Bitonic Tour Problem

Rivers

Nanjing University

September 15, 2018

In the **euclidean traveling-salesman problem**, we are given a set of n points in the plane, and we wish to find the shortest closed tour that connects all n points. Figure 15.11(a) shows the solution to a 7-point problem. The general problem is NP-hard, and its solution is therefore believed to require more than polynomial time (see Chapter 34).

J. L. Bentley has suggested that we simplify the problem by restricting our attention to **bitonic tours** , that is, tours that start at the leftmost point, go strictly rightward to the rightmost point, and then go strictly leftward back to the starting point.Figure 15.11(b) shows the shortest bitonic tour of the same 7 points. In this case, a polynomial-time algorithm is possible.

Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour. You may assume that no two points have the same x-coordinate and that all operations on real numbers take unit time. (*Hint: Scan left to right, maintaining optimal possibilities for the two parts of the tour.*)

J. L. Bentley has suggested that we simplify the problem by restricting our attention to **bitonic tours** , that is, tours that start at the leftmost point, go strictly rightward to the rightmost point, and then go strictly leftward back to the starting point.Figure 15.11(b) shows the shortest bitonic tour of the same 7 points. In this case, a polynomial-time algorithm is possible.
Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour. You may assume that no two points have the same x-coordinate and that all operations on real numbers take unit time. (*Hint: Scan left to right, maintaining optimal possibilities for the two parts of the tour.*)

Restricting our attention to **bitonic tours**

Tours that start at the leftmost point, go strictly rightward to the rightmost point, and then go strictly leftward back to the starting point.

Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour.

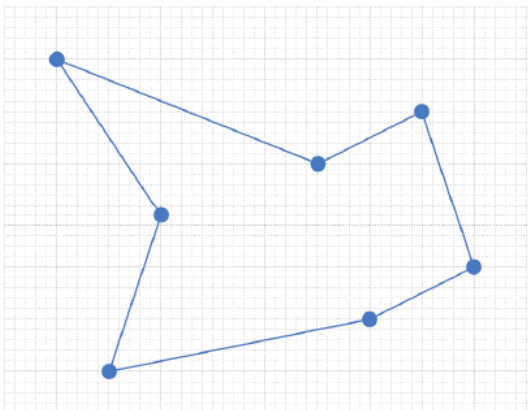


Figure: Tour

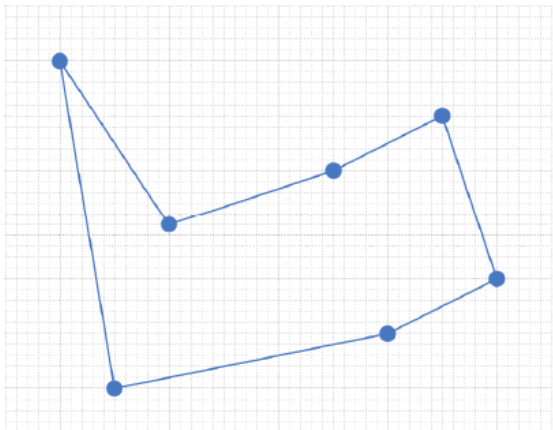


Figure: Bitonic Tour

You may assume that no two points have the same x-coordinate and that all operations on real numbers take unit time.

Hint: Scan left to right, maintaining optimal possibilities for the two parts of the tour.

What does ‘Bitonic’ mean?

