Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

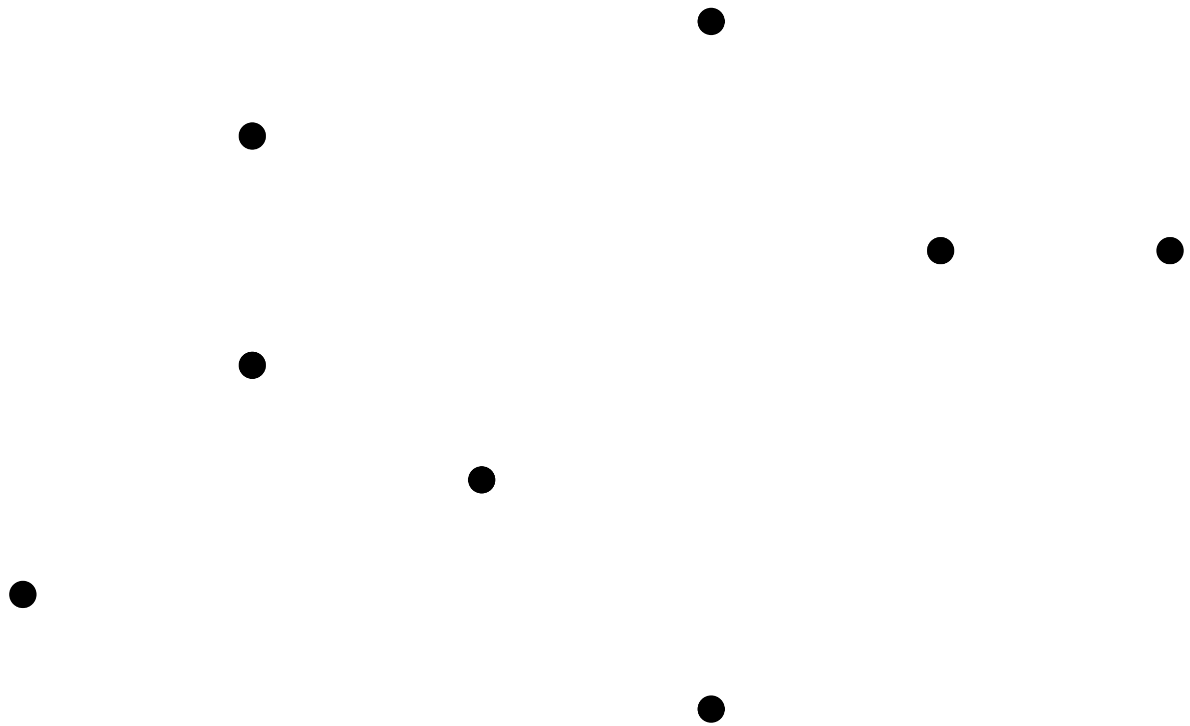
The Traveling Salesman Problem with Few Inner Points

Vladimir Deĭneko	U Warwick	UK
Michael Hoffmann	ETH Zurich	CH
Yoshio Okamoto	ETH Zurich	CH
Gerhard Woeginger	TU Eindhoven	NL

August 18, 2004 @ COCOON 2004, Jeju Island, Republic of Korea

The 2DTSP

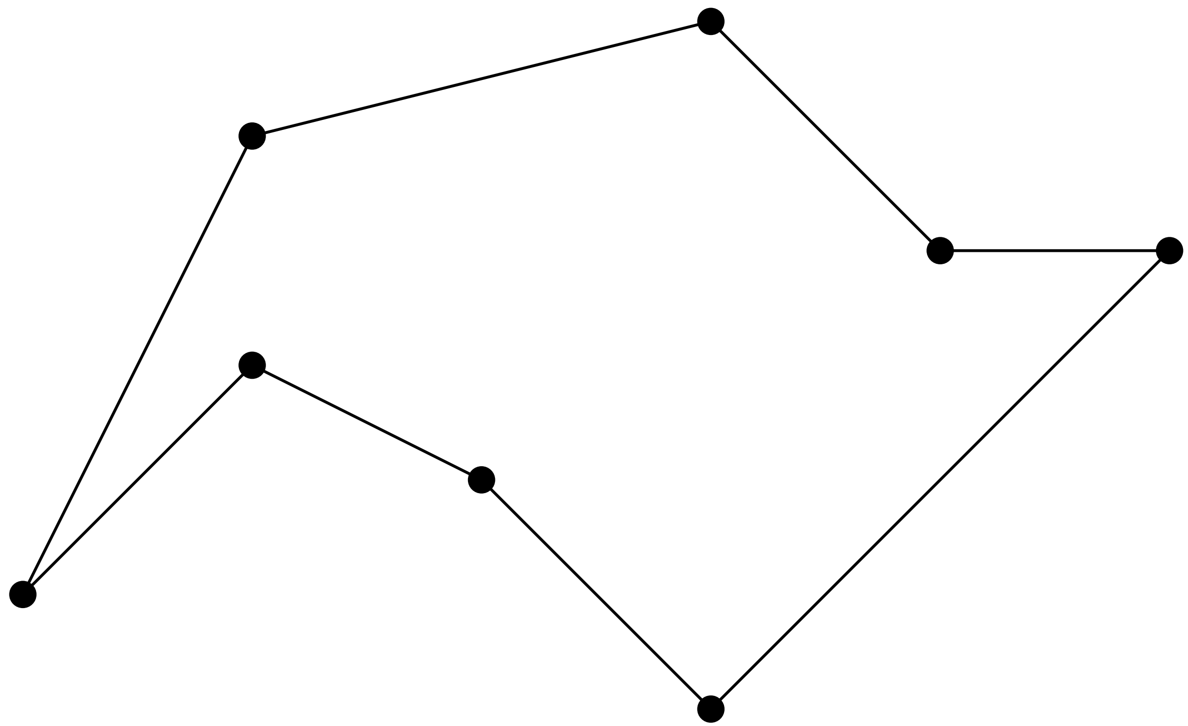
Given: finite set of points on $\mathbb{R}^2$
Find: a minimum-length tour



The 2DTSP

Given: finite set of points on $\mathbb{R}^2$

Find: a minimum-length tour

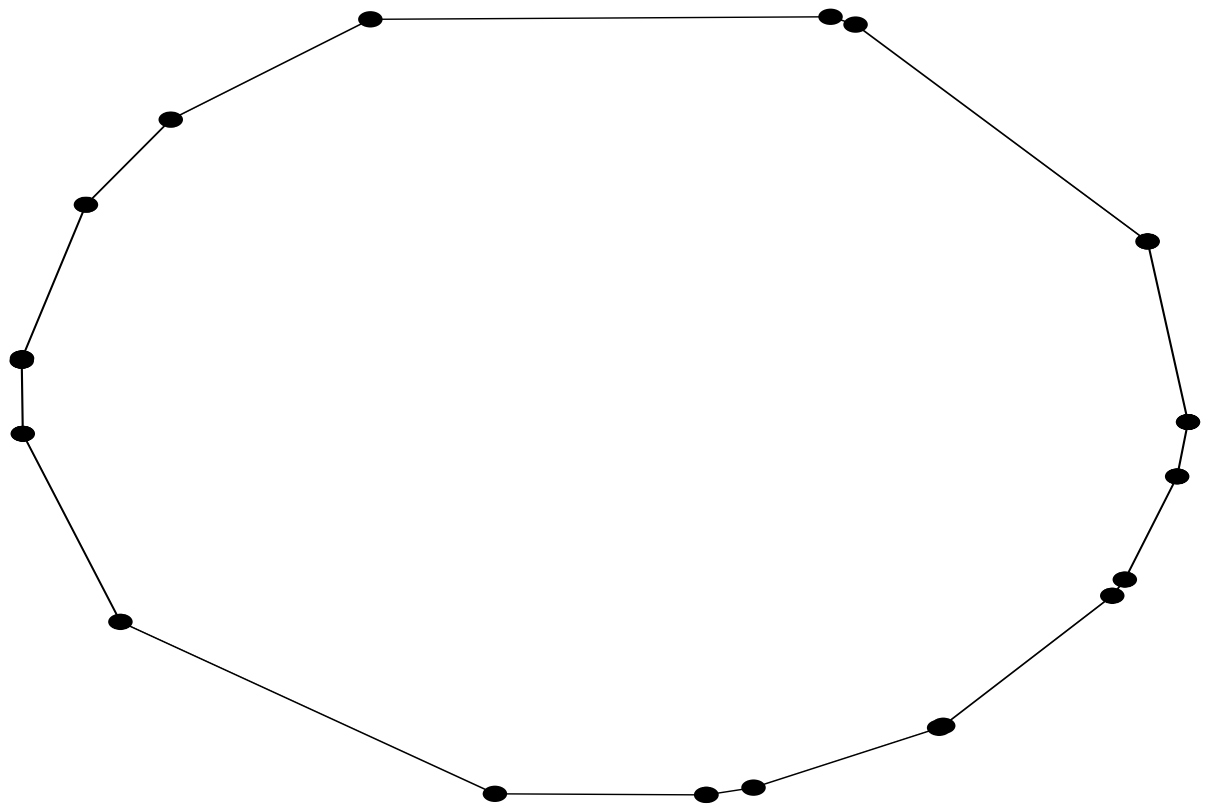


Known facts about 2DTSP

- In general, it is NP-hard.

(Garey, Graham & Johnson '76
Papadimitriou '77)

- When the points are in convex position, the problem is easy.

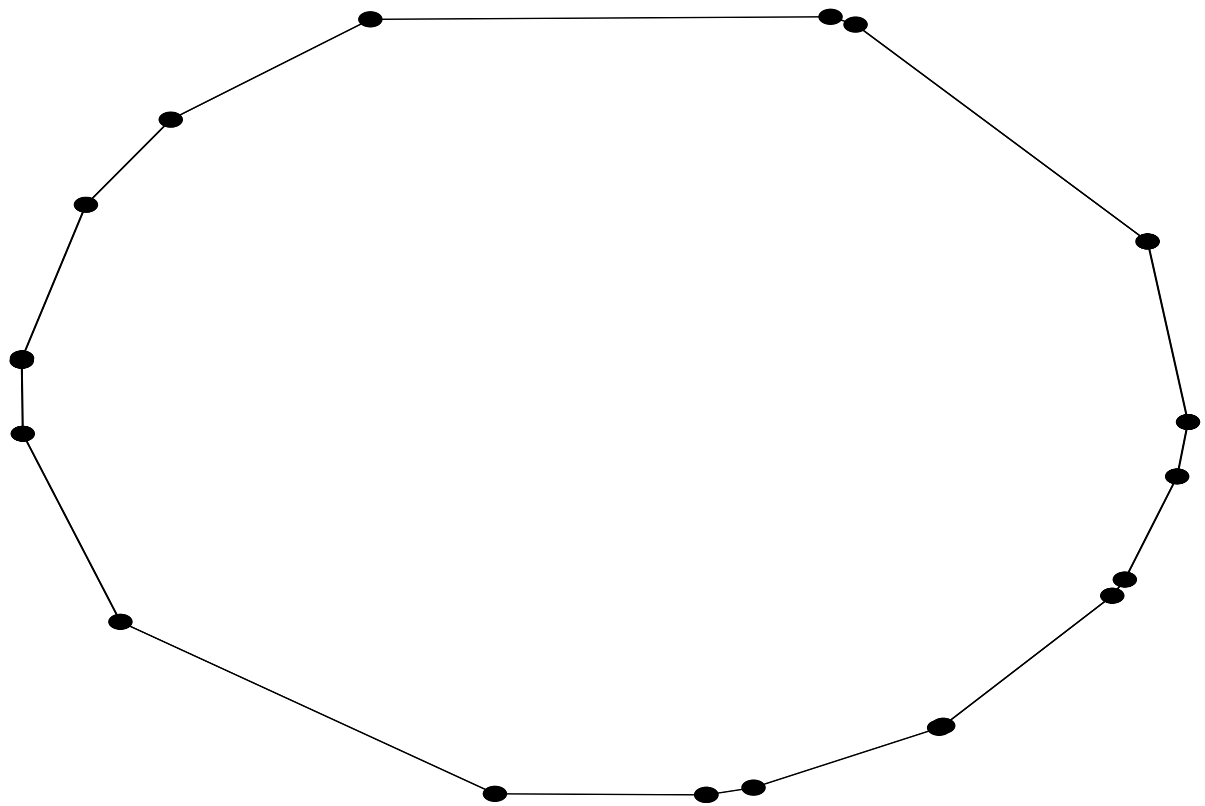


Known facts about 2DTSP

- In general, it is NP-hard.

