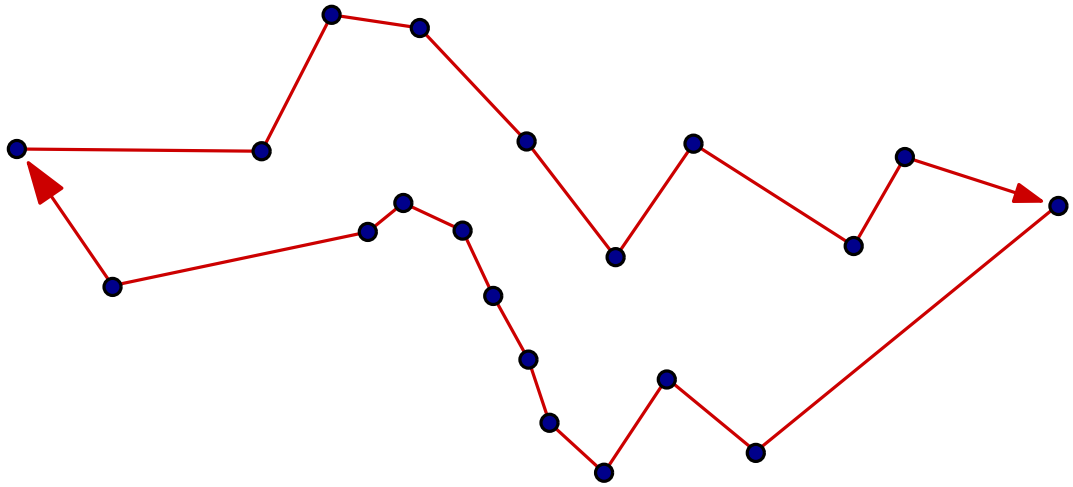
# Fine-Grained Complexity Analysis of Two Classic TSP Variants



Mark de Berg  Kevin Buchin  Bart Jansen  Gerhard Woeginger

TU Eindhoven

TU/e

# Traveling Salesman Problem (TSP)

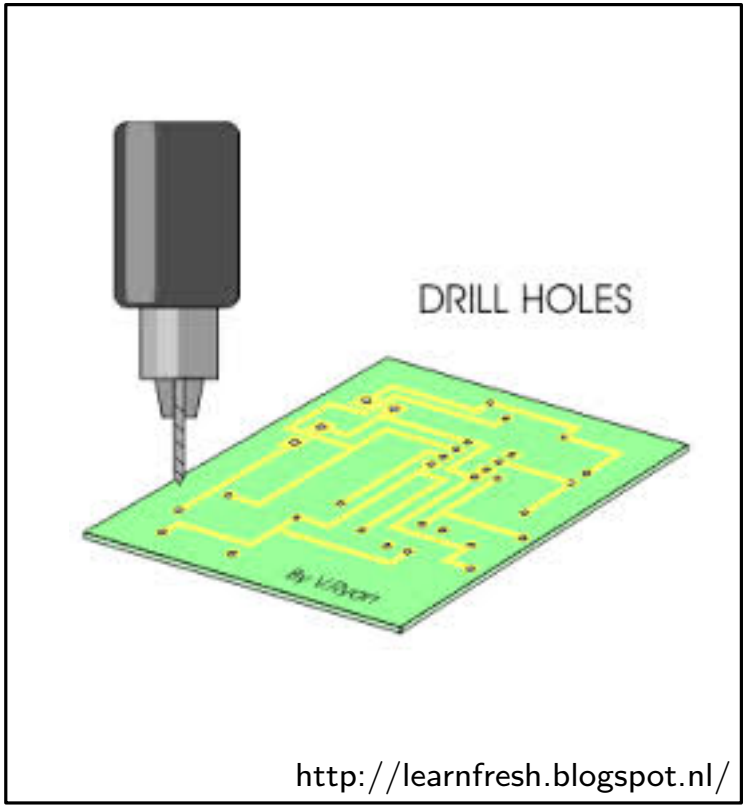

Compute min-length tour visiting all cities

# Traveling Salesman Problem (TSP)


Compute min-length tour visiting all cities

# Traveling Salesman Problem (TSP)

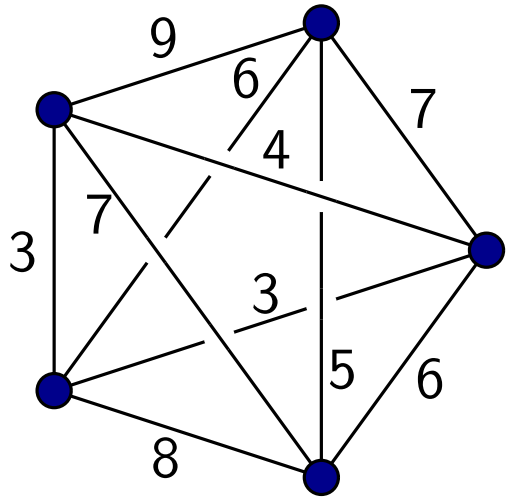

Compute min-length tour visiting all cities




Compute min-length tour visiting all holes to be drilled

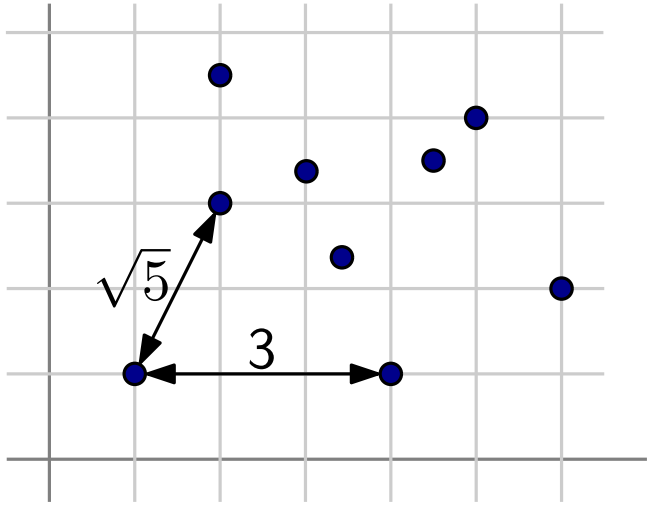
# TSP: general setting vs Euclidean setting

## General setting




- cities = nodes in a graph
- arbitrary edge lengths (satisfying triangle inequality)

## Euclidean setting

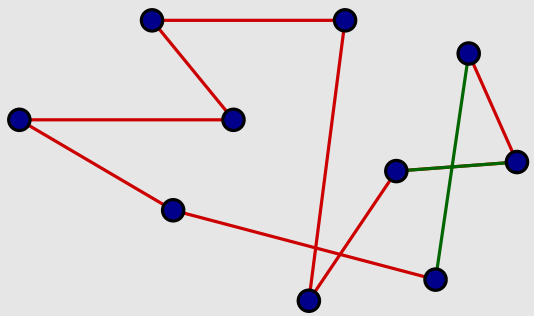



- cities = points in the plane
- edge length = Euclidean distance

TSP is NP-hard, even in Euclidean setting

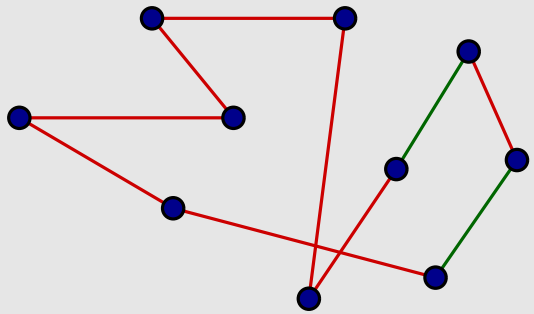
→ fast algorithm impossible (unless $\mathrm{P} = \mathrm{NP}$)

# $k$-OPT: a classic heuristic for TSP



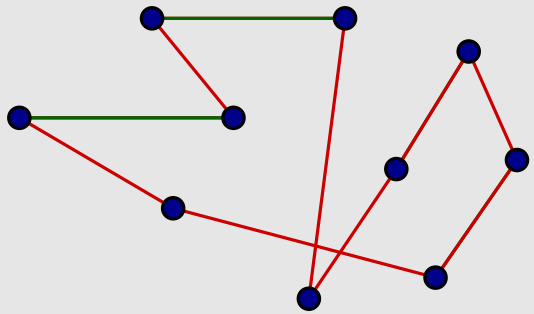
2-OPT: apply 2-swaps as long as they reduce tour length

# $k$-OPT: a classic heuristic for TSP



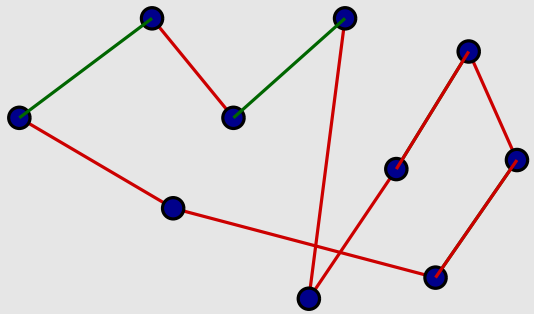
2-OPT: apply 2-swaps as long as they reduce tour length

# $k$-OPT: a classic heuristic for TSP



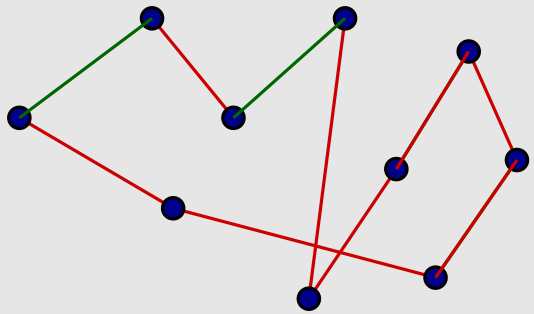
2-OPT: apply 2-swaps as long as they reduce tour length

# $k$-OPT: a classic heuristic for TSP



2-OPT: apply 2-swaps as long as they reduce tour length

# $k$-OPT: a classic heuristic for TSP



2-OPT: apply 2-swaps as long as they reduce tour length

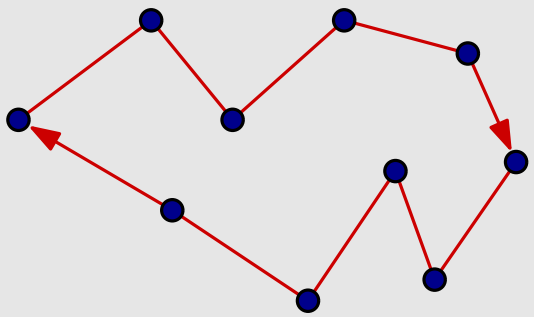
## Quality of $k$-OPT solution

- $k = 2$: within few percent of optimum
- $k > 2$: even slightly better

## How fast can we find $k$-swaps?

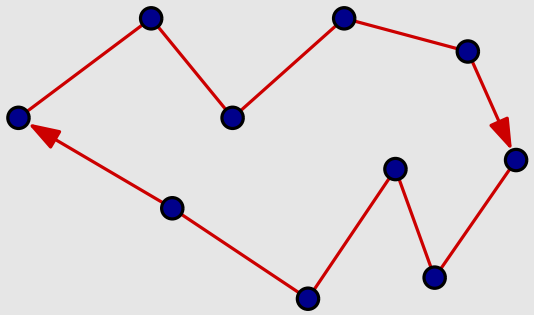
- $O(n^k)$ is trivial
- for $k = 2$ this is optimal
- what about $k \geqslant 3$?

# Bitonic TSP: a classic geometric variant



bitonic TSP (Euclidean variant):
only allow tours that go
left-to-right and then right-to-left

# Bitonic TSP: a classic geometric variant

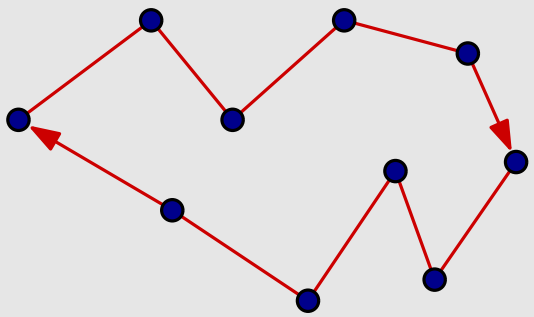


bitonic TSP (Euclidean variant):
only allow tours that go
left-to-right and then right-to-left


Classic exercise:
$O(n^2)$ algorithm using dynamic programming

no faster algorithms known

# Bitonic TSP: a classic geometric variant



bitonic TSP (Euclidean variant):
only allow tours that go
left-to-right and then right-to-left


Classic exercise:
$O(n^2)$ algorithm using dynamic programming

no faster algorithms known

pyramidal TSP:

- cities are numbered $1, \ldots, n$
- only allow unimodal permutations:
  1, 3, 6, 9, 10, 8, 7, 5, 4, 2, 1

# FINE-GRAINED COMPLEXITY ANALYSIS

# Fine-grained complexity

Traditional complexity analysis: Distinguish tractable (polynomial-time solvable) problems from intractable problems.

- Problem X can be solved in polynomial time
- Problem Y cannot be solved in polynomial time, unless P=NP

(in other words: under P≠NP Hypothesis)

# Fine-grained complexity

Traditional complexity analysis: Distinguish tractable (polynomial-time solvable) problems from intractable problems.

- Problem X can be solved in polynomial time
- Problem Y cannot be solved in polynomial time, unless P=NP

(in other words: under P≠NP Hypothesis)

Fine-grained analysis: prove more precise bounds

- Problem X can be solved in $O(n^3)$ time
- Problem X cannot be solved in $O(n^{3-\varepsilon})$, unless ...
- Problem Y can be solved in $2^{O(n)}$ time
- Problem Y cannot be solved in $2^{O(n/\log n)}$ time, unless ...

# Fine-grained complexity: conditional lower bounds

Problem X cannot be solved in $O(\cdots)$ time, under ...

# Fine-grained complexity: conditional lower bounds

Problem X cannot be solved in $O(\cdots)$ time, under ...

- Strong Exponential-Time Hypothesis (SETH)

  CNF-SAT: $(x_1 \vee \overline{x}_2) \wedge (x_1 \vee x_3 \vee x_4 \vee \overline{x}_5) \wedge (\overline{x}_1 \vee \overline{x_3} \vee x_5)$ satisfiable?

  SETH: CNF-SAT cannot be solved in $O(2^{(1-\varepsilon)n})$ time

# Fine-grained complexity: conditional lower bounds

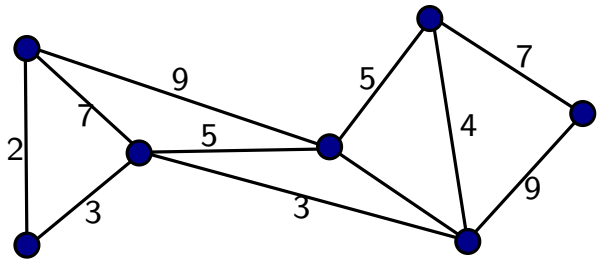
Problem X cannot be solved in $O(\cdots)$ time, under ...

- Strong Exponential-Time Hypothesis (SETH)

  CNF-SAT: $(x_1 \vee \overline{x}_2) \wedge (x_1 \vee x_3 \vee x_4 \vee \overline{x}_5) \wedge (\overline{x}_1 \vee \overline{x_3} \vee x_5)$ satisfiable?

  SETH: CNF-SAT cannot be solved in $O(2^{(1-\varepsilon)n})$ time

- or Exponential-Time Hypothesis (ETH), or 3-SUM Conjecture
- All-Pairs Shortest Paths (APSP) Conjecture




APSP Conjecture: Computing all pairwise distances in weighted graph cannot be done in $O(n^{3-\varepsilon})$ time.

