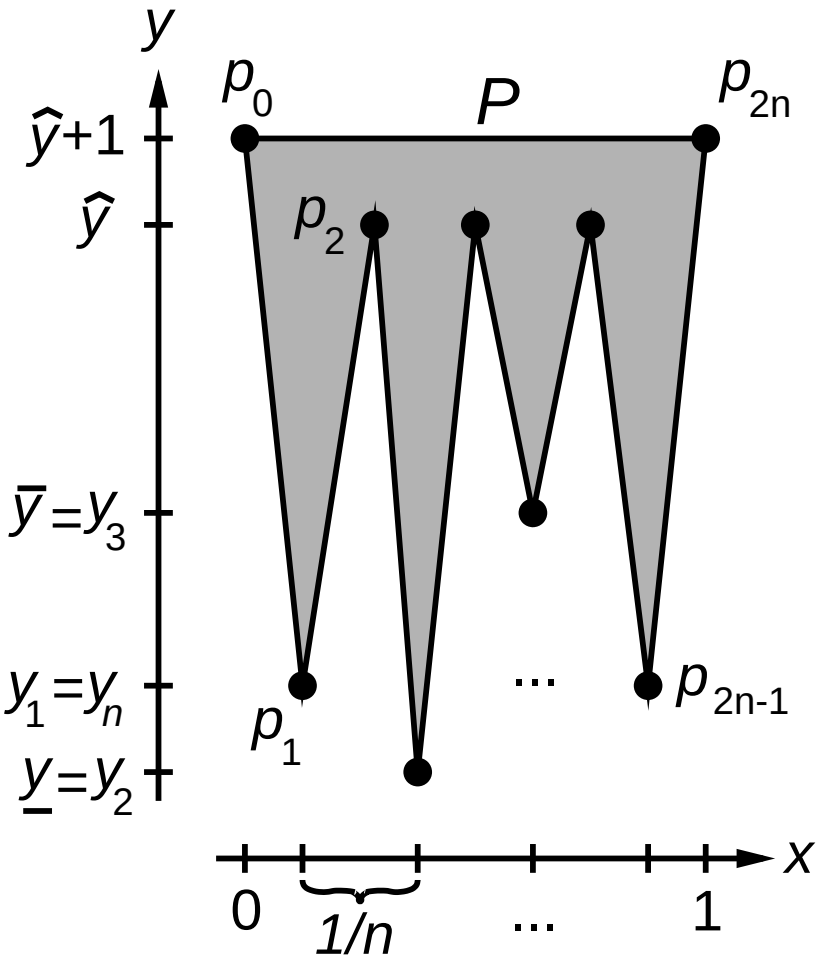
- computing detour maximum of vertices $\delta_V(P) := \max_{p,q \in V} \delta(p,q)$
- reduction to Element-Uniqueness of integers (Yao, 89): $\Omega(n \log n)$
  - build special polygon P from given integers $y_1, \ldots, y_n$ in O(n)
  - compute vertex detour $\delta_V(P)$
  - Element-Uniqueness $\Leftrightarrow \delta_V(P) \leq C$

Lower Time Bound of Euclidean Vertex Detour

- computing detour maximum of vertices $\delta_V(P) := \max_{p,q \in V} \delta(p,q)$
- reduction to Element-Uniqueness of integers (Yao, 89): $\Omega(n \log n)$
  - build special polygon P from given integers $y_1, \ldots, y_n$ in O(n)
  - compute vertex detour $\delta_V(P)$
  - Element-Uniqueness $\Leftrightarrow \delta_V(P) \leq C$

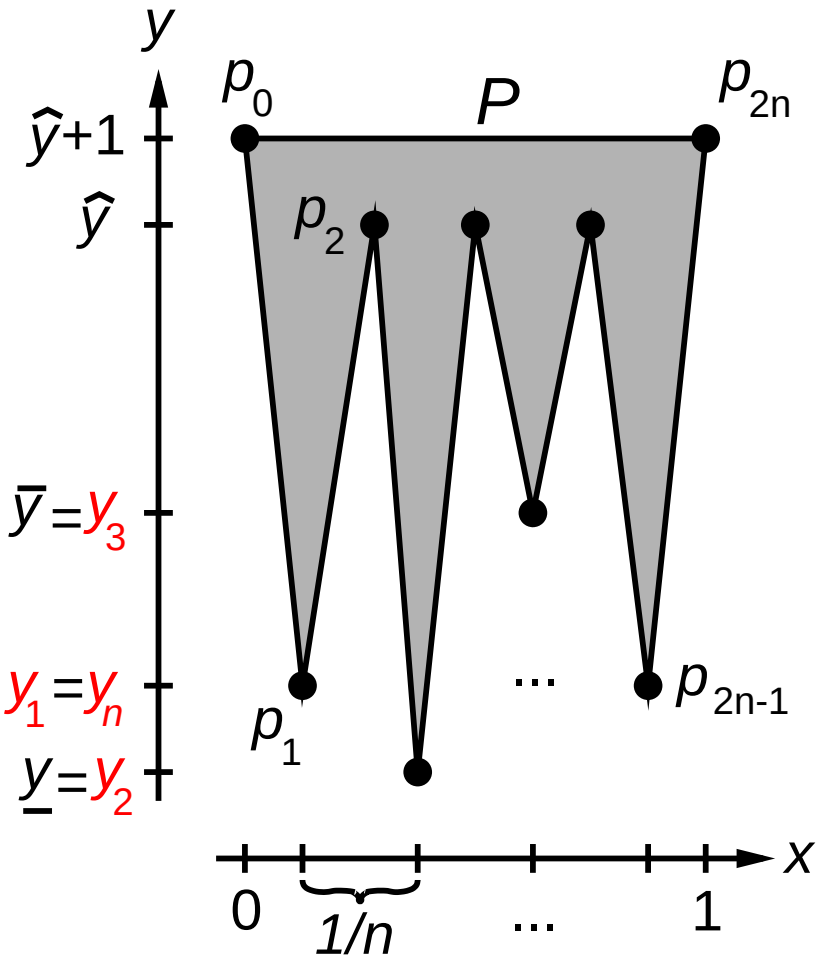
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