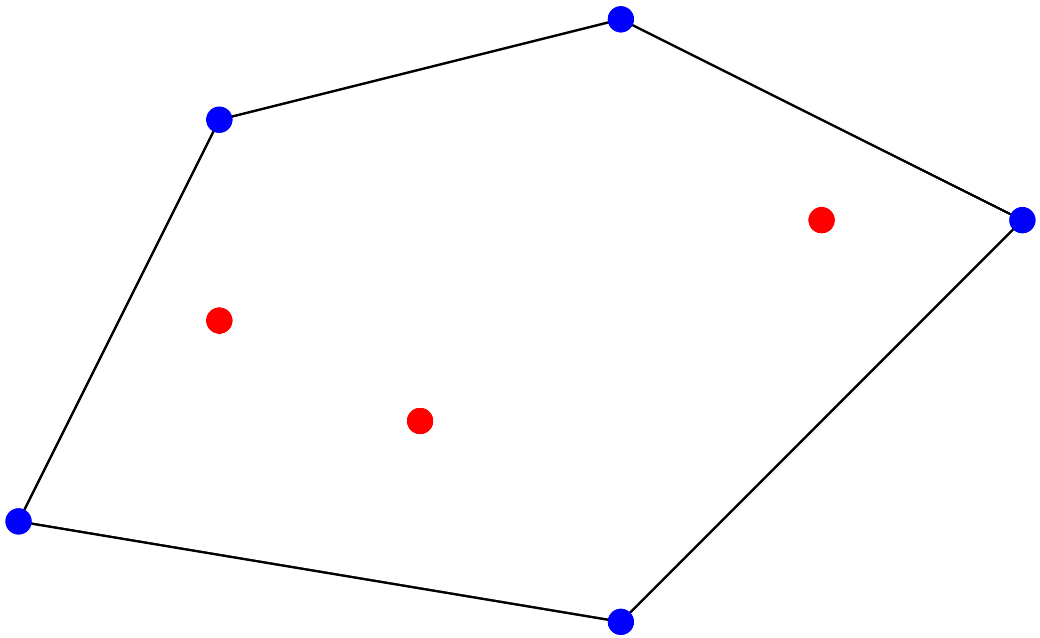
(Garey, Graham & Johnson '76
Papadimitriou '77)

- When the points are in convex position, the problem is easy.



Observation

The **inner points** make the problem difficult.



Observation

The **inner points** make the problem difficult.

Motivation

How many inner points can we have in order to obtain a polynomial-time algorithm?

Result

We give three simple algorithms.

n := the total number of points
k := the number of inner points

- First algorithm runs in polynomial time when $k = O(1)$.
- Second algorithm runs in polynomial time when $k = O(\log n / \log \log n)$.
- Third algorithm runs in polynomial time when $k = O(\log n)$.

Open problem: Improve the bound!

Result

We give three simple algorithms.

$n :=$ the total number of points
$k :=$ the number of inner points

- First algorithm runs in polynomial time when $k = O(1)$.
- Second algorithm runs in polynomial time when $k = O(\log n / \log\log n)$.
- Third algorithm runs in polynomial time when $k = O(\log n)$.

Open problem: Improve the bound!

Result

We give three simple algorithms.

n := the total number of points
k := the number of inner points

- First algorithm runs in polynomial time when $k = O(1)$.
- Second algorithm runs in polynomial time when $k = O(\log n / \log\log n)$.
- Third algorithm runs in polynomial time when $k = O(\log n)$.

Open problem: Improve the bound!

Result

We give three simple algorithms.

$n :=$ the total number of points
$k :=$ the number of inner points

- First algorithm runs in polynomial time when $k = O(1)$.
- Second algorithm runs in polynomial time when $k = O(\log n / \log\log n)$.
- Third algorithm runs in polynomial time when $k = O(\log n)$.

Open problem: Improve the bound!

Result

We give three simple algorithms.

n := the total number of points
k := the number of inner points

- First algorithm runs in polynomial time when $k = O(1)$.
- Second algorithm runs in polynomial time when $k = O(\log n / \log \log n)$.
- Third algorithm runs in polynomial time when $k = O(\log n)$.

Open problem: Improve the bound!

Result

We give three simple algorithms.

n := the total number of points
k := the number of inner points

- First algorithm runs in polynomial time when $k = O(1)$.
- Second algorithm runs in polynomial time when $k = O(\log n / \log\log n)$.
- Third algorithm runs in polynomial time when $k = O(\log n)$.

Open problem: Improve the bound!

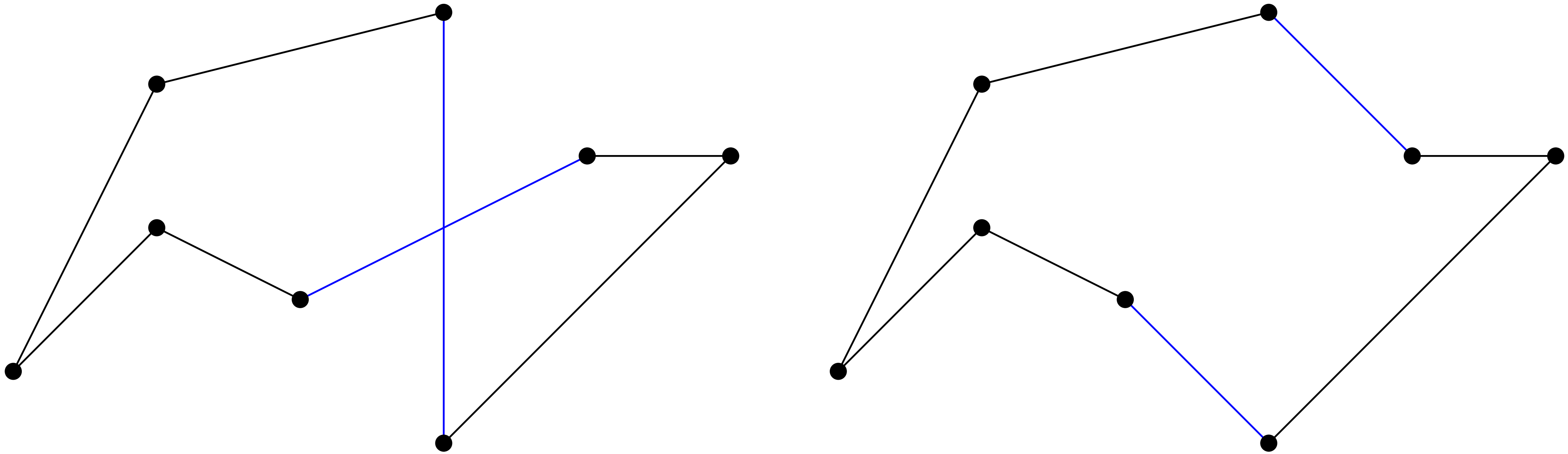
A useful fact

Fact

(Flood '56)

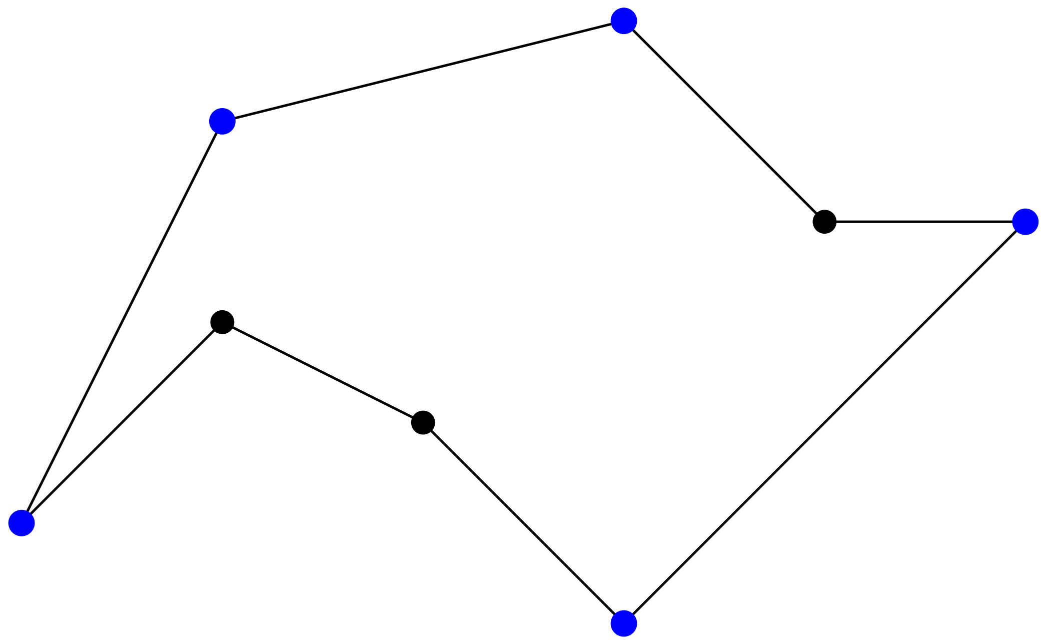
An optimal tour has no self-crossing.

Proof



Corollary

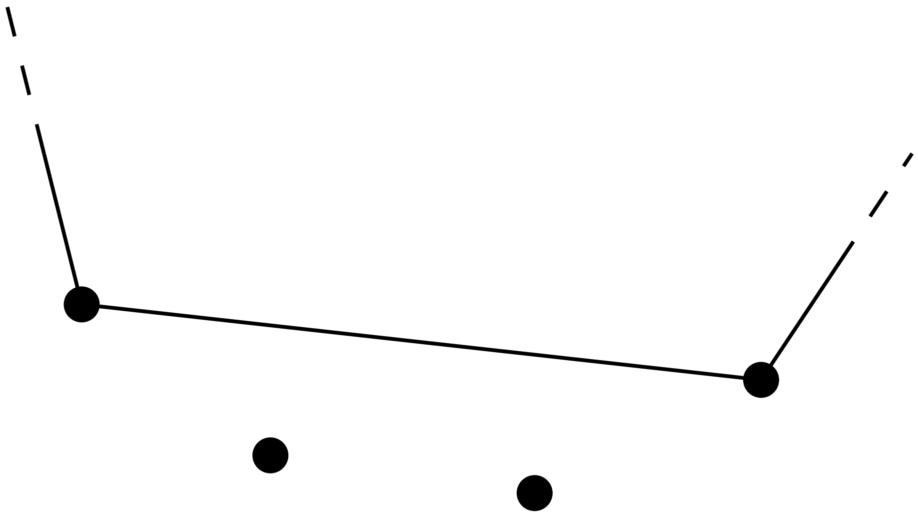
An optimal tour visits the non-inner points in a cyclic order.



