Randomised Algorithms
Lecture 8: Solving a TSP Instance using Linear Programming

Thomas Sauerwald (tms41@cam.ac.uk)
Outline

Introduction

Examples of TSP Instances

Demonstration
The Traveling Salesman Problem (TSP)

Given a set of cities along with the cost of travel between them, find the cheapest route visiting all cities and returning to your starting point.

Formal Definition

- **Given:** A complete undirected graph $G = (V, E)$ with nonnegative integer cost $c(u, v)$ for each edge $(u, v) \in E$
- **Goal:** Find a Hamiltonian cycle of $G$ with minimum cost.

Solution space consists of at most $n!$ possible tours!

Actually the right number is $(n - 1)!/2$

Special Instances

- **Metric TSP:** costs satisfy triangle inequality:

  $\forall u, v, w \in V : \quad c(u, w) \leq c(u, v) + c(v, w)$.

- **Euclidean TSP:** cities are points in the Euclidean space, costs are equal to their (rounded) Euclidean distance.

Even this version is NP hard (Ex. 35.2-2)
Outline

Introduction

Examples of TSP Instances

Demonstration
The traveling salesman problem recently achieved national prominence when a soap company used it as the basis of a promotional contest. Prizes up to $10,000 were offered.

Thus Flood realized that the Nearest Neighbor method is not a good estimate of the TSP but it created a decent first solution.
532 cities (1987 [Padberg, Rinaldi])
13,509 cities (1999 [Applegate, Bixby, Chavatal, Cook])
SOLUTION OF A LARGE-SCALE TRAVELING-SALESMAN PROBLEM*

G. DANTZIG, R. FULKERSON, AND S. JOHNSON
The Rand Corporation, Santa Monica, California
(Received August 9, 1954)

It is shown that a certain tour of 49 cities, one in each of the 48 states and Washington, D. C., has the shortest road distance.

THE TRAVELING-SALESMAN PROBLEM might be described as follows: Find the shortest route (tour) for a salesman starting from a given city, visiting each of a specified group of cities, and then returning to the original point of departure. More generally, given an $n$ by $n$ symmetric matrix $D = (d_{ij})$, where $d_{ij}$ represents the 'distance' from $I$ to $J$, arrange the points in a cyclic order in such a way that the sum of the $d_{ij}$ between consecutive points is minimal. Since there are only a finite number of possibilities (at most $\frac{1}{2} (n-1)!$) to consider, the problem is to devise a method of picking out the optimal arrangement which is reasonably efficient for fairly large values of $n$. Although algorithms have been devised for problems of similar nature, e.g., the optimal assignment problem, little is known about the traveling-salesman problem. We do not claim that this note alters the situation very much; what we shall do is outline a way of approaching the problem that sometimes, at least, enables one to find an optimal path and prove it so. In particular, it will be shown that a certain arrangement of 49 cities, one in each of the 48 states and Washington, D. C., is best, the $d_{ij}$ used representing road distances as taken from an atlas.
### The 42 (49) Cities

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10. Minneapolis, Minn.</td>
<td></td>
<td>A. Baltimore, Md.</td>
</tr>
<tr>
<td>13. Helena, Mont.</td>
<td></td>
<td>D. Newark, N. J.</td>
</tr>
<tr>
<td>15. Portland, Ore.</td>
<td></td>
<td>F. Hartford, Conn.</td>
</tr>
<tr>
<td>16. Boise, Idaho</td>
<td></td>
<td>G. Providence, R. I.</td>
</tr>
<tr>
<td>17. Salt Lake City, Utah</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Combinatorial Explosion

8. Solving TSP via Linear Programming © T. Sauerwald

Examples of TSP Instances
Solution of this TSP problem

Dantzig, Fulkerson and Johnson found an optimal tour through 42 cities.

http://www.math.uwaterloo.ca/tsp/history/img/dantzig_big.html
Hence this is an instance of the **Metric TSP**, but not **Euclidean TSP**.