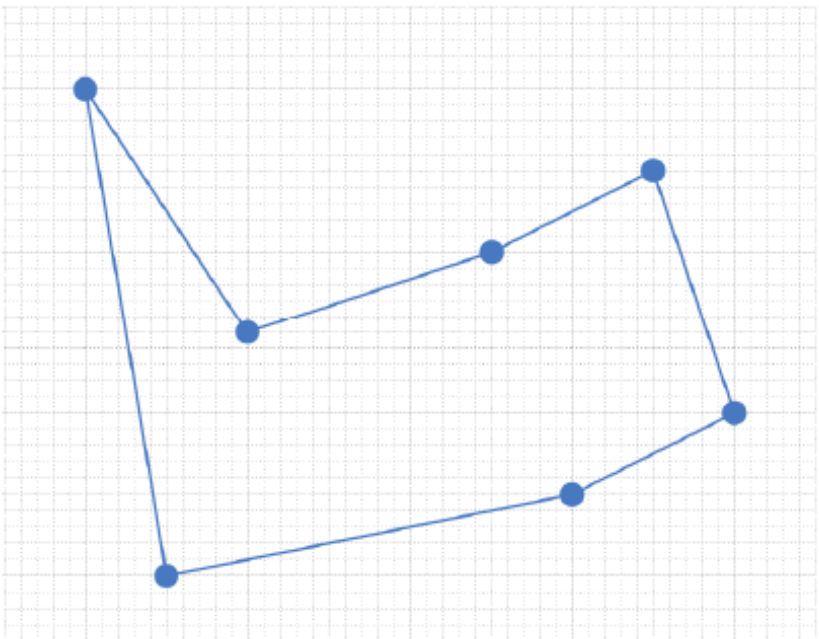
What does 'Bitonic' mean?

Bitonic

(computing theory, of a sequence) Having the property $x_0 \leq \cdots \leq x_k \geq \cdots \geq x_{n-1}$ for some $k, 0 \leq k < n$, or being a circular shift of such a sequence.

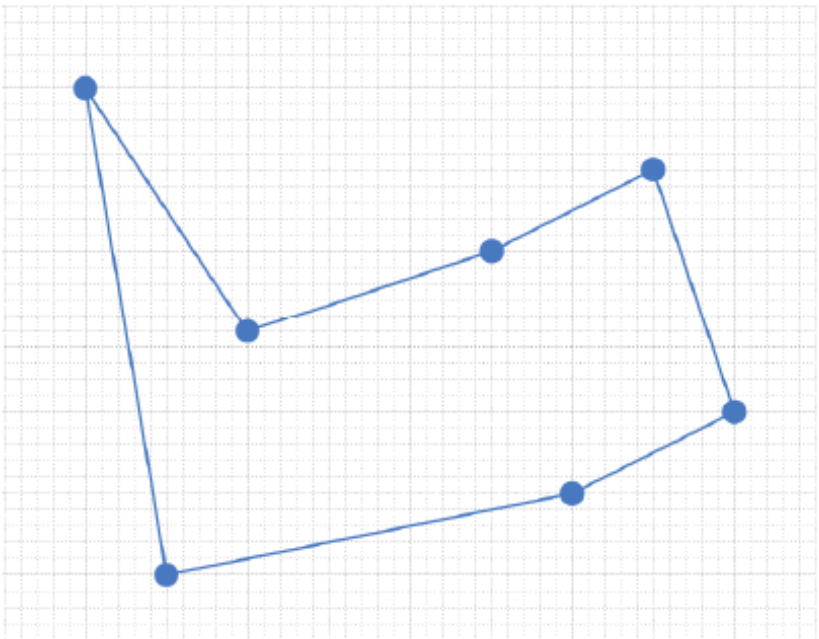
A set of points $\{p_1, p_2, \ldots, p_n\}$

A set of points $\{p_1, p_2, \ldots, p_n\}$
The shortest bitonic path $P_{i,j}$, where $i \leq j$, includes all points $p_1, p_2, \ldots, p_j$; it starts at some point p_i, goes strictly left to point p1, and then goes strictly right to point p_j.