Corollary

An optimal tour visits the non-inner points in a cyclic order.

Proof



Suppose not.
Then $\exists$ a "skip."
Skipped points must be visited later,
which causes a self-crossing.
A contradiction.

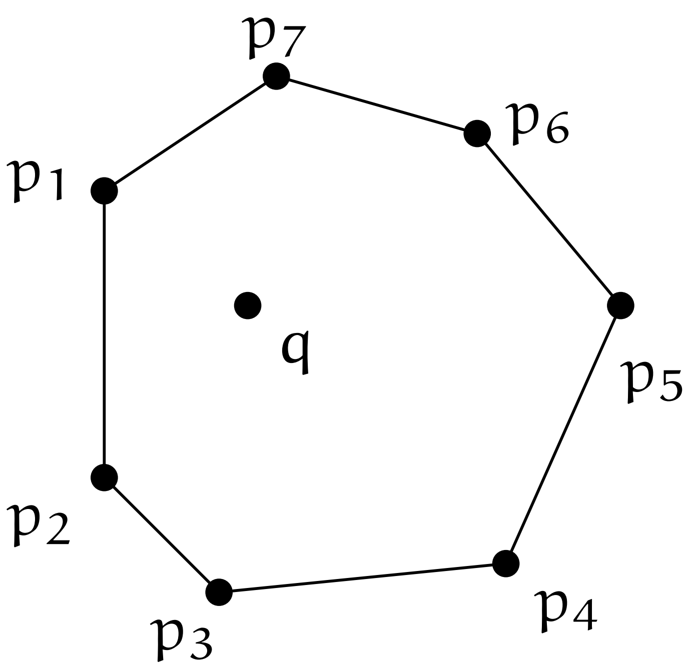
One inner point

Consider the case $k = 1$. ($k := \#$ of inner pts)

Inner point: q

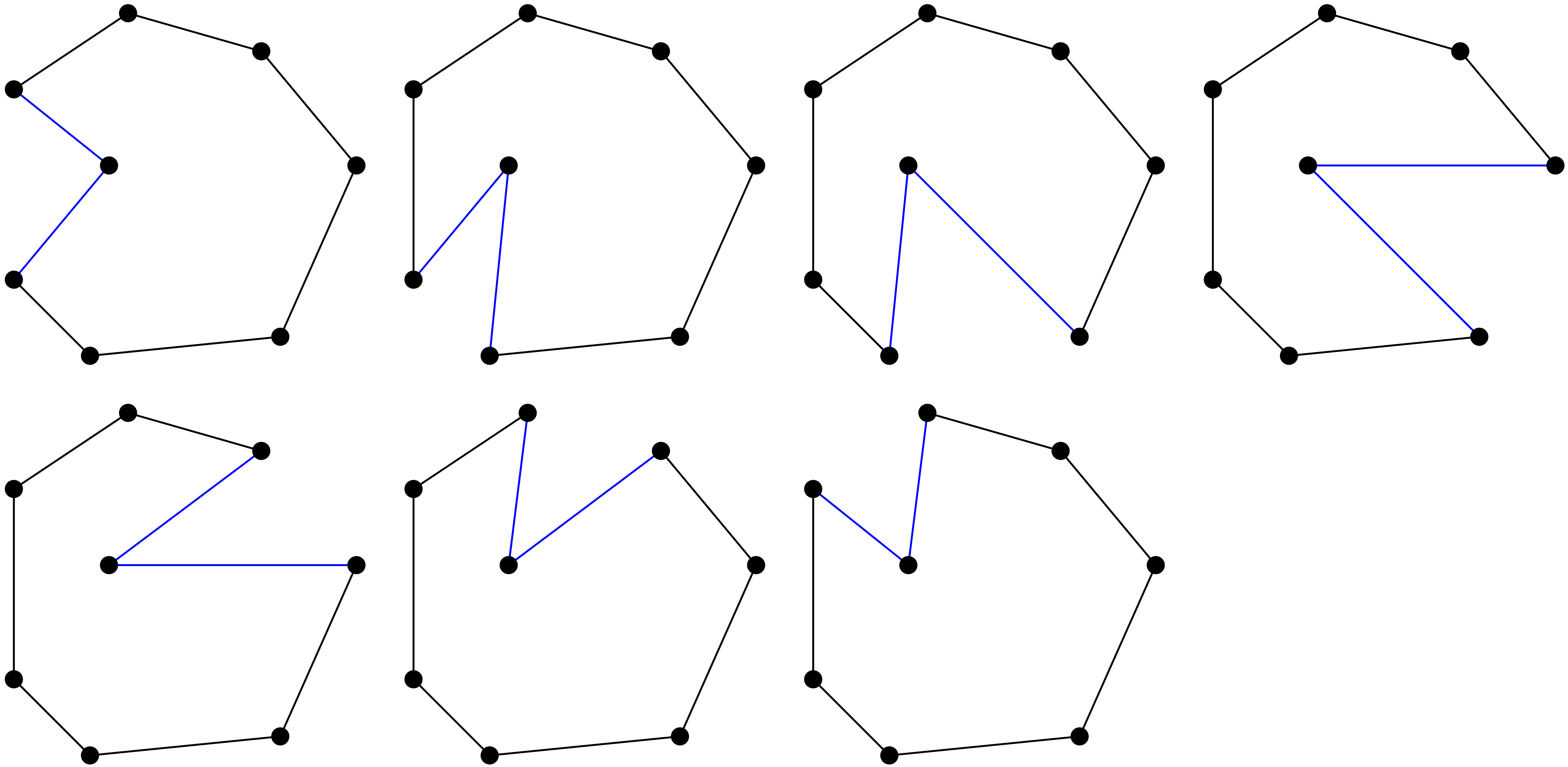
Non-inner points: $p_1, p_2, \ldots, p_{n-1}$

labeled according to a cyclic order



One inner point

of tours which "respect" the cycl. order $= n-1$.



Choose the best one.

First algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each tour which respects the cyclic order

  (a) Compute the length of the tour;

(4) Choose the best one among them.

First algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each tour which respects the cyclic order

  (a) Compute the length of the tour;

(4) Choose the best one among them.

First algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each tour which respects the cyclic order

   (a) Compute the length of the tour;

(4) Choose the best one among them.

First algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each tour which respects the cyclic order

  (a) Compute the length of the tour;

(4) Choose the best one among them.

First algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each tour which respects the cyclic order

   (a) Compute the length of the tour;

(4) Choose the best one among them.

First algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each tour which respects the cyclic order

  (a) Compute the length of the tour;

(4) Choose the best one among them.

Running time

There are k inner points.

- $\#$ of tours which "respect" the cycl. order $= O(k!n^k)$.
- They can be enumerated in $O(1)$ time per tour.
- The length of each tour can be computed in $O(n)$ time.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(k!n^{k+1})$.

When k is a constant, this is polynomial in n.

Running time

There are k inner points.

- $\#$ of tours which "respect" the cycl. order $= O(k!n^k)$.
- They can be enumerated in $O(1)$ time per tour.
- The length of each tour can be computed in $O(n)$ time.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(k!n^{k+1})$.

When k is a constant, this is polynomial in n.

Running time

There are k inner points.

- $\#$ of tours which "respect" the cycl. order $= O(k!n^k)$.
- They can be enumerated in $O(1)$ time per tour.
- The length of each tour can be computed in $O(n)$ time.

The running time $= \underbrace{O(n\log n)}_{\text{convex hull computation}} + O(k!n^{k+1})$.

When k is a constant, this is polynomial in n.

Running time

There are k inner points.

- # of tours which "respect" the cycl. order $= O(k!n^k)$.
- They can be enumerated in $O(1)$ time per tour.
- The length of each tour can be computed in $O(n)$ time.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(k!n^{k+1})$.

When k is a constant, this is polynomial in n.

Running time

There are k inner points.

- $\#$ of tours which "respect" the cycl. order $= O(k!n^k)$.
- They can be enumerated in $O(1)$ time per tour.
- The length of each tour can be computed in $O(n)$ time.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(k!n^{k+1})$.

When k is a constant, this is polynomial in n.

Running time

There are k inner points.

- \# of tours which "respect" the cycl. order $= O(k!n^k)$.
- They can be enumerated in $O(1)$ time per tour.
- The length of each tour can be computed in $O(n)$ time.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(k!n^{k+1})$.

When k is a constant, this is polynomial in n.

Result

We give three simple algorithms.

n := the total number of points
k := the number of inner points

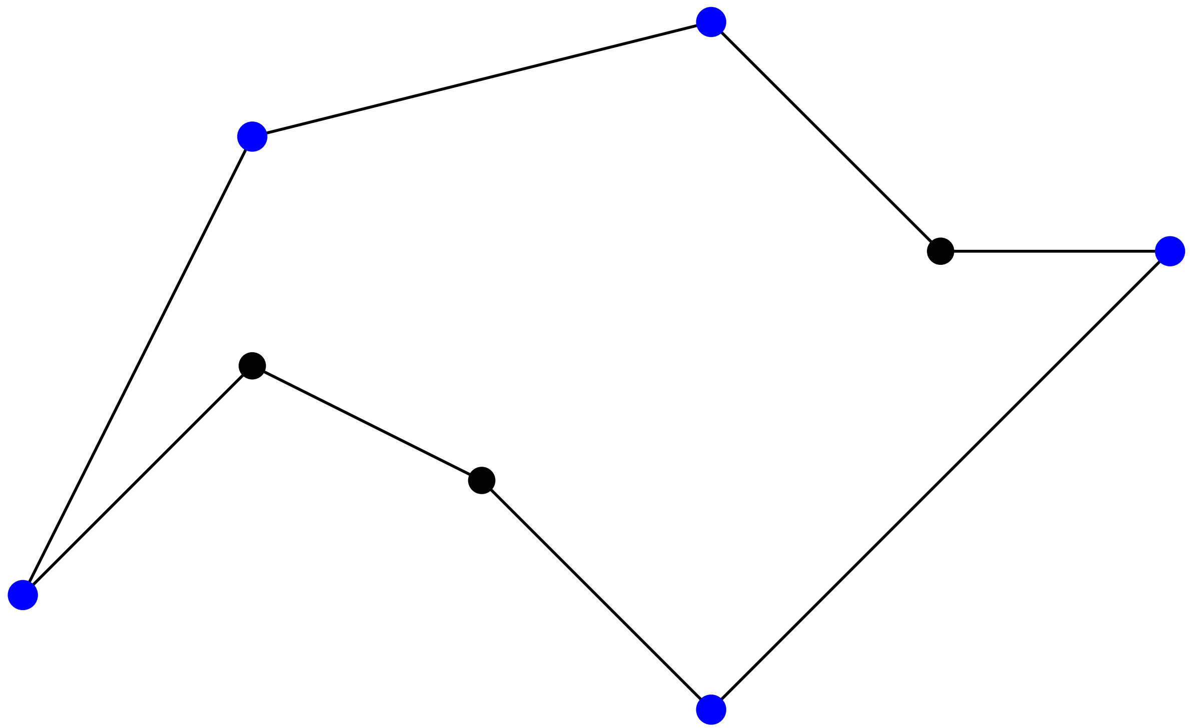
- First algorithm runs in polynomial time when $k = O(1)$.
- Second algorithm runs in polynomial time when $k = O(\log n / \log \log n)$.
- Third algorithm runs in polynomial time when $k = O(\log n)$.

Open problem: Improve the bound!