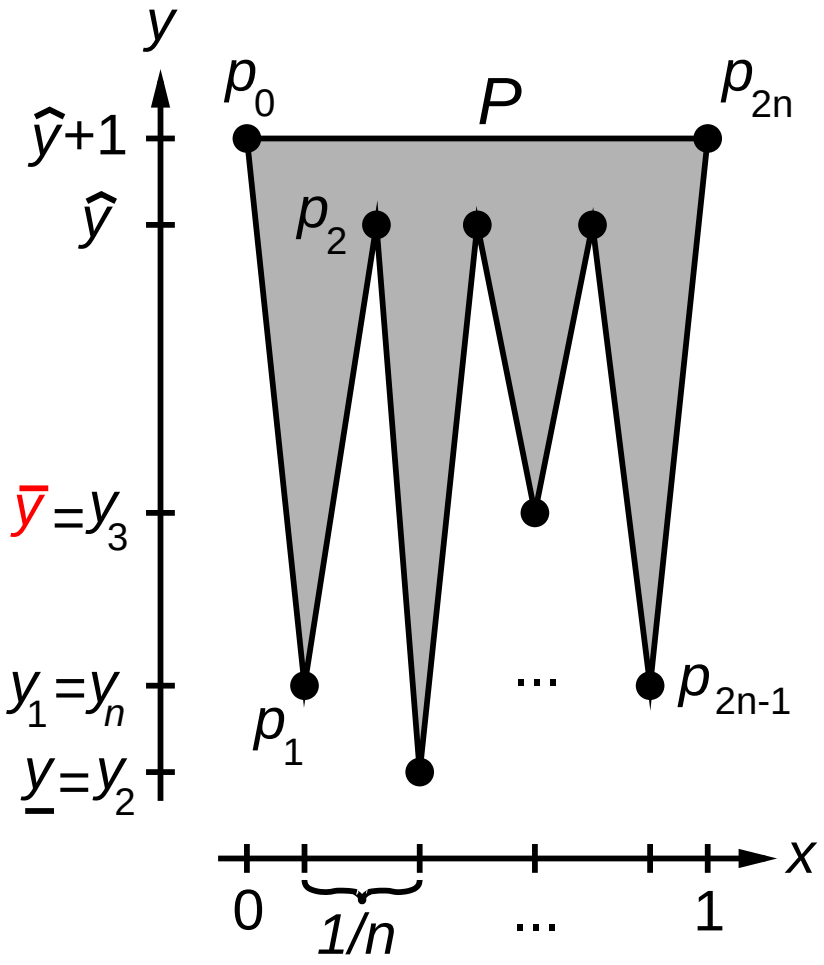
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



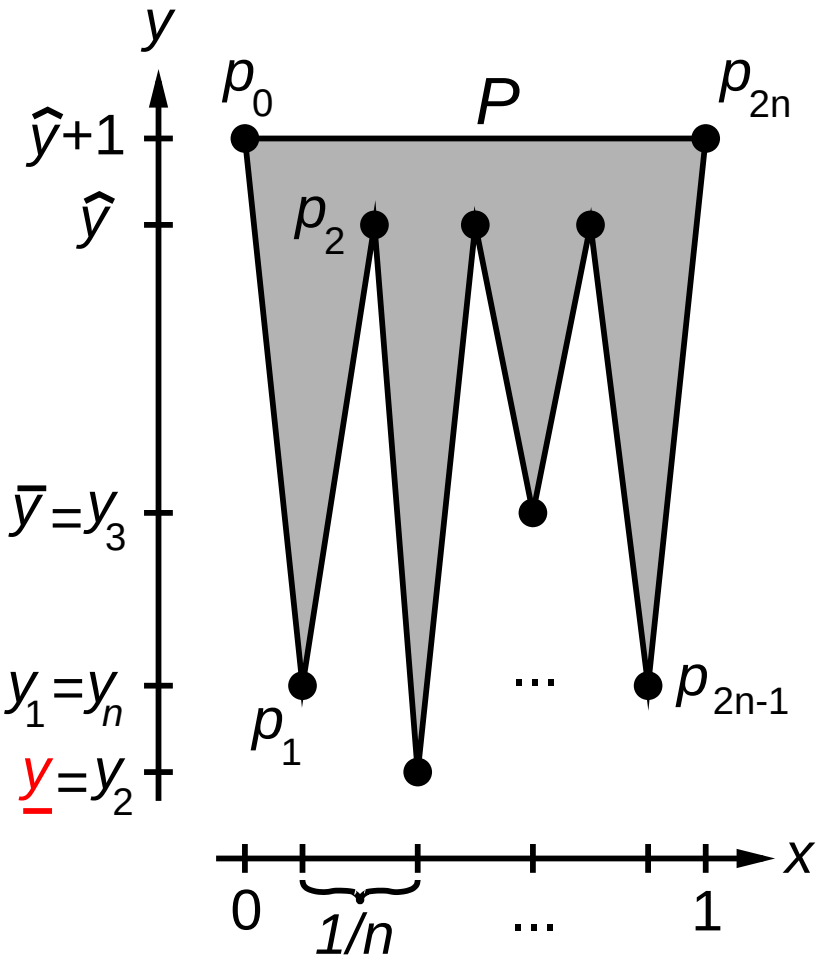
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



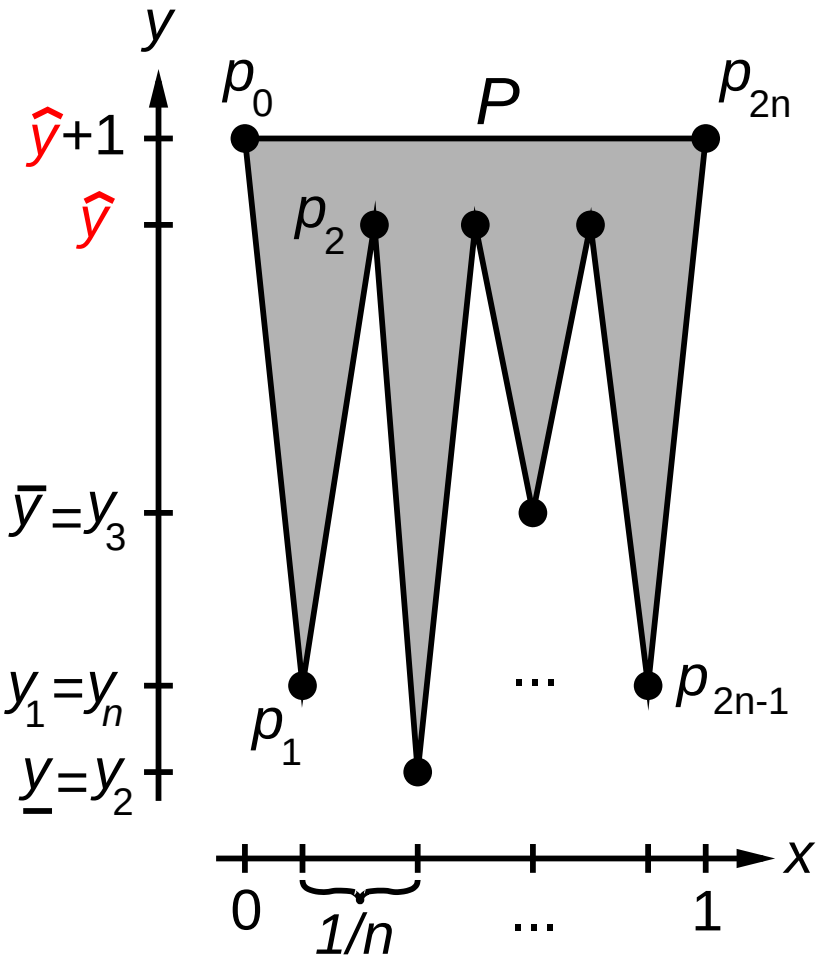
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