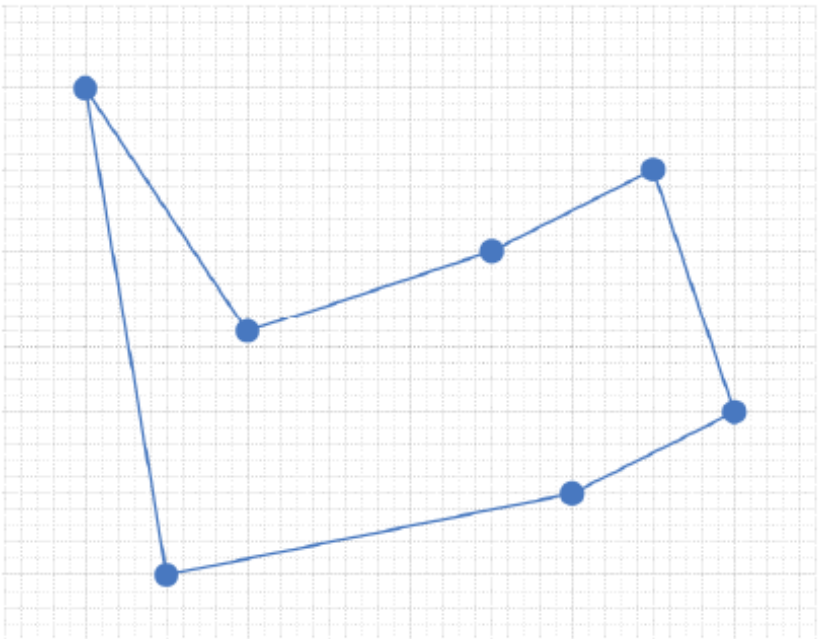
A set of points $\{p_1, p_2, \ldots, p_n\}$
The shortest bitonic path $P_{i,j}$, where $i \leq j$, includes all points $p_1, p_2, \ldots, p_j$; it starts at some point p_i, goes strictly left to point p1, and then goes strictly right to point p_j.

$$P_{n,n}$$



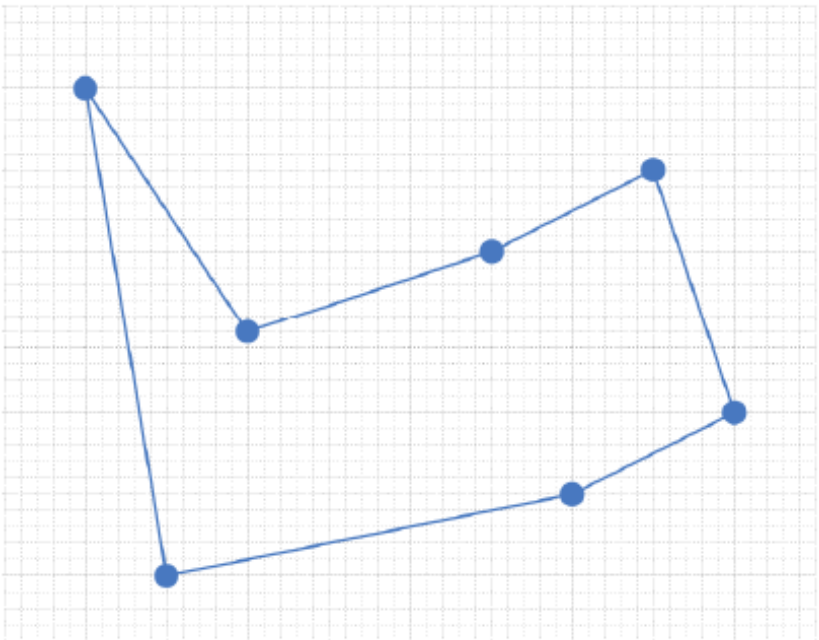
A set of points $\{p_1, p_2, \ldots, p_n\}$
The shortest bitonic path $P_{i,j}$, where $i \leq j$, includes all points $p_1, p_2, \ldots, p_j$; it starts at some point p_i, goes strictly left to point p1, and then goes strictly right to point p_j.