Idea for the second algorithm

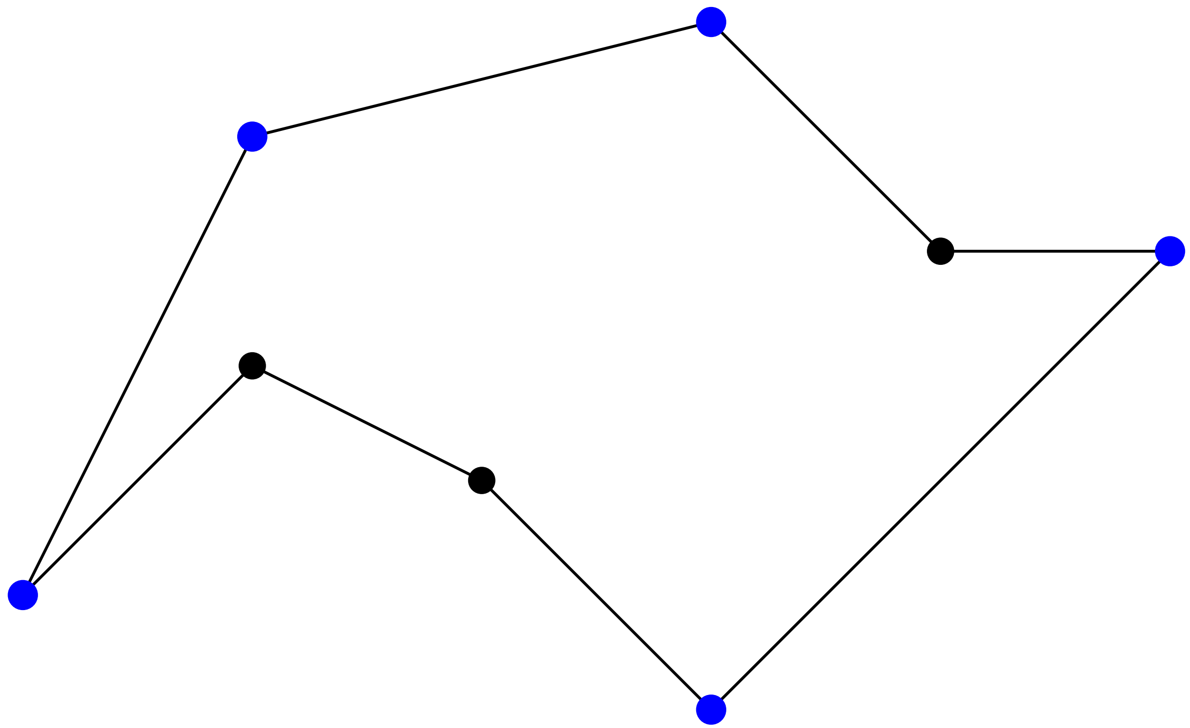
Fact we already saw

An optimal tour respects
a cyclic order on the non-inner points.



Another fact

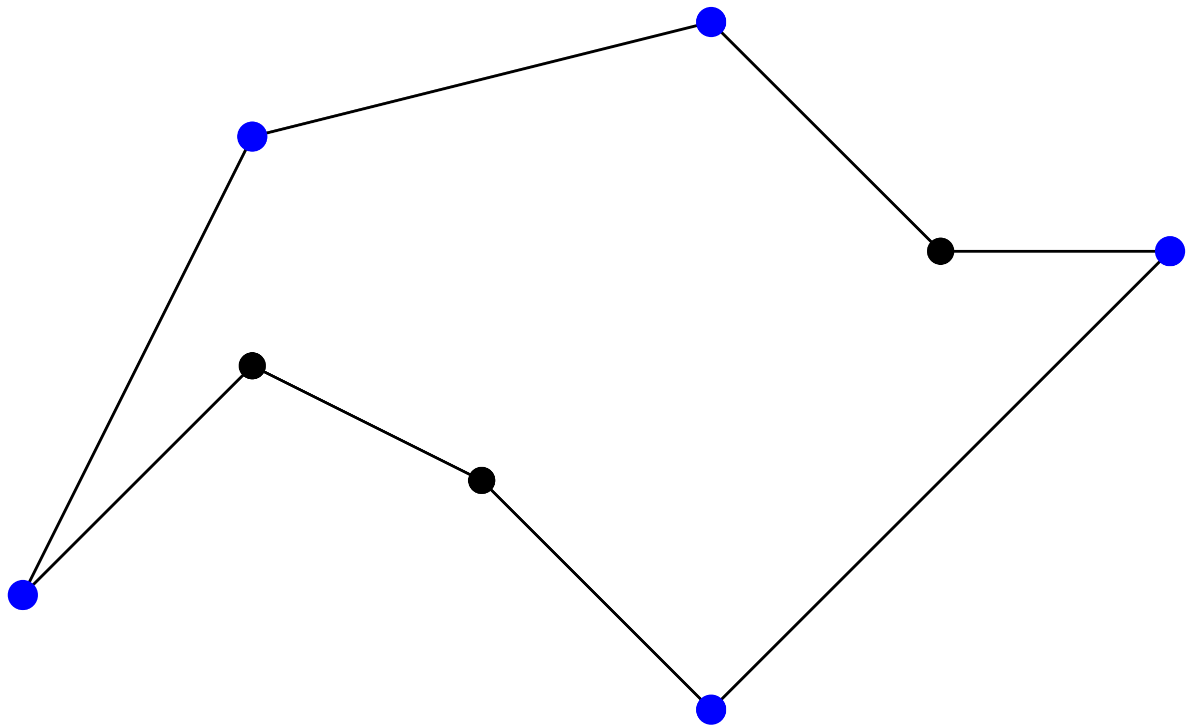
An optimal tour respects
some linear order on the inner points.



Idea for the second algorithm

Idea

Try all linear orders on the inner points.



Outline of the second algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each linear order on the inner points

(a) Compute an optimal tour among those which respect these two orders;

(4) Choose the best one among them.

Outline of the second algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each linear order on the inner points

   (a) Compute an optimal tour among those which respect these two orders;

(4) Choose the best one among them.

Outline of the second algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each linear order on the inner points

  (a) Compute an optimal tour among those which respect these two orders;

(4) Choose the best one among them.

Outline of the second algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each linear order on the inner points

(a) Compute an optimal tour among those which respect these two orders;

(4) Choose the best one among them.

Outline of the second algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each linear order on the inner points

  (a) Compute an optimal tour among those which respect these two orders;

(4) Choose the best one among them.

Outline of the second algorithm

(1) Distinguish the inner points and the non-inner points;

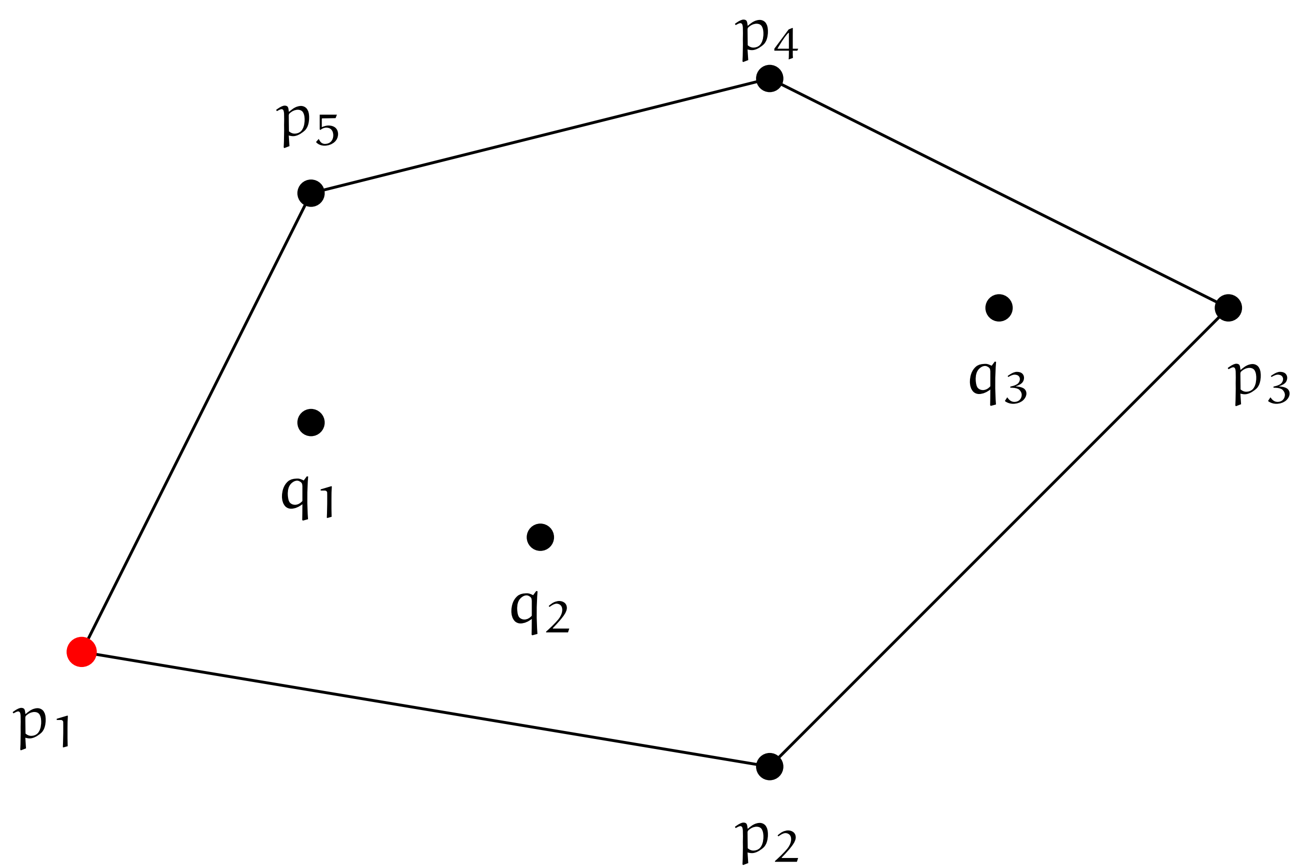
(2) Fix a cyclic order on the non-inner points;