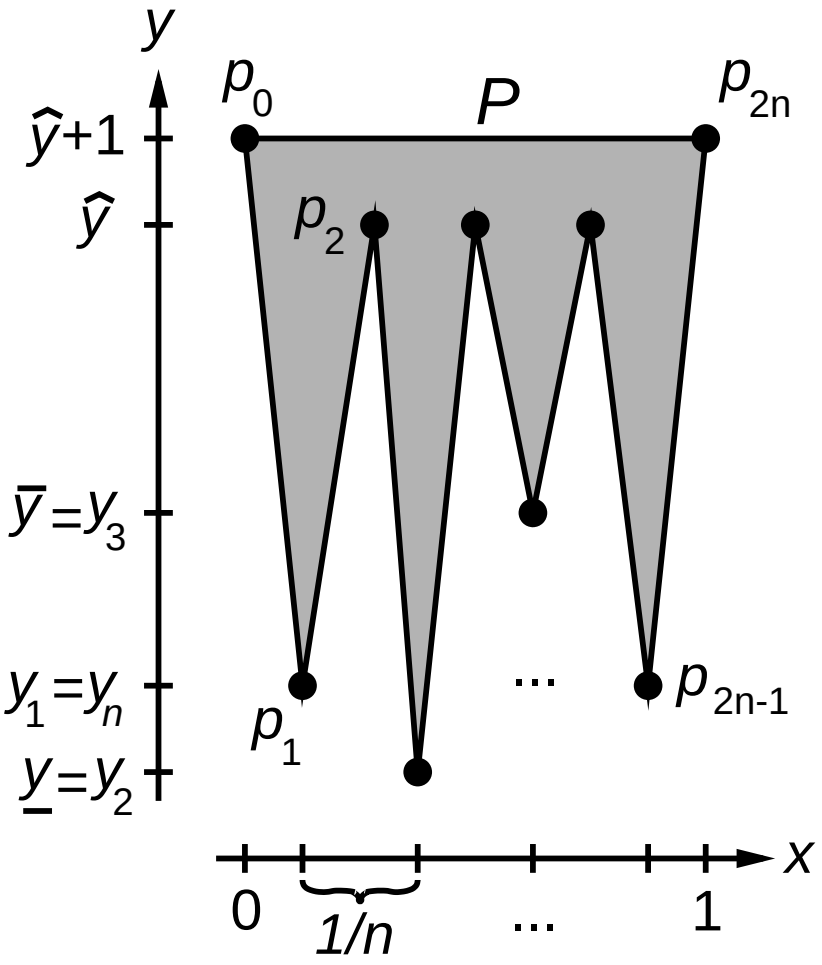
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



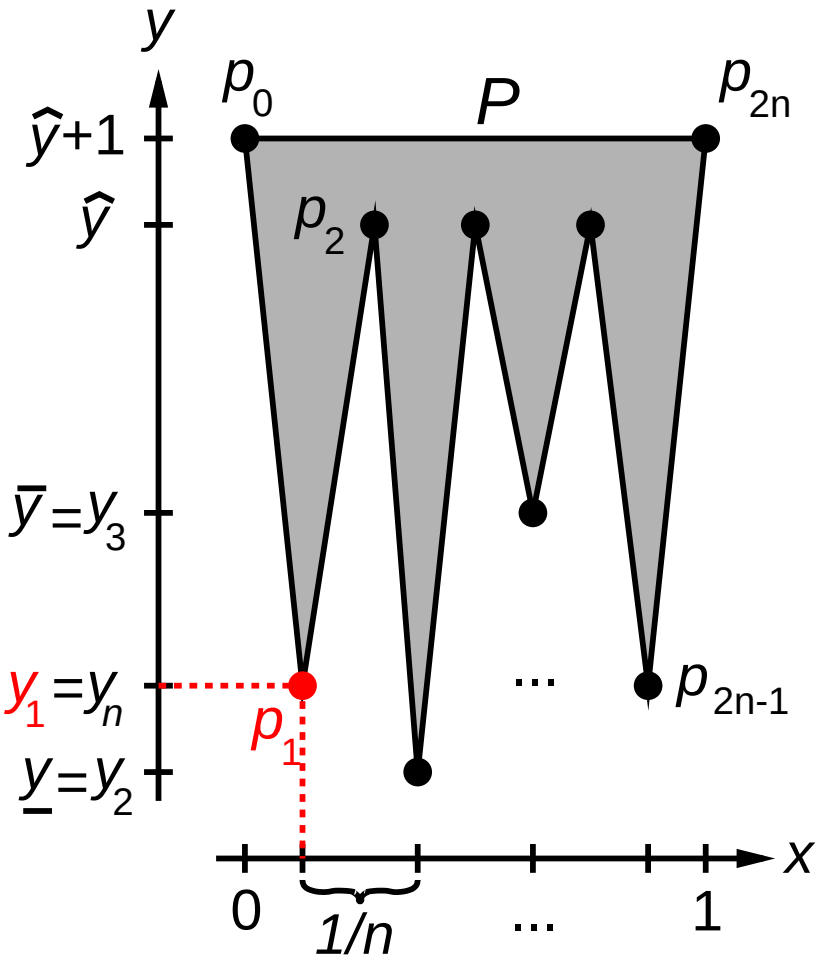
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



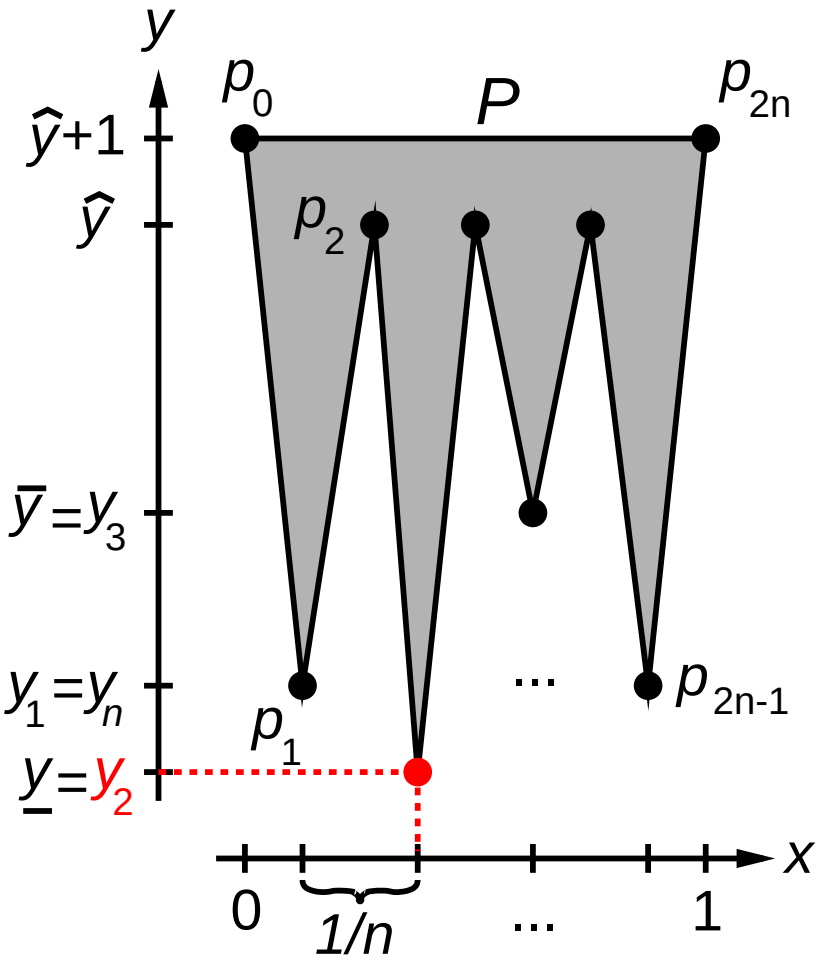
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