$$P_{n,n}=P_{n-1,n} + l_{n-1,n}$$



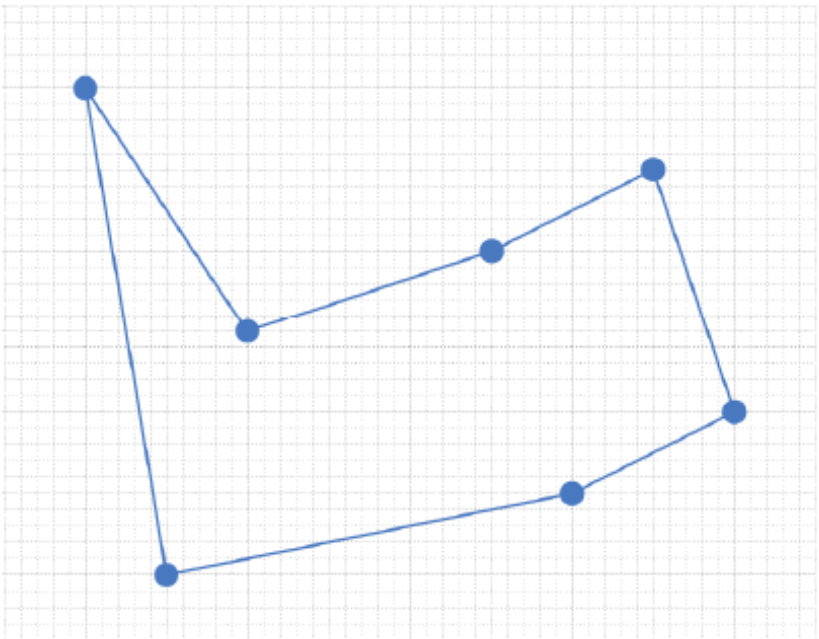
Optimal Substructure

$P_{i,j}(i < j)$ is the shortest bitonic tour.
(1). $i < j - 1$, remove the line $p_{j-1}p_j$, it becomes $P_{i,j-1}$.
(2). $i = j - 1$. let k be the predecessor of p_j. Remove $p_k p_j$ and it becomes $P_{k,j-1}$.



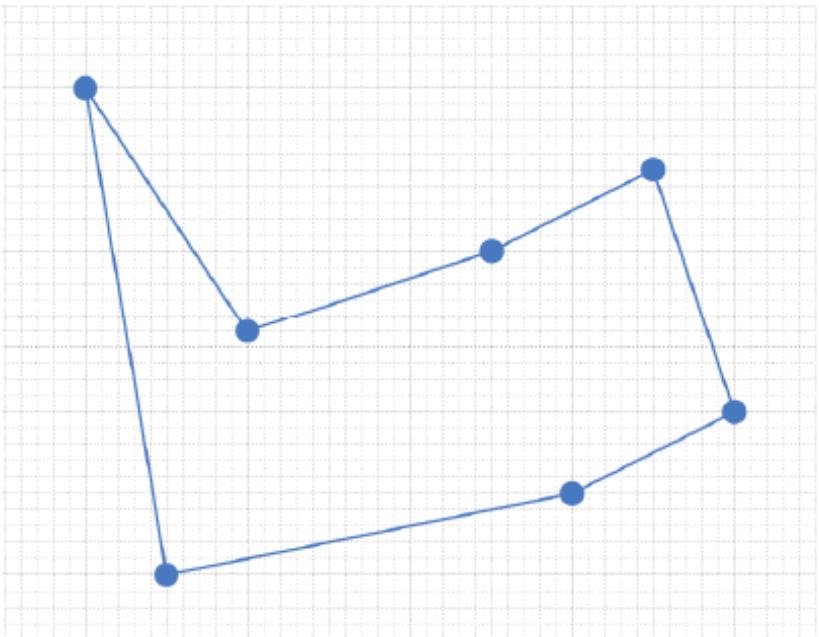
Case 1: $i < j - 1$

$i < j - 1$. But $P_{i,j}$ includes all points $p_1, \ldots, p_{j-1}, p_j$. p_{j-1} is the predecessor of p_j.
cut-and-paste