# Fine-grained complexity: conditional lower bounds

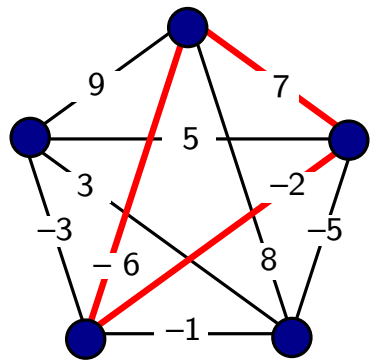
Examples:

- Longest Common Subsequence (LCS)
  cannot be solved in $O(n^{2-\varepsilon})$ time,
  under SETH [Bringmann, Künnemann 2015]


- Fréchet Distance cannot be solved in $O(n^{2-\varepsilon})$ time,
  under SETH [Bringmann 2014]

- Negative-Weight Triangle cannot be solved
  in $O(n^{3-\varepsilon})$ time, under APSP Conjecture
  [Vassilevska-Williams, Williams 2010]

  (in fact, Negative-Weight Triangle
  and APSP are "equivalent")



# Our Results

## $k$-OPT in general setting

- $k = 3$: lower bound of $\Omega(n^{3-\varepsilon})$ under APSP Conjecture
- $k > 3$: algorithm with $O(n^{\lfloor 2k/3 \rfloor + 1})$ running time
- iterated $k$-OPT: $O(n^2)$ preprocessing, $O(n \log n)$ per iteration $(k = 2)$

  [ $O(n^3)$ preprocessing, $O(n^2 \log n)$ per iteration $(k = 3)$ ]
- Euclidean setting: algorithm with $O(n^{8/5+\varepsilon})$ running time $(k = 2)$

  algorithm with $O(n^{80/31+\varepsilon})$ running time $(k = 3)$

## Bitonic (more generally: pyramidal) Euclidean TSP in the plane

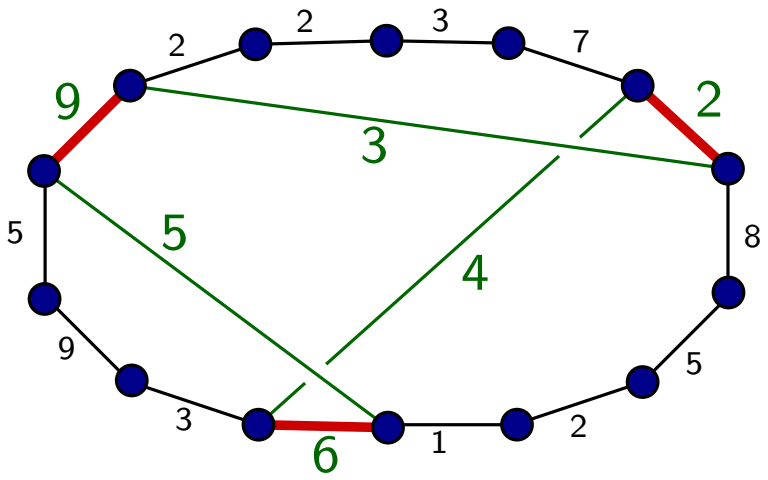
- algorithm with $O(n \log^2 n)$ running time
- bottleneck variant: $O(n \log^3 n)$

# RESULTS ON $k$-OPT

## 3-Opt Detection

Given: Tour $T$ in undirected complete graph $G$ on $n$ vertices
with positive edge weights satisfying triangle inequality

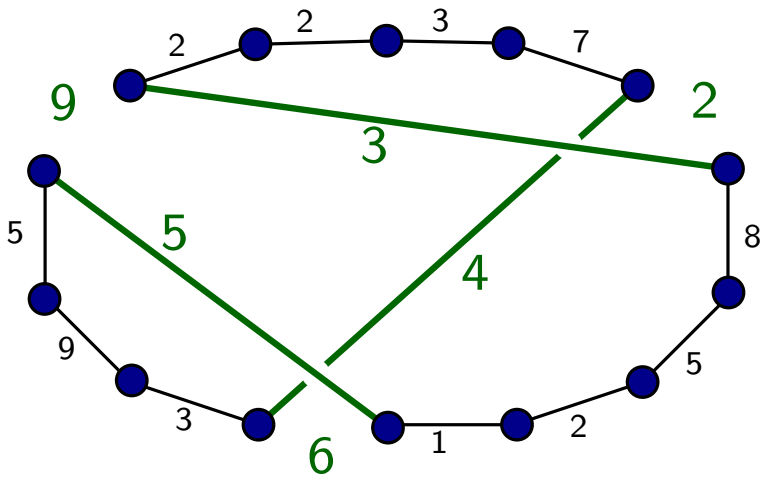
Question: Is there a 3-swap that reduces tour length?



## 3-Opt Detection

Given: Tour $T$ in undirected complete graph $G$ on $n$ vertices
with positive edge weights satisfying triangle inequality

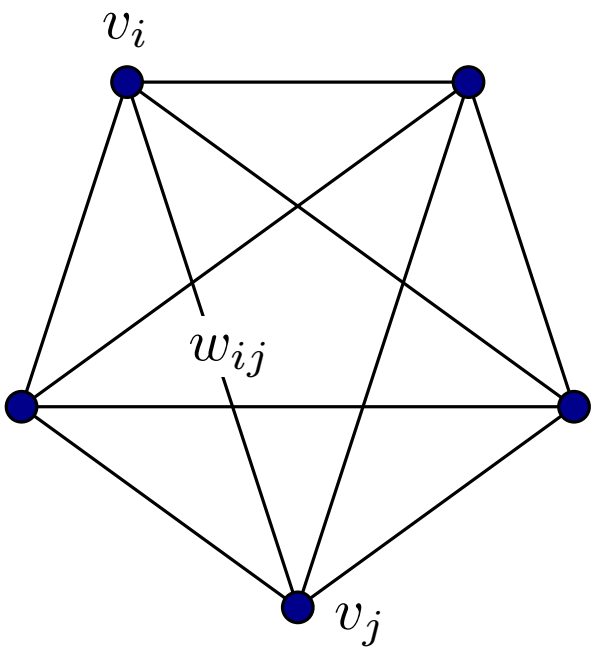
Question: Is there a 3-swap that reduces tour length?



# Conditional Lower Bound for 3-Opt Detection

Theorem. 3-Opt Detection cannot be solved in $O(n^{3-\varepsilon})$ time under the APSP Conjecture.

Proof. Reduction from Negative-Weight Triange




transform input graph $G$ into graph $G'$ with $O(n)$ vertices and tour $T$ in $G'$ such that

negative weight triangle in $G$

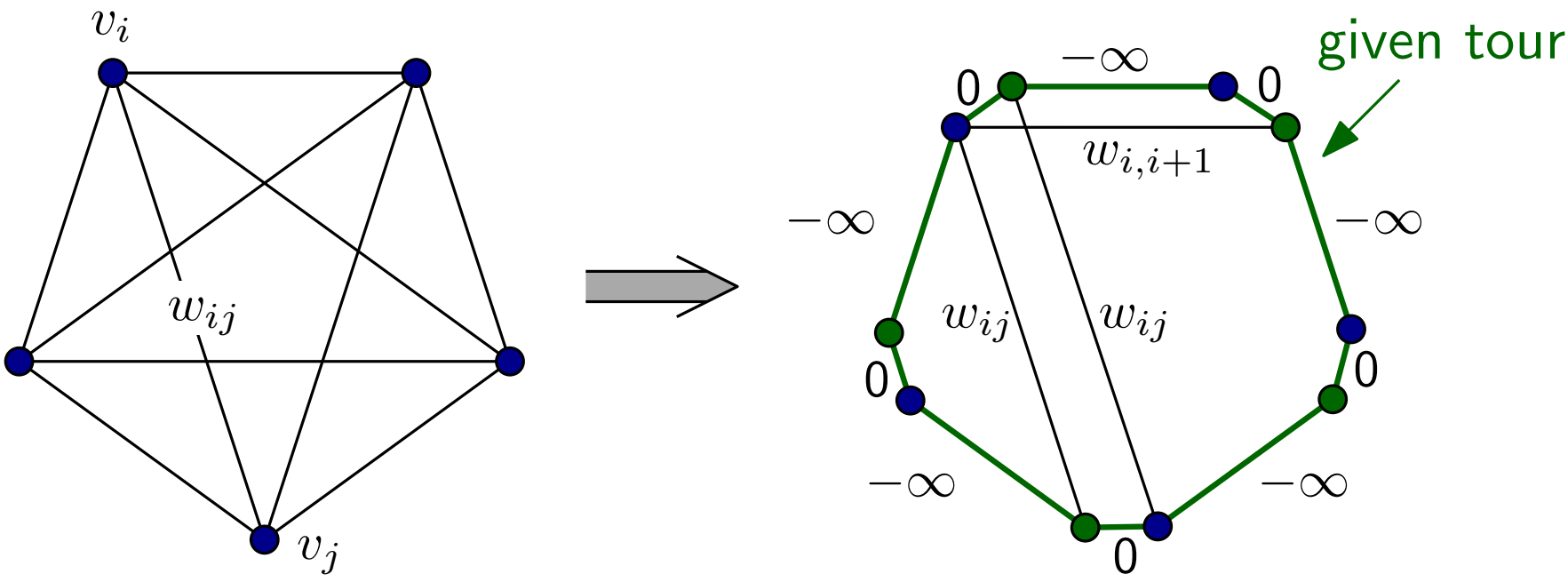
$\equiv$

improving 3-swap in $T$

# Conditional Lower Bound for 3-Opt Detection

Theorem. 3-Opt Detection cannot be solved in $O(n^{3-\varepsilon})$ time under the APSP Conjecture.

Proof. Reduction from Negative-Weight Triange

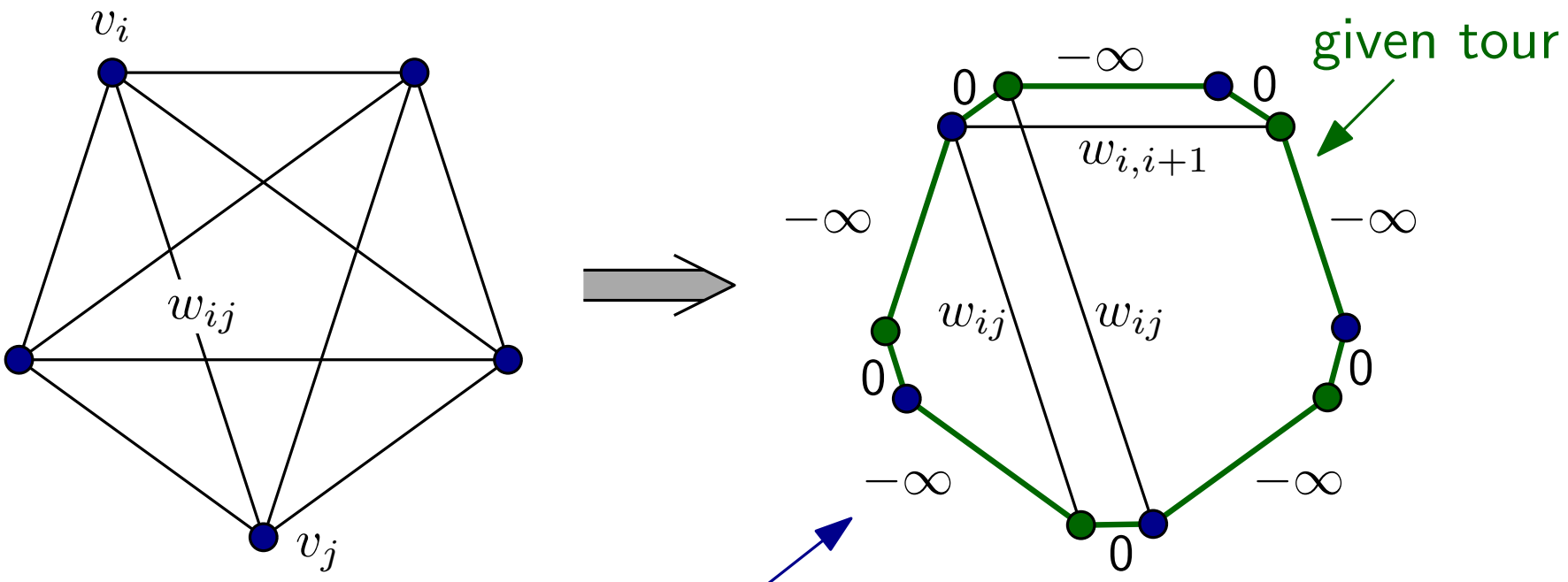



monochromatic edges have weight $+\infty$

# Conditional Lower Bound for 3-Opt Detection

**Theorem.** 3-Opt Detection cannot be solved in $O(n^{3-\varepsilon})$ time under the APSP Conjecture.

**Proof.** Reduction from Negative-Weight Triange



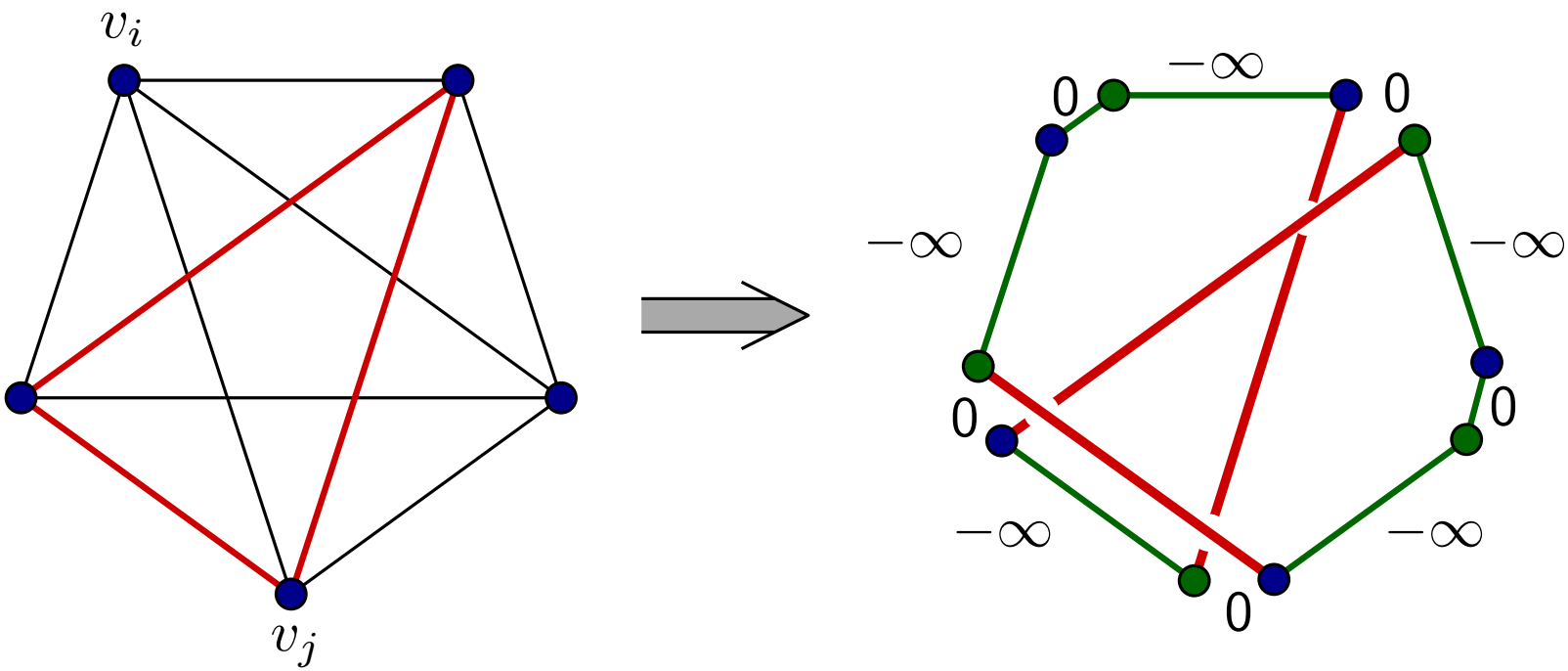

(negative weights can be avoided)

