LEHIGH UNIVERSITY

IE 426 PROJECT REPORT

The Traveling Salesman Problem with Time Windows

Author:
Ziyi GUO, Weiyue REN,
Yawei YUAN, Xiao YIN

Lecturer:
Prof. Katya SCHEINBERG

December 13, 2013

The Traveling Salesman Problem with Time Windows

Ziyi Guo, Weiyue Ren, Yawei Yuan, Xiao Yin
Lehigh University

December 13, 2013

I Introduction

In the field of operations research and theoretical computer science, the Traveling Salesman Problem (TSP) refers to the following question: given a set of cities and the traveling cost between each pair of cities, find the shortest tour: a directed cycle containing all cities. This problem was first formulated in 1930 and is one of the most well-known optimization problems. TSP is a fundamental task since it is important to further research and real-world applications, such as postal planning, logistics and biological sequencing. For example, in the topic of DNA sequencing, each city represents one DNA fragment, the traveling cost represents the similarity of each pair of fragments and the sequencing can be viewed as the process of determining the order of DNA nucleotides.

TSP is a NP-hard (Non-deterministic Polynomial-time hard) problem in computational complexity theory, indicating that it is at least as hard as the hardest problems in NP, and the decision version of TSP is shown to be NP-complete, indicating the problem is in the set of NP problems and also in the set of NP-hard problems.

On the basis of TSP, Traveling Salesman Problem with Time Windows (TSPTW) requires that each city must be visited during a specific time window, which makes the problem considerably harder. Therefore, various heuristic algorithms, such as tabu search [1] and simulated annealing [2], have been used to solve TSPTW.

In this project, we first formulate TSP as an Integer Programming (IP) problem. We show that the time constrains cannot be satisfied. Furthermore, to avoid sub-tour elimination constrains, we formulate the TSPTW as a scheduling problem and allocate each customer to one visiting position. In the end, we show that the results of TSPTW and relaxed TSP are comparable.

II TSP Relaxation

Given one complete directed graph $D = (V, E)$, we try to find a directed cycle that contains all n nodes of minimum length. First, we define variables:

$$x_{ij} = \begin{cases} 1 \text{ if arc } (i,j) \text{ is in the tour,} \\ 0 \text{ otherwise.} \end{cases} \tag{1}$$

The node constrains can be formulated as follows:

$$\begin{aligned} \min \quad & \sum_{i \in V} \sum_{j \in V: i \neq j} c_{ij} x_{ij} + \sum_{i} p_i \\ \text{s.t.} \quad & \sum_{j \in V: i \neq j} x_{ij} = 1 \quad \forall i \in V, \\ & \sum_{j \in V: i \neq j} x_{ji} = 1 \quad \forall i \in V, \\ & x_{ij} \in \{0,1\} \quad \forall i, j \in V, i \neq j, \end{aligned} \tag{2}$$

The constrains of (2) are also called the assignment constrains [3]. However, the feasible solutions of this integer programming may contain several directed cycles, named sub-tours, which is illustrated in Figure 1.

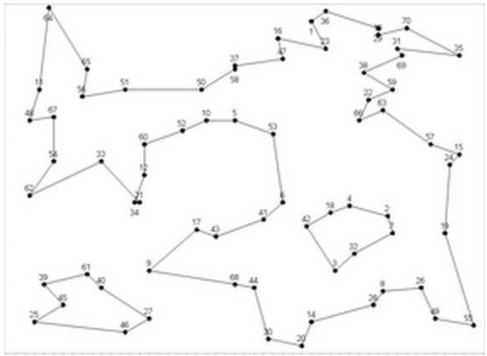

Figure 1: TSP Example with 3 Sub-tours.