(3) For each linear order on the inner points

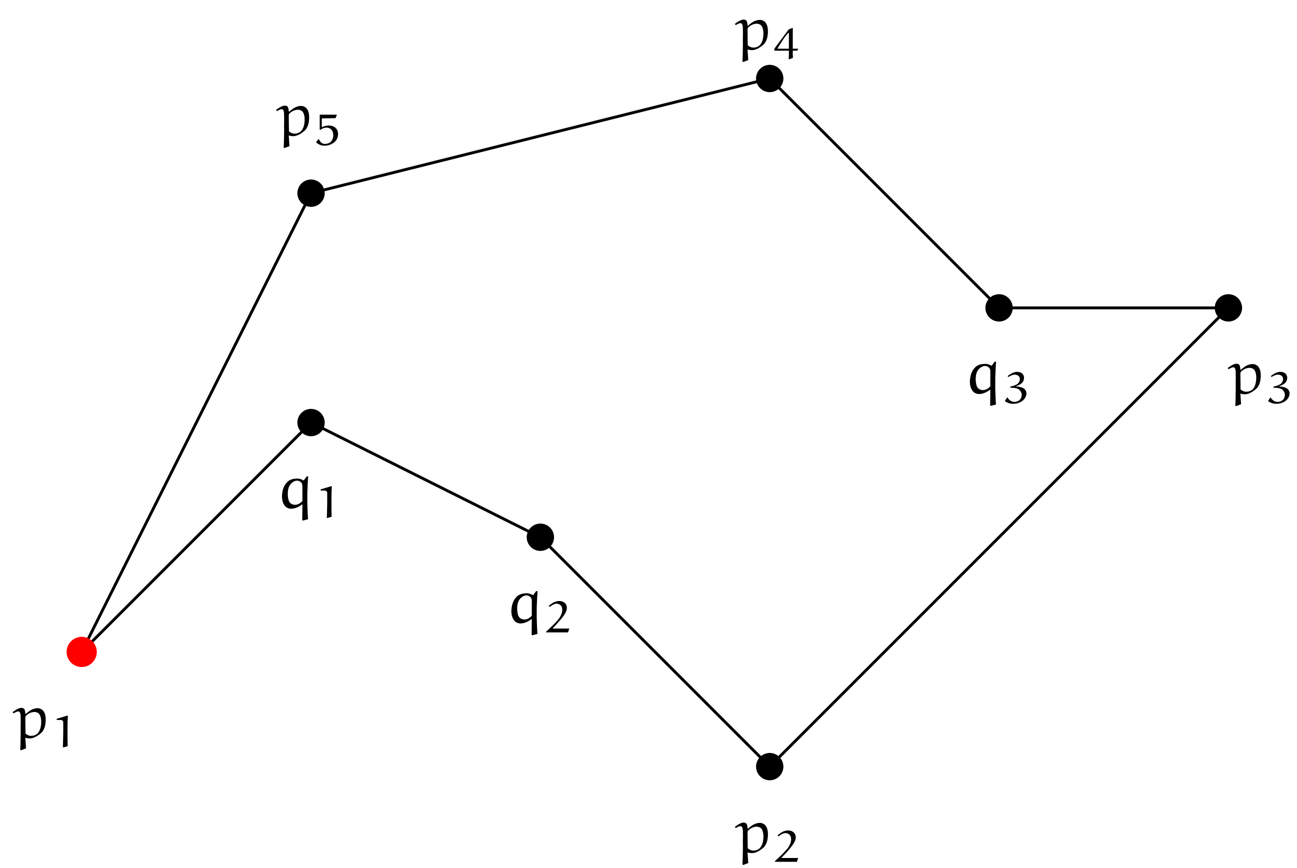
(a) Compute an optimal tour among those which respect these two orders;

(4) Choose the best one among them.

Example

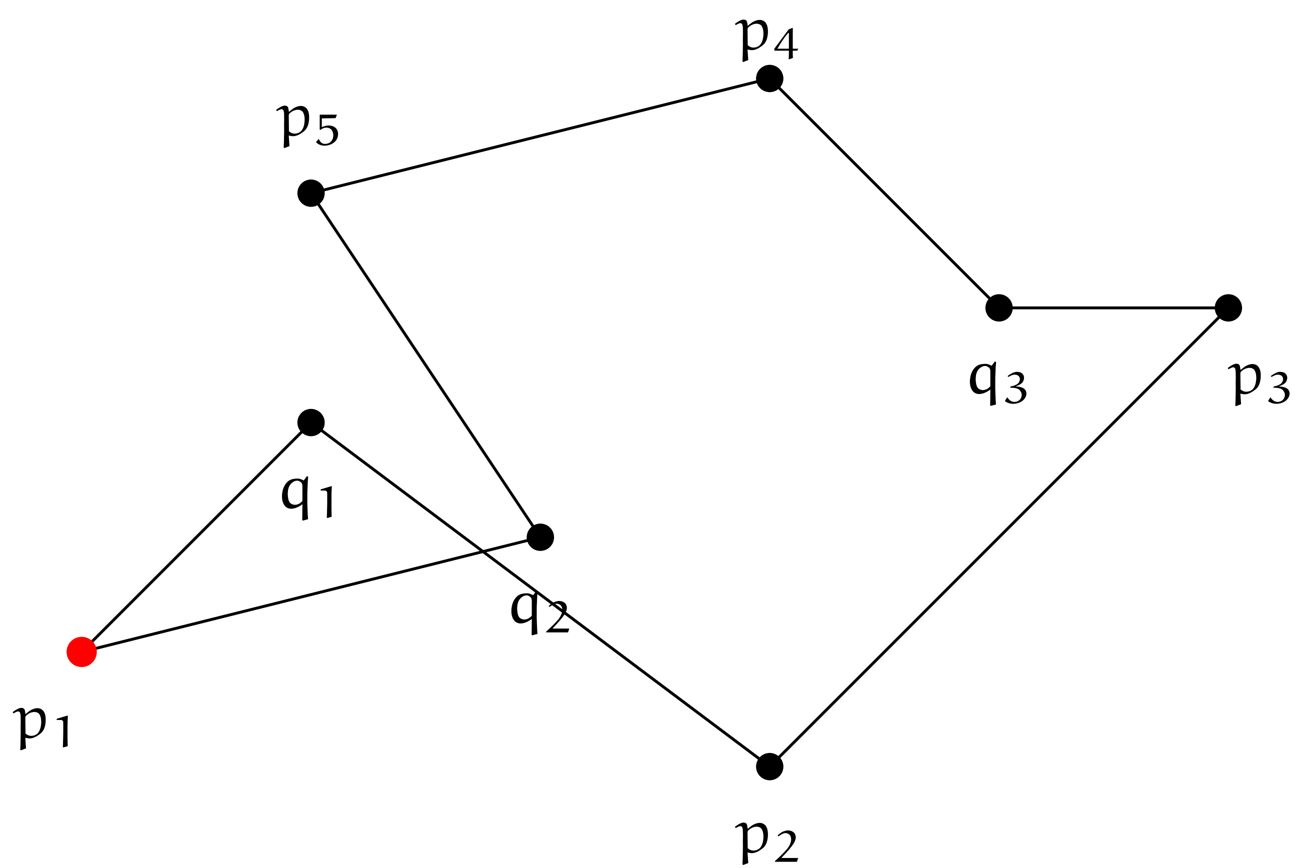


Example



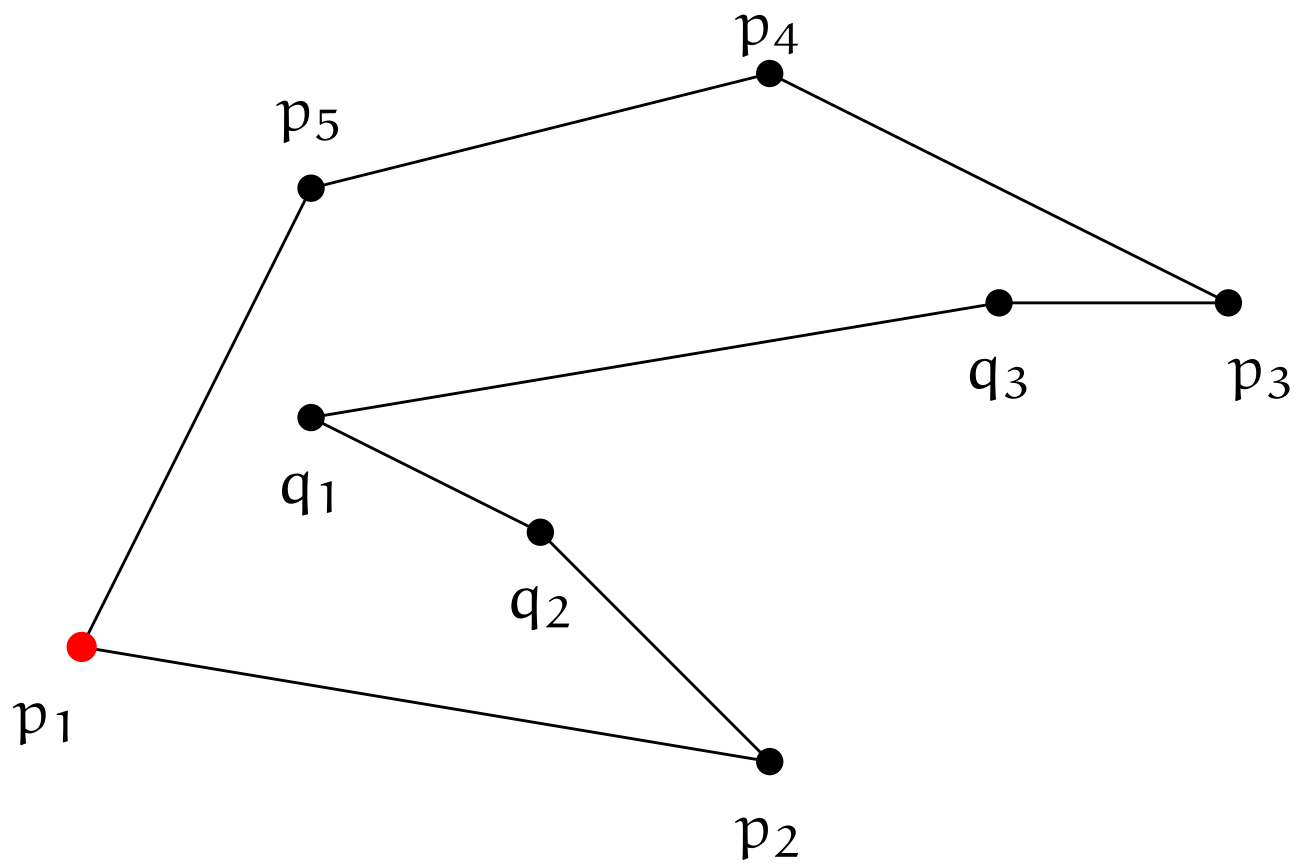
Optimal tour among those which respect the cyclic order and the order “1–2–3.”

Example



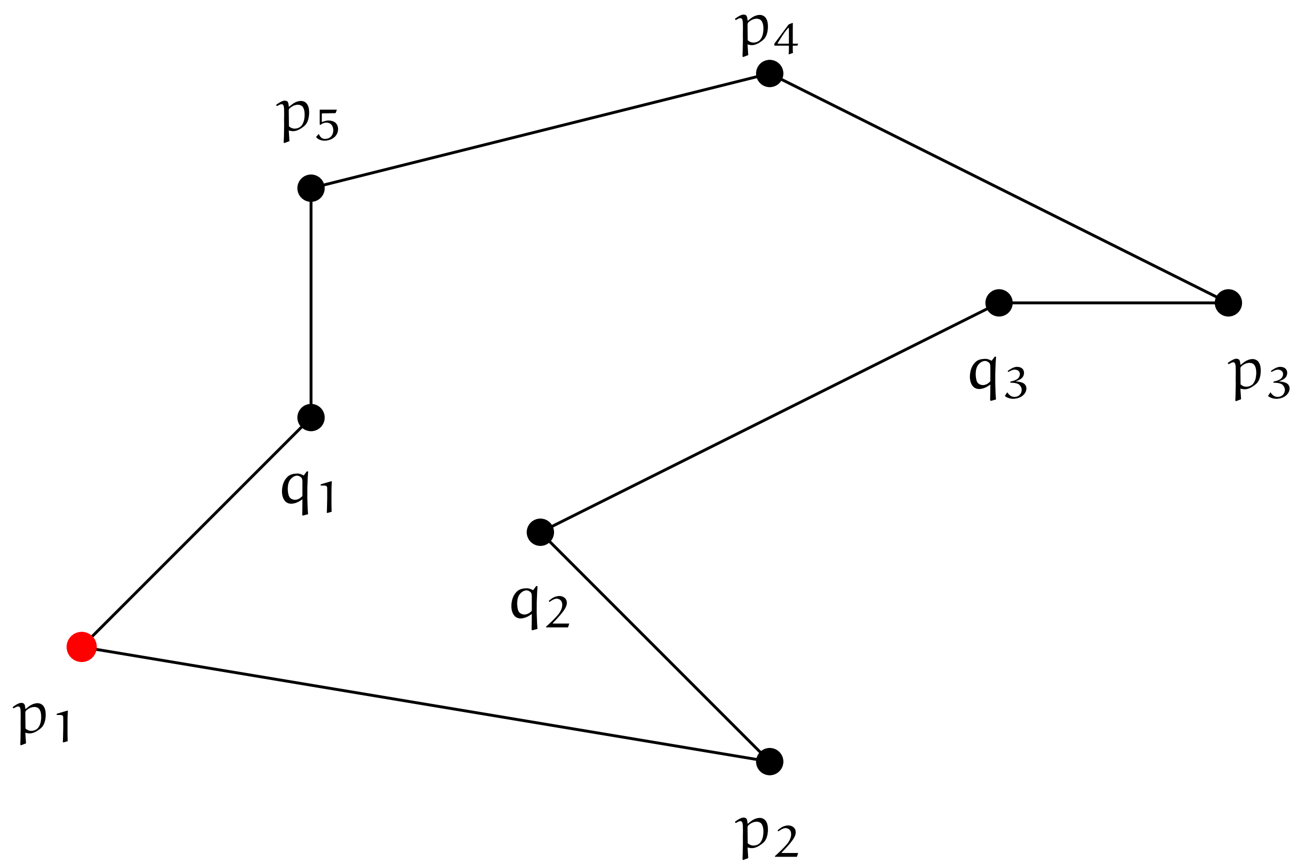
Optimal tour among those which respect the cyclic order and the order “1–3–2.”