### TABLE I

**Road Distances between Cities in Adjusted Units**

The figures in the table are mileages between the two specified numbered cities, less 11, divided by 17, and rounded to the nearest integer.
Modelling TSP as a Linear Program Relaxation

Idea: Indicator variable $x(i, j)$, $i > j$, which is one if the tour includes edge $\{i, j\}$ (in either direction)

minimize $\sum_{i=1}^{42} \sum_{j=1}^{i-1} c(i, j) x(i, j)$
subject to

$\sum_{j<i} x(i, j) + \sum_{j>i} x(j, i) = 2$ for each $1 \leq i \leq 42$

$0 \leq x(i, j) \leq 1$ for each $1 \leq j < i \leq 42$

Constraints $x(i, j) \in \{0, 1\}$ are not allowed in a LP!

Branch & Bound to solve an Integer Program:
- As long as solution of LP has fractional $x(i, j) \in (0, 1)$:
  - Add $x(i, j) = 0$ to the LP, solve it and recurse
  - Add $x(i, j) = 1$ to the LP, solve it and recurse
  - Return best of these two solutions
- If solution of LP integral, return objective value

Bound-Step: If the best known integral solution so far is better than the solution of a LP, no need to explore branch further!
Outline

Introduction

Examples of TSP Instances

Demonstration
In the following, there are a few different runs of the demo.
Iteration 1: Eliminate Subtour 1, 2, 41, 42

Objective value: \(-641.000000\), 861 variables, 945 constraints, 1809 iterations

Disallow subtour \((1, 2, 42, 41)\) by adding this constraint to the LP:

\[ x(2, 1) + x(41, 1) + x(42, 1) + x(41, 2) + x(42, 2) + x(42, 41) \leq 3 \]

Equivalent to:

\[ S = \{1, 2, 41, 42\}, \quad \sum_{i \in S, j \in V \setminus S} x(\max(i, j), \min(i, j)) \geq 2 \]
Iteration 2: Eliminate Subtour 3 – 9

Objective value: $-676.000000$, 861 variables, 946 constraints, 1802 iterations
Iteration 3: Eliminate Subtour 24, 25, 26, 27

Objective value: \(-681.000000\), 861 variables, 947 constraints, 1984 iterations
Iteration 4: Eliminate Cut 11 – 23

Objective value: $-682.500000$, 861 variables, 948 constraints, 1492 iterations

Tour has to include at least two edges between $S = \{11, 12, \ldots, 23\}$ and $V \setminus S$:

$$\sum_{i \in S, j \in V \setminus S} x(\max(i, j), \min(i, j)) \geq 2.$$
Iteration 5: Eliminate Subtour 13 – 23

Objective value: $-686.000000$, 861 variables, 949 constraints, 2446 iterations
Iteration 6: Eliminate Cut 13 – 17

Objective value: $-694.500000$, 861 variables, 950 constraints, 1690 iterations
Iteration 7: Branch 1a $x_{18,15} = 0$

Objective value: $-697.000000$, 861 variables, 951 constraints, 2212 iterations
Iteration 8: Branch 2a $x_{17,13} = 0$

Objective value: $-698.000000$, 861 variables, 952 constraints, 1878 iterations
Iteration 9: Branch 2b $x_{17,13} = 1$

Objective value: $-699.000000$, 861 variables, 953 constraints, 2281 iterations
Iteration 10: Branch 1b $x_{18,15} = 1$

Objective value: $-700.000000$, 861 variables, 954 constraints, 2398 iterations

Branch & Bound procedure would stop here, since value of the best LP solution for $x_{18,15} = 0$ is worse than a previously found tour.
Iteration 11: Branch & Bound terminates

Objective value: $-701.000000$, 861 variables, 953 constraints, 2506 iterations
### Branch & Bound Overview

