Modeling and Optimization of Networked Systems: Network Flows Modeling with Mixed Integer Programming

Andrés D. González

Assistant Professor
School of Industrial and Systems Engineering, The University of Oklahoma

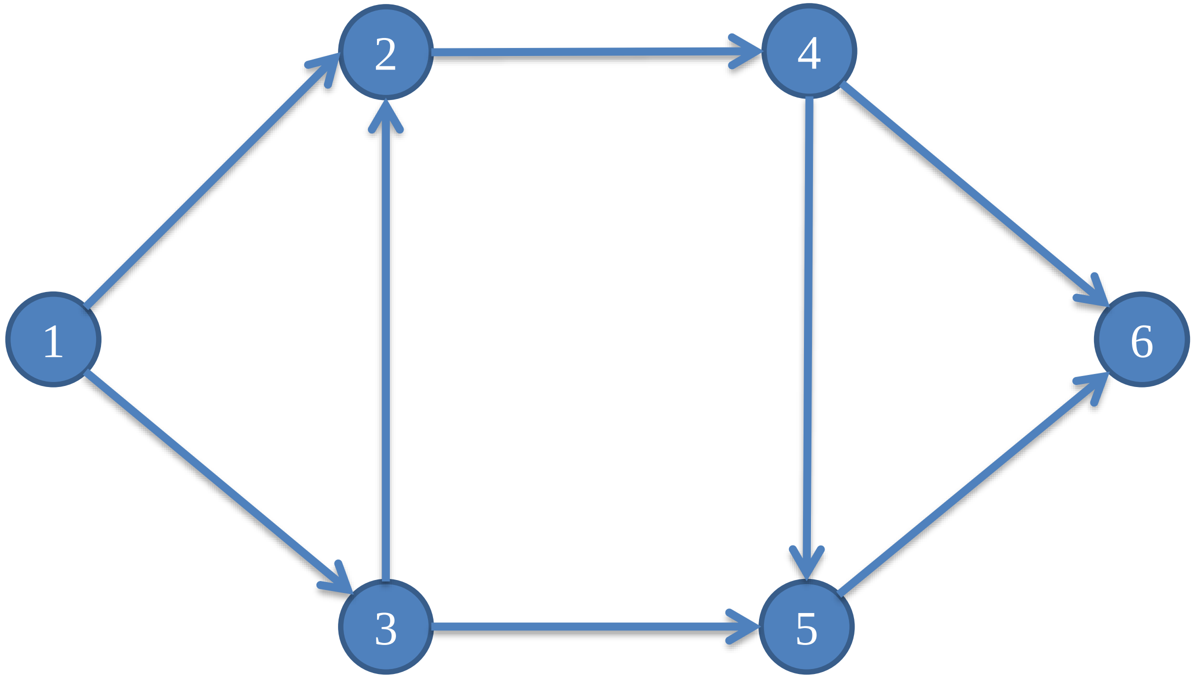
ISE 4623/5023: Deterministic Systems Models / Systems Optimization

The University of Oklahoma, Norman, OK, USA

Minimum Cost Flow Problem (MCFP)

Given:

- *G = directed network* (*N*, *A*)
- *N* = set of *n nodes*
- *A* = set of *m directed arcs*.

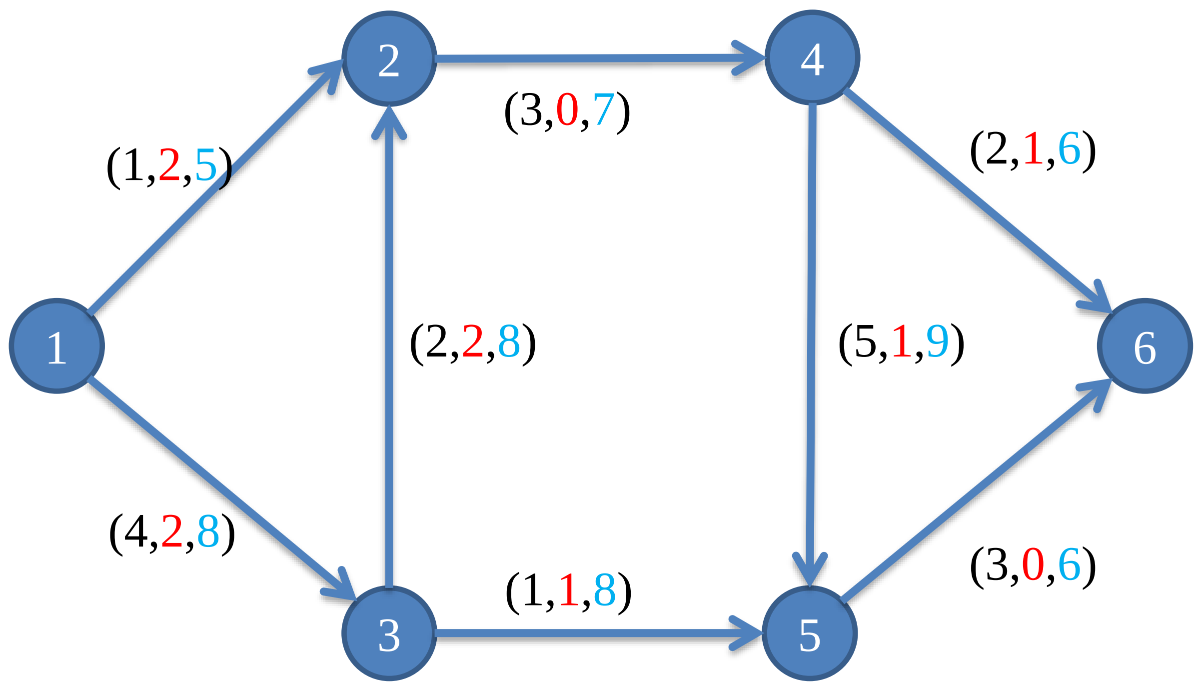


(Ahuja, Magnanti & Orlin, 1993; Kennington, 2006; Vaidyanathan, 2010)

Minimum Cost Flow Problem (MCFP)

Each arc $(i, j) \in A$ has an associated:

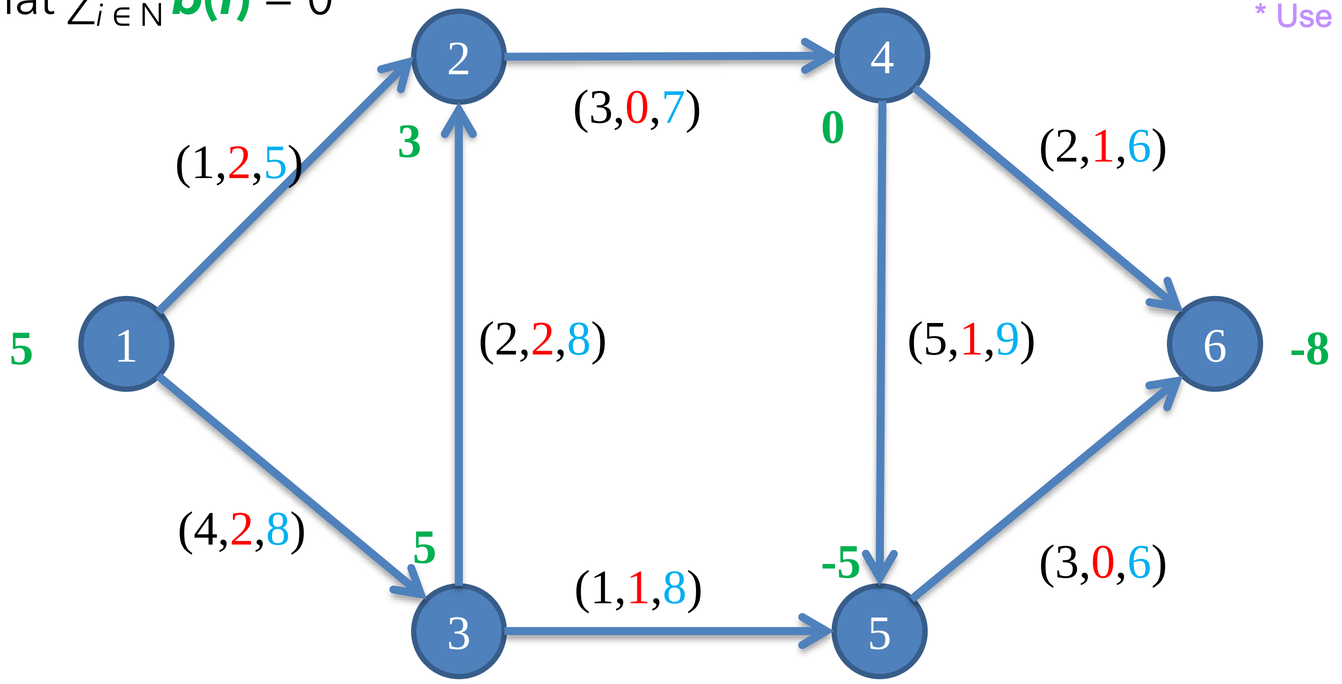
- *Cost* $\boldsymbol{c_{ij}}$ (per unit flow on that arc),
- Lower bound $\boldsymbol{l_{ij}}$ *and a capacity* $\boldsymbol{u_{ij}}$ (minimum and maximum flow on the arc).