monochromatic edges have weight $+\infty$

# Conditional Lower Bound for 3-Opt Detection

**Theorem.** 3-Opt Detection cannot be solved in $O(n^{3-\varepsilon})$ time under the APSP Conjecture.

**Proof.** negative-weight triangles $\equiv$ improving 3-swaps

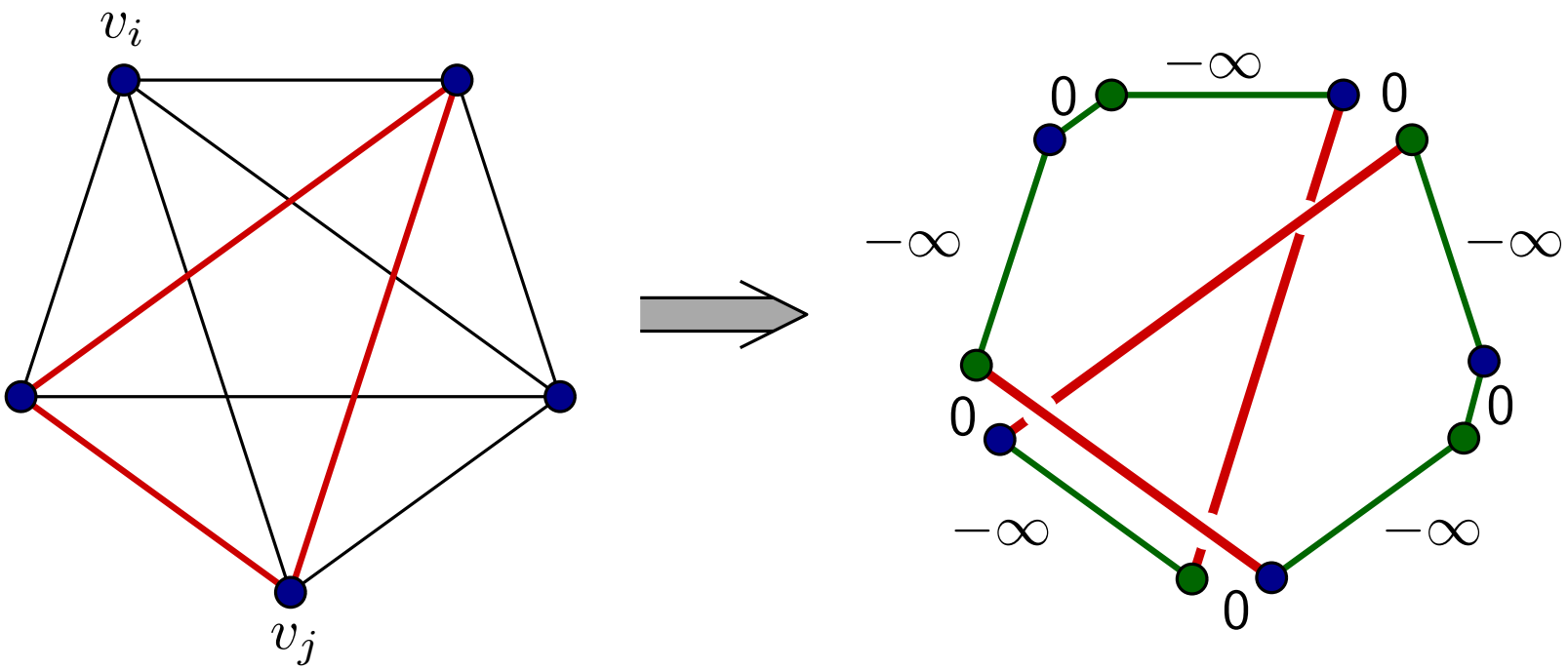



monochromatic edges have weight $+\infty$

# Conditional Lower Bound for 3-Opt Detection

**Theorem.** 3-Opt Detection cannot be solved in $O(n^{3-\varepsilon})$ time under the APSP Conjecture.

**Proof.** negative-weight triangles $\equiv$ improving 3-swaps




monochromatic edges have weight $+\infty$

# Complexity of $k$-OPT

complexity bounds for $k$-OPT

- trivial upper bound: $O(n^k)$
- lower bounds
  - $k = 2$: $\Omega(n^2)$ (must look at every edge)
  - $k = 3$: $O(n^{3-\varepsilon})$ time impossible under the APSP Conjecture

What about $k > 3$? Do we basically need $\Omega(n^k)$ time?

# A faster algorithm for $k$-OPT for $k \geqslant 4$

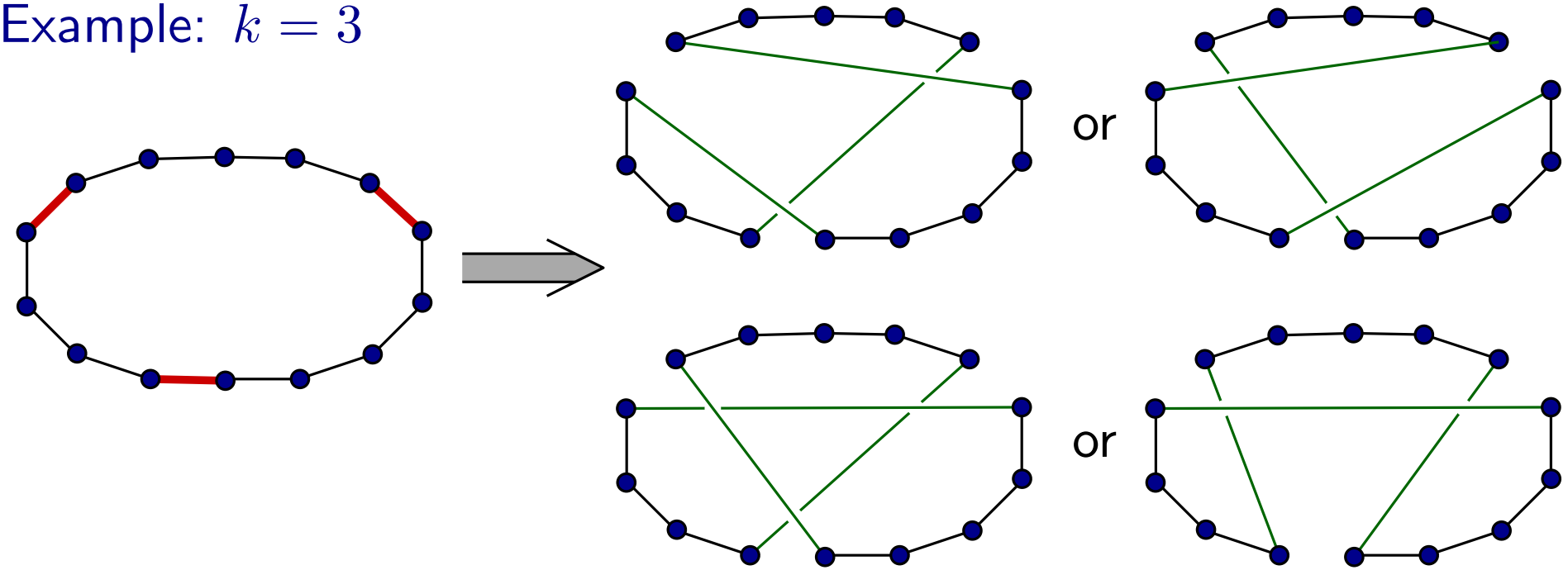
Theorem. For any fixed $k \geqslant 4$ we can find the best $k$-swap in $O(n^{\lfloor 2k/3 \rfloor + 1})$ time.

Proof. ...

# A faster algorithm for $k$-OPT for $k \geqslant 4$

For $k > 2$ there are several combinations of edges that can replace $k$ given edges.

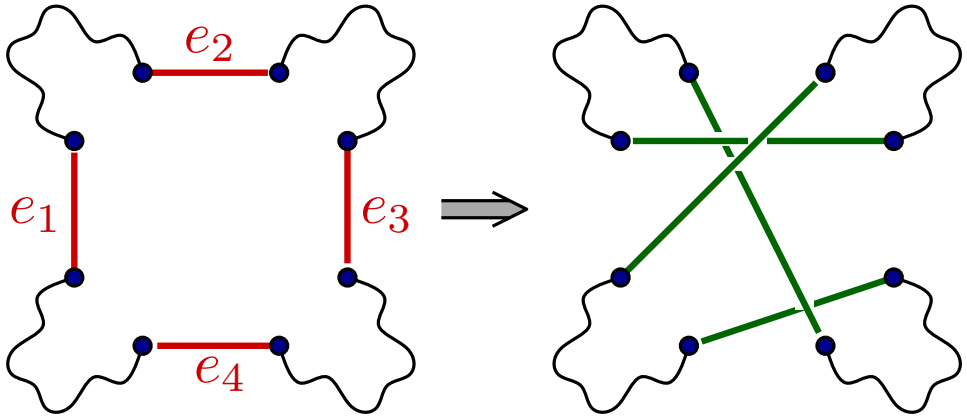
Example: $k = 3$




Let's handle them separately

# A faster algorithm for $k$-OPT for $k \geqslant 4$

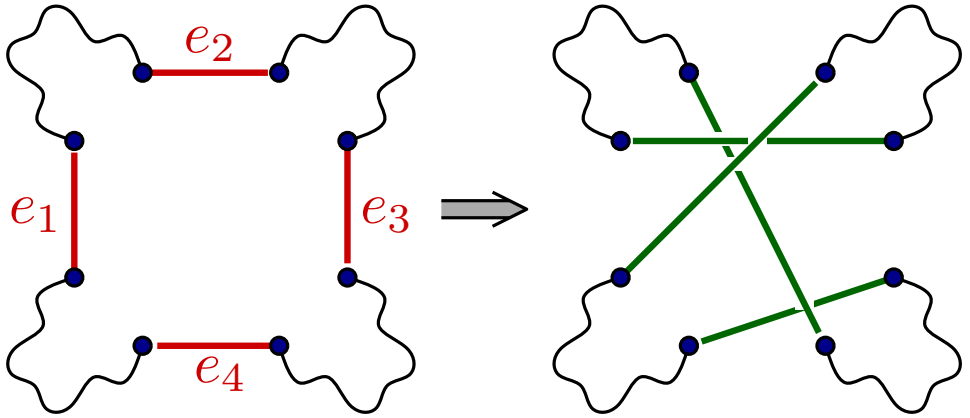
signature of a $k$-swap $E = \{e_1, e_2, \ldots, e_k\} \rightarrow F = \{f_1, f_2, \ldots, f_k\}$

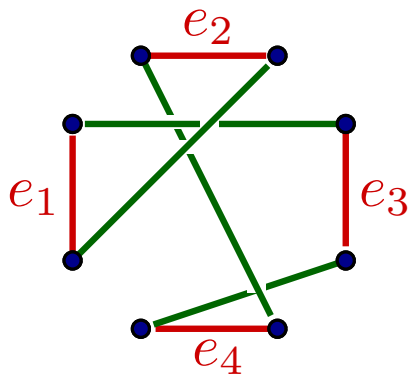



signature

# A faster algorithm for $k$-OPT for $k \geqslant 4$

signature of a $k$-swap $E = \{e_1, e_2, \ldots, e_k\} \rightarrow F = \{f_1, f_2, \ldots, f_k\}$




signature

find best $k$-swap for each signature separately

# A faster algorithm for $k$-OPT for $k \geqslant 4$

finding best $k$-swap $E = \{e_1, \ldots, e_k\} \to F$ for given signature

$\equiv$ finding best locations for edges in $E$

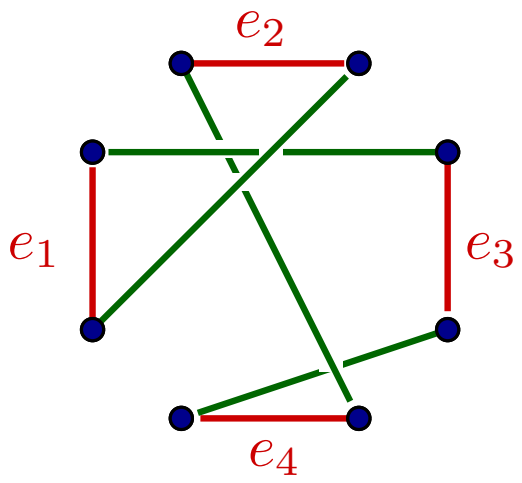
How to do this without checking all $O(n^k)$ possibilities for $E$?

# A faster algorithm for $k$-OPT for $k \geqslant 4$

finding best $k$-swap $E = \{e_1, \ldots, e_k\} \to F$ for given signature

$\equiv$ finding best locations for edges in $E$

How to do this without checking all $O(n^k)$ possibilities for $E$?




$e_i$ and $e_j$ interfere for given signature:

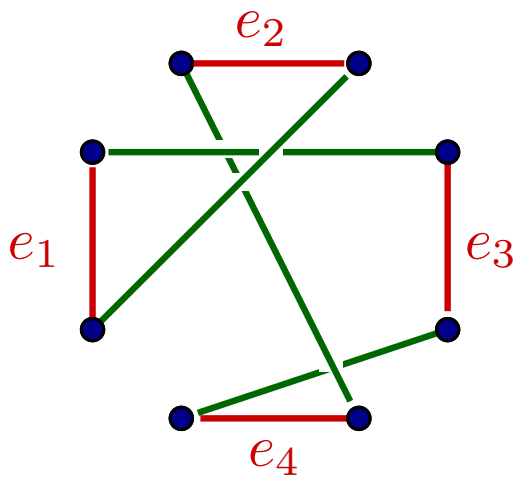
there is an edge in $F$ connecting $e_i$ to $e_j$

# A faster algorithm for $k$-OPT for $k \geqslant 4$

finding best $k$-swap $E = \{e_1, \ldots, e_k\} \to F$ for given signature

$\equiv$ finding best locations for edges in $E$

How to do this without checking all $O(n^k)$ possibilities for $E$?




$e_i$ and $e_j$ interfere for given signature:

there is an edge in $F$ connecting $e_i$ to $e_j$

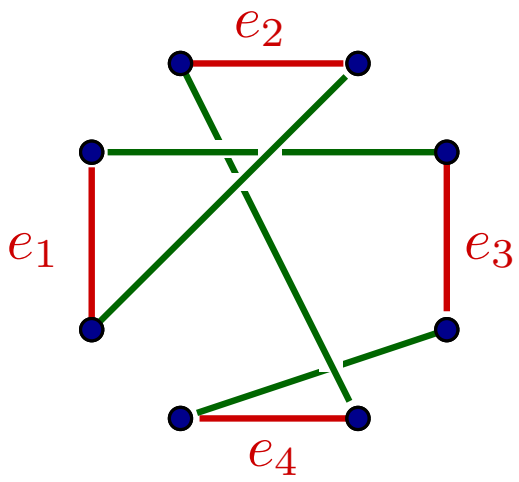
$e_2$ interferes with $e_1$ and $e_4$, but not with $e_3$

# A faster algorithm for $k$-OPT for $k \geqslant 4$

Lemma. For any signature, there is a subset of at least $\lceil k/3 \rceil$ pairwise non-interfering edges.

# A faster algorithm for $k$-OPT for $k \geqslant 4$

Lemma. For any signature, there is a subset of at least $\lceil k/3 \rceil$ pairwise non-interfering edges.




- $E \cup F$ consists of one or more cyles
- along each cycle the edges from $E$ and $F$ alternate

per cycle at least $1/3$ of edges from $E$ are pairwise non-interfering

# A faster algorithm for $k$-OPT for $k \geqslant 4$

finding best $k$-swap $E = \{e_1, \ldots, e_k\} \to F$ for given signature

$\equiv$ finding best locations to "embed" edges in $E$ into tour

How to do this without checking all $O(n^k)$ possibilities for $E$?