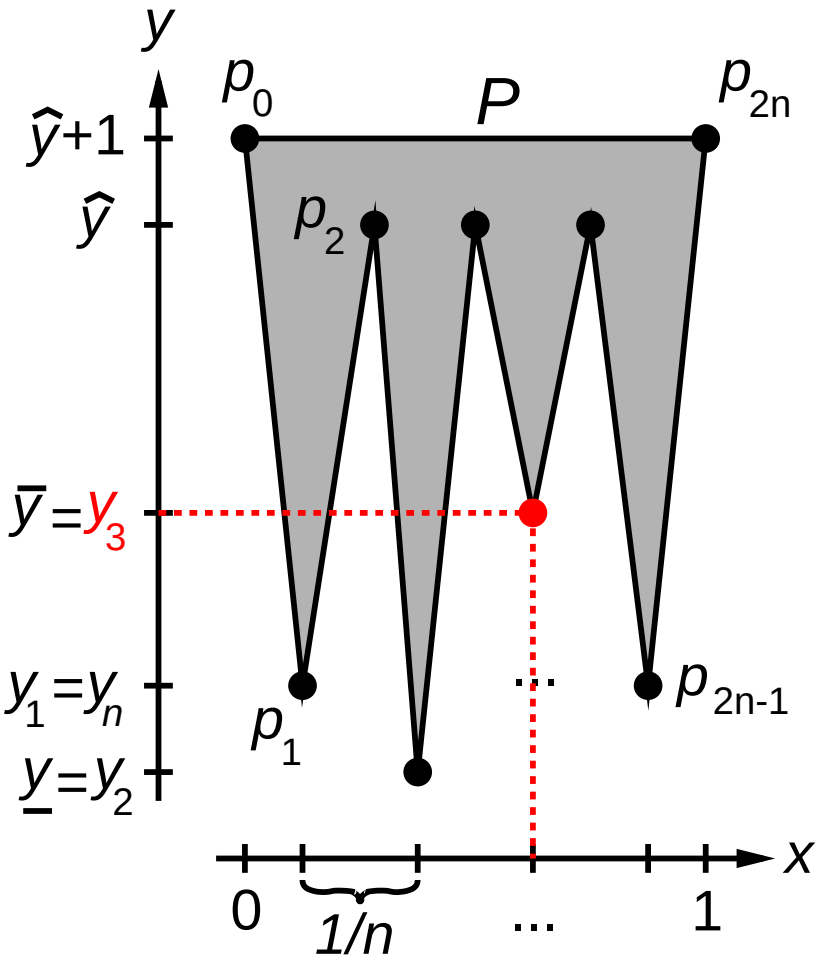
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



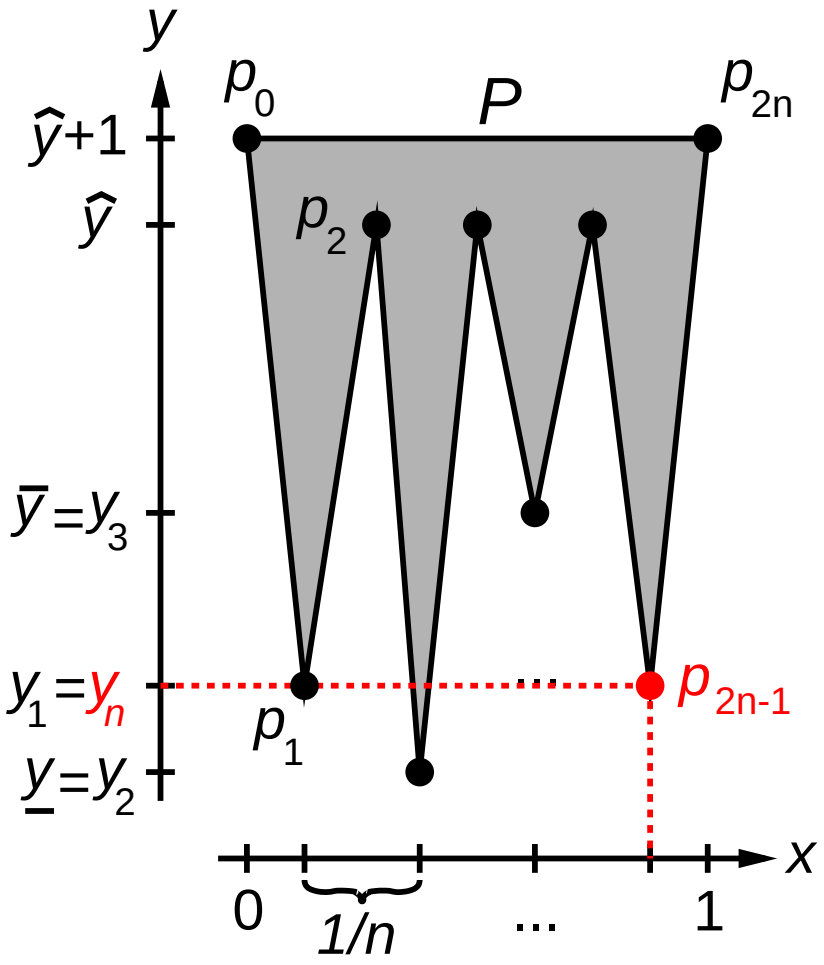
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



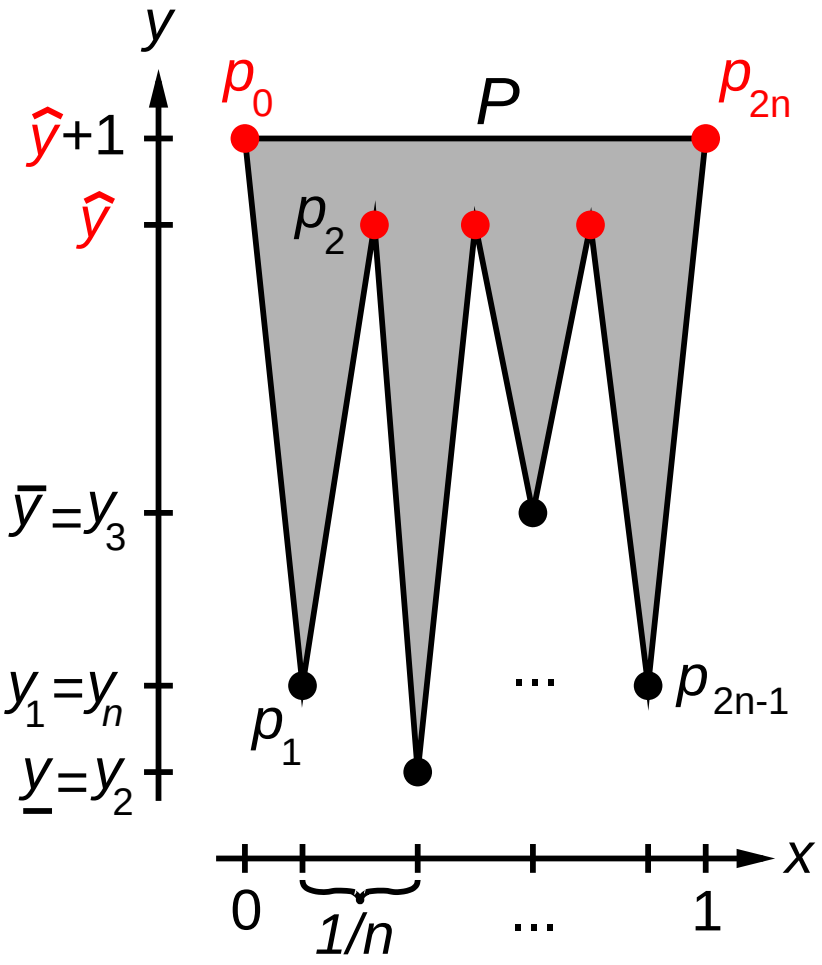
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