Minimum Cost Flow Problem (MCFP)

- We associate with each node $i \in N$ an integer ***b(i)*** representing its supply/demand.
  - If ***b(i)*** > 0, node *i* is a *supply node*
  - If ***b(i)*** < 0, then node *i* is a *demand node* with a demand of – ***b(i)***
  - If ***b(i)*** = 0, then node *i* is a *transshipment node*
  - We assume that $\sum_{i \in N} b(i) = 0$

* Use slacks if unbalanced Networks

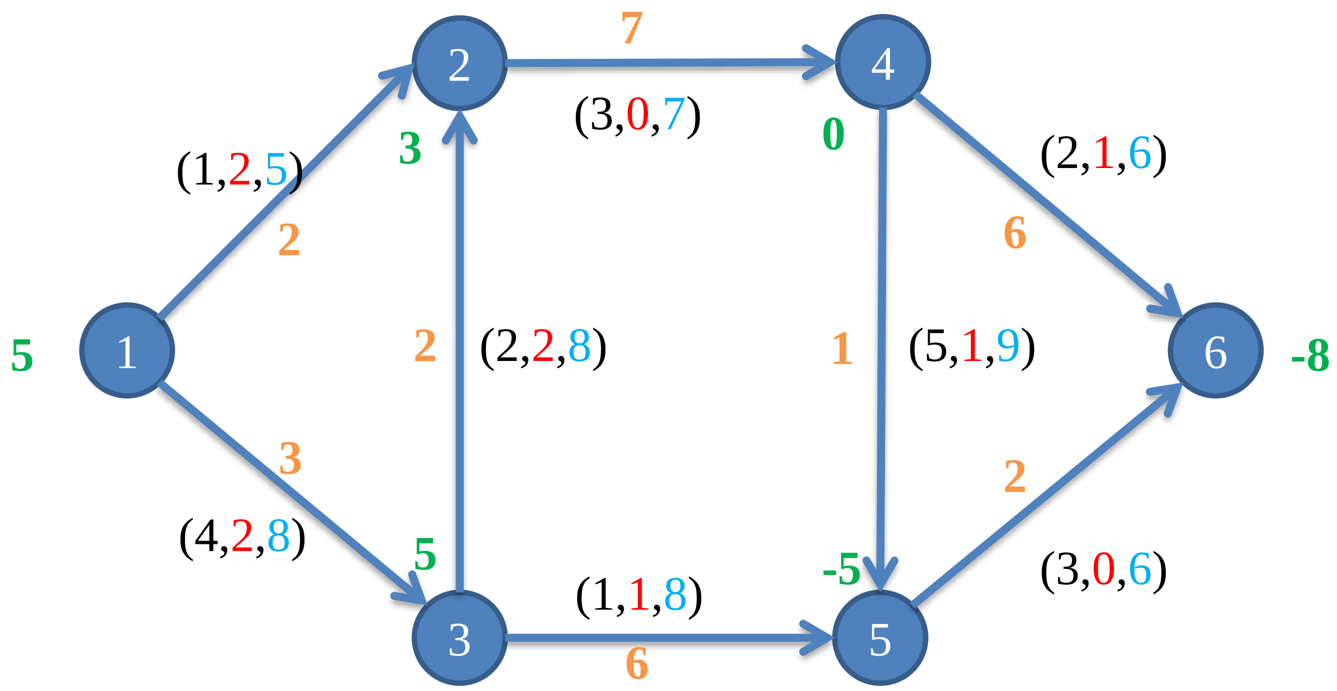


Minimum Cost Flow Problem (MCFP)

Objective
The *minimum cost flow problem* seeks a least cost shipment of a commodity through a network to satisfy demands at certain nodes by available supplies at other nodes.

The decision variables x_{ij} are arc flows defined for each arc $(i, j) \in A$.

Total cost = 68

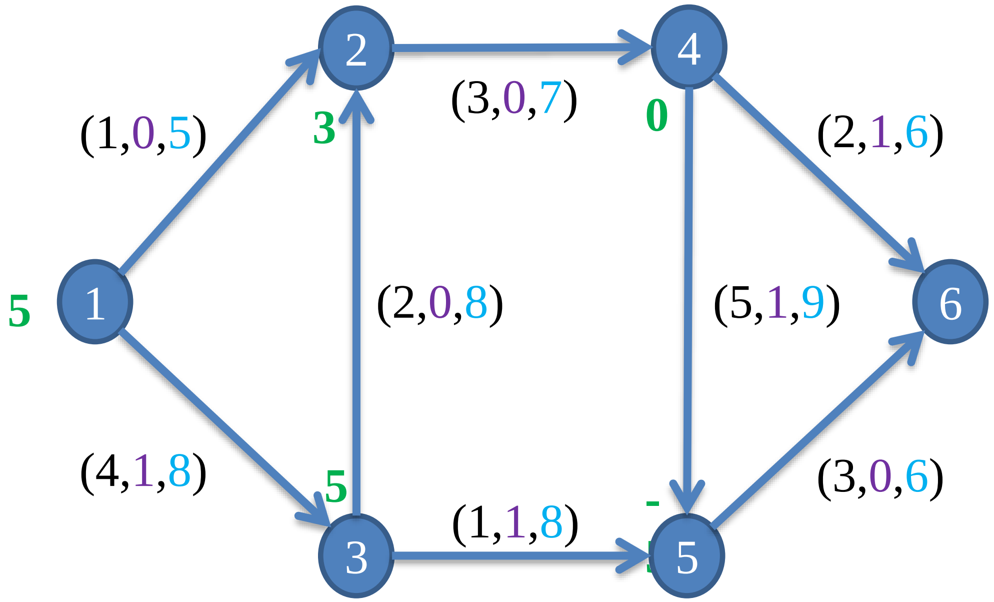


Network Design Problem (NDP)

Given:

- G = *directed network* (N, A)
- N = set of n nodes
- A = set of m directed arcs
- $\mathbf{c_{ij}}$ = cost per unit flow on the *arc* $(i, j) \in A$
- $\mathbf{u_{ij}}$ = maximum flow on the *arc* $(i, j) \in A$
- $\mathbf{b_i}$ = supply/demand associated to *node* $i \in N$
- $\mathbf{f_{ij}}$ = fixed cost of using the *arc* $(i, j) \in A$

f = fixed cost of the arc,
e.g., using a toll road from point a to b

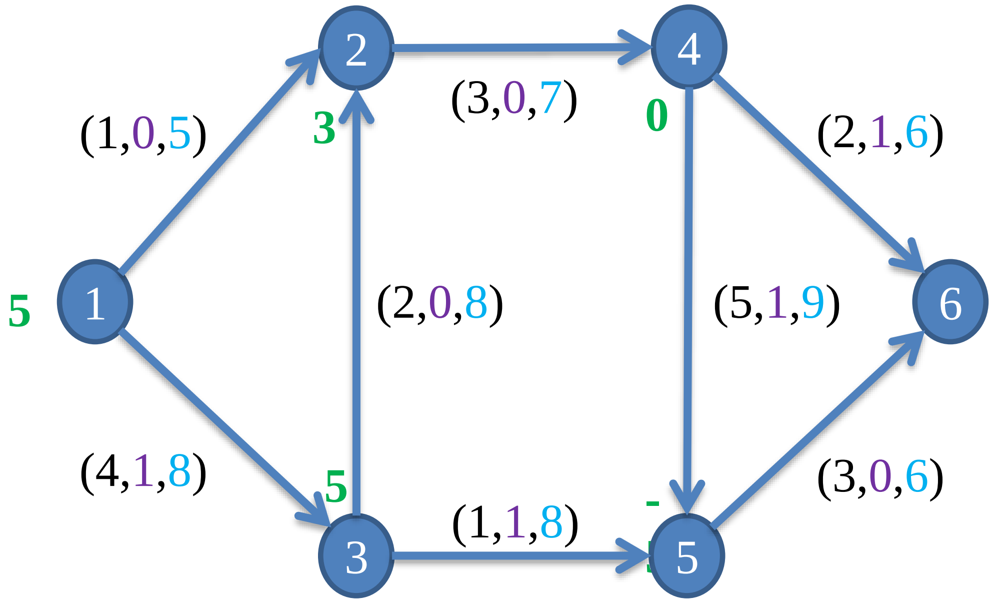


Network Design Problem (NDP)

To be determined from optimization

Variables:

- x_{ij} = amount of flow on the *arc* $(i, j) \in A$
- y_{ij} = design variables, that equal 1 if *arc* $(i, j) \in A$ is selected in the final design (and 0 otherwise)



Network Design Problem (NDP)

- Sets:

N: Set of nodes $\{1,2,3,\ldots,n\}$
A: Set of arcs

- Parameters:

c_{ij}: unit cost of sending a commodity through arc $(i,j) \in A$
f_{ij}: fixed cost of using/building arc $(i,j) \in A$ New addition
b_i: demand/supply of commodities in node $i \in N$
u_{ij}: maximum flow through arc $(i,j) \in A$
l_{ij}: minimum flow through arc $(i,j) \in A$ Not used but could be

- Variables:

x_{ij}: flow through arc $(i,j) \in A$
y_{ij}: binary variable that indicates if arc $(i,j) \in A$ is used (=1) or not (=0)

- Objective function:

$$\min z = \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{(i,j)\in A} f_{ij}y_{ij}$$

- Constraints:

$$\sum_{j:(i,j)\epsilon\, A} x_{ij} - \sum_{j:(j,i)\epsilon\, A} x_{ji} = b_i, \qquad \forall i \in N$$

$$x_{ij} \le u_{ij}y_{ij}, \qquad \forall (i,j) \in A$$
$$x_{ij} \ge l_{ij}, \qquad \forall (i,j) \in A$$
$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A$$

Use 0 instead of l for our assignment

Note: The basic NDP uses zero as the lower bound for x_{ij}, but this can be easily generalized to use any l_{ij} as in the Minimum Cost Flow Problem

Network Design Problem (NDP)

Objective

The *network design problem* seeks a least cost shipment of a commodity through a network to satisfy demands at certain nodes by available supplies at other nodes.

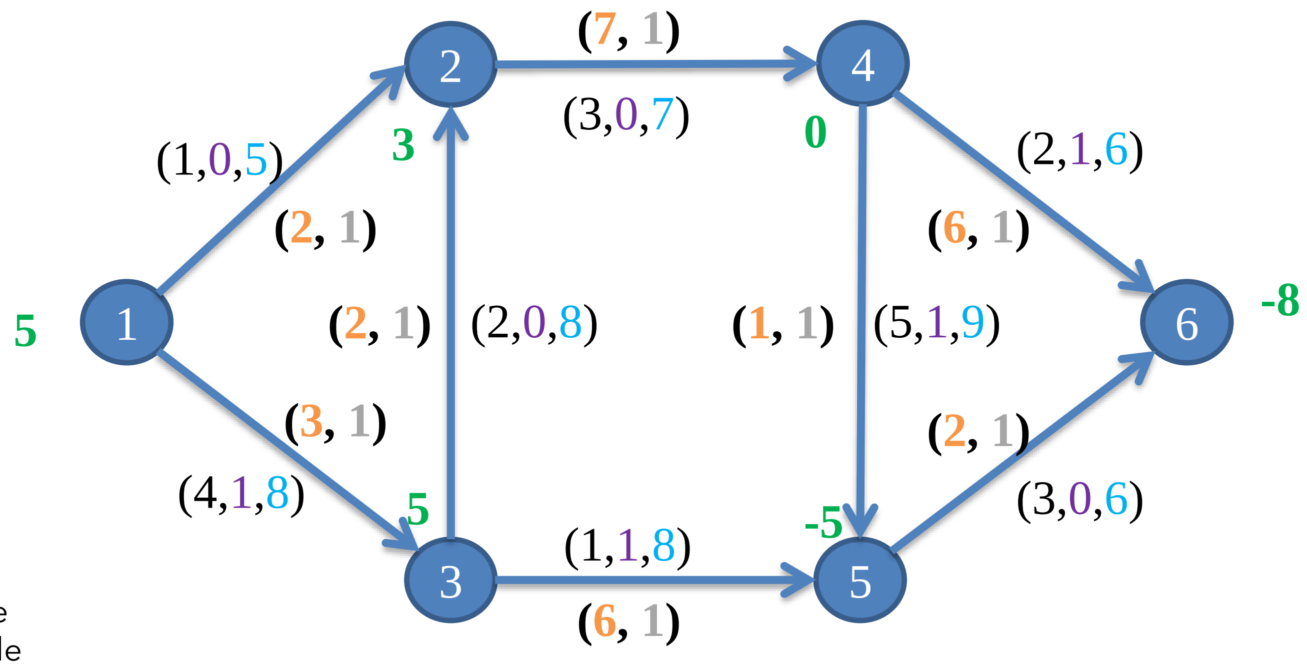
f = fixed cost of the arc

(c_{ij}, f_{ij}, u_{ij})

(x_{ij}, y_{ij})

$b(i)$

Total cost = 72*



* Using the lower bounds used in the Minimum Cost Flow Problem example

NDP – Gurobi/Python Example

```python
#Deterministic Systems Models/ Systems Optimization. The University of Oklahoma.
#Prof: Andrés D. González.

#network design problem (NDP) - Class example

from gurobipy import *

# N -> set of Nodes
# b -> demand/supply of each node
N, b = multidict({
  ('node1'):  5,
  ('node2'):  3,
  ('node3'):  5,
  ('node4'):  0,
  ('node5'):  -5,
  ('node6'):  -8})

# A -> set of Arcs: (startNode,endNode)
# l -> flow lower bound for each arc
# u -> flow lower bound for each arc
# c -> flow unit cost for each arc
# f -> fixed cost associated with using each arc

A, l, u, c, f = multidict({
  ('node1', 'node2'):   [2,5,1,0],
  ('node1', 'node3'):   [2,8,4,1],
  ('node3', 'node2'):   [2,8,2,0],
  ('node2', 'node4'):   [0,7,3,0],
  ('node3', 'node5'):   [1,8,1,1],
  ('node4', 'node5'):   [1,9,5,1],
  ('node4', 'node6'):   [1,6,2,0],
  ('node5', 'node6'):   [0,6,3,1] })
```

```python
# Create optimization model
m = Model('ndp')

# Create variables
# flow variable "x_{ij}"
x = m.addVars(A, obj=c, name="x")

# design variable "y_{ij}"
y = m.addVars(A, obj=f, name="y", vtype=GRB.BINARY)

# Flow-conservation/balance constraints (1)
m.addConstrs(
    (  x.sum(i,'*')-x.sum('*',i)==b[i]
     for i in N), "balanceConstraint")

# Arc-capacity constraints (2)
m.addConstrs(
    (x[i,j] <= u[i,j]*y[i,j] for i,j in A), "upperBoundConstraint")

#the basic "network design problem" doesn't include lower bound constraints, but for this example
#we will assume that the problem is asking us to keep these

# Lower bound constraints
m.addConstrs(
    (x[i,j] >= l[i,j] for i,j in A), "lowerBoundConstraint")


# Compute optimal solution
m.optimize()

# Print solution
if m.status == GRB.Status.OPTIMAL:
    solution_x = m.getAttr('x', x)
    solution_y = m.getAttr('x', y)
    print('\nOptimal solution:')
    for i,j in A:
        if solution[i,j] >= 0:
            print('%s -> %s: total flow: %g    usign arc? %g' % (i, j, solution_x[i,j], solution_y[i,j]))
```

