

Mathematical Programming: Modeling and Applications

Giacomo Nannicini

`giacomon@lix.polytechnique.fr`

The Traveling Salesman Problem

- A very well known problem in the literature
- Several applications: logistics, crane control, placing circuits on a board minimizing the required time, and many more
- Unfortunately, it is a *very* difficult problem
- For not too large instances, it can be done on a desktop machine

Problem Definition

- A salesman must visit all cities to see his customers, and return to the starting point
- He wants to minimize the total travel distance
- We assume that the distance between two cities is symmetric, and that our starting point is the first city
- Write model, data and run file to solve an instance *automatically*
- Remember to check if the solution is correct

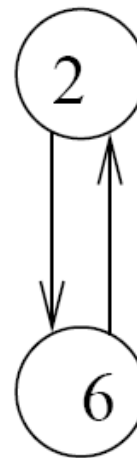
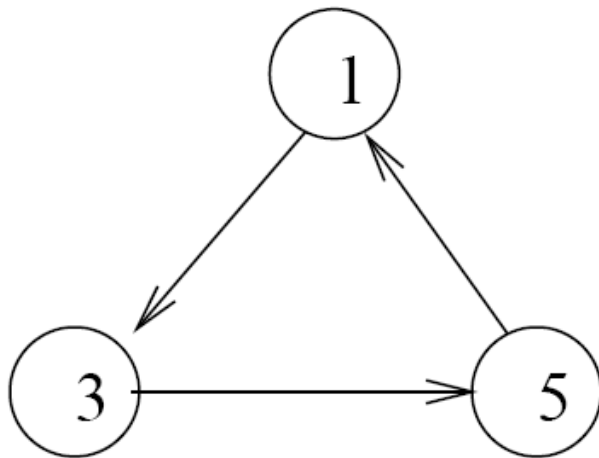
Data

- We have 7 cities
- Distance matrix:

	1	2	3	4	5	6	7
1	-	86	49	57	31	69	50
2		-	68	79	93	24	5
3			-	16	7	72	67
4				-	90	69	1
5					-	86	59
6						-	81

Hint: Formulation

- The TSP can be formalized as the search for a Hamiltonian tour (i.e. a tour that touches all vertices at most once) on a complete directed graph $G = (V, A)$
- Constraints: we enter each vertex once, we leave each vertex once
- But we do not want subtours!



Hint: Formulation

- We have to eliminate subtours
- Let x_{ij} be a variable which indicates if the arc between city i and j is chosen or not
- For each $S \subset V : 1 \leq |S| \leq |V|$, we have to add the constraint

$$\sum_{i \in S, j \in V \setminus S} x_{ij} \geq 1$$

- If S is the set of nodes in a subtour, this constraint breaks the subtour
- Problem: the number of subsets $S \subset V$ is exponential in the cardinality of V !

Solution Method

- We solve the TSP without subtour elimination constraint
- We check the solution: if there is no subtour, we are done
- If there is a subtour, we add the corresponding subtour elimination constraint, and resolve
- We iterate until we find a solution with no subtours
- We have to do this *automatically*

Solution Method

- Write a run file that solves the TSP without subtour elimination constraints, checks the solution, and if there is a subtour, it adds the corresponding violated constraint
- Remember that you can declare/use parameters in a run file
- You can also declare constraints on a set:
subject to subtour_elim{k in 1..subtours}:
- Useful instructions:
for { i in V } { ... }
repeat while (condition) { ... }
repeat { ... } until (condition);
- Use printf!

Solution: Formulation

- Sets: set V of vertices (cities)
- Parameters: distance d_{ij} between two cities
 $i, j \in V, i \neq j$
- Variables: for each $i, j \in V, i \neq j$, binary variable x_{ij} which is 1 iff the salesman travels directly from i to j
- Objective:

$$\min \sum_{i \neq j \in V} d_{ij} x_{ij}$$

Solution: Constraints

- One successor:

$$\forall i \in V \quad \sum_{j \in V, j \neq i} x_{ij} = 1$$

- One predecessor:

$$\forall j \in V \quad \sum_{i \in V, i \neq j} x_{ij} = 1$$

- Subtour elimination:

$$\forall S \subset V \quad \sum_{i \in S, j \in V \setminus S} x_{ij} \geq 1$$

Solution: Model

```
# tsp.mod

param n > 0, integer;
set V := 1..n;
param d{V,V} >= 0;
param subtours >= 0, integer, default 0;
set S{1..subtours};

var x{V,V} binary;

minimize distance : sum{i in V, j in V : i != j} d[i,j] * x[i,j];

subject to successor {i in V} : sum{j in V : i != j} x[i,j] = 1;
subject to predecessor {j in V} : sum{i in V : i != j} x[i,j] = 1;
subject to subtour_elim {k in 1..subtours} :
    sum{i in S[k], j in V diff S[k]} x[i,j] >= 1;
```

Solution: Data

```
# tsp.dat
```

```
param n := 7;
```

```
param d : 1 2 3 4 5 6 7 :=
```

```
1 0 86 49 57 31 69 50
```

```
2 0 0 68 79 93 24 5
```

```
3 0 0 0 16 7 72 67
```

```
4 0 0 0 0 90 69 1
```

```
5 0 0 0 0 0 86 59
```

```
6 0 0 0 0 0 0 81
```

```
7 0 0 0 0 0 0 0 ;
```

Solution: Run File (1)

```
# tsp.run
model tsp.mod;
data tsp.dat;

# distance matrix must be symmetric
for {i in V, j in V : i > j} {
    let d[i,j] := d[j,i];
}
option solver cplex;
# dont print CPLEX messages
option solver_msg 0;

# data structures required for the algorithm
let subtours := 0;
param successorvertex{V} >= 0, integer;
param currentvertex >= 0, integer;
param termination binary;
let termination := 0;
(...)
```

Solution: Run File (2)

```
(...)  
# iterative part  
repeat while (termination == 0) {  
    # solve the problem  
    solve;  
    let subtours := subtours + 1;  
    # find the successors of each vertex  
    for {i in V} {  
        let successorvertex[i] := sum{j in V : j != i} j * x[i,j];  
    }  
    # find a subtour  
    let currentvertex := 1;  
    let S[subtours] := {};  
    repeat {  
        let S[subtours] := S[subtours] union {currentvertex};  
        let currentvertex := successorvertex[currentvertex];  
    } until (currentvertex = 1);  
    # print the subtour we wish to eliminate  
    (...)
```

Solution: Run File (3)

```
(...)  
# print the subtour we wish to eliminate  
printf "(sub)tour: (";  
for {i in S[subtours]} {  
    printf "%d -> ", i ;  
}  
printf "1)\n";  
# termination condition  
if (card(S[subtours]) >= n) then {  
    # Hamiltonian tour, terminate  
    let termination := 1;  
}  
}  
  
printf "travelled distance in the optimal tour: %f\n", distance;
```

Solution: Output

```
ILOG AMPL 11.010, licensed to "ecolepolytechnique-palaiseau".  
AMPL Version 20080219 (Linux 2.6.9-5.ELsmp)  
ILOG CPLEX 11.010, licensed to "ecolepolytechnique-palaiseau",  
options: e m b q use=1  
(sub)tour: (1 -> 3 -> 5 -> 1)  
ILOG CPLEX 11.010, licensed to "ecolepolytechnique-palaiseau",  
options: e m b q use=1  
(sub)tour: (1 -> 6 -> 2 -> 7 -> 4 -> 3 -> 5 -> 1)  
travelled distance in the optimal tour: 153.000000
```