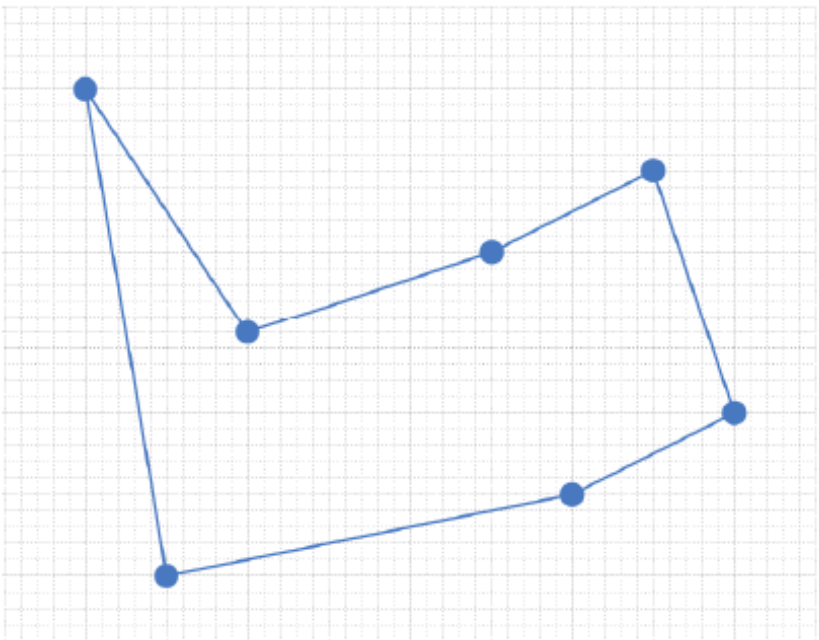
Case 2: $i = j - 1$

$i = j - 1$. p_j must have a predecessor $p_k (1 \leq k \leq j - 2)$.
cut-and-paste



Case 2: $i = j - 1$

$i = j - 1$. p_j must have a predecessor $p_k (1 \leq k \leq j - 2)$.
cut-and-paste
Why $P_{k,j-1}$? Why not $P_{j-1,k}$?



$$P[i,j] = \begin{cases} l[1,2] & i = 1 \text{ and } j = 2 \\ P[i,j-1] + l[j-1,j] & i < j-1 \\ \min_{1 \le k \le j-2} P[k,j-1] + l[k,j] & i = j-1 \end{cases}$$

$$P[i,j] = \begin{cases} l[1,2] & i = 1 \text{ and } j = 2 \\ P[i,j-1] + l[j-1,j] & i < j-1 \\ \min_{1 \le k \le j-2} P[k,j-1] + l[k,j] & i = j-1 \end{cases}$$

$$P[1,j] = \sum_{k=2}^{j} P[k-1,k]?$$

EUCLIDEAN-TSP(p)

```
 1  SORT(p)
 2  let b[1..n, 2..n] and r[1..n − 2, 3..n] be new arrays
 3  b[1, 2] = LEN(p[1], p[2])
 4  for j = 3 to n
 5      for i = 1 to j − 2
 6          b[i, j] = b[i, j − 1] + LEN(p_{j−1}, p_j)
 7          r[i, j] = j − 1
 8      b[j − 1, i] = ∞
 9      for k = 1 to j − 2
10          q = b[k, j − 1] + LEN(p_k, p_j)
11          if q < b[j, j − 1]
12              b[j − 1, j] = q
13              r[j − 1, j] = k
14  b[n, n] = b[n − 1, n] + LEN(p_{n−1}, p_n)
15  return b and r
```

```
PRINT-TOUR(R,N)
1  print p[n]
2  print p[n − 1]
3  k = r[n − 1, n]
4  PRINT-PATH
5  (r, k, n − 1)
6  print p[k]
```

```
PRINT-PATH(r, i, j)
 1  if i < j
 2        k = r[i, j]
 3        if k ≠ i
 4              print p[k]
 5        if k > 1
 6              PRINT-PATH(r, i, k)
 7  else
 8        k = r[j, i]
 9        if k > 1
10              PRINT-PATH(r, k, j)
11              print p[k]
```


Thank
you!!