Example



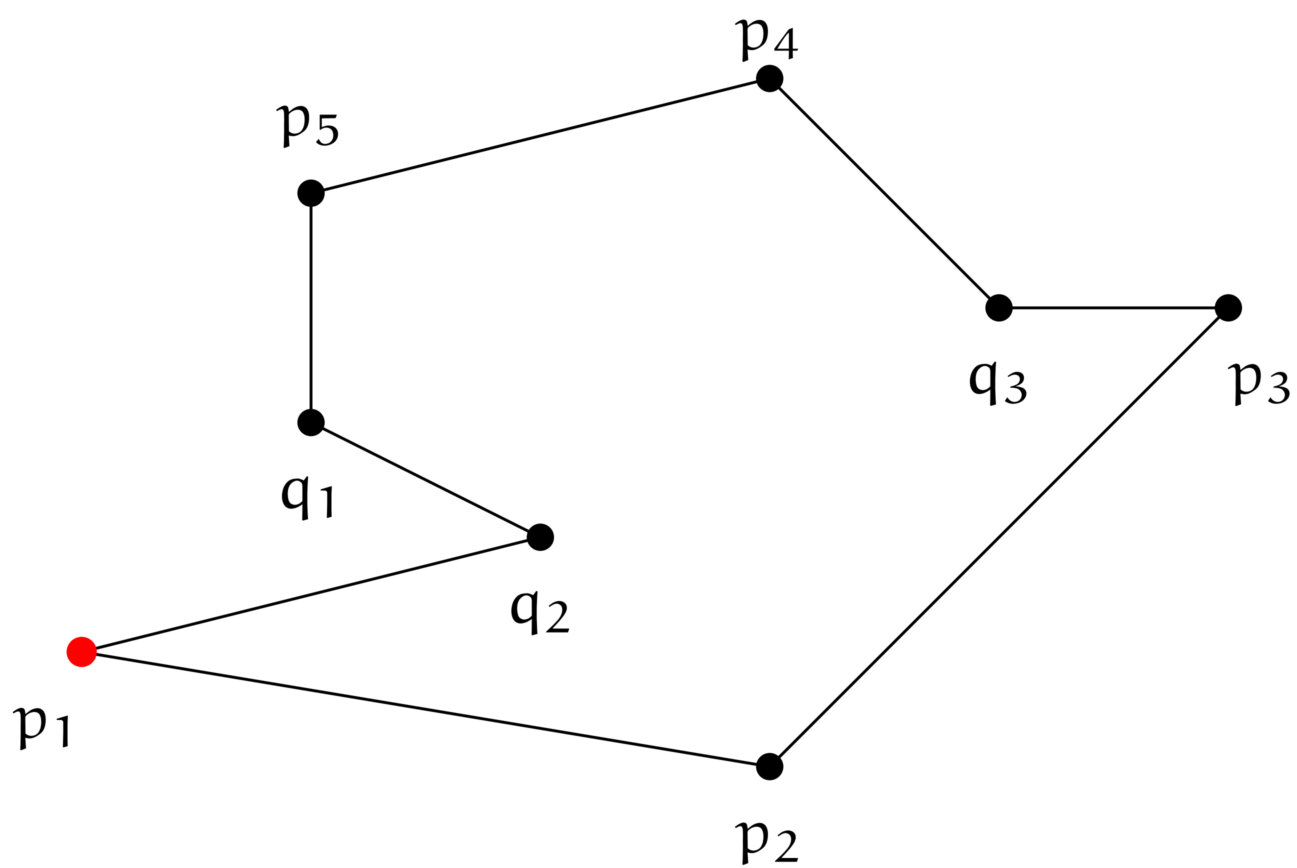
Optimal tour among those which respect the cyclic order and the order "2–1–3."

Example



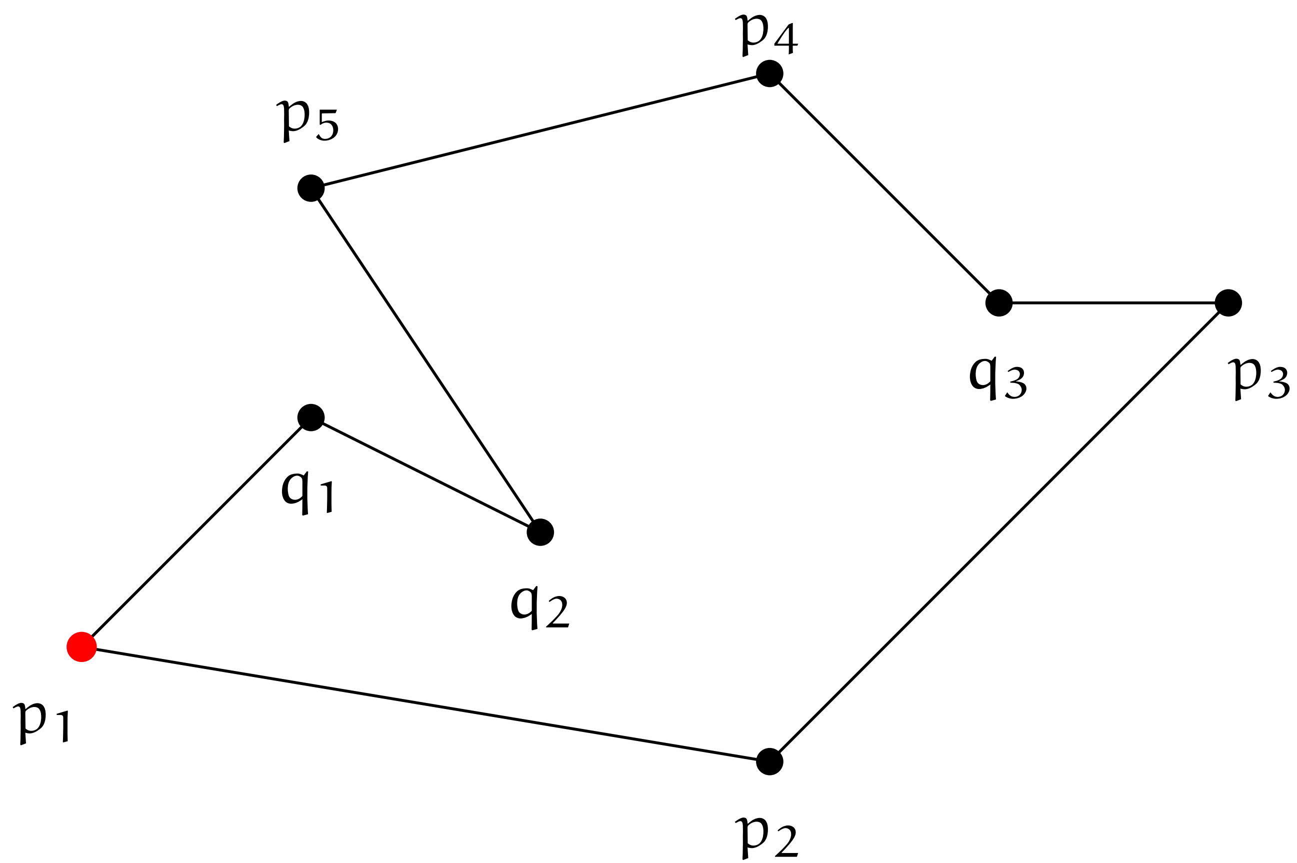
Optimal tour among those which respect the cyclic order and the order "2–3–1."

Example



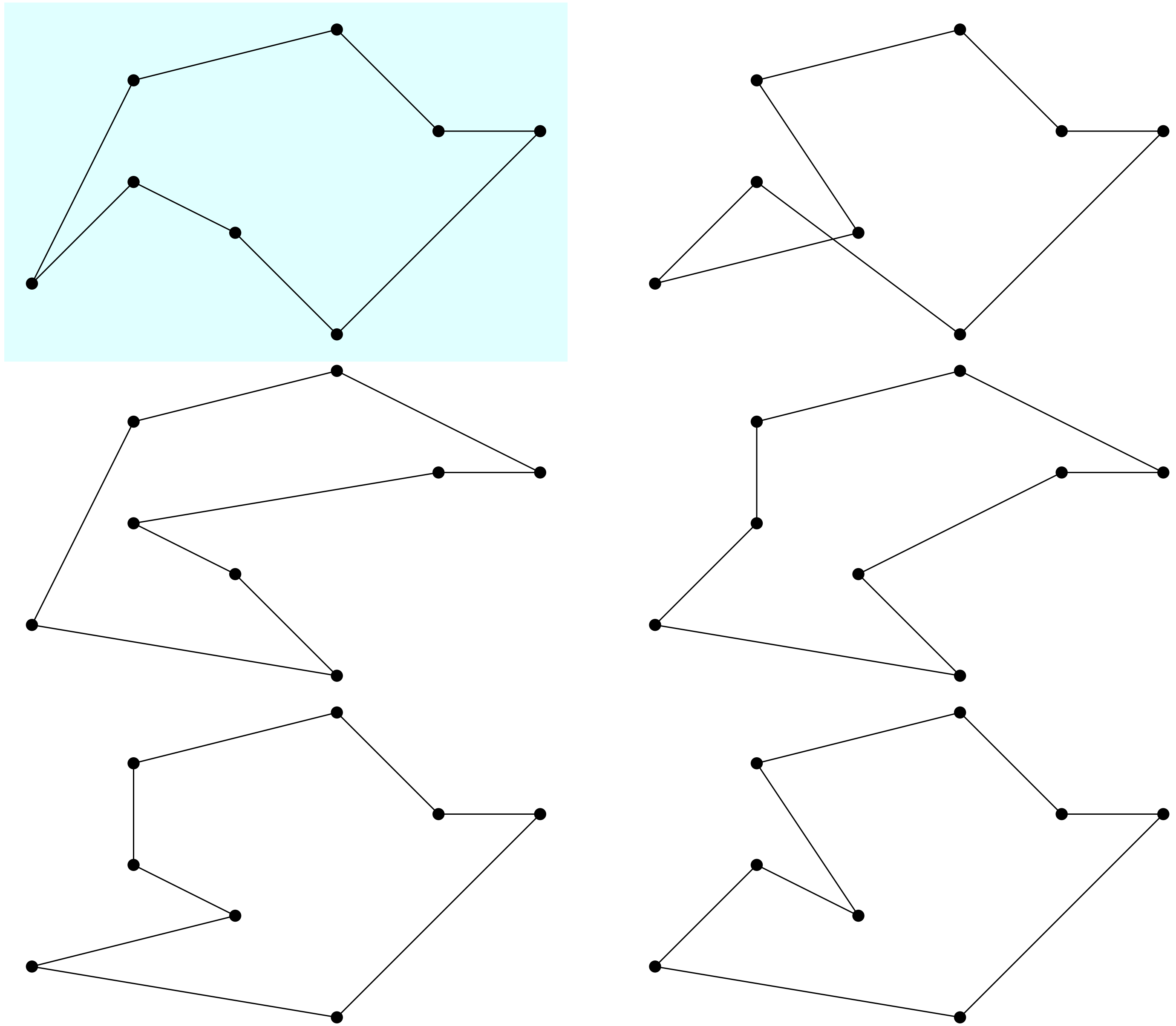
Optimal tour among those which respect the cyclic order and the order “3–1–2.”