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LP solution 641</td>
</tr>
<tr>
<td>2</td>
<td>Eliminate Subtour 1, 2, 41, 42</td>
</tr>
<tr>
<td>3</td>
<td>LP solution 676</td>
</tr>
<tr>
<td>4</td>
<td>Eliminate Subtour 3 – 9</td>
</tr>
<tr>
<td>5</td>
<td>LP solution 681</td>
</tr>
<tr>
<td>6</td>
<td>Eliminate Subtour 24, 25, 26, 27</td>
</tr>
<tr>
<td>7</td>
<td>LP solution 682.5</td>
</tr>
<tr>
<td>8</td>
<td>Eliminate Cut 11 – 23</td>
</tr>
<tr>
<td>9</td>
<td>LP solution 686</td>
</tr>
<tr>
<td>10</td>
<td>Eliminate Subtour 10, 11, 12</td>
</tr>
<tr>
<td>11</td>
<td>LP solution 686</td>
</tr>
<tr>
<td>12</td>
<td>Eliminate Subtour 10, 11, 12</td>
</tr>
<tr>
<td>13</td>
<td>LP solution 694.5</td>
</tr>
<tr>
<td>14</td>
<td>Eliminate Cut 13 – 17</td>
</tr>
<tr>
<td>15</td>
<td>LP solution 697</td>
</tr>
<tr>
<td>16</td>
<td>Eliminate Cut 13 – 17</td>
</tr>
<tr>
<td>17</td>
<td>LP solution 697</td>
</tr>
<tr>
<td>18</td>
<td>x_{18,15} = 0</td>
</tr>
<tr>
<td>19</td>
<td>x_{18,15} = 1</td>
</tr>
<tr>
<td>20</td>
<td>x_{17,13} = 0</td>
</tr>
<tr>
<td>21</td>
<td>x_{17,13} = 1</td>
</tr>
<tr>
<td>22</td>
<td>Valid tour 698</td>
</tr>
<tr>
<td>23</td>
<td>Valid tour 699</td>
</tr>
<tr>
<td>24</td>
<td>LP solution 700</td>
</tr>
<tr>
<td>25</td>
<td>Cut branch, since LP solution worse than current best possible tour.</td>
</tr>
<tr>
<td>26</td>
<td>Valid tour 701</td>
</tr>
</tbody>
</table>

8. Solving TSP via Linear Programming © T. Sauerwald
What about choosing a different branching variable?
Solving Progress (Alternative Branch 1)

1. LP solution 641
   Eliminate Subtour 1, 2, 41, 42
2. LP solution 676
   Eliminate Subtour 3 – 9
3. LP solution 681
   Eliminate Subtour 24, 25, 26, 27
4. LP solution 682.5
   Eliminate Cut 13 – 17
5. LP solution 686
   Eliminate Subtour 10, 11, 12
6. LP solution 686
   Eliminate Subtour 13 – 23
7. LP solution 688
   Eliminate Subtour 11 — 23
8. LP solution 697

\[ x_{15,18} = 1 \]
\[ x_{15,18} = 0 \]
9. ???
10. ???

8. Solving TSP via Linear Programming © T. Sauerwald

Demonstration
Alternative Branch 1: $x_{18,15}$, Objective 697
Alternative Branch 1a: $x_{18,15} = 1$, Objective 701 (Valid Tour)
Alternative Branch 1b: \( x_{18,15} = 0 \), Objective 698
Solving Progress (Alternative Branch 1)

1: LP solution 641
   Eliminate Subtour 1, 2, 41, 42

2: LP solution 676
   Eliminate Subtour 3 − 9

3: LP solution 681
   Eliminate Subtour 24, 25, 26, 27

4: LP solution 682.5
   Eliminate Cut 13 − 17

5: LP solution 686
   Eliminate Subtour 10, 11, 12

6: LP solution 686
   Eliminate Subtour 13 − 23

7: LP solution 688
   Eliminate Subtour 11 − 23

8: LP solution 697
   \[ x_{18,15} = 1 \]