over all j's

Note balanced network, so would have to use slacks if not balanced

NDP – Gurobi/Python Example (Solution)

```
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 22 rows, 16 columns and 40 nonzeros
Model fingerprint: 0xd0b9ec1f
Variable types: 8 continuous, 8 integer (8 binary)
Coefficient statistics:
  Matrix range     [1e+00, 9e+00]
  Objective range  [1e+00, 5e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 8e+00]
Presolve removed 22 rows and 16 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.01 seconds
Thread count was 1 (of 4 available processors)

Solution count 1: 72

Optimal solution found (tolerance 1.00e-04)
Best objective 7.200000000000e+01, best bound 7.200000000000e+01, gap 0.0000%

Optimal solution:
node1 -> node2: total flow: 2    usign arc? 1
node1 -> node3: total flow: 3    usign arc? 1
node3 -> node2: total flow: 2    usign arc? 1
node2 -> node4: total flow: 7    usign arc? 1
node3 -> node5: total flow: 6    usign arc? 1
node4 -> node5: total flow: 1    usign arc? 1
node4 -> node6: total flow: 6    usign arc? 1
node5 -> node6: total flow: 2    usign arc? 1
```

Traveling Salesperson Problem (TSP)

Sets:

- N: Set of nodes $\{1,2,3, \dots, n\}$
- A: Set of arcs

Parameters:

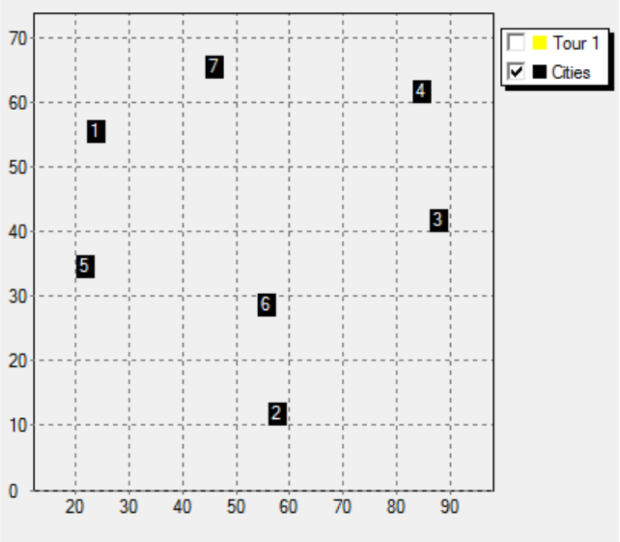
- c_{ij}: cost of traveling through arc $(i.j) \in A$

Objective:

- Minimize the total travel cost, while guaranteeing that all nodes are visited

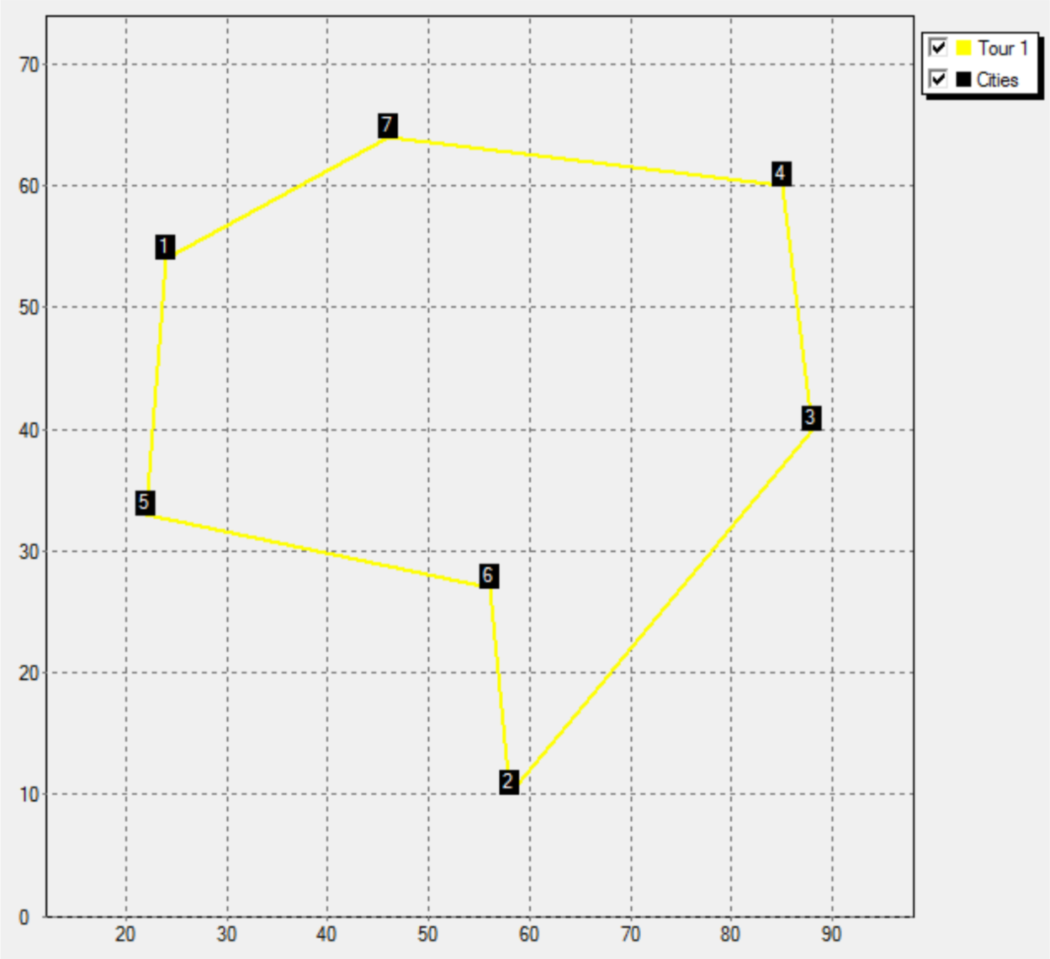
TSP - Example

```
TYPE:         Symmetric traveling salesman problem
DIFFICULTY:   5
FEATURES:     MIP problem, loop over problem solving, TSP subtour
              elimination algorithm;
              procedure for generating additional constraints, recursive
              subroutine calls, working with sets, `forall-do',
              `repeat-until', `getsize', `not', graphical representation
              of solutions
DESCRIPTION:  A flight tour starts from airport 1, visits all
              the other airports and then comes back to the starting point.
              The distances between the airports are symmetric. In which
              order should the airports be visited to minimize the total
              distance covered?
```