# A faster algorithm for $k$-OPT for $k \geqslant 4$

finding best $k$-swap $E = \{e_1, \ldots, e_k\} \to F$ for given signature

$\equiv$ finding best locations to "embed" edges in $E$ into tour

How to do this without checking all $O(n^k)$ possibilities for $E$?

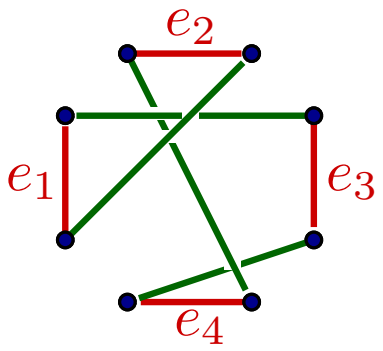
1. determine set $E^* \subset E$ of $\lceil k/3 \rceil$ pairwise non-interfering edges for given signature
2. try all possible choices for the $\lfloor 2k/3 \rfloor$ edges in $E \setminus E^*$

   for each of these $O(n^{\lfloor 2k/3 \rfloor})$ choices, compute the best way to embed remaining $\lfloor k/3 \rfloor$ edges

   → can be done in $O(n)$ time

# A faster algorithm for $k$-OPT for $k \geqslant 4$

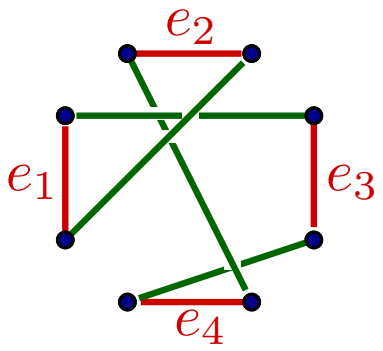
How to add pairwise non-interfering edges in $O(n)$ time

Example: k=4 and signature 

and $e_2, e_3$ non-interfering

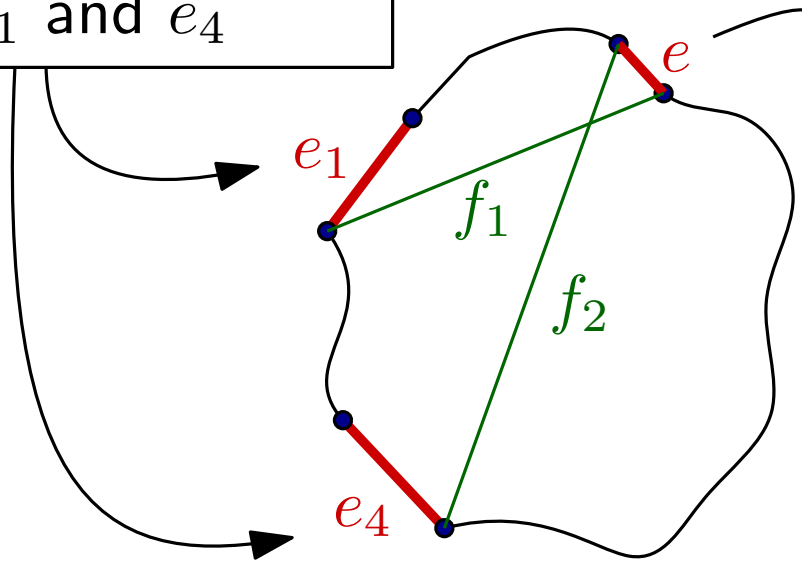
# A faster algorithm for $k$-OPT for $k \geqslant 4$

How to add pairwise non-interfering edges in $O(n)$ time

Example: k=4 and signature 

and $e_2, e_3$ non-interfering

fixed choice for embedding $e_1$ and $e_4$

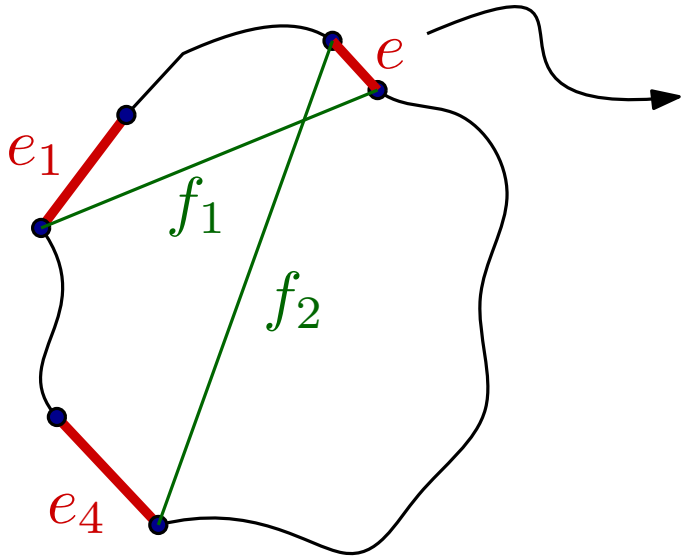



- cost of embedding $e_2$ into $e$:

  $\text{cost}(e_2 \to e) = |f_1| + |f_2| - |e|$

- independent of where we embed $e_3$

# A faster algorithm for $k$-OPT for $k \geqslant 4$

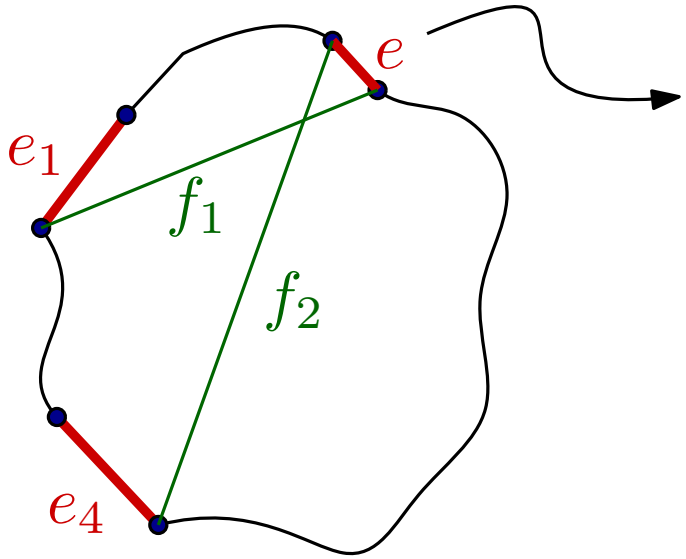



- cost of embedding $e_2$ into $e$:

  $\mathsf{cost}(e_2 \to e) = |f_1| + |f_2| - |e|$

- independent of where we embed $e_3$

# A faster algorithm for $k$-OPT for $k \geqslant 4$




- cost of embedding $e_2$ into $e$:

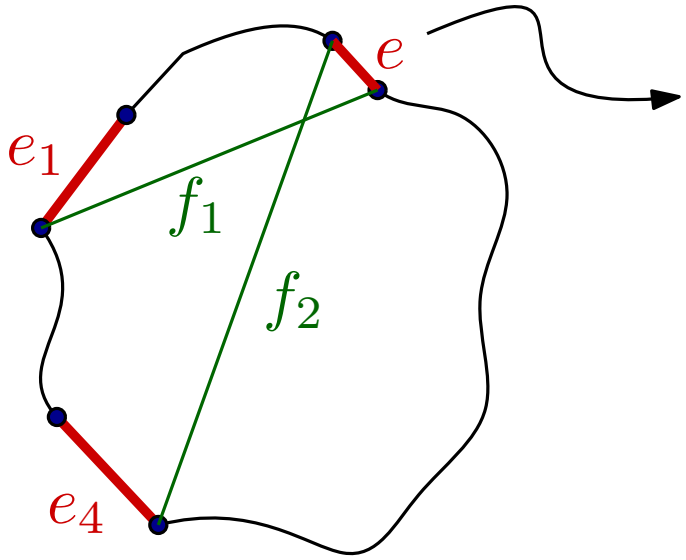
  $\mathsf{cost}(e_2 \to e) = |f_1| + |f_2| - |e|$

- independent of where we embed $e_3$

- $e_1^*, \ldots, e_{k/3}^*$ = edges we still need to embed
- $e_1', \ldots, e_n'$ = tour edges
- $\mathsf{Cost}[\ell, r]$ := min cost of embedding $e_1^*, \ldots, e_\ell^*$ into $e_1', \ldots, e_r'$

$$\mathsf{Cost}[\ell, r] = \min\left(\mathsf{Cost}[\ell, r-1], \mathsf{Cost}[\ell-1, r-1] + \mathsf{cost}(e_\ell^* \to e_r')\right)$$

# A faster algorithm for $k$-OPT for $k \geqslant 4$




- cost of embedding $e_2$ into $e$:

  $\text{cost}(e_2 \to e) = |f_1| + |f_2| - |e|$

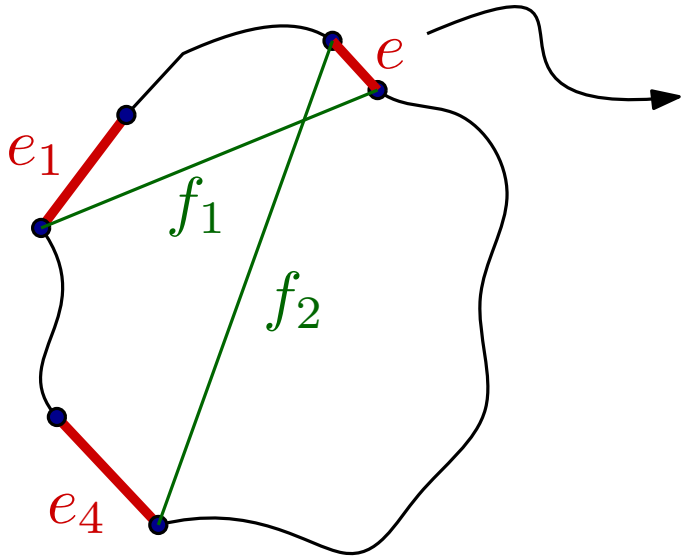
- independent of where we embed $e_3$

- $e_1^*, \ldots, e_{k/3}^*$ = edges we still need to embed
- $e_1', \ldots, e_n'$ = tour edges
- $\text{Cost}[\ell, r]$ := min cost of embedding $e_1^*, \ldots, e_\ell^*$ into $e_1', \ldots, e_r'$

$$\text{Cost}[\ell, r] = \min\left(\text{Cost}[\ell, r-1], \text{Cost}[\ell-1, r-1] + \text{cost}(e_\ell^* \to e_r')\right)$$

if we are allowed to embed $e_\ell^*$ into $e_r'$

# A faster algorithm for $k$-OPT for $k \geqslant 4$




- cost of embedding $e_2$ into $e$:

  $\text{cost}(e_2 \to e) = |f_1| + |f_2| - |e|$

- independent of where we embed $e_3$

- $e_1^*, \ldots, e_{k/3}^*$ = edges we still need to embed
- $e_1', \ldots, e_n'$ = tour edges
- $\text{Cost}[\ell, r]$ := min cost of embedding $e_1^*, \ldots, e_\ell^*$ into $e_1', \ldots, e_r'$

$$\text{Cost}[\ell, r] = \min\left(\text{Cost}[\ell, r-1], \text{Cost}[\ell-1, r-1] + \text{cost}(e_\ell^* \to e_r')\right)$$

if we are allowed to embed $e_\ell^*$ into $e_r'$

- final answer: $\text{Cost}[k/3, n]$
- time: $O(nk) = O(n)$

# Complexity of $k$-OPT

Theorem. For any fixed $k \geqslant 4$ we can find the best $k$-swap in $O(n^{\lfloor 2k/3 \rfloor + 1})$ time.

# Complexity of $k$-OPT

Theorem. For any fixed $k \geqslant 4$ we can find the best $k$-swap in $O(n^{\lfloor 2k/3 \rfloor + 1})$ time.

complexity bounds for $k$-OPT

- $k = 2$: $\Theta(n^2)$
- $k = 3$: $O(n^3)$, $O(n^{3-\varepsilon})$ impossible under the APSP Conjecture
- $k \geqslant 4$: $O(n^{\lfloor 2k/3 \rfloor + 1})$

  $n^{o(k/\log k)}$ impossible under ETH [Guo et al.]

# Complexity of $k$-OPT

Theorem. For any fixed $k \geqslant 4$ we can find the best $k$-swap in $O(n^{\lfloor 2k/3 \rfloor + 1})$ time.

complexity bounds for $k$-OPT

- $k = 2$: $\Theta(n^2)$
- $k = 3$: $O(n^3)$, $O(n^{3-\varepsilon})$ impossible under the APSP Conjecture
- $k \geqslant 4$: $O(n^{\lfloor 2k/3 \rfloor + 1})$

  $n^{o(k/\log k)}$ impossible under ETH [Guo et al.]

other results: bounds for $k = 2, 3$ can be improved for

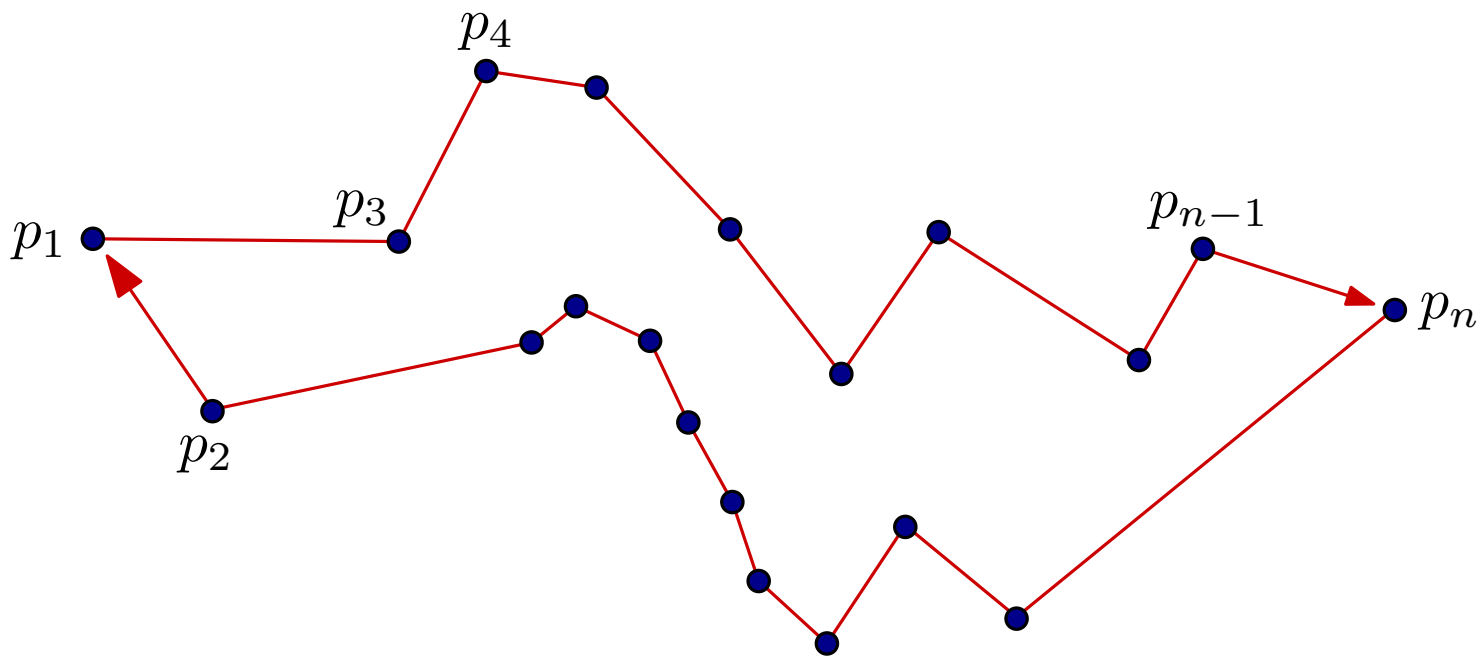
- iterated 2-OPT $O(n \log n)$ [ iterated 3-OPT $O(n^2 \log n)$ ]
- Euclidean $k$-OPT: $O(n^{8/5+\varepsilon})$ resp. $O(n^{80/31+\varepsilon})$