To eliminate sub-tours solutions, we add extra variables $u_i (i = 1, \ldots, n)$ and the constraints:

$$u_i - u_j + n x_{ij} \leqslant n - 1 \quad 1 \leqslant i \neq j \leqslant n \tag{3}$$

From the assignment constrains, we conclude that there is only one sub-tour that passes through city 0. For any sub-tour with k cities that does not pass through city 0, we sum all inequalities in (3) and obtain $nk \leqslant (n-1)k$, which is a contradiction. Therefore, this constraint called Miller-Tucker-Zemlin (MTZ) formulation of TSP is capable of removing sub-tours [4].

We give an IP formulation of TSP via $(2)\&(3)$ and the AMPL code can be found in Appendix A. We find that the optimal path is $\{0 \to 8 \to 1 \to 4 \to 9 \to 7 \to 2 \to 3 \to 6 \to 5 \to 0\}$ and

the objective function value is 66. Furthermore, we display the starting time and ending time for each customer in the optimality, in which the red color means that this value does not satisfy time window constrain in TSPTW.

customer	0	8	1	4	9	7	2	3	6	5
start time	0	1	10	14	17	26	31	37	48	58
end time	66	6	11	16	20	27	36	46	53	65

Table 1: Time Window of Each Customer in TSP.

III TSPTW Formulation

In this part, we formulate TSPTW as an scheduling problem and allocate each customer to one fixed position, so the the sequencing of positions indicate the traveling path. First we define variables:

$$\begin{aligned} rank_{jk} &= \begin{cases} 1 \text{ if customer } j \text{ is allocated in kth position,} \\ 0 \text{ otherwise.} \end{cases} \\ start_k &: \text{the start time in kth position} \\ y_{i(k-1)jk} &: rank_{i(k-1)} rank_{jk} \end{aligned} \tag{4}$$

Given the set $V = \{0...n\}$, TSPTW can be formulated in (5). The constraints on *rank* variables guarantee that the mapping between customers and allocation positions is one-to-one and the first customer is the beginning point of the traveling path. The constraints on *start* variables guarantee the time window conditions are satisfied as (1) if the driver arrives at one customer in advance, the starting time is determined by its lower time window bound; (2) if the drive does not arrive in advance, the starting time is determined by the sum of the starting time of previous customer, the loading time of previous customer and the traveling time; (3) the sum of the start time of current customer and the loading time of current customer cannot exceed its upper time window bound. Also, we introduce variable $y_{i(k-1)jk}$ along with its linear constraints to avoid nonlinear constraints.

We give an IP formulation of TSPTW via (5) and the AMPL code can be found in Appendix B. We find that the optimal path is $\{0 \to 1 \to 2 \to 3 \to 6 \to 5 \to 9 \to 4 \to 7 \to 8 \to 0\}$ and the total traveling time is 75, which is comparable with TSP optimal traveling time. Similarly, we display the starting time and ending time for each customer of this optimal path and we find that all time constraints are satisfied.

customer	0	1	2	3	6	5	9	4	7	8
start time	0	5	9	15	26	36	47	53	58	64
end time	75	6	14	24	31	43	50	5	59	69

Table 2: Time Window of Each Customer in TSPTW.

$$
\begin{aligned}
\min \quad & start_n + \sum_{j \in V} p_j rank_{jn} + \sum_{j \in V} c_{j0} rank_{jn} \\
\text{s.t.} \quad & \sum_{k \in V} rank_{jk} = 1 \quad \forall j \in V, \\
& \sum_{j \in V} rank_{jk} = 1 \quad \forall k \in V, \\
& rank_{00} = 1, \\
& rank_{jk} \in \{0,1\} \quad \forall j,k \in V, \\
& start_k \geqslant start_{k-1} + \sum_{j \in V} p_j rank_{j(k-1)} + \sum_{i \in V} \sum_{j \in V : i \neq j} c_{ij} y_{i(k-1)jk} \quad k = 1..n, \\
& start_k \geqslant \sum_{j \in V} r_j rank_{jk} \quad k = 1..n, \\
& start_k + \sum_{j \in V} p_j rank_{jk} \leqslant \sum_{j \in V} d_j rank_{jk} \quad k = 1..n, \\
& start_k \geqslant 0 \quad k = 1..n, \\
& start_0 = 0, \\
& y_{i(k-1)jk} \leqslant rank_{i(k-1)} \quad \forall i,j \in V, k = 1..n, \\
& y_{i(k-1)jk} \leqslant rank_{jk} \quad \forall i,j \in V, k = 1..n, \\
& y_{i(k-1)jk} \geqslant rank_{i(k-1)} + rank_{jk} - 1 \quad \forall i,j \in V, k = 1..n, \\
& y_{i(k-1)jk} \in \{0,1\} \quad \forall i,j \in V, k = 1..n,
\end{aligned}
\tag{5}
$$