9: valid tour 701

10: LP solution 698
   \[ x_{18,15} = 0 \]
Solving Progress (Alternative Branch 2)

1: LP solution 641
   Eliminate Subtour 1, 2, 41, 42
2: LP solution 676
   Eliminate Subtour 3 – 9
3: LP solution 681
   Eliminate Subtour 24, 25, 26, 27
4: LP solution 682.5
   Eliminate Cut 13 – 17
5: LP solution 686
   Eliminate Subtour 10, 11, 12
6: LP solution 686
   Eliminate Subtour 13 – 23
7: LP solution 688
   Eliminate Subtour 11 – 23
8: LP solution 697
   $x_{27,22} = 1$
   $x_{27,22} = 0$
9: ???
10: ???

8. Solving TSP via Linear Programming © T. Sauerwald Demonstration
Alternative Branch 2: $x_{27,22}$, Objective 697
Alternative Branch 2a: $x_{27,22} = 1$, Objective 708 (Valid tour)
Alternative Branch 2b: $x_{27,22} = 0$, Objective 697.75
Solving Progress (Alternative Branch 2)

1: LP solution 641
   - Eliminate Subtour 1, 2, 41, 42

2: LP solution 676
   - Eliminate Subtour 3 – 9

3: LP solution 681
   - Eliminate Subtour 24, 25, 26, 27

4: LP solution 682.5
   - Eliminate Cut 13 – 17

5: LP solution 686
   - Eliminate Subtour 10, 11, 12

6: LP solution 686
   - Eliminate Subtour 13 – 23

7: LP solution 688
   - Eliminate Subtour 11 – 23

8: LP solution 697
   - \( x_{27,22} = 1 \)
   - \( x_{27,22} = 0 \)

9: valid tour 708

10: LP solution 697.75
   - Connected valid tour
Solving Progress (Alternative Branch 3)

1: LP solution 641
   Eliminate Subtour 1, 2, 41, 42

2: LP solution 676
   Eliminate Subtour 3 – 9

3: LP solution 681
   Eliminate Subtour 24, 25, 26, 27

4: LP solution 682.5
   Eliminate Cut 13 – 17

5: LP solution 686
   Eliminate Subtour 10, 11, 12

6: LP solution 686
   Eliminate Subtour 13 – 23

7: LP solution 688
   Eliminate Subtour 11 – 23

8: LP solution 697
   \[ x_{27,24} = 1 \]
   \[ x_{27,24} = 0 \]

9: ???
10: ???
Alternative Branch 3: $x_{27,24}$, Objective 697
Alternative Branch 3a: $x_{27,24} = 1$, Objective 697.75
Alternative Branch 3b: $x_{27,24} = 0$, Objective 698
Solving Progress (Alternative Branch 3)

1: LP solution 641
   Eliminate Subtour 1, 2, 41, 42

2: LP solution 676
   Eliminate Subtour 3 – 9

3: LP solution 681
   Eliminate Subtour 24, 25, 26, 27

4: LP solution 682.5

Not only do we have to explore (and branch further in) both subtrees, but also the optimal tour is in the subtree with larger LP solution!

5: LP solution 686
   Eliminate Subtour 13 – 23

6: LP solution 688
   Eliminate Subtour 11 – 23

7: LP solution 688

8: LP solution 697
   \[ x_{27,24} = 1 \]

9: LP solution 697.75

10: LP solution 698
    \[ x_{27,24} = 0 \]
Conclusion (1/2)

- How can one generate these constraints automatically?
  Subtour Elimination: Finding Connected Components
  Small Cuts: Finding the Minimum Cut in Weighted Graphs

- Why don’t we add all possible Subtour Elimination constraints to the LP?
  There are exponentially many of them!

- Should the search tree be explored by BFS or DFS?
  BFS may be more attractive, even though it might need more memory.