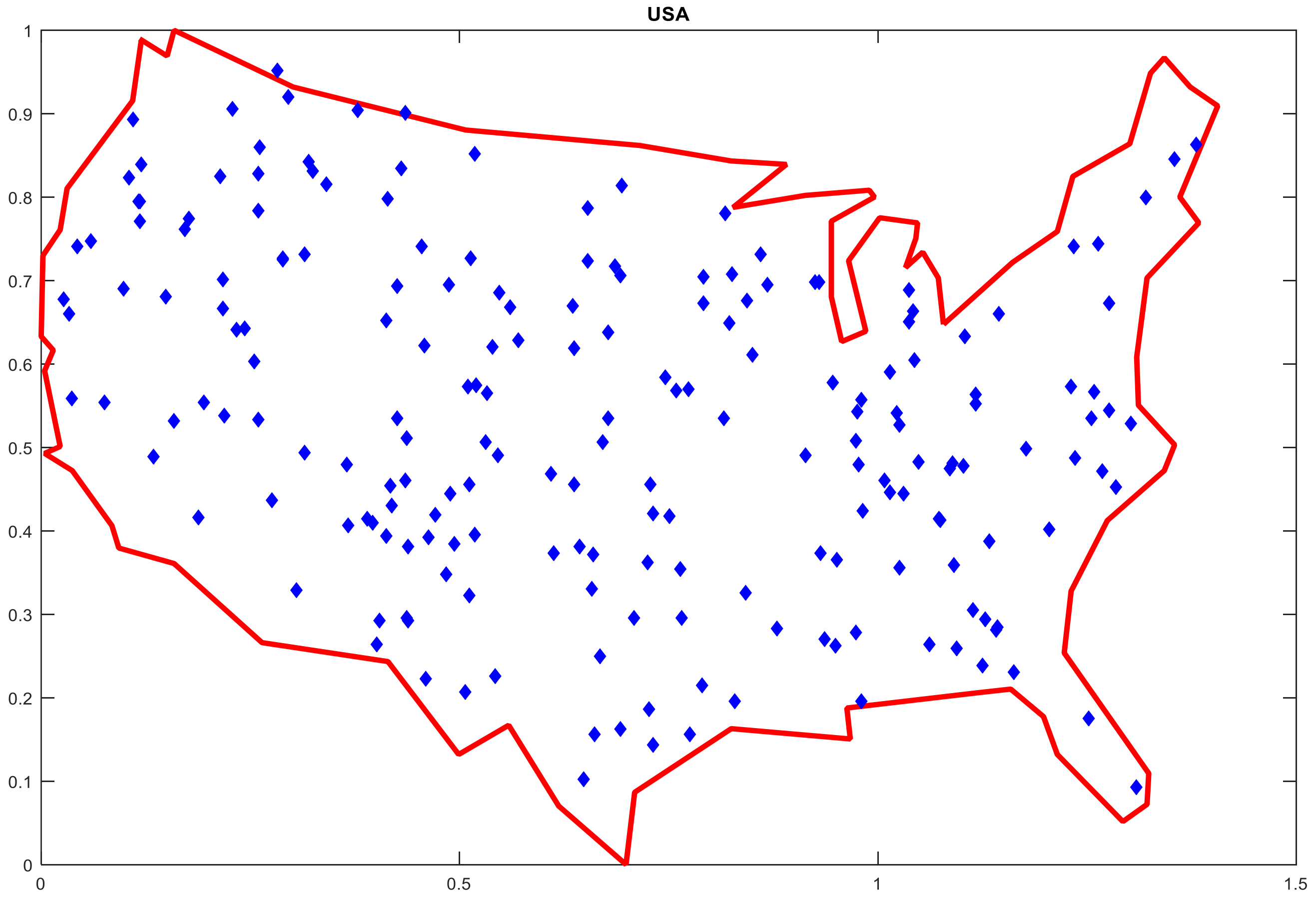


TSP - Example

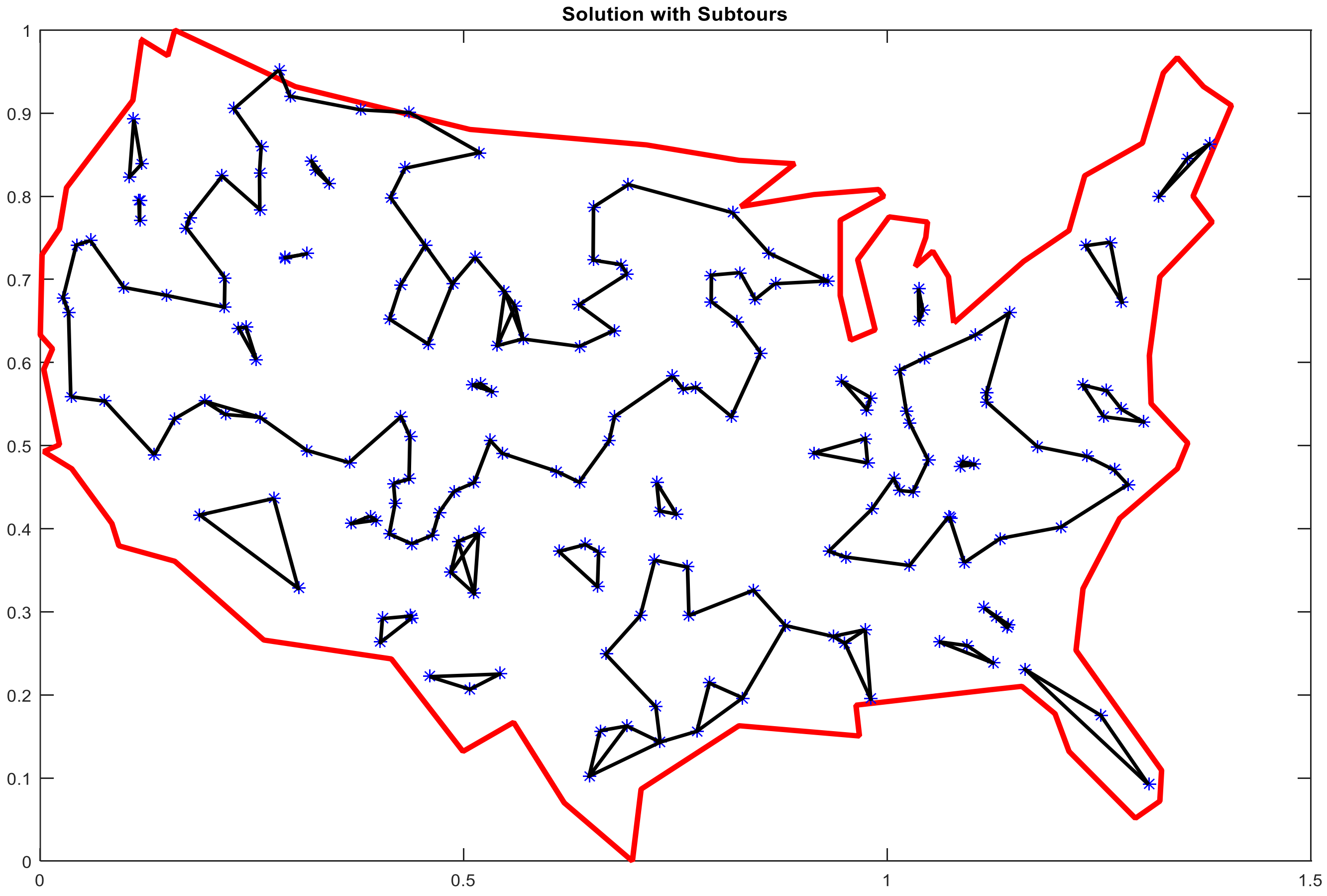
```
Total distance: 2220
1 - 7 - 5 - 1
2 - 6 - 2
3 - 4 - 3
Total distance: 2335
1 - 7 - 1
2 - 6 - 5 - 2
3 - 4 - 3
Total distance: 2575
1 - 7 - 4 - 3 - 2 - 6 - 5 - 1
```