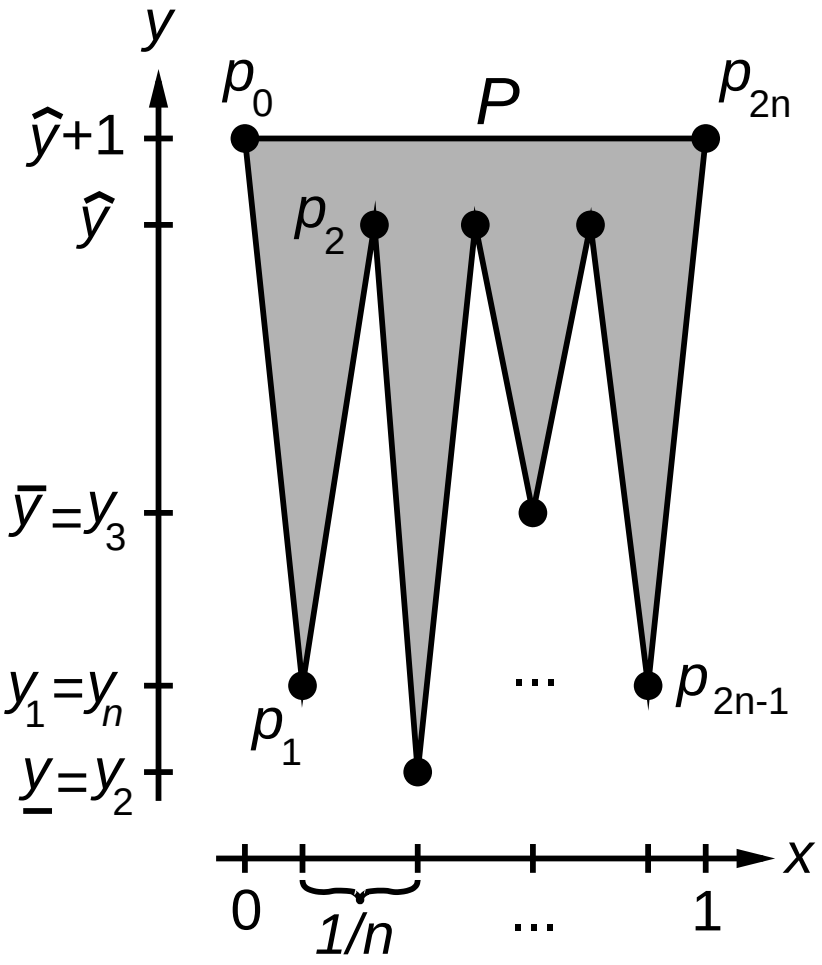
Lower Time Bound of Euclidean Vertex Detour

- given integers $y_1, \ldots, y_n$
- compute $\overline{y} := \max y_i$ and $\underline{y} := \min y_i$
- select high value $\hat{y}$ depending on $\overline{y}$, $\underline{y}$ and n
- build comb-like polygon P



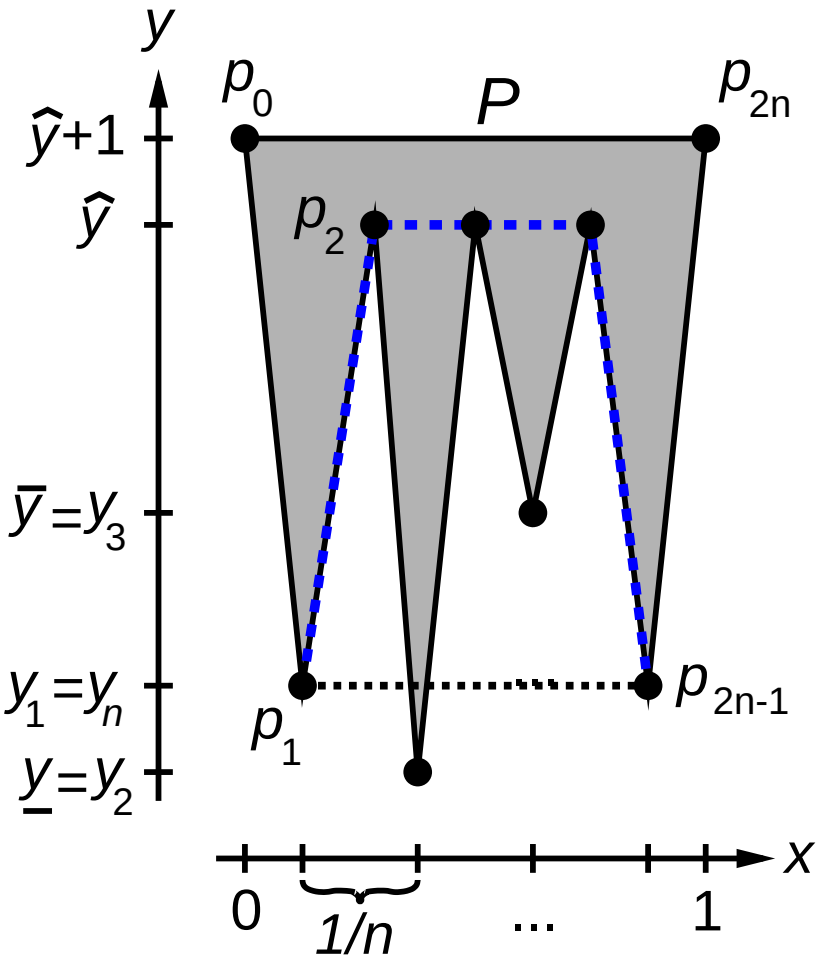
Lower Time Bound of Euclidean Vertex Detour

- $y_i = y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) > C_1(\overline{y}, \hat{y})$
- $y_i \neq y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) \leq C_2(\underline{y}, \hat{y}, n)$
- $\hat{y}$ big enough $\Rightarrow C_1 > C_2$
- $\Rightarrow$ Element-Uniqueness iff
  $\delta_V(P) \leq C_2$