# RESULTS ON BITONIC TSP

# Bitonic TSP in the plane

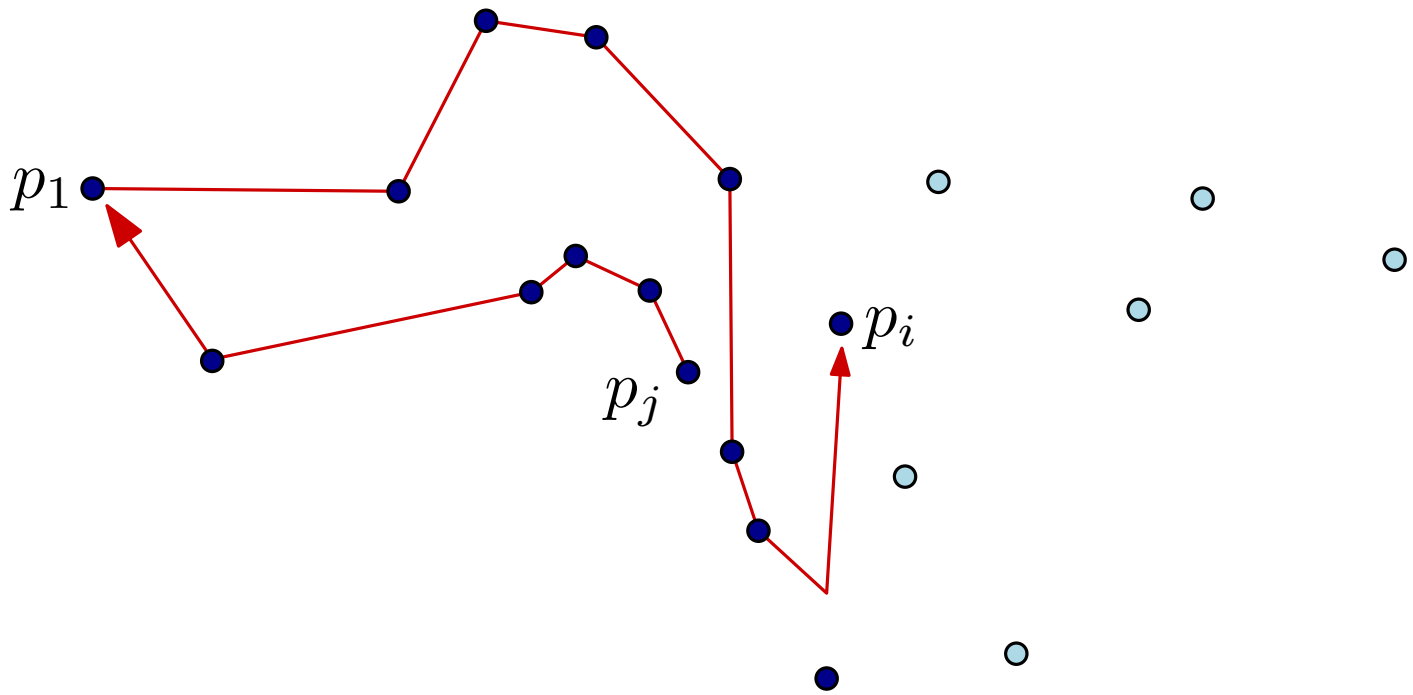
## Bitonic TSP

- find min-length tour on points in the plane, Euclidean distances
- only allow tours that go left-to-right and then right-to-left




$p_1, p_2, \ldots, p_n$: points numbered from left to right

# Bitonic TSP in the plane: textbook solution



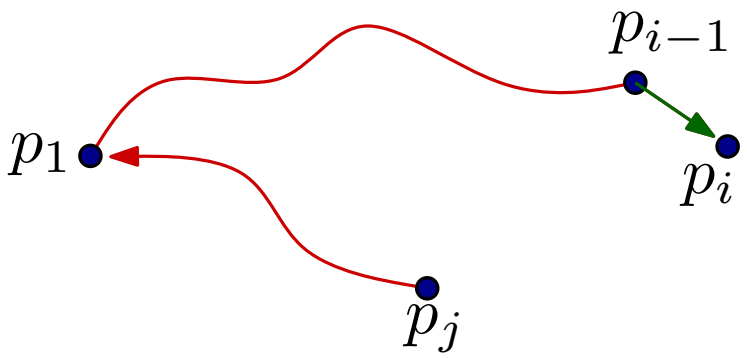

For each pair $p_i, p_j$ with $1 \leqslant j < i \leqslant n$, compute

$A[i,j] =$ min length of left-to-right path from $p_1$ to $p_i$ and right-to-left path from $p_j$ back to $p_1$ that together visit all points $p_1, \ldots p_i$

# Bitonic TSP in the plane: textbook solution

## Recursive formula for $A[i, j]$

case (i): $j < i - 1$




must connect $p_i$ to $p_i$

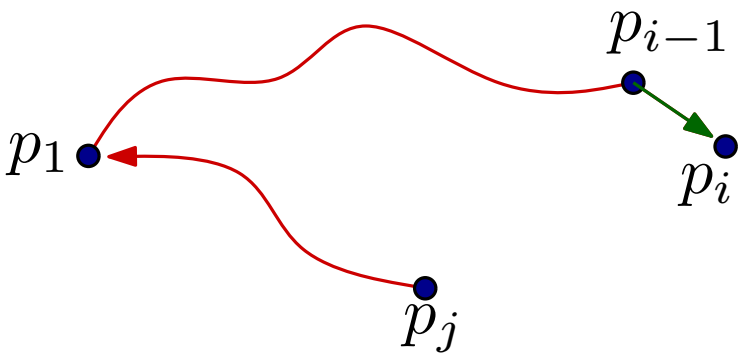
$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

where $A[2,1] = |p_1 p_2|$.

# Bitonic TSP in the plane: textbook solution

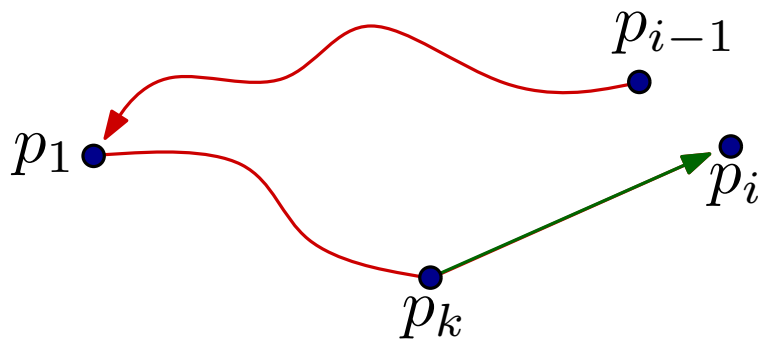
## Recursive formula for $A[i, j]$

case (i): $j < i - 1$




must connect $p_i$ to $p_i$

case (ii): $j = i - 1$




can connect to any $p_k$ with $k < i - 1$

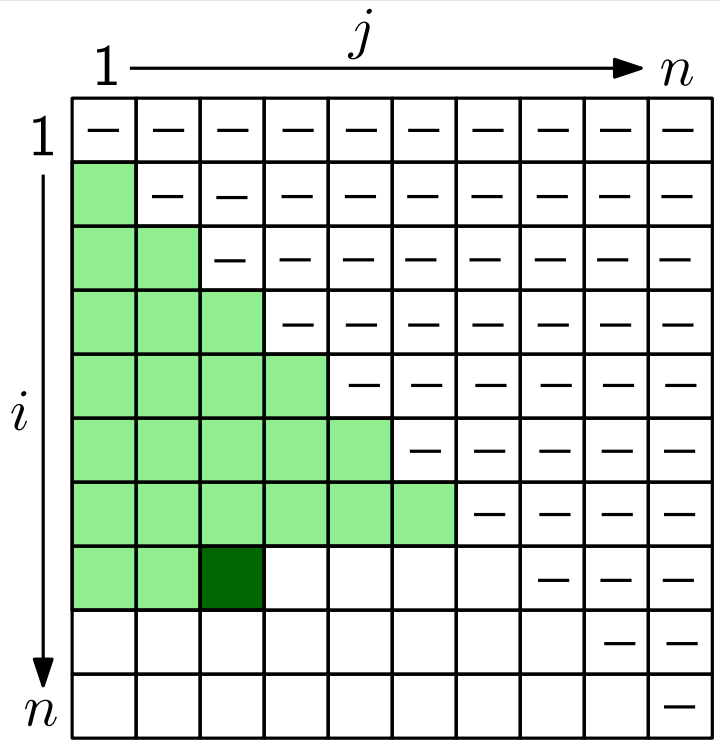
$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

where $A[2, 1] = |p_1 p_2|$.

# Bitonic TSP in the plane: textbook solution

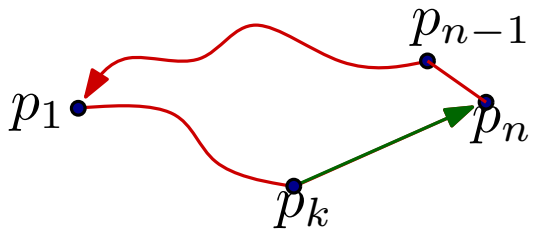
$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

where $A[2,1] = |p_1 p_2|$.




## Algorithm

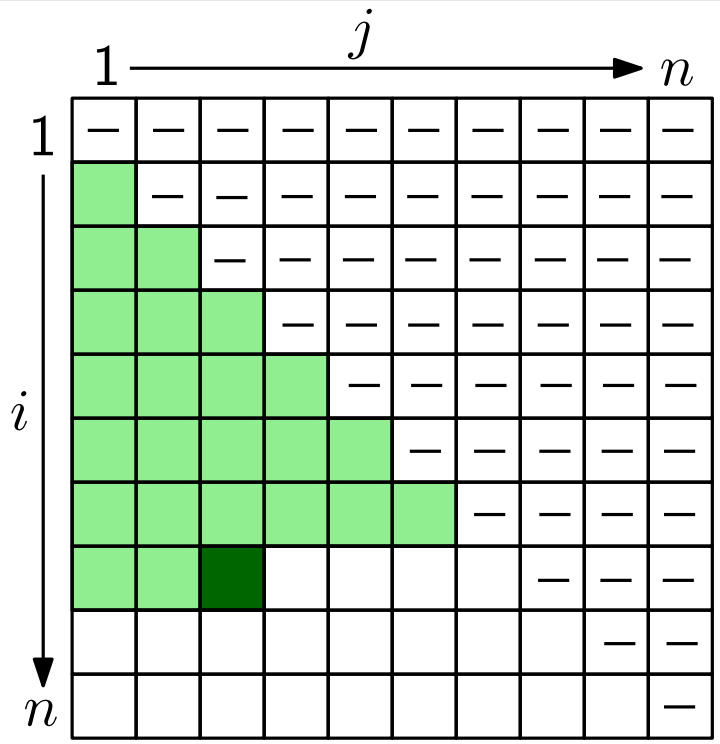
1. Fill in table row by row, using recursive formula.
2. Compute final solution: $\min_{1 \leqslant k < n} \left(A[n,k] + |p_k p_n|\right)$



# Bitonic TSP in the plane: textbook solution

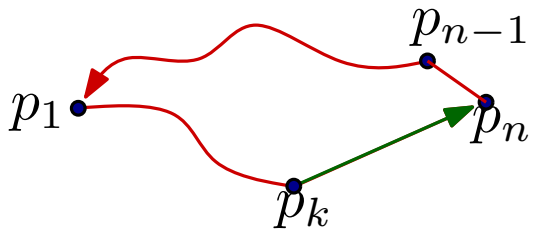
$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

where $A[2,1] = |p_1 p_2|$.




Algorithm runs in $O(n^2)$ time

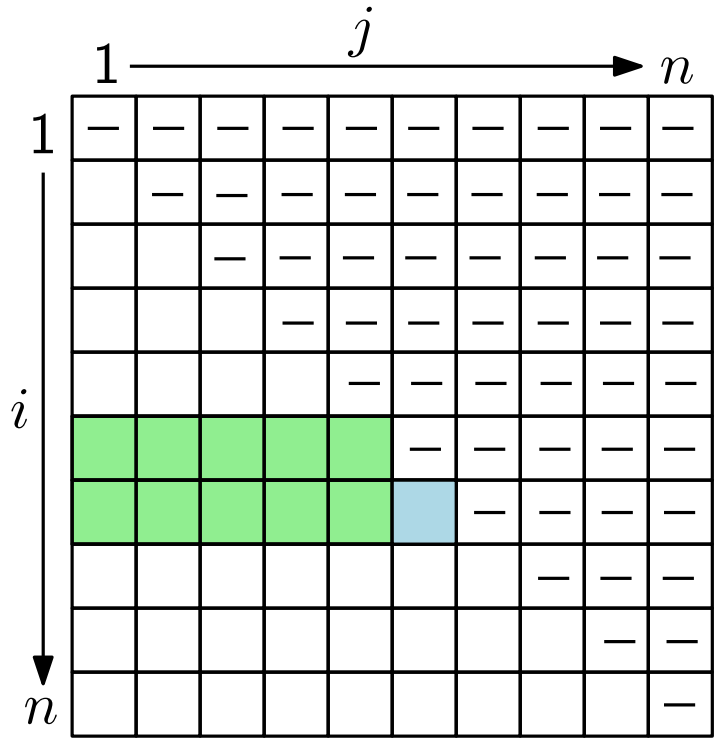
1. Fill in table row by row, using recursive formula.
2. Compute final solution: $\min_{1 \leqslant k < n} \left(A[n,k] + |p_k p_n|\right)$



# Bitonic TSP in the plane: our solution

$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

where $A[2,1] = |p_1 p_2|$.



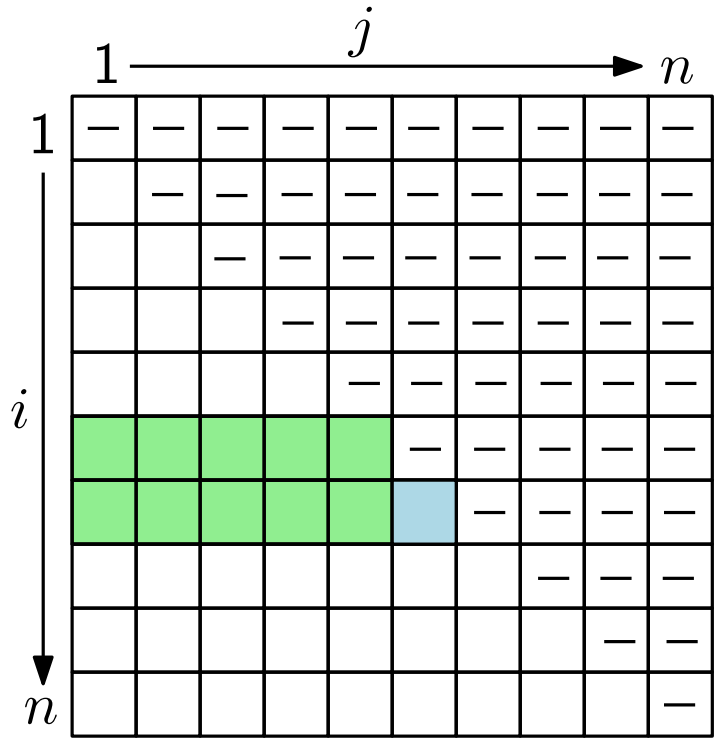

Construct implicit representation of rows, in $O(\log^2 n)$ time per row

- need to find best $k$ quickly
- entries change in same way

# Bitonic TSP in the plane: our solution

$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

where $A[2,1] = |p_1 p_2|$.