References

[1] W. B. Carlton and J. W. Barnes, "Solving the traveling-salesman problem with time windows using tabu search," *IIE transactions*, vol. 28, no. 8, pp. 617–629, 1996.

[2] J. W. Ohlmann and B. W. Thomas, "A compressed-annealing heuristic for the traveling salesman problem with time windows," *INFORMS Journal on Computing*, vol. 19, no. 1, pp. 80–90, 2007.

[3] G. Pataki, "Teaching integer programming formulations using the traveling salesman problem," *SIAM review*, vol. 45, no. 1, pp. 116–123, 2003.

[4] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM (JACM)*, vol. 7, no. 4, pp. 326–329, 1960.

Appendix

TSP Model File

```
param n > 0,integer;
set V := 0..n;
set V2 := 1..n;
param c{V,V} >= 0;
param p{V} >= 0;

var x{V,V} binary;
var u{V2} integer;

minimize total_time : sum{i in V, j in V : i!=j} (x[i,j]*c[i,j])+sum{i in V}(p[i]);

subject to constrain1 {i in V}: sum{j in V:i!=j} x[i,j] = 1;
subject to constrain2 {j in V}: sum{i in V:i!=j} x[i,j] = 1;
subject to constrain3 {i in V2,j in V2:i!=j}: u[i]-u[j]+n*x[i,j] <= (n-1);
```

TSP Data File

```
param n := 9;
param c:
        0   1  2   3   4   5   6  7   8   9 :=
0   0   5  4   4   4   6   3  2   1   8
1   7   0  2   5   3   5   4  4   4   9
2   3   4  0   1   1  12   4  3  11   6
3   2   2  3   0   2  23   2  9  11   4
4   6   4  7   2   0   9   8  3   2   1
5   1   4  6   7   3   0   8  5   7   4
6  12  32  5  12  18   5   0  7   9   6
7   9  11  4  12  32   5  12  0   5  22
8   6   4  7   3   5   8   6  9   0   5
9   4   6  4   7   3   5   8  6   9   0;

param p:=
        0 0         1 1
        2 5         3 9
        4 2         5 7
        6 5         7 1
        8 5         9 3;
```

Running Result

```
ampl: model tsp.mod;
ampl: data tsp.dat;
ampl: option solver cplex;
ampl: solve;
CPLEX 12.5.1.0: optimal integer solution; objective 66
54 MIP simplex iterations
0 branch-and-bound nodes
ampl: display x;
x [*,*]
:   0  1  2  3  4  5  6  7  8  9   :=
0   0  0  0  0  0  0  0  0  1  0
1   0  0  0  0  1  0  0  0  0  0
2   0  0  0  1  0  0  0  0  0  0
3   0  0  0  0  0  0  1  0  0  0
4   0  0  0  0  0  0  0  0  0  1
5   1  0  0  0  0  0  0  0  0  0
6   0  0  0  0  0  1  0  0  0  0
7   0  0  1  0  0  0  0  0  0  0
8   0  1  0  0  0  0  0  0  0  0
9   0  0  0  0  0  0  0  1  0  0
;

ampl: display total_time;
```