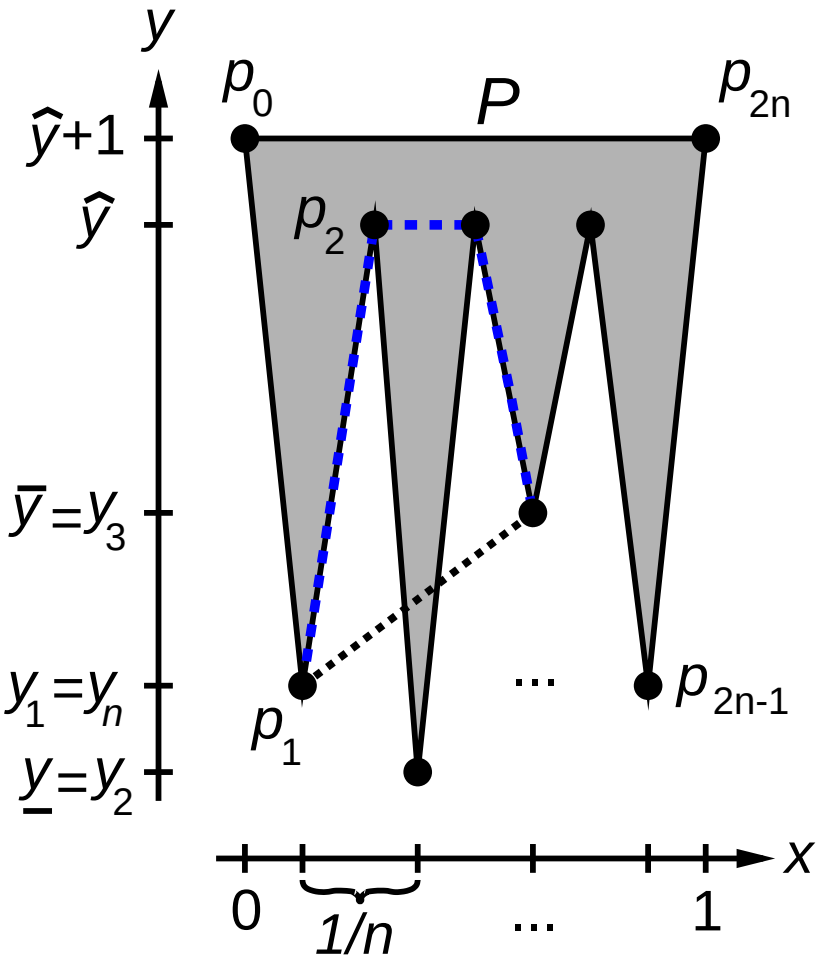
Lower Time Bound of Euclidean Vertex Detour

- $y_i = y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) > C_1(\overline{y}, \hat{y})$
- $y_i \neq y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) \leq C_2(\underline{y}, \hat{y}, n)$
- $\hat{y}$ big enough $\Rightarrow C_1 > C_2$
- $\Rightarrow$ Element-Uniqueness iff
  $\delta_V(P) \leq C_2$



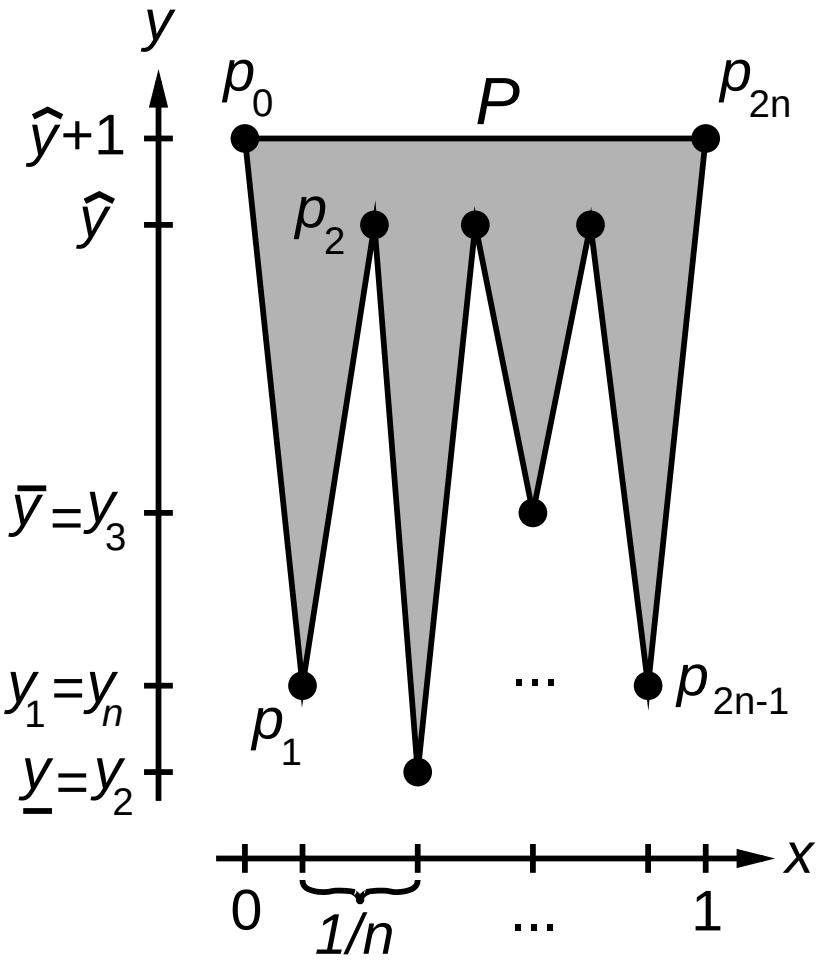
Lower Time Bound of Euclidean Vertex Detour

- $y_i = y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) > C_1(\overline{y}, \hat{y})$
- $y_i \neq y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) \leq C_2(\underline{y}, \hat{y}, n)$
- $\hat{y}$ big enough $\Rightarrow C_1 > C_2$
- $\Rightarrow$ Element-Uniqueness iff
  $\delta_V(P) \leq C_2$



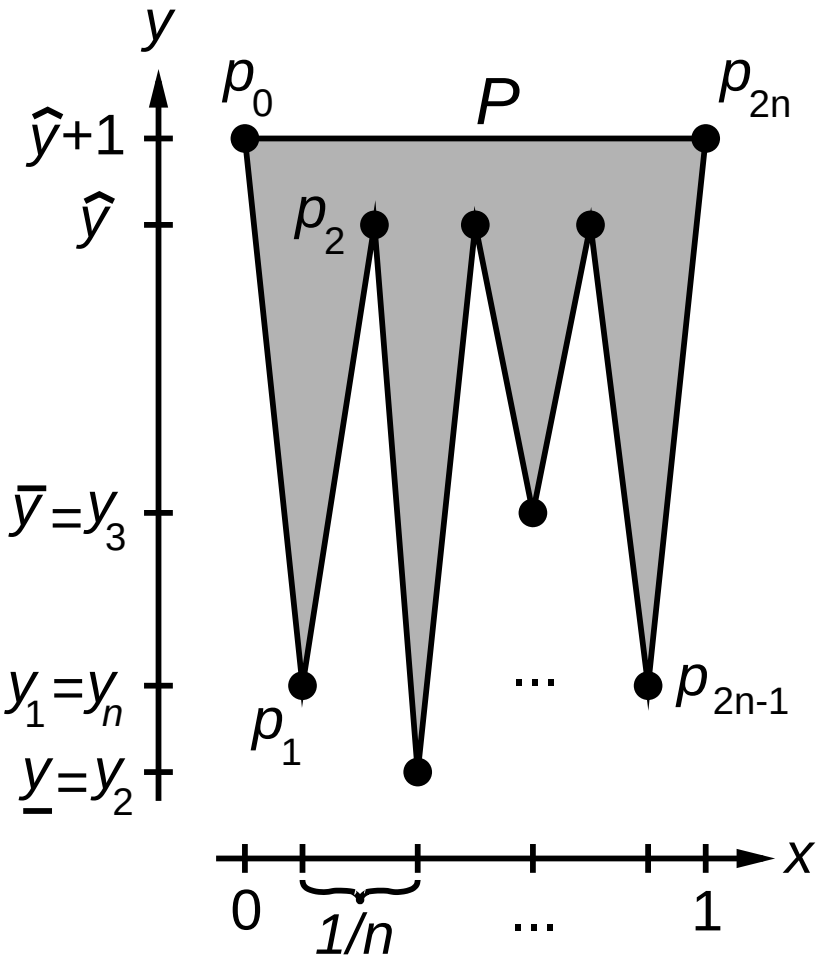
Lower Time Bound of Euclidean Vertex Detour