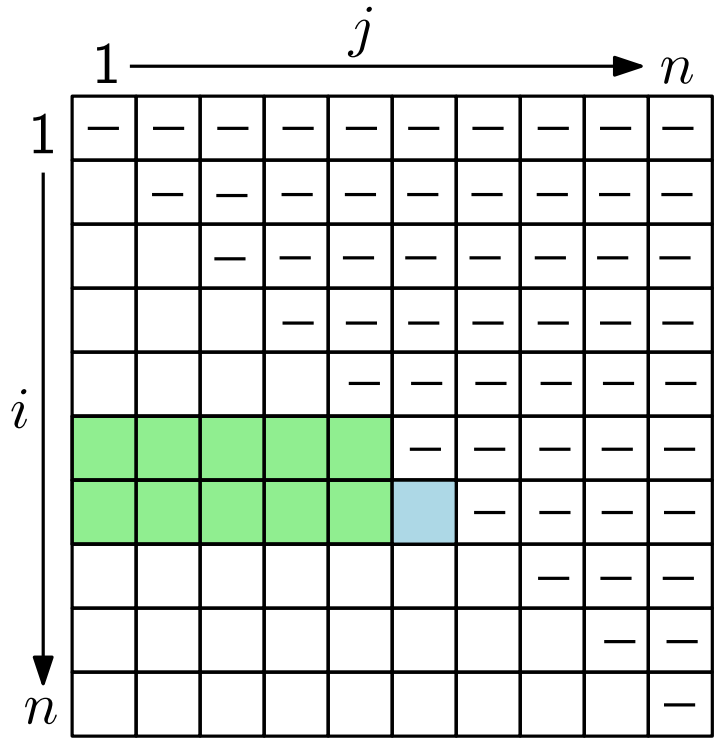

Construct implicit representation of rows, in $O(\log^2 n)$ time per row

- need to find best $k$ quickly
- entries change in same way

# Bitonic TSP in the plane: our solution

$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

where $A[2,1] = |p_1 p_2|$.




Construct implicit representation of rows, in $O(\log^2 n)$ time per row

- need to find best $k$ quickly
- entries change in same way

Need data structure supporting queries, insertions and bulk updates

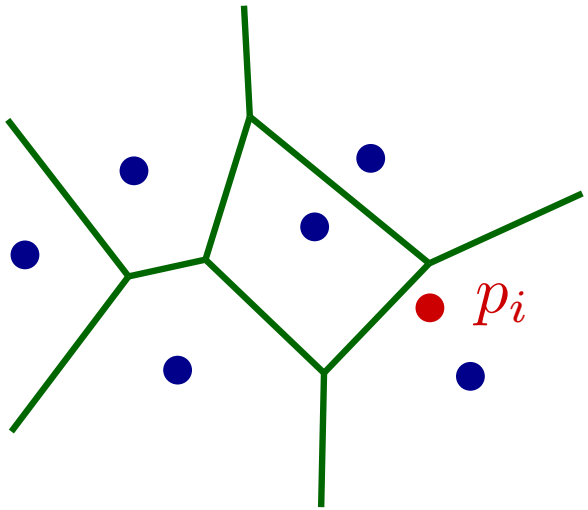
# The data structure

$$A[i,j] \;=\; \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

# The data structure

$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

without term $A[k-1]$ we would be looking for nearest neighbor

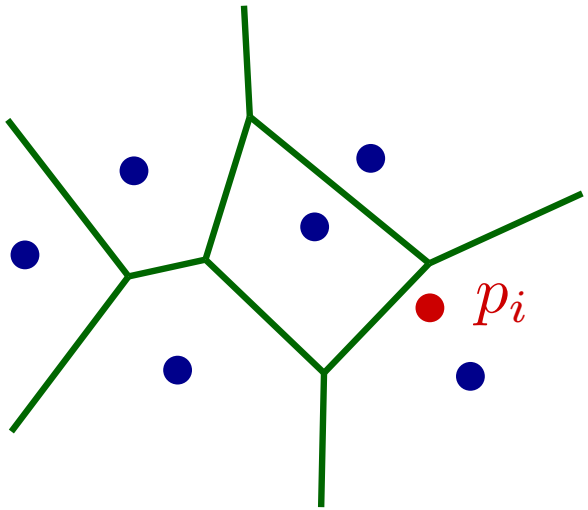



- point location with $p_i$ in Voronoi diagram of $\{p_1, \ldots, p_{i-1}\}$
- $O(i \log i)$ preprocessing, $O(\log i)$ query

# The data structure

$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

without term $A[k-1]$ we would be looking for nearest neighbor




- point location with $p_i$ in Voronoi diagram of $\{p_1, \ldots, p_{i-1}\}$
- $O(i \log i)$ preprocessing, $O(\log i)$ query

- view $w_k := A[i-1,k]$ as weight of $p_k$
- work with additively weighted distance function

## The data structure

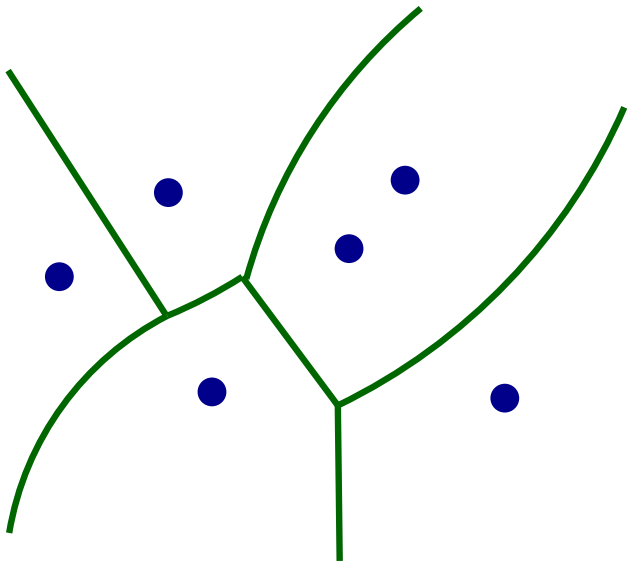
- $P =$ set of point sites
- $w_k =$ weight of point $p_k \in P$
- distance of $p_k \in P$ to any point $q \in \mathbb{R}^2$: $\text{dist}(p_k, q) = w_k + |p_k q|$

additively weighted Voronoi diagram $=$ subdivision of $\mathbb{R}^2$ into cells such that $\text{Cell}(p_k)$ contains points $q \in \mathbb{R}^2$ for which $p_k$ is closest site

# The data structure

- $P =$ set of point sites
- $w_k =$ weight of point $p_k \in P$
- distance of $p_k \in P$ to any point $q \in \mathbb{R}^2$: $\text{dist}(p_k, q) = w_k + |p_k q|$

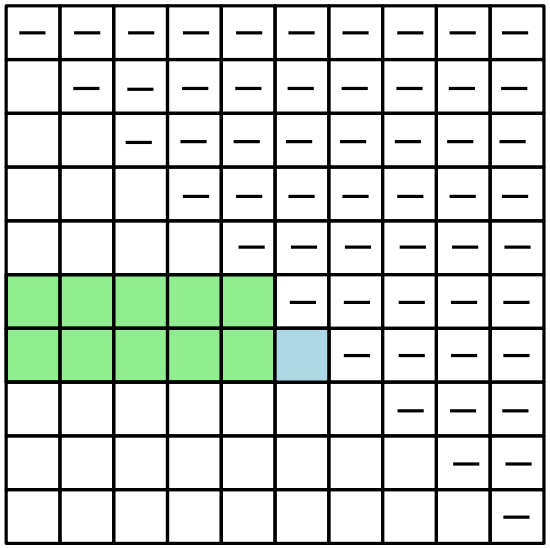
additively weighted Voronoi diagram = subdivision of $\mathbb{R}^2$ into cells such that Cell$(p_k)$ contains points $q \in \mathbb{R}^2$ for which $p_k$ is closest site



- cell boundaries are line segments or hyperbolic arcs
- some points may not have a corresponding cell
- complexity still $O(i)$ for $i$ points
- can be computed (with point location structure) in $O(i \log i)$ time

# The data structure

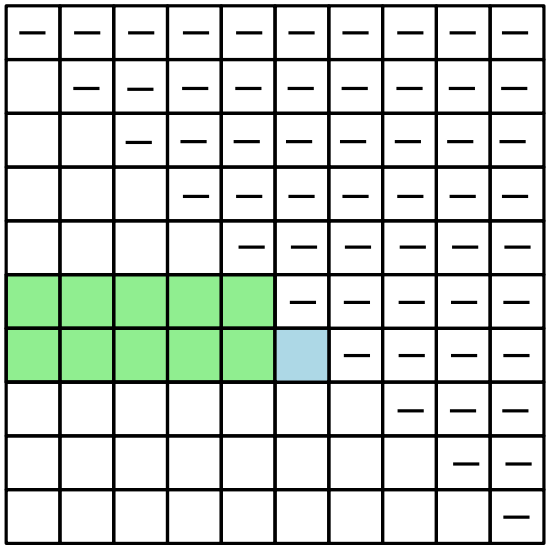
$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$



Computing the next row

# The data structure

$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$

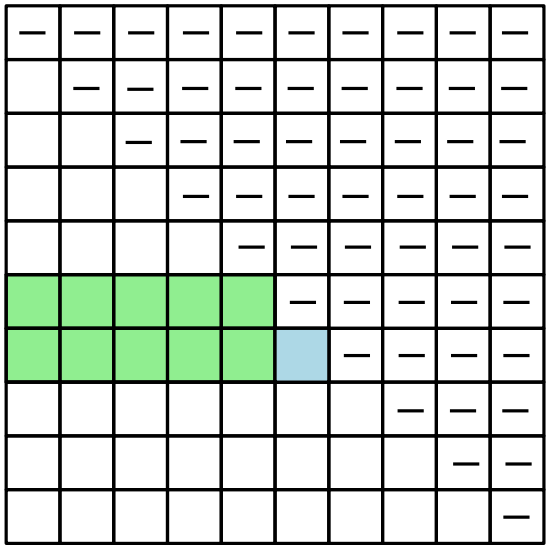


## Computing the next row

1. do point location in AW-VD to compute point $p_k$ defining $A[i, i-1]$
   - set $w_{i-1} := w_k + |p_k p_i|$
2. update data structure
   - add $|p_{i-1}p_i|$ to weight of each $p_j$ currently in data structure
   - insert $p_{i-1}$ with weight $w_{i-1}$

# The data structure

$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$



## Computing the next row

1. do point location in AW-VD to compute point $p_k$ defining $A[i, i-1]$
   - set $w_{i-1} := w_k + |p_k p_i|$
2. update data structure
   - add $|p_{i-1}p_i|$ to weight of each $p_j$ currently in data structure
   - insert $p_{i-1}$ with weight $w_{i-1}$

How can we do this quickly?

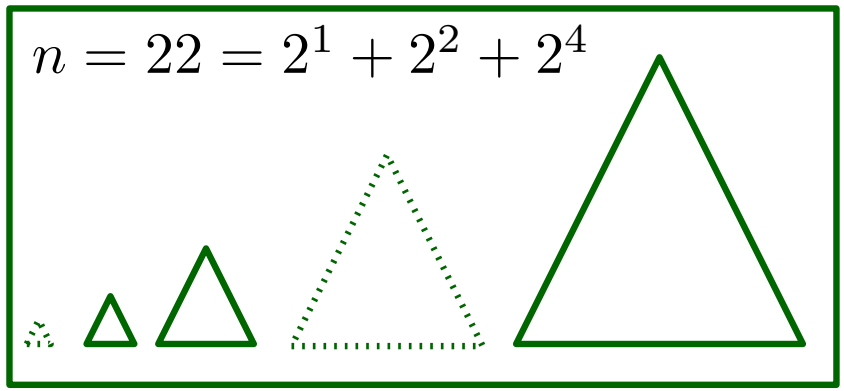
# The data structure

insertions and bulk updates in additively weighted Voronoi diagrams

# The data structure

insertions and bulk updates in additively weighted Voronoi diagrams

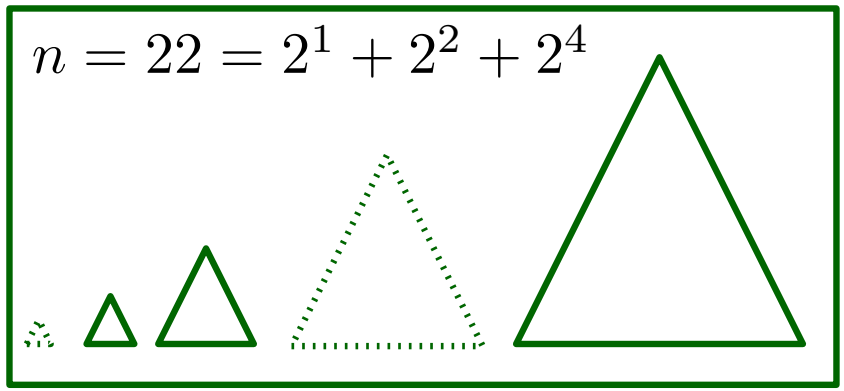
use logarithmic method $\Longrightarrow$ maintain $O(\log n)$ data structures on subsets of exponentially increasing sizes



# The data structure

insertions and bulk updates in additively weighted Voronoi diagrams

use logarithmic method $\Longrightarrow$ maintain $O(\log n)$ data structures on subsets of exponentially increasing sizes

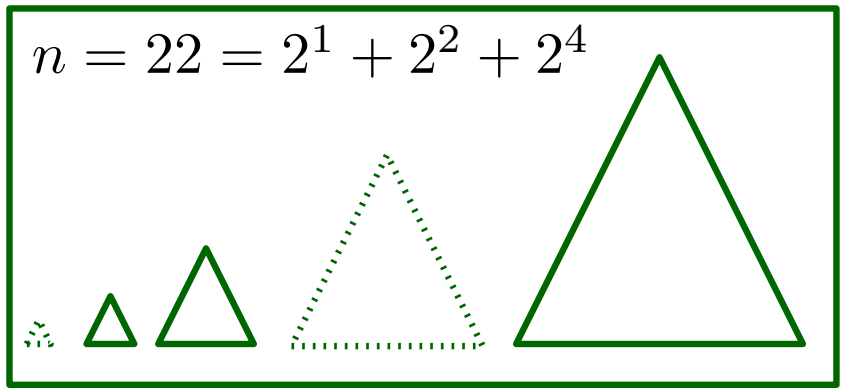



- query: do on all structures and take best answer

# The data structure

insertions and bulk updates in additively weighted Voronoi diagrams

use logarithmic method $\Longrightarrow$ maintain $O(\log n)$ data structures on subsets of exponentially increasing sizes

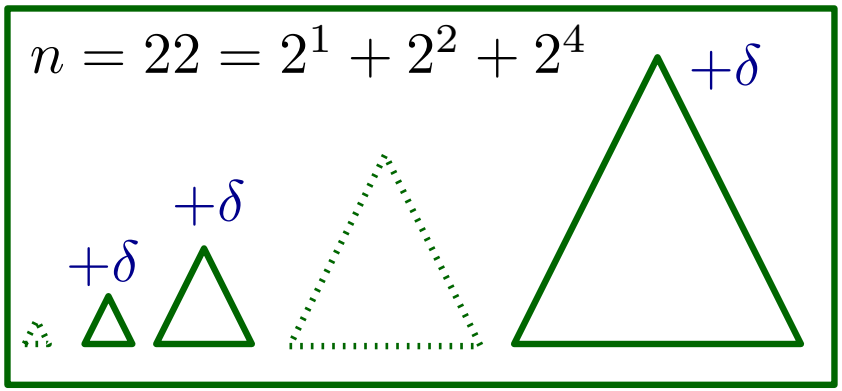



- query: do on all structures and take best answer
- insertion: find first empty structure, destroy all smaller structures, build new one from scratch