Example



Optimal tour among those which respect the cyclic order and the order "3–2–1."

Choose the best one



Outline of the second algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each linear order on the inner points

   (a) Compute an optimal tour among those which respect these two orders;

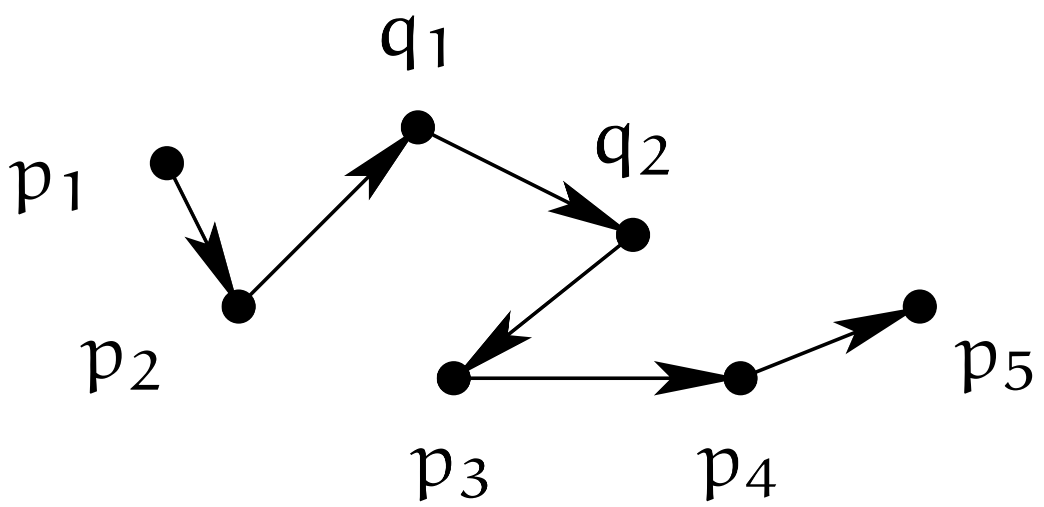
(4) Choose the best one among them.

Not yet clear: How to do Step (3a)??

$p_1, \ldots, p_{n-k}$ a cycl. order on the non-inner pts

$q_1, \ldots, q_k$ a linear order on the inner pts

$F(\underline{i}, j) :=$ the length of a shortest path from p_1 to p_i via $p_1, \ldots, p_i$ and $q_1, \ldots, q_j$ which respects these two orders



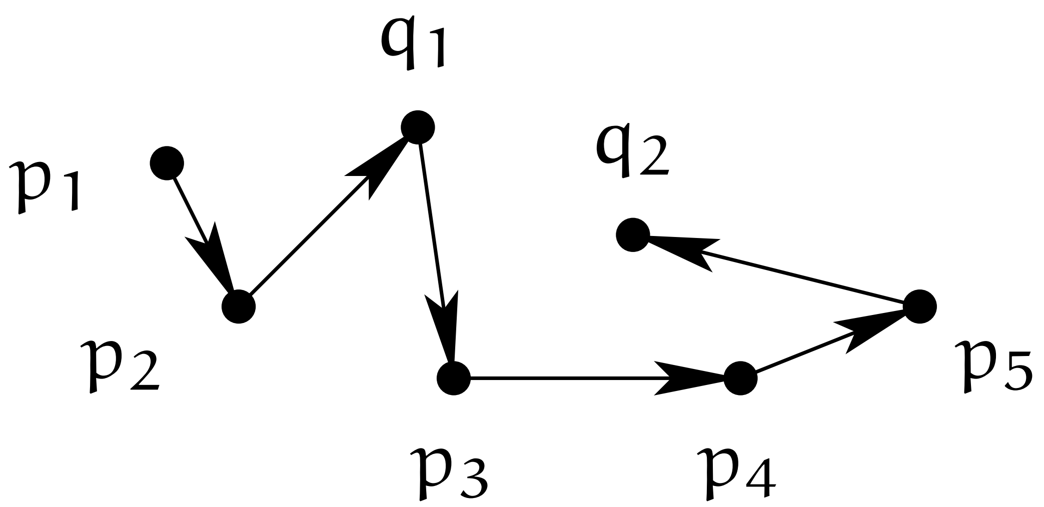
$(i = 5,\ j = 2)$

Dynamic programming

$p_1, \ldots, p_{n-k}$ a cycl. order on the non-inner pts

$q_1, \ldots, q_k$ a linear order on the inner pts

$F(i, \underline{j}) :=$ the length of a shortest path
from p_1 to q_j
via $p_1, \ldots, p_i$ and $q_1, \ldots, q_j$
which respects these two orders



$(i = 5,\ j = 2)$

- The length of a shortest tour which respects these two orders is

  the minimum of
  $F(\underline{n-k}, k) + d(p_{n-k}, p_1)$ and
  $F(n-k, \underline{k}) + d(q_k, p_1)$.

- By the dynamic programming technique,

  $F(\underline{n-k}, k)$ and $F(n-k, \underline{k})$ can be computed in $O(kn)$ time.

Dynamic programming

- The length of a shortest tour which respects these two orders is

  the minimum of
  $F(\underline{n-k}, k) + d(p_{n-k}, p_1)$ and
  $F(n-k, \underline{k}) + d(q_k, p_1)$.

- By the dynamic programming technique,

  $F(\underline{n-k}, k)$ and $F(n-k, \underline{k})$ can be computed in $O(kn)$ time.

Outline of the second algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) For each linear order on the inner points

   (a) Compute an optimal tour among those which respect these two orders;

(4) Choose the best one among them.

What remains: the analysis of the running time

Running time

There are k inner points.

- $\#$ of linear orders on k points $= k!$.
- They can be enumerated in $O(1)$ time per order.
- The length of an optimal tour which respects the two orders can be computed in $O(kn)$ time.

$$\text{The running time} = \underbrace{O(n \log n)}_{\substack{\text{convex hull}\\ \text{computation}}} + O(k!kn).$$

When $k = O(\log n / \log \log n)$, this is poly. in n.

Running time

There are k inner points.

- $\#$ of linear orders on k points $= k!$.
- They can be enumerated in $O(1)$ time per order.
- The length of an optimal tour which respects the two orders can be computed in $O(kn)$ time.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(k!kn)$.

When $k = O(\log n / \log \log n)$, this is poly. in n.

Running time

There are k inner points.

- $\#$ of linear orders on k points $= k!$.
- They can be enumerated in $O(1)$ time per order.
- The length of an optimal tour which respects the two orders can be computed in $O(kn)$ time.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(k!kn)$.

When $k = O(\log n / \log \log n)$, this is poly. in n.

Running time

There are k inner points.

- $\#$ of linear orders on k points $= k!$.
- They can be enumerated in $O(1)$ time per order.
- The length of an optimal tour which respects the two orders can be computed in $O(kn)$ time.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(k!kn)$.

When $k = O(\log n / \log \log n)$, this is poly. in n.

Running time

There are k inner points.

- $\#$ of linear orders on k points $= k!$.
- They can be enumerated in $O(1)$ time per order.
- The length of an optimal tour which respects the two orders can be computed in $O(kn)$ time.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(k!kn)$.

When $k = O(\log n / \log \log n)$, this is poly. in n.

Running time

There are k inner points.

- $\#$ of linear orders on k points $= k!$.
- They can be enumerated in $O(1)$ time per order.
- The length of an optimal tour which respects the two orders can be computed in $O(kn)$ time.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(k!kn)$.

When $k = O(\log n / \log \log n)$, this is poly. in n.

Result

We give three simple algorithms.

$n :=$ the total number of points
$k :=$ the number of inner points

- First algorithm runs in polynomial time when $k = O(1)$.
- Second algorithm runs in polynomial time when $k = O(\log n / \log \log n)$.
- Third algorithm runs in polynomial time when $k = O(\log n)$.

Open problem: Improve the bound!

Dynamic programming across the subsets

Technique: **"Dynamic Programming across the Subsets"**
(Held & Karp '62)

- The traveling salesman problem
  - Trivial: $n!$
  - DPatS: 2^n (Held & Karp '62, Bellman '62)
- Total completion time scheduling under prec. constraints
  - Trivial: $n!$
  - DPatS: 2^n (Held & Karp '62)
- Chromatic number of a graph
  - Trivial: Bell number B_n (# of partitions)
  - DPatS: 2.4422^n (Lawler '76)

Nice survey: Woeginger '03

Dynamic programming across the subsets

Technique: **"Dynamic Programming across the Subsets"**
(Held & Karp '62)

- The traveling salesman problem
  - Trivial: $n!$
  - DPatS: 2^n (Held & Karp '62, Bellman '62)
- Total completion time scheduling under prec. constraints
  - Trivial: $n!$
  - DPatS: 2^n (Held & Karp '62)
- Chromatic number of a graph
  - Trivial: Bell number B_n (# of partitions)
  - DPatS: 2.4422^n (Lawler '76)

Nice survey: Woeginger '03

Technique: **“Dynamic Programming across the Subsets”**
(Held & Karp ’62)

- The traveling salesman problem
  - Trivial: $n!$
  - DPatS: 2^n (Held & Karp ’62, Bellman ’62)
- Total completion time scheduling under prec. constraints
  - Trivial: $n!$
  - DPatS: 2^n (Held & Karp ’62)
- Chromatic number of a graph
  - Trivial: Bell number B_n (# of partitions)
  - DPatS: 2.4422^n (Lawler ’76)

Nice survey: Woeginger ’03

Dynamic programming across the subsets

Technique: **"Dynamic Programming across the Subsets"**
(Held & Karp '62)

- The traveling salesman problem
  - Trivial: $n!$
  - DPatS: 2^n (Held & Karp '62, Bellman '62)
- Total completion time scheduling under prec. constraints
  - Trivial: $n!$
  - DPatS: 2^n (Held & Karp '62)
- Chromatic number of a graph
  - Trivial: Bell number B_n (# of partitions)
  - DPatS: 2.4422^n (Lawler '76)

Nice survey: Woeginger '03

Dynamic programming across the subsets

Technique: **"Dynamic Programming across the Subsets"**
(Held & Karp '62)

- The traveling salesman problem
  - Trivial: $n!$
  - DPatS: 2^n (Held & Karp '62, Bellman '62)
- Total completion time scheduling under prec. constraints
  - Trivial: $n!$
  - DPatS: 2^n (Held & Karp '62)
- Chromatic number of a graph
  - Trivial: Bell number B_n (# of partitions)
  - DPatS: 2.4422^n (Lawler '76)

Nice survey: Woeginger '03

Basic scheme for the third algorithm

- Second algorithm
  - Perform a DP $k!$ times.
  - Each DP takes $O(kn)$ time.
- Third algorithm
  - Perform a DP once.
  - Each DP takes $O(2^k k^2 n)$ time.