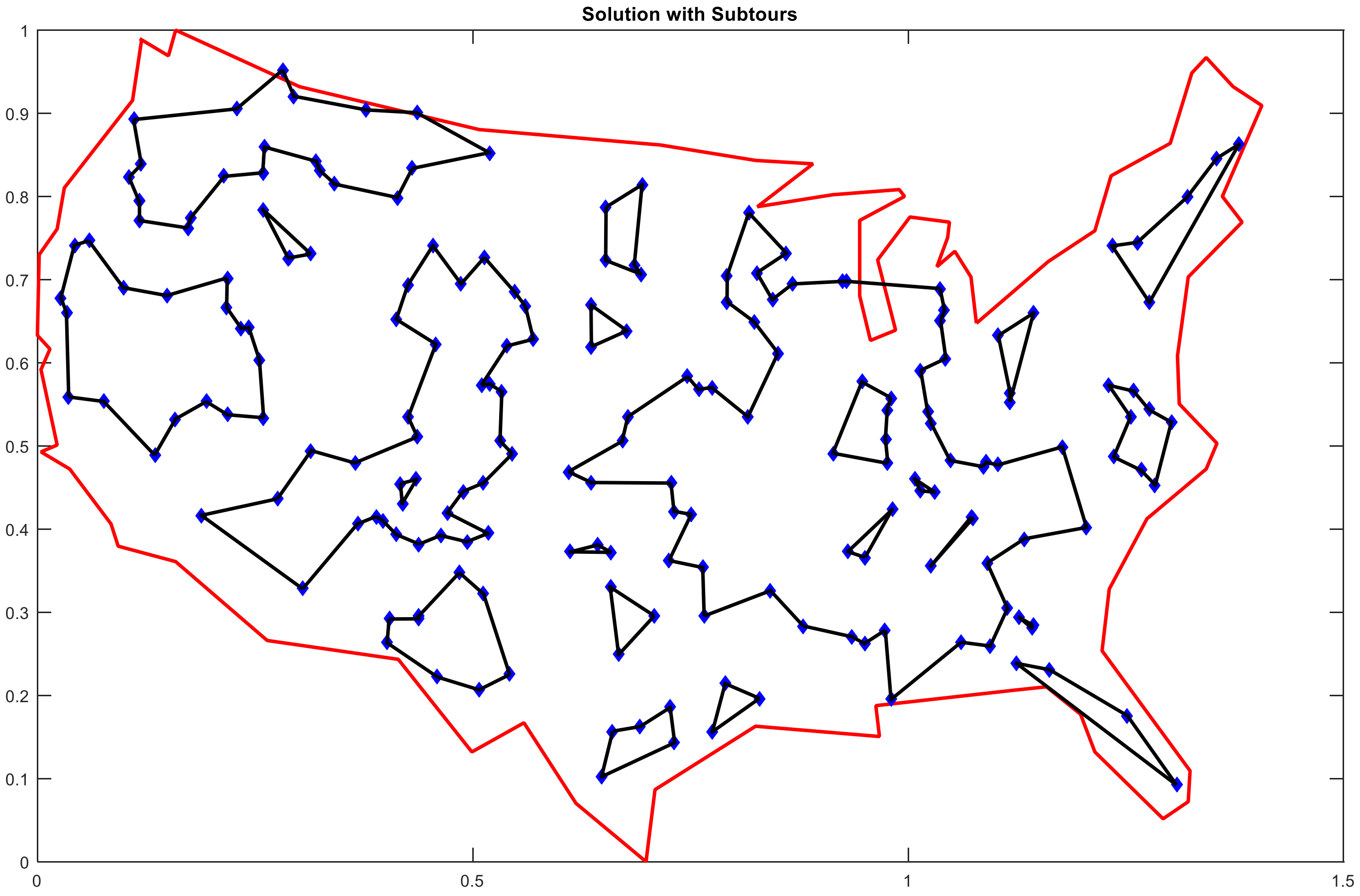
Traveling Salesperson Problem (TSP)



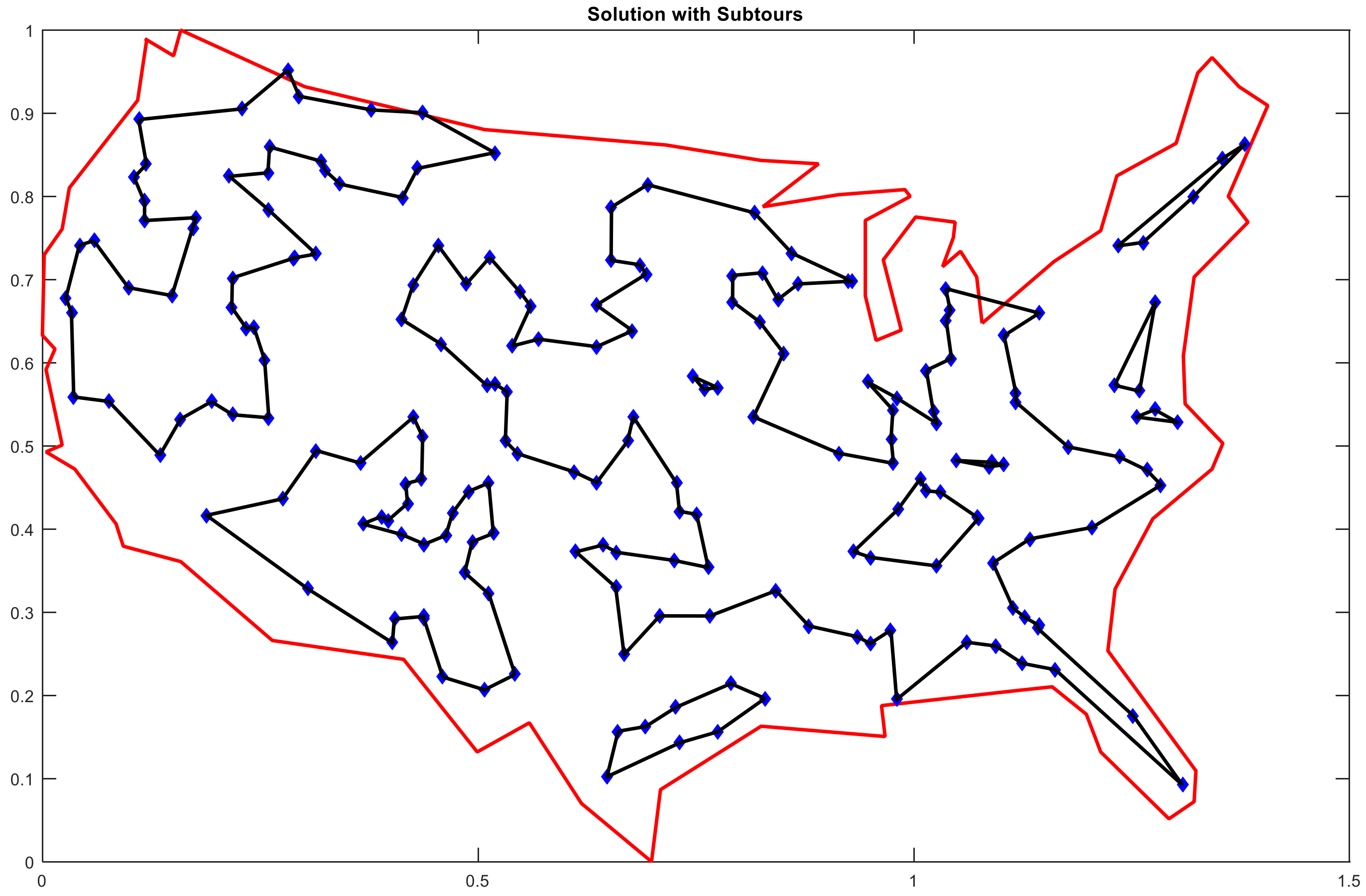
Traveling Salesperson Problem (TSP) – 28 subtours