- $y_i = y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) > C_1(\overline{y}, \hat{y})$
- $y_i \neq y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) \leq C_2(\underline{y}, \hat{y}, n)$
- $\hat{y}$ big enough $\Rightarrow C_1 > C_2$
- $\Rightarrow$ Element-Uniqueness iff
  $\delta_V(P) \leq C_2$



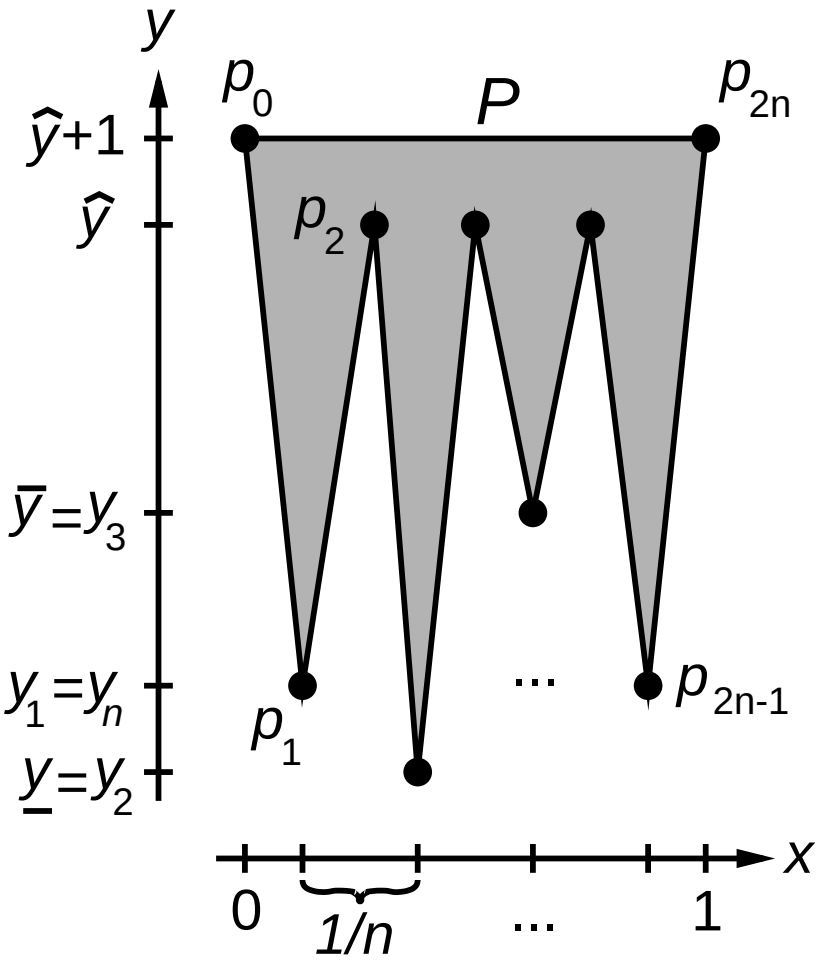
Lower Time Bound of Euclidean Vertex Detour

- $y_i = y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) > C_1(\overline{y}, \hat{y})$
- $y_i \neq y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) \leq C_2(\underline{y}, \hat{y}, n)$
- $\hat{y}$ big enough $\Rightarrow C_1 > C_2$
- $\Rightarrow$ Element-Uniqueness iff
  $\delta_V(P) \leq C_2$



Lower Time Bound of Euclidean Vertex Detour

- $y_i = y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) > C_1(\overline{y}, \hat{y})$
- $y_i \neq y_j \Rightarrow$
  $\delta(p_{2i-1}, p_{2j-1}) \leq C_2(\underline{y}, \hat{y}, n)$
- $\hat{y}$ big enough $\Rightarrow C_1 > C_2$
- $\Rightarrow$ Element-Uniqueness iff
  $\delta_V(P) \leq C_2$



More Lower Time Bounds

- $\Omega(n \log n)$-time bound extends to good approximations of $\delta_V(P)$
- same arguments hold for L_1-vertex detour $\delta_V^{L_1}(P)$
  - in particular: $\Omega(n \log n)$ for $\delta_V^{L_1}(P)$ of x-monotone rectilinear polygons but $O(n)$ for $\delta^{L_1}(P)$
- can also be transferred to polygonal chains
  - Narasimhan and Smid, 2000: $\Omega(n \log n)$ approximating δ_V using self-intersecting chains

More Lower Time Bounds

- $\Omega(n \log n)$-time bound extends to good approximations of $\delta_V(P)$
- same arguments hold for L_1-vertex detour $\delta_V^{L_1}(P)$
  - in particular: $\Omega(n \log n)$ for $\delta_V^{L_1}(P)$ of x-monotone rectilinear polygons but $O(n)$ for $\delta^{L_1}(P)$
- can also be transferred to polygonal chains
  - Narasimhan and Smid, 2000: $\Omega(n \log n)$ approximating δ_V using self-intersecting chains

More Lower Time Bounds

- $\Omega(n \log n)$-time bound extends to good approximations of $\delta_V(P)$
- same arguments hold for L_1-vertex detour $\delta_V^{L_1}(P)$
  - in particular: $\Omega(n \log n)$ for $\delta_V^{L_1}(P)$ of x-monotone rectilinear polygons but $O(n)$ for $\delta^{L_1}(P)$
- can also be transferred to polygonal chains
  - Narasimhan and Smid, 2000: $\Omega(n \log n)$ approximating δ_V using self-intersecting chains

More Lower Time Bounds

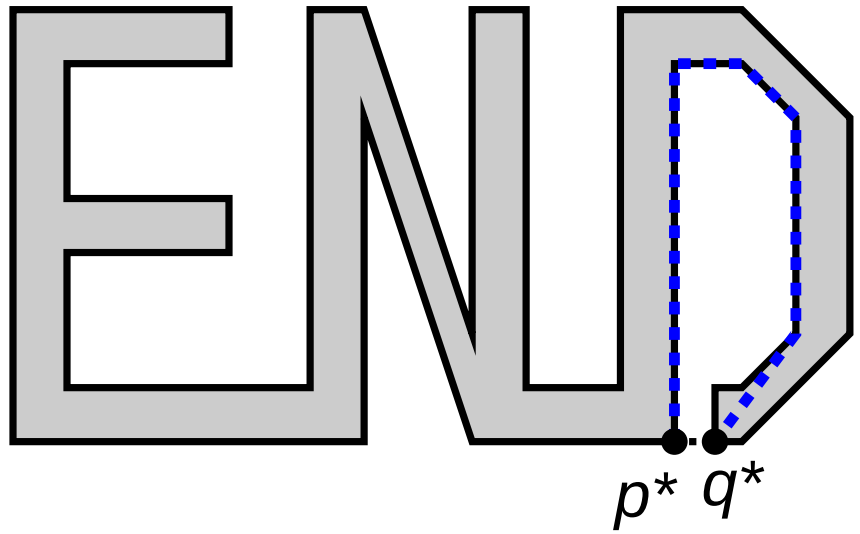
- $\Omega(n \log n)$-time bound extends to good approximations of $\delta_V(P)$
- same arguments hold for L_1-vertex detour $\delta_V^{L_1}(P)$
  - in particular: $\Omega(n \log n)$ for $\delta_V^{L_1}(P)$ of x-monotone rectilinear polygons but $O(n)$ for $\delta^{L_1}(P)$
- can also be transferred to polygonal chains
  - Narasimhan and Smid, 2000: $\Omega(n \log n)$ approximating δ_V using self-intersecting chains