Basic scheme for the third algorithm

- Second algorithm
  - Perform a DP $k!$ times.
  - Each DP takes $O(kn)$ time.
- Third algorithm
  - Perform a DP once.
  - Each DP takes $O(2^k k^2 n)$ time.

Basic scheme for the third algorithm

- Second algorithm
  - Perform a DP $k!$ times.
  - Each DP takes $O(kn)$ time.
- Third algorithm
  - Perform a DP once.
  - Each DP takes $O(2^k k^2 n)$ time.

Basic scheme for the third algorithm

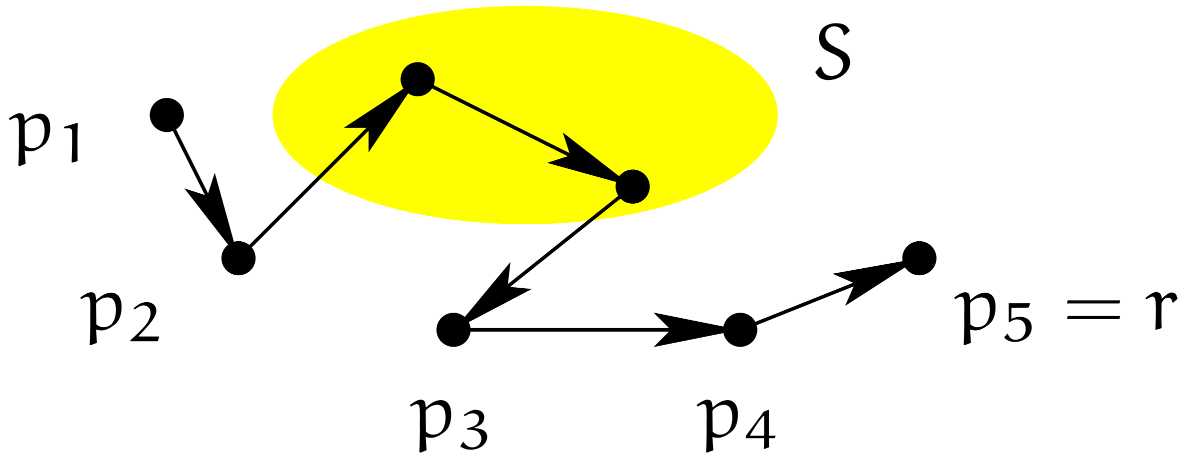
- Second algorithm
  - Perform a DP $k!$ times. HEAVY
  - Each DP takes $O(kn)$ time. LIGHT
- Third algorithm
  - Perform a DP once. LIGHT
  - Each DP takes $O(2^k k^2 n)$ time. HEAVY

DPatS for our problem

$p_1, \ldots, p_{n-k}$	a cycl. order on the non-inner pts
Q	the set of inner pts

For $i \in \{1, \ldots, n{-}k\}, S \subseteq Q, r \in S \cup \{p_i\}$,

$F(i, S, r) :=$ the length of a shortest path from p_1 to r via $p_1, \ldots, p_i$ and the pts in S which respects the cycl. order



$(i = 5,\ |S| = 2, r = p_i)$

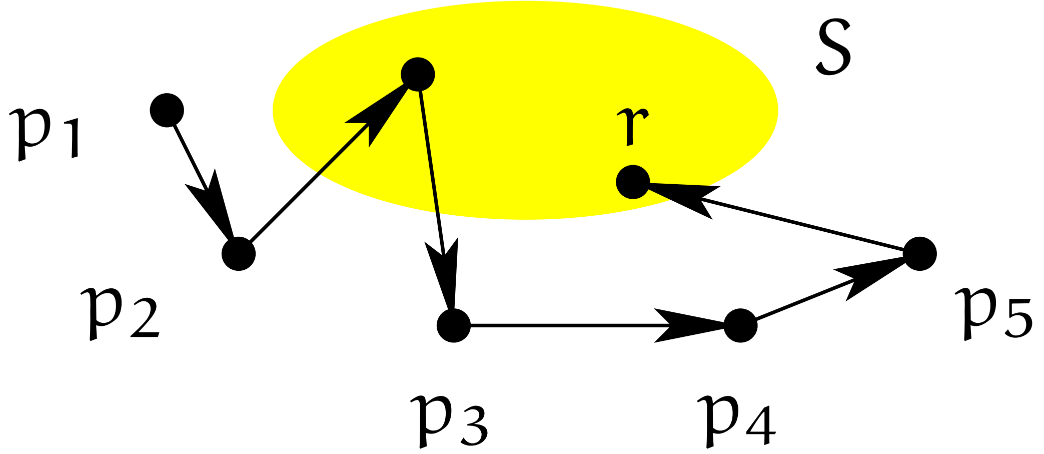
DPatS for our problem

$p_1, \ldots, p_{n-k}$ a cycl. order on the non-inner pts

Q the set of inner pts

For $i \in \{1, \ldots, n-k\}, S \subseteq Q, r \in S \cup \{p_i\}$,

$F(i, S, r) :=$ the length of a shortest path from p_1 to r via $p_1, \ldots, p_i$ and the pts in S which respects the cycl. order



$(i = 5,\ |S| = 2, r \in S)$

- The length of a shortest tour which respects the cycl. order is

  the minimum of $F(n-k, Q, r) + d(r, p_1)$
  among all $r \in Q \cup \{p_{n-k}\}$.

- By the dynamic programming technique,

  these values can be computed in $O(2^k k^2 n)$ time.

Dynamic Programming

- The length of a shortest tour
  which respects the cycl. order is
  the minimum of $F(n-k, Q, r) + d(r, p_1)$
  among all $r \in Q \cup \{p_{n-k}\}$.
- By the dynamic programming technique,
  these values can be computed in $O(2^k k^2 n)$ time.

Outline of the third algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) Perform the DPatS above.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(2^k k^2 n)$.

When $k = O(\log n)$, this is polynomial in n.

Outline of the third algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) Perform the DPatS above.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(2^k k^2 n)$.

When $k = O(\log n)$, this is polynomial in n.

Outline of the third algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) Perform the DPatS above.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(2^k k^2 n)$.

When $k = O(\log n)$, this is polynomial in n.

Outline of the third algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) Perform the DPatS above.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(2^k k^2 n)$.

When $k = O(\log n)$, this is polynomial in n.

Outline of the third algorithm

(1) Distinguish the inner points and the non-inner points;

(2) Fix a cyclic order on the non-inner points;

(3) Perform the DPatS above.

The running time $= \underbrace{O(n \log n)}_{\text{convex hull computation}} + O(2^k k^2 n)$.

When $k = O(\log n)$, this is polynomial in n.

Result

We gave three simple algorithms.

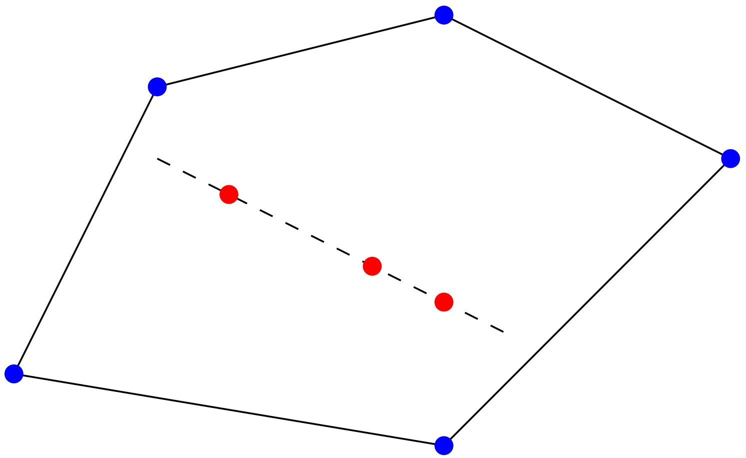
$n :=$ the total number of points
$k :=$ the number of inner points

	Time	Space	PTIME if $k =$
First	$O(k!n^{k+1})$		$O(1)$
Second	$O(k!kn)$	$O(k)$	$O(\log n / \log\log n)$
Third	$O(2^k k^2 n)$	$O(2^k kn)$	$O(\log n)$

Related work

Thm (Deĭneko, van Dal & Rote '96)

The convex-hull-and-line TSP can be solved in $O(kn)$ time.

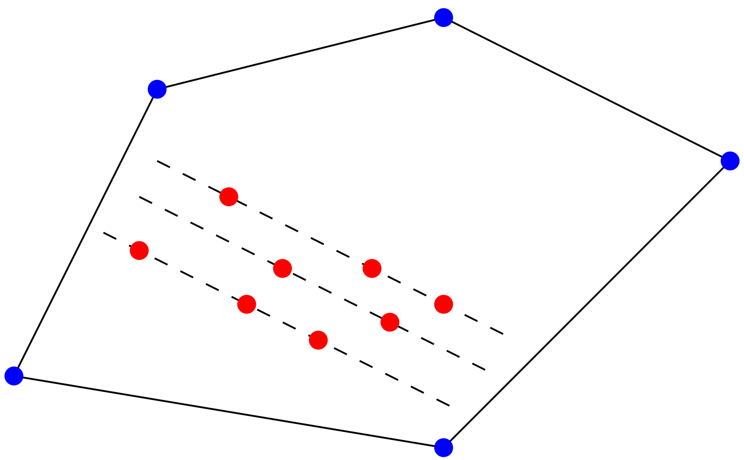


Our work $\left\{ \begin{array}{l} \text{deals with the most general case.} \\ \text{still runs in linear time in } n. \end{array} \right.$

Related work

Thm (Deĭneko & Woeginger '96)

The convex-hull-and-ℓ-line TSP can be solved in $O(f(k, \ell)n^2)$ time for some fn f.



Our work $\left\{ \begin{array}{l} \text{deals with the most general case.} \\ \text{still runs in linear time in } n. \end{array} \right.$

The same strategy works for other problems.

- The prize-collecting TSP
- The partial TSP

Result

The 2D versions of these problems
with k inner points
can be solved in polynomial time
when $k = O(\log n)$.

Many problems can be solved in poly time when some parameters are bounded.

- Graph optimization problems
  - bounded treewidth
  - bounded genus
  - ...

Many problems can be solved in poly time when some parameters are bounded.

- Graph optimization problems
  - bounded treewidth
  - bounded genus
  - ...
- Geometric optimization problems in 2D
  - bounded number of inner points

Many problems can be solved in poly time when some parameters are bounded.

- Graph optimization problems
  - bounded treewidth
  - bounded genus
  - ...
- Geometric optimization problems in 2D
  - bounded number of inner points
  - ...?

General framework

Many problems can be solved in poly time when some parameters are bounded.

- Graph optimization problems
  - bounded treewidth
  - bounded genus
  - ...
- Geometric optimization problems in 2D
  - bounded number of inner points
  - ...?

$\Rightarrow$ **Distance-from-Triviality approach**

(Guo, Hüffner, Niedermeier IWPEC '04
Niedermeier MFCS '04)

Thank you

감사합니다.