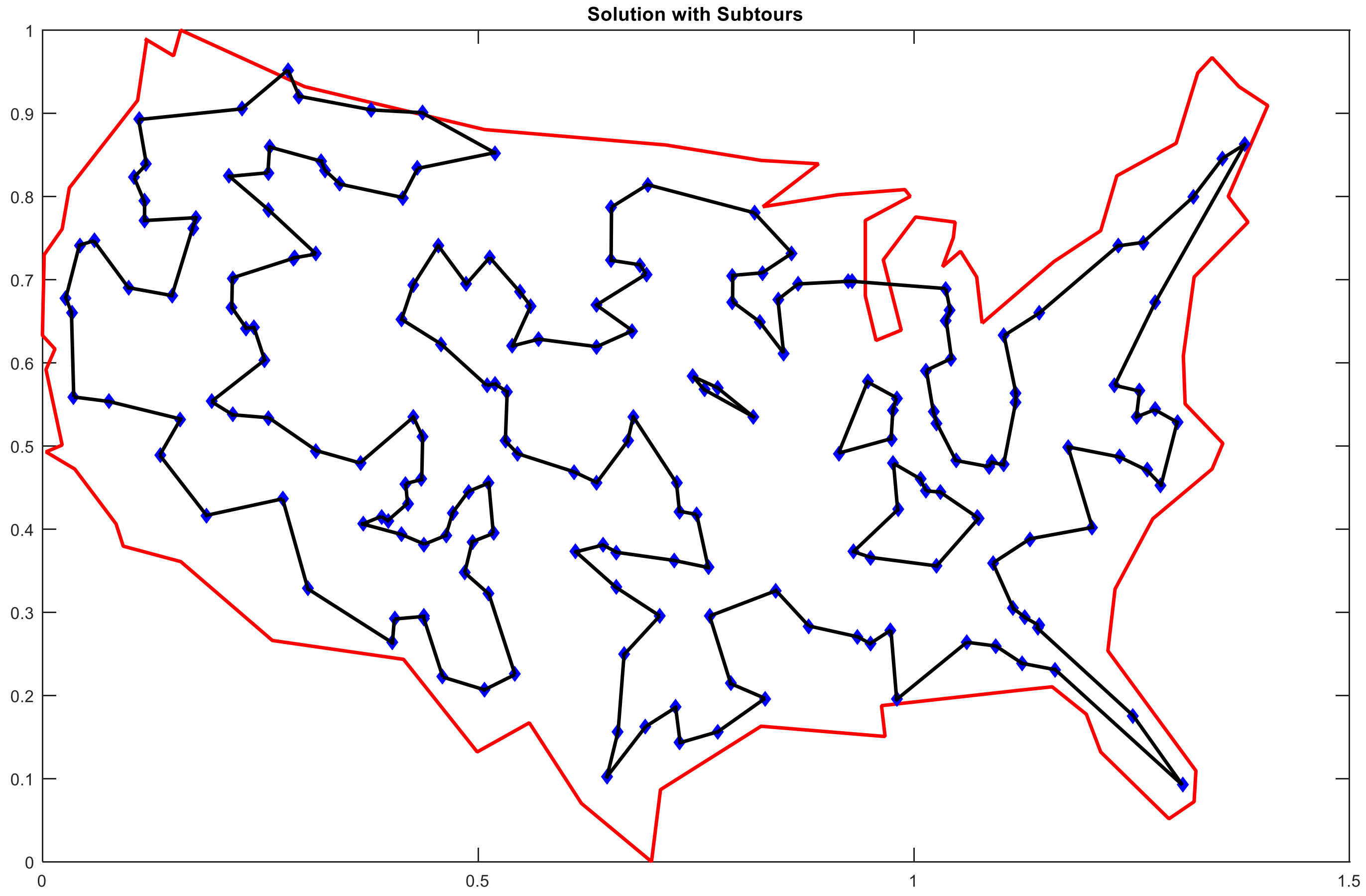
Traveling Salesperson Problem (TSP) – 22 subtours



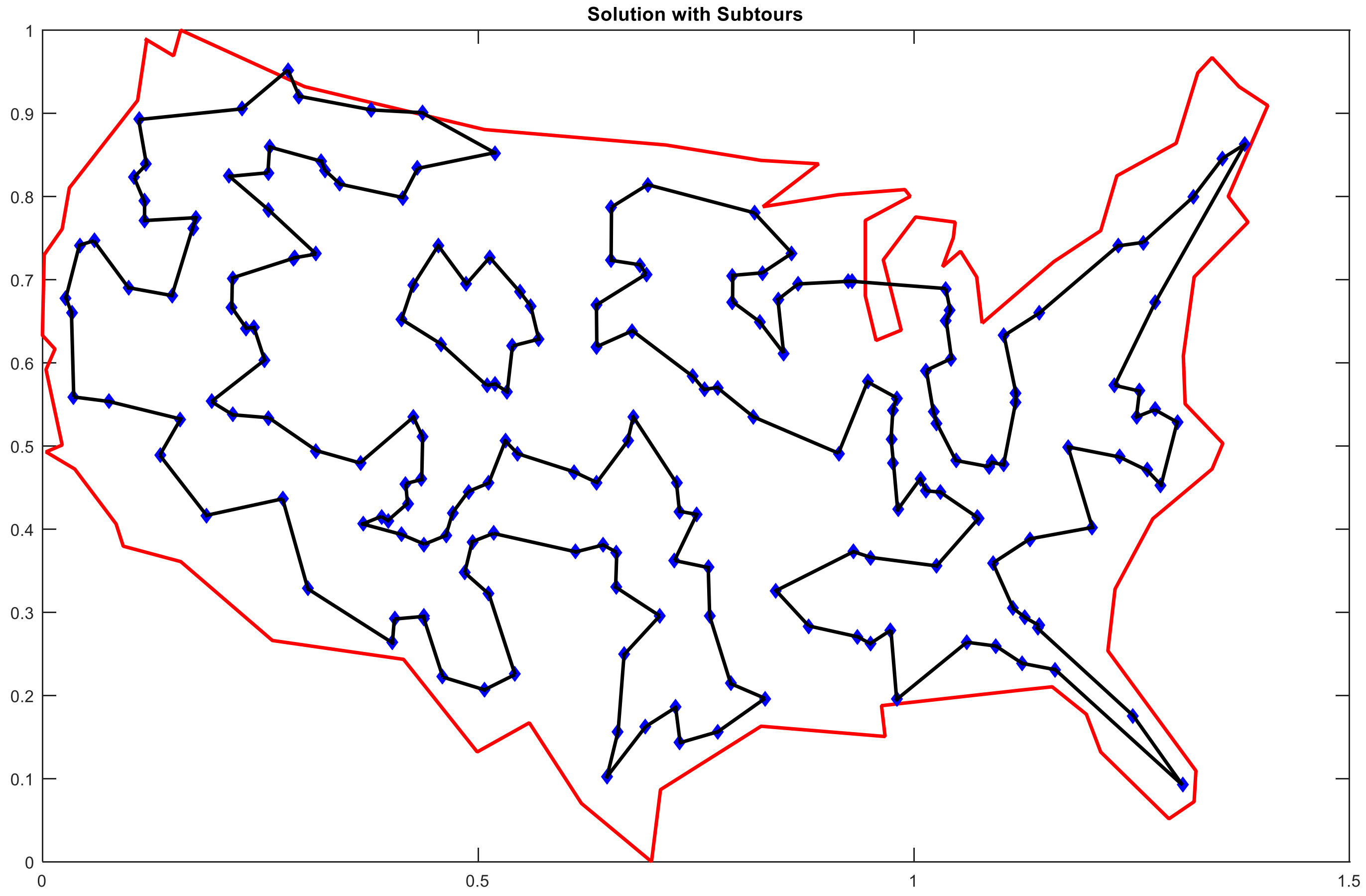
Traveling Salesperson Problem (TSP) – 10 subtours