# The data structure

insertions and bulk updates in additively weighted Voronoi diagrams

use logarithmic method $\Longrightarrow$ maintain $O(\log n)$ data structures on subsets of exponentially increasing sizes

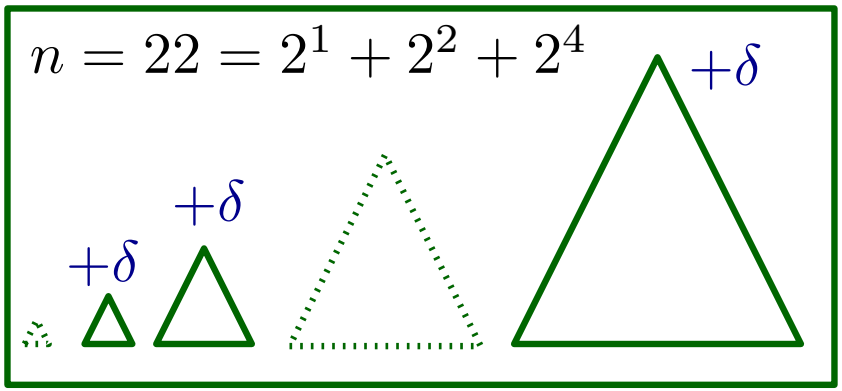



- query: do on all structures and take best answer
- insertion: find first empty structure, destroy all smaller structures, build new one from scratch
- bulk update: add correction term to each structure

# The data structure

insertions and bulk updates in additively weighted Voronoi diagrams

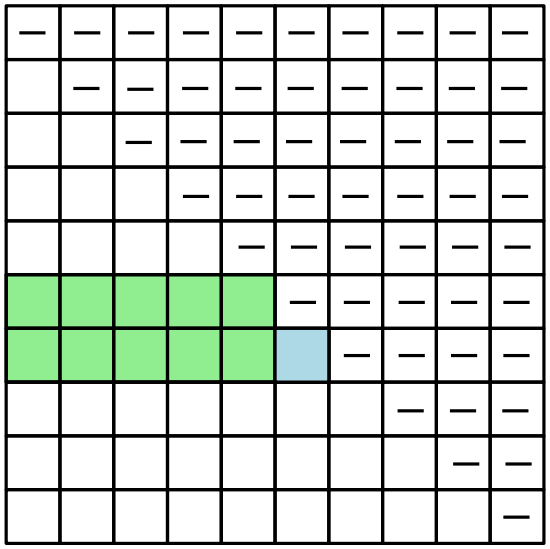
use logarithmic method $\Longrightarrow$ maintain $O(\log n)$ data structures on subsets of exponentially increasing sizes




- query: do on all structures and take best answer $O(\log n \cdot Q(n))$
- insertion: find first empty structure, destroy all smaller structures, build new one from scratch $O(\log n \cdot B(n)/n)$ amortized
- bulk update: add correction term to each structure $O(\log n)$

# The data structure

$$A[i,j] = \begin{cases} A[i-1,j] + |p_{i-1}p_i| & \text{if } 1 \leqslant j < i-1 \\ \min_{1 \leqslant k < i-1} \left(A[i-1,k] + |p_k p_i|\right) & \text{if } j = i-1 \end{cases}$$



## Computing the next row

1. do point location in AW-VD to compute point $p_k$ defining $A[i, i-1]$
   - set $w_{i-1} := w_k + |p_k p_i|$
2. update data structure
   - add $|p_{i-1}p_i|$ to weight of each $p_j$ currently in data structure
   - insert $p_{i-1}$ with weight $w_{i-1}$

$O(\log n)$

$O(\log^2 n)$

# Complexity of $k$-OPT

Theorem. Finding the shortest bitonic tour (or: pyramidal tour) on a set of $n$ points in the plane can be done in $O(n \log^2 n)$ time.

# Complexity of $k$-OPT

Theorem. Finding the shortest bitonic tour (or: pyramidal tour) on a set of $n$ points in the plane can be done in $O(n \log^2 n)$ time.

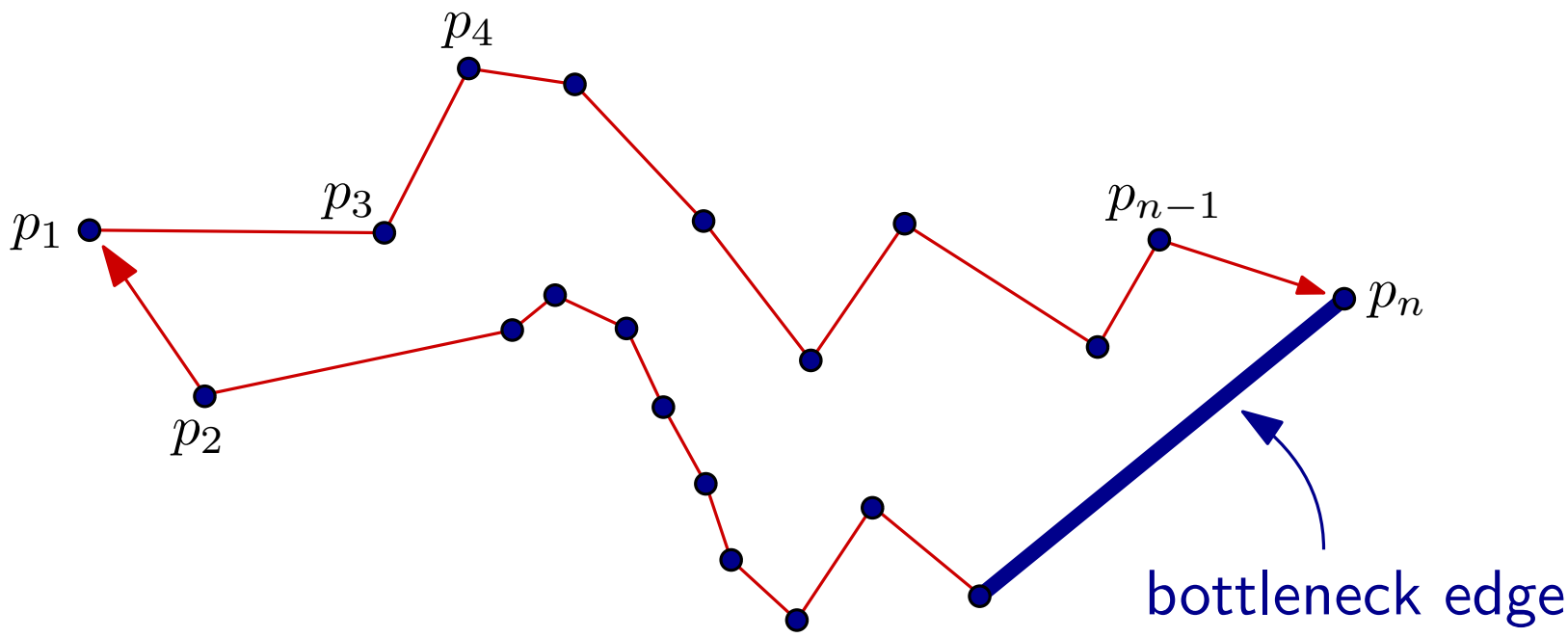
results on bottleneck bitonic TSP


- decision problem: $O(n \log n)$
- optimization problem: $O(n \log^3 n)$

# Bitonic Bottleneck TSP in the plane

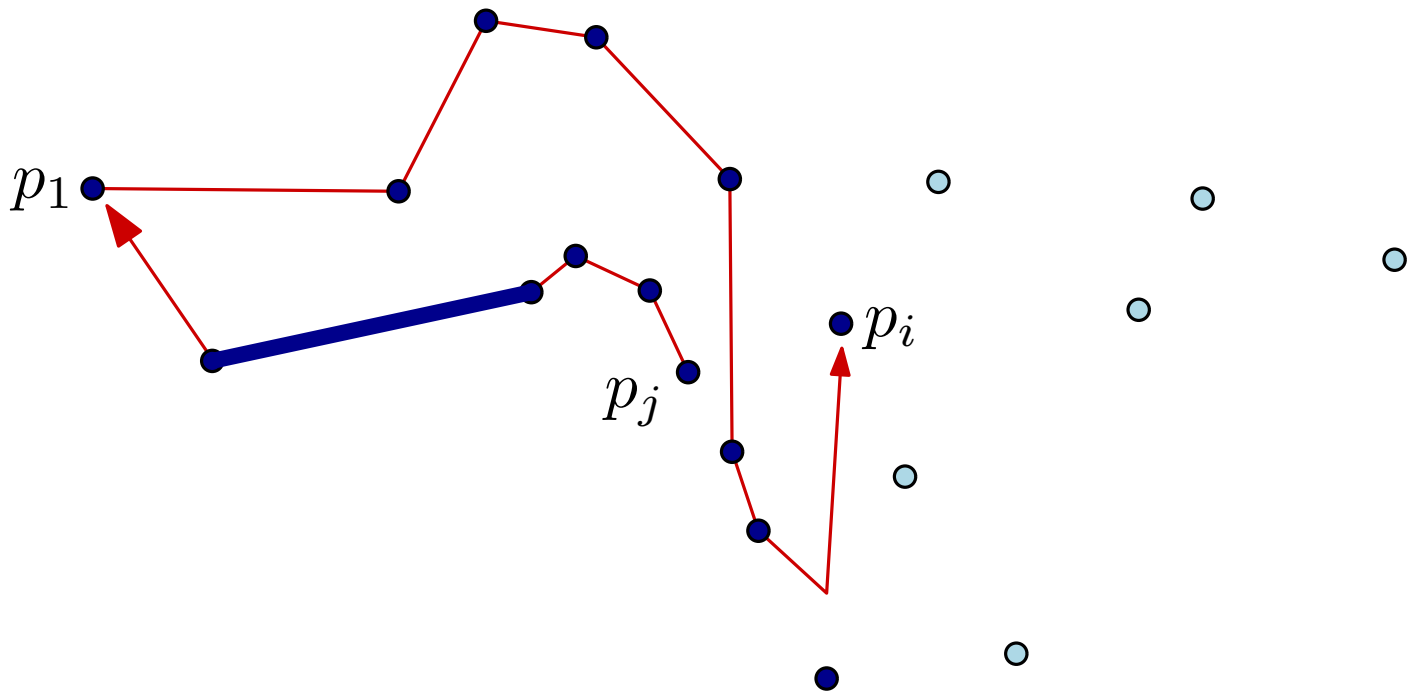
## Bitonic Bottleneck TSP — decision variant

- is there a tour whose bottleneck edge has length $\leqslant B$?
- only allow tours that go left-to-right and then right-to-left




$p_1, p_2, \ldots, p_n$: points numbered from left to right

# Bitonic TSP in the plane: standard DP solution




For each pair $p_i, p_j$ with $1 \leqslant j < i \leqslant n$, compute

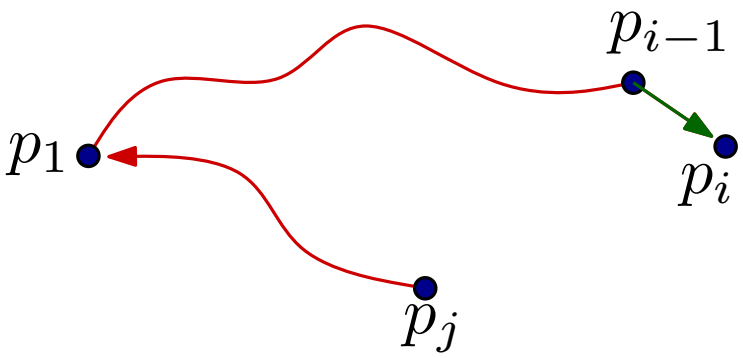
$A[i,j] =$ TRUE if there is a "partial solution" whose bottleneck edge has length $\leqslant B$

FALSE otherwise

# Bitonic TSP in the plane: standard DP solution

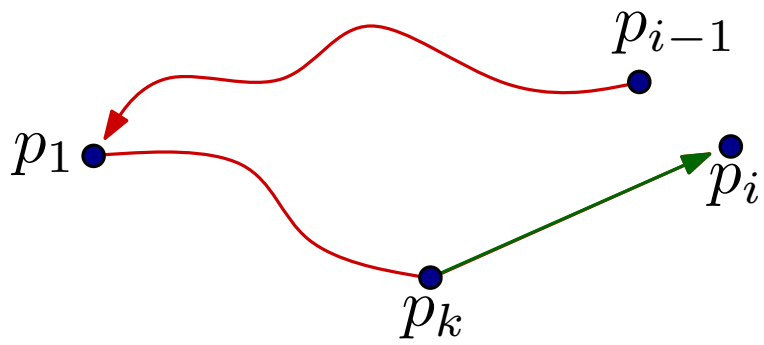
## Recursive formula for $A[i, j]$

case (i): $j < i - 1$




must connect $p_i$ to $p_i$

case (ii): $j = i - 1$



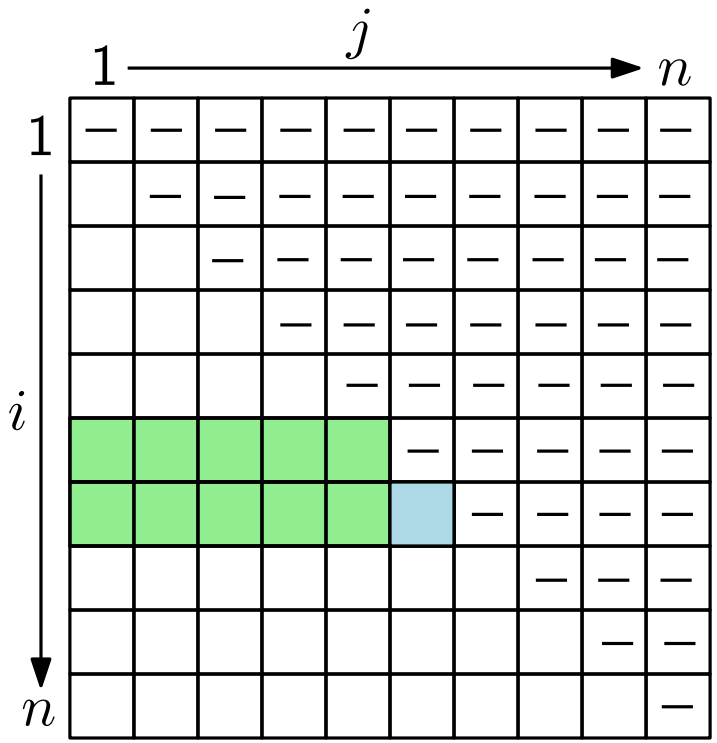

can connect to any $p_k$ with $k < i - 1$

$$A[i,j] = \begin{cases} A[i-1,j] \wedge (|p_{i-1}p_i| \leqslant B) & \text{if } 1 \leqslant j < i-1 \\ \vee_{1 \leqslant k < i-1} \left(A[i-1,k] \wedge (|p_k p_i| \leqslant B)\right) & \text{if } j = i-1 \end{cases}$$

where $A[2,1] = \textsc{True}$ if $|p_1 p_2| \leqslant B$.