CONCLUDING REMARK

It is clear that we have left unanswered practically any question one might pose of a theoretical nature concerning the traveling-salesman problem; however, we hope that the feasibility of attacking problems involving a moderate number of points has been successfully demonstrated, and that perhaps some of the ideas can be used in problems of similar nature.
Conclusion (2/2)

- Eliminate Subtour 1, 2, 41, 42
- Eliminate Subtour 3 – 9
- **Eliminate Subtour 10, 11, 12**
- Eliminate Subtour 11 – 23
- Eliminate Subtour 13 – 23
- Eliminate Cut 13 – 17
- Eliminate Subtour 24, 25, 26, 27

THE 49-CITY PROBLEM*

The optimal tour $\bar{x}$ is shown in Fig. 16. The proof that it is optimal is given in Fig. 17. To make the correspondence between the latter and its programming problem clear, we will write down in addition to 42 relations in non-negative variables (2), a set of 25 relations which suffice to prove that $D(x)$ is a minimum for $\bar{x}$. We distinguish the following subsets of the 42 cities:

- $S_1 = \{1, 2, 41, 42\}$
- $S_2 = \{3, 4, \ldots, 9\}$
- $S_3 = \{1, 2, \ldots, 9, 29, 30, \ldots, 42\}$
- $S_4 = \{11, 12, \ldots, 23\}$
- $S_5 = \{13, 14, \ldots, 23\}$
- $S_6 = \{13, 14, 15, 16, 17\}$
- $S_7 = \{24, 25, 26, 27\}$. 
From Wikipedia, the free encyclopedia

**IBM ILOG CPLEX Optimization Studio** (often informally referred to simply as CPLEX) is an optimization software package. In 2004, the work on CPLEX earned the first INFORMS Impact Prize.

The CPLEX Optimizer was named for the simplex method as implemented in the C programming language, although today it also supports other types of mathematical optimization and offers interfaces other than just C. It was originally developed by Robert E. Bixby and was offered commercially starting in 1988 by CPLEX Optimization Inc., which was acquired by ILOG in 1997; ILOG was subsequently acquired by IBM in January 2009. CPLEX continues to be actively developed under IBM.

The IBM ILOG CPLEX Optimizer solves integer programming problems, very large linear programming problems using either primal or dual variants of the simplex method or the barrier interior
Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.6.1.0
with Simplex, Mixed Integer & Barrier Optimizers
5725-A06 5725-A29 5725-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corp. 1988, 2014. All Rights Reserved.

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more information on commands.

CPLEX> read tsp.lp
Problem 'tsp.lp' read.
Read time = 0.00 sec. (0.06 ticks)
CPLEX> primopt
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 49 rows, 860 columns, and 2483 nonzeros.
Presolve time = 0.00 sec. (0.36 ticks)

Iteration log . . .
Iteration:  1  Infeasibility = 33.999999
Iteration:  26  Objective = 1510.000000
Iteration:  90  Objective = 923.000000
Iteration: 155  Objective = 711.000000

Primal simplex - Optimal: Objective = 6.9900000000e+02
Solution time =  0.00 sec.  Iterations = 168 (25)
Deterministic time = 1.16 ticks (288.86 ticks/sec)

CPLEX>
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Solution Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_2_1</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_42_1</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_3_2</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_4_3</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_5_4</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_6_5</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_7_6</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_8_7</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_9_8</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_10_9</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_11_10</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_12_11</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_13_12</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_14_13</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_15_14</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_16_15</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_17_16</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_18_17</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_19_18</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_20_19</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_21_20</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_22_21</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_23_22</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_24_23</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_25_24</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_26_25</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_27_26</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_28_27</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_29_28</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_30_29</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_31_30</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_32_31</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_33_32</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_34_33</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_35_34</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_36_35</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_37_36</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_38_37</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_39_38</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_40_39</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_41_40</td>
<td>1.000000</td>
</tr>
<tr>
<td>x_42_41</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

All other variables in the range 1-861 are 0.