More Lower Time Bounds

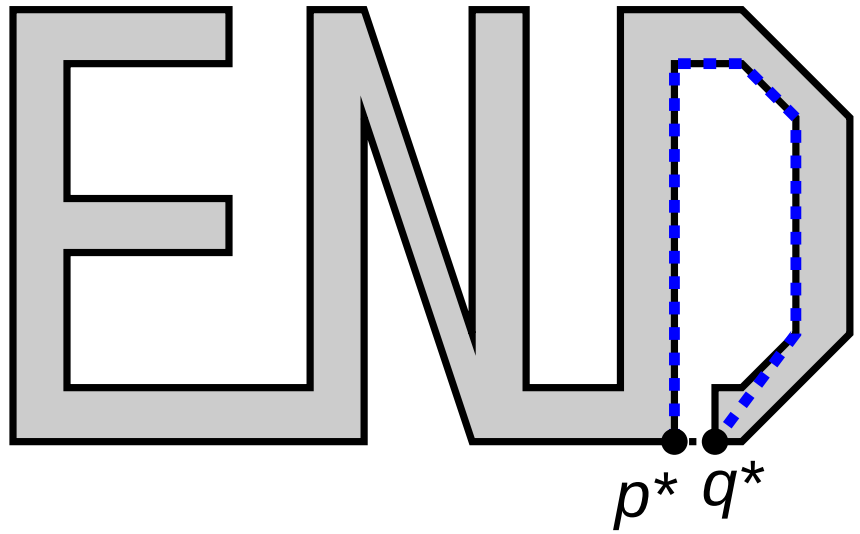
- $\Omega(n \log n)$-time bound extends to good approximations of $\delta_V(P)$
- same arguments hold for L_1-vertex detour $\delta_V^{L_1}(P)$
  - in particular: $\Omega(n \log n)$ for $\delta_V^{L_1}(P)$ of x-monotone rectilinear polygons but $O(n)$ for $\delta^{L_1}(P)$
- can also be transferred to polygonal chains
  - Narasimhan and Smid, 2000: $\Omega(n \log n)$ approximating δ_V using self-intersecting chains

More Lower Time Bounds

- $\Omega(n \log n)$-time bound extends to good approximations of $\delta_V(P)$
- same arguments hold for L_1-vertex detour $\delta_V^{L_1}(P)$
  - in particular: $\Omega(n \log n)$ for $\delta_V^{L_1}(P)$ of x-monotone rectilinear polygons but $O(n)$ for $\delta^{L_1}(P)$
- can also be transferred to polygonal chains
  - Narasimhan and Smid, 2000: $\Omega(n \log n)$ approximating δ_V using self-intersecting chains

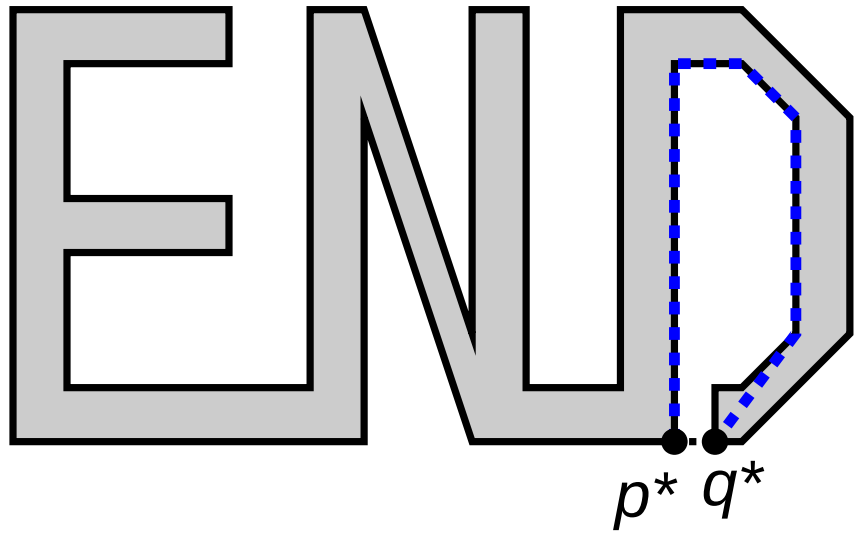


- Euclidean Results in “Umwege in Polygonen”, A. Grüne, master’s thesis
- Java-Applet Computing Euclidean Detour of Polygons at http://web.informatik.uni-bonn.de/I/staff/gruene.html

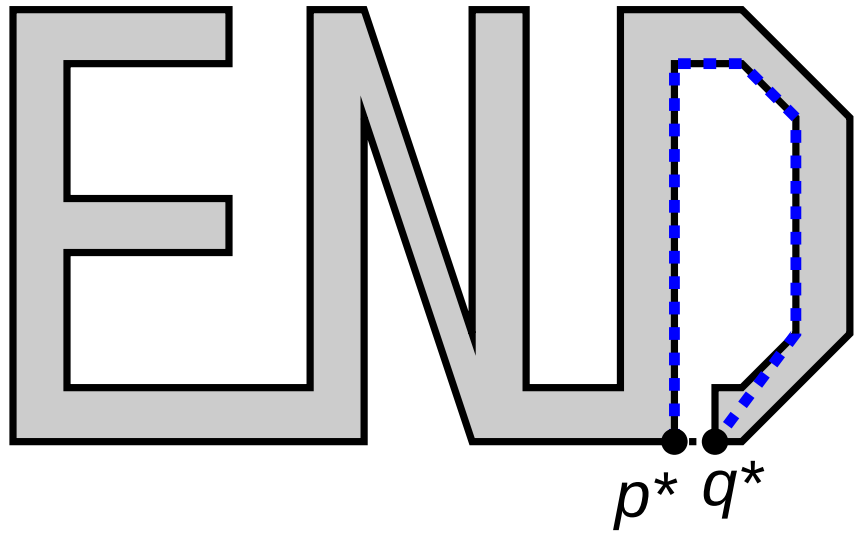


- Euclidean Results in "Umwege in Polygonen", A. Grüne, master's thesis

- Java-Applet Computing Euclidean Detour of Polygons at http://web.informatik.uni-bonn.de/I/staff/gruene.html



- Euclidean Results in "Umwege in Polygonen", A. Grüne, master's thesis
- Java-Applet Computing Euclidean Detour of Polygons at http://web.informatik.uni-bonn.de/I/staff/gruene.html



- Euclidean Results in “Umwege in Polygonen”, A. Grüne, master’s thesis
- Java-Applet Computing Euclidean Detour of Polygons at http://web.informatik.uni-bonn.de/I/staff/gruene.html