total_time = 66

TSPTW Model File

```
param n > 0,integer;
set V := 0..n;
set V2 := 1..n;
param c{V,V} >= 0;
param p{V} >= 0;
param r{V} >= 0;
param d{V} >= 0;

var rank{V,V} binary;
var start{V} >= 0;
var y{V,V,V,V2} binary;

minimize total_time : start[n]+sum{j in V:j!=0}(c[j,0]*rank[j,n])+sum{j in V:j!=0} (p[j]*rank[j,n]);

subject to constrain11 {j in V}: sum{k in V} rank[j,k]=1;
subject to constrain12 {k in V}: sum{j in V} rank[j,k]=1;
subject to constrain13: rank[0,0] = 1;

subject to constrain21 {k in V2}: start[k] >= start[k-1] + sum{i in V}(p[i]*rank[i,k-1])+sum{i in V,j in V:i!=j}(c[i,j])*y[i,k-1,j,k];
subject to constrain22: start[0] = 0;

subject to constrain31 {i in V,j in V,k in V2}: y[i,k-1,j,k]<=rank[i,k-1];
subject to constrain32 {i in V,j in V,k in V2}: y[i,k-1,j,k]<=rank[j,k];
subject to constrain33 {i in V,j in V,k in V2}: y[i,k-1,j,k]>=rank[i,k-1]+rank[j,k]-1;

subject to constrain4 {k in V2}: start[k] >= sum{j in V}(r[j]*rank[j,k]);
subject to constarin5 {k in V2}: start[k]+sum{j in V}(p[j]*rank[j,k]) <= sum{j in V}(d[j]*rank[j,k]);
```

TSPTW Data File

```
param n := 9;
param c:
        0   1   2   3   4   5   6   7   8   9 :=
0   0   5   4   4   4   6   3   2   1   8
1   7   0   2   5   3   5   4   4   4   9
2   3   4   0   1   1  12   4   3  11   6
3   2   2   3   0   2  23   2   9  11   4
4   6   4   7   2   0   9   8   3   2   1
5   1   4   6   7   3   0   8   5   7   4
6  12  32   5  12  18   5   0   7   9   6
7   9  11   4  12  32   5  12   0   5  22
8   6   4   7   3   5   8   6   9   0   5
9   4   6   4   7   3   5   8   6   9   0;

param p:=
        0 0         1 1
        2 5         3 9
        4 2         5 7
        6 5         7 1
        8 5         9 3;

param r:=
        0   0       1   2
        2   9       3   4
        4  12       5   0
        6  23       7   9
        8  15       9  10;

param d:=
        0 150       1   45
        2  42       3   40
        4 150       5   48
        6  96       7  100
        8 127       9   66;
```

Running Result

```
ampl: model tsptw.mod;
ampl: data tsptw.dat;
ampl: option solver cplexamp;
ampl: solve;
CPLEX 12.5.0.0: optimal integer solution; objective 75
52894 MIP simplex iterations
3429 branch-and-bound nodes
ampl: display rank;
rank [*,*]
:  0  1  2  3  4  5  6  7  8  9  :=
0  1  0  0  0  0  0  0  0  0  0
1  0  1  0  0  0  0  0  0  0  0
2  0  0  1  0  0  0  0  0  0  0
3  0  0  0  1  0  0  0  0  0  0
4  0  0  0  0  0  0  0  1  0  0
5  0  0  0  0  0  1  0  0  0  0
6  0  0  0  0  1  0  0  0  0  0
7  0  0  0  0  0  0  0  0  1  0
8  0  0  0  0  0  0  0  0  0  1
9  0  0  0  0  0  0  1  0  0  0
;

ampl: display start;
start [*] :
0    0
1    5
2    9
3  15
4  26
5  36
6  47
7  53
8  58
9  64
;

ampl: display total_time;
total_time = 75
```