# Bitonic TSP in the plane: our solution

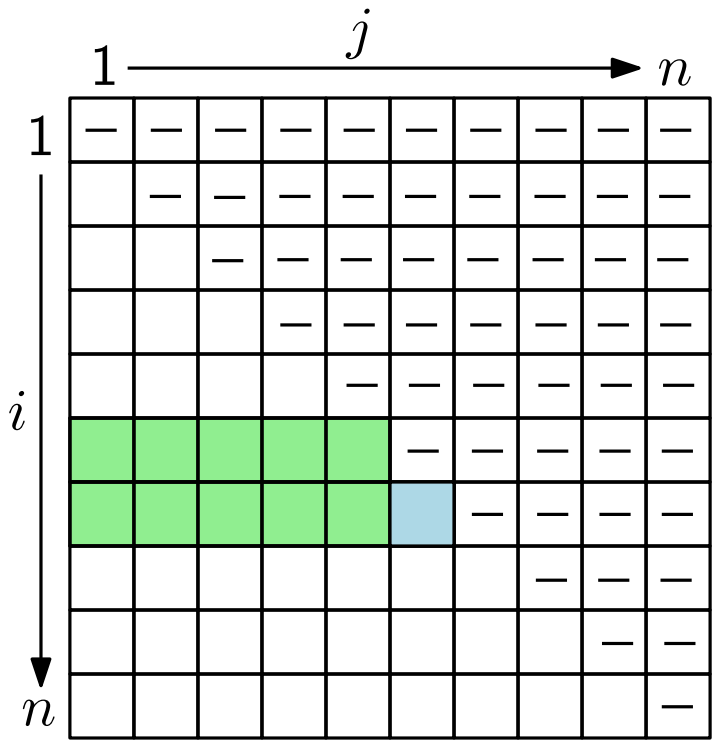
$$A[i,j] = \begin{cases} A[i-1,j] \wedge (|p_{i-1}p_i| \leqslant B) & \text{if } 1 \leqslant j < i-1 \\ \vee_{1 \leqslant k < i-1} \left(A[i-1,k] \wedge (|p_k p_i| \leqslant B)\right) & \text{if } j = i-1 \end{cases}$$




Implicit representation of $i$-th row:
set $S$ containing those points $p_j$
with $j < i$ such that $A[i,j]$=TRUE

# Bitonic TSP in the plane: our solution

$$A[i,j] = \begin{cases} A[i-1,j] \wedge (|p_{i-1}p_i| \leqslant B) & \text{if } 1 \leqslant j < i-1 \\ \vee_{1 \leqslant k < i-1} \left(A[i-1,k] \wedge (|p_k p_i| \leqslant B)\right) & \text{if } j = i-1 \end{cases}$$




Implicit representation of $i$-th row:
set $S$ containing those points $p_j$
with $j < i$ such that $A[i,j]$=TRUE

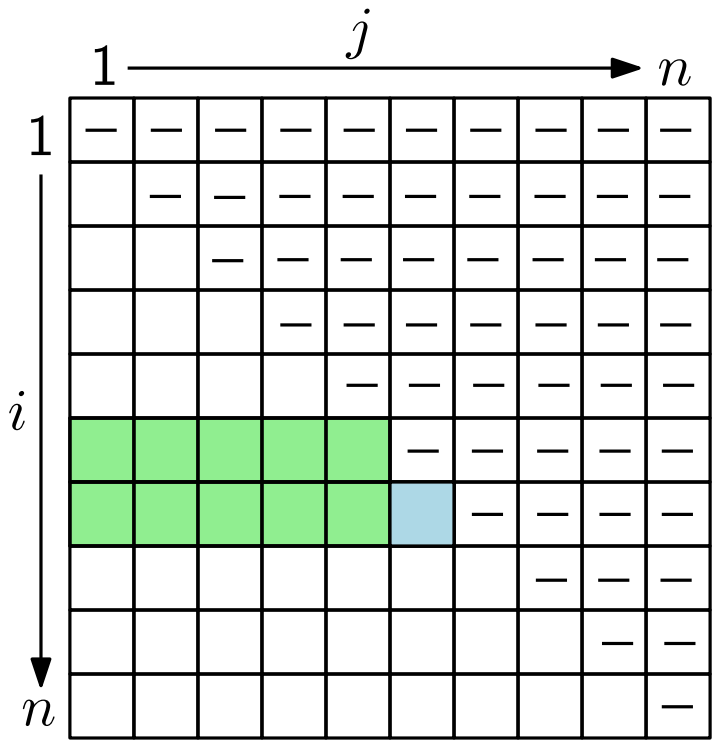
To handle $p_i$ we check if

(i) $|p_{i-1}p_i| \leqslant B$
(ii) there is a $p_k \in S$ with $|p_k p_i| \leqslant B$

answer to (i) is yes $\Longrightarrow$ keep $S$
answer to (i) is no $\Longrightarrow$ empty $S$
answer to (ii) is yes $\Longrightarrow$ add $p_i$ to $S$

# Bitonic TSP in the plane: our solution

$$A[i,j] = \begin{cases} A[i-1,j] \wedge (|p_{i-1}p_i| \leqslant B) & \text{if } 1 \leqslant j < i-1 \\ \vee_{1 \leqslant k < i-1} \left(A[i-1,k] \wedge (|p_k p_i| \leqslant B)\right) & \text{if } j = i-1 \end{cases}$$




data structure needed

Implicit representation of $i$-th row:
set $S$ containing those points $p_j$
with $j < i$ such that $A[i,j]$=TRUE

To handle $p_i$ we check if

(i) $|p_{i-1}p_i| \leqslant B$

(ii) there is a $p_k \in S$ with $|p_k p_i| \leqslant B$

answer to (i) is yes $\Longrightarrow$ keep $S$

answer to (i) is no $\Longrightarrow$ empty $S$

answer to (ii) is yes $\Longrightarrow$ add $p_i$ to $S$

# The data structure

- $S =$ set of point sites
- $B =$ threshold distance

queries: given point $q$, is there a point $p_k \in S$ with $|p_k q| \leqslant B$?

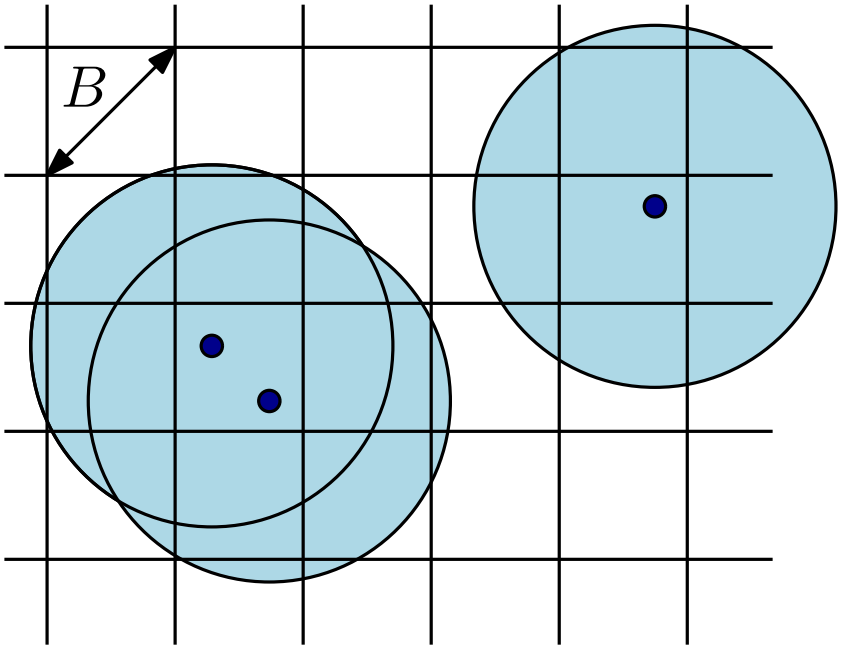
insertions must also be supported

# The data structure

- $S =$ set of point sites
- $B =$ threshold distance

queries: given point $q$, is there a point $p_k \in S$ with $|p_k q| \leqslant B$?

insertions must also be supported



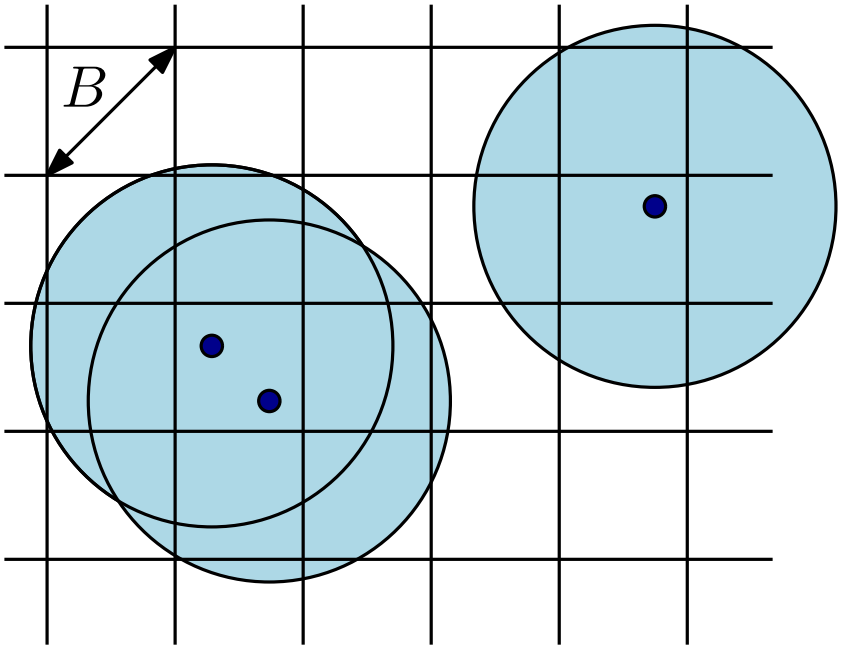

for each cell $C$: maintain point location structure on union of radius-$B$ disks centered at points in $S \cap C$

# The data structure

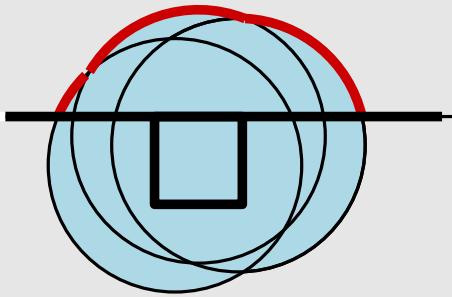
- $S =$ set of point sites
- $B =$ threshold distance

queries: given point $q$, is there a point $p_k \in S$ with $|p_k q| \leqslant B$?

insertions must also be supported




for each cell $C$: maintain point location structure on union of radius-$B$ disks centered at points in $S \cap C$



order of pieces on boundary $\equiv$ left-to-right order of centers

# The data structure

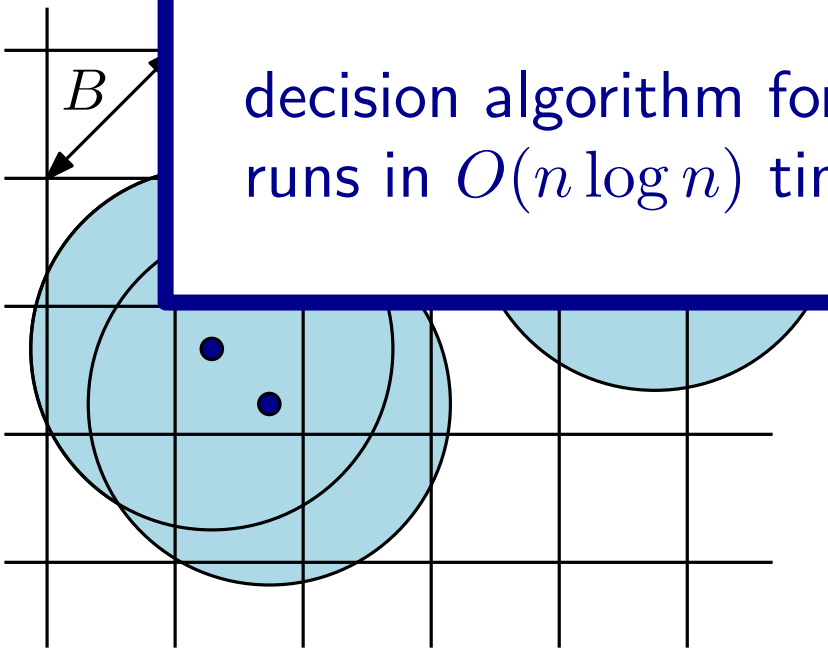
- $S =$ set of point sites
- $B =$ threshold distance

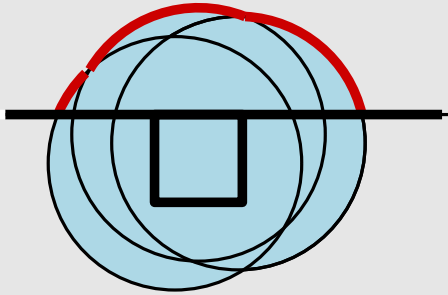
queries: g

insertions

- queries: $O(\log n)$ time
- insertions: $O(\log n)$ time amortized

decision algorithm for Bitonic Bottleneck TSP
runs in $O(n \log n)$ time



order of pieces on boundary ≡ left-to-right order of centers

# Our Results

## $k$-OPT in general setting

- $k = 3$: lower bound of $\Omega(n^{3-\varepsilon})$ under APSP Conjecture
- $k > 3$: algorithm with $O(n^{\lfloor 2k/3 \rfloor + 1})$ running time
- iterated $k$-OPT: $O(n^2)$ preprocessing, $O(n \log n)$ per iteration $(k = 2)$

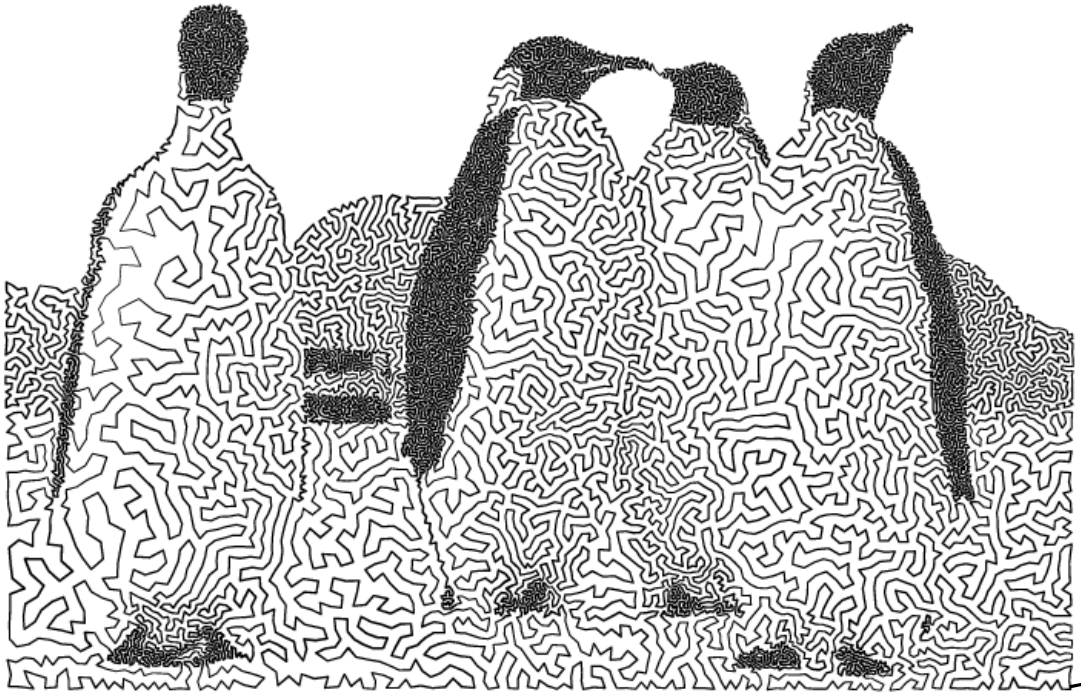
  [ $O(n^3)$ preprocessing, $O(n^2 \log n)$ per iteration $(k = 3)$ ]
- Euclidean setting: algorithm with $O(n^{8/5+\varepsilon})$ running time $(k = 2)$

  algorithm with $O(n^{80/31+\varepsilon})$ running time $(k = 3)$

## Bitonic (more generally: pyramidal) Euclidean TSP in the plane

- algorithm with $O(n \log^2 n)$ running time
- bottleneck variant: $O(n \log^3 n)$

# Thanks for your attention!



TSP Art by Carig Kaplan and Robert Bosch