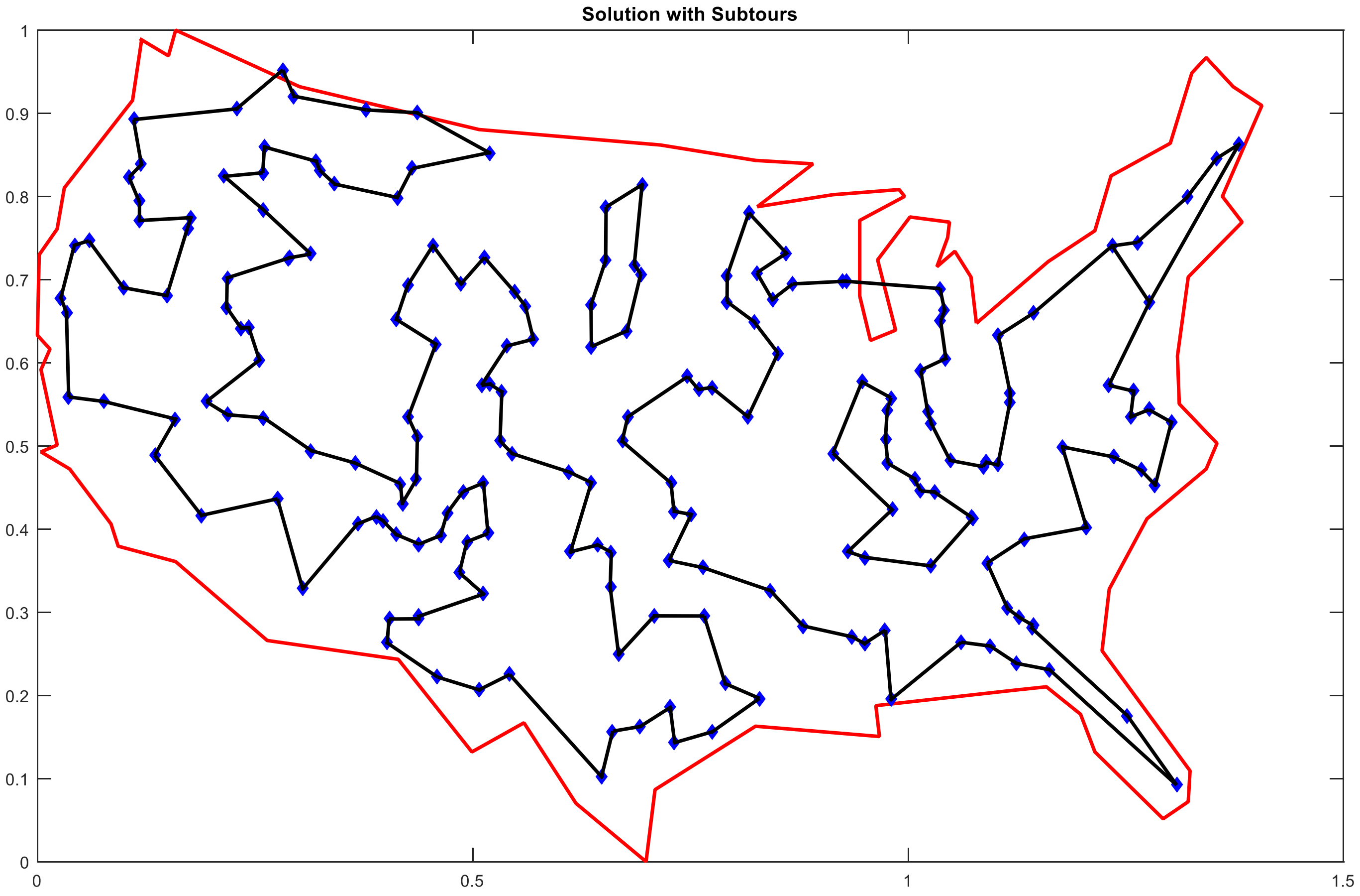
Traveling Salesperson Problem (TSP) – 5 subtours



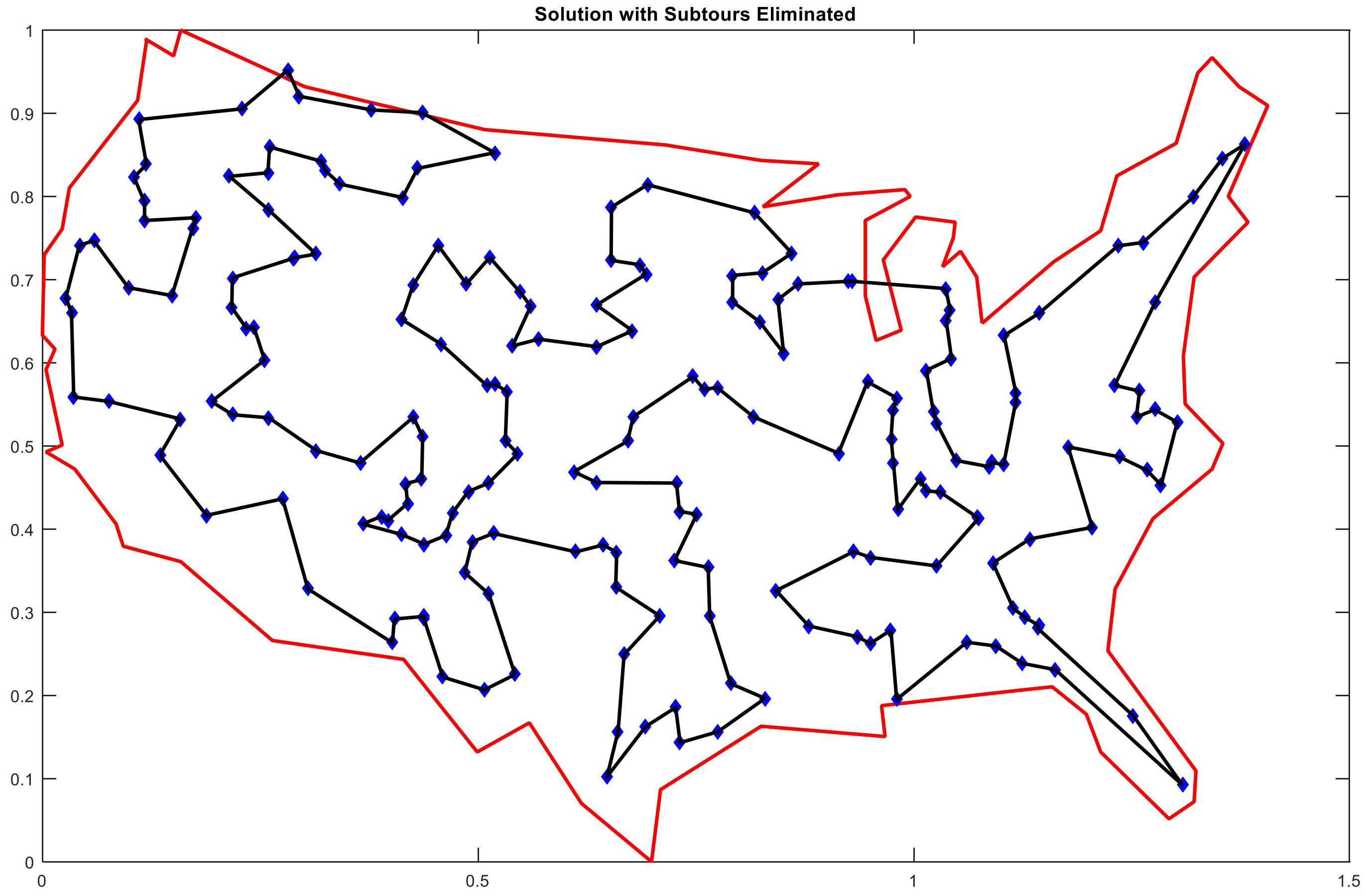
Traveling Salesperson Problem (TSP) – 3 subtours



Traveling Salesperson Problem (TSP) – 4 subtours



Traveling Salesperson Problem (TSP) – 1 tour



Traveling Salesperson Problem (TSP)

Sets:

- N: Set of nodes $\{1,2,3,\ldots,n\}$
- A: Set of arcs

Parameters:

- c_{ij}: cost of traveling through arc $(i.j) \in A$

Traveling Salesperson Problem (TSP)

Sets:

- N: Set of nodes $\{1,2,3,\dots,n\}$
- A: Set of arcs

Parameters:

- c_{ij}: cost of traveling through arc $(i.j) \in A$

Variables:

- x_{ij}: binary variable that is 1 if salesman travels through arc $(i.j) \in A$, and is 0 otherwise.
- u_i: label / order of visit of node $i \in N \backslash \{1\}$ (to eliminate subtours)

- Objective function:

$$\min z = \sum_{(i,j)\in A} c_{ij} x_{ij}$$

- Constraints:

$$\sum_{i:(i,j)\in A} x_{ij} = 1, \qquad \forall j \in N$$

Only depart from one node

$$\sum_{j:(i,j)\in A} x_{ij} = 1, \qquad \forall i \in N$$

Only arrive at one node

$$u_i \le u_j - 1 + n(1 - x_{ij}),$$
$$\forall (i,j) \in A: j \neq 1$$
$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in A$$
$$u_i \ge 0, \quad \forall i \in N \backslash \{1\}$$

TSP – Gurobi/Python Example

```
from gurobipy import *

# # The original distances from the Xpress example are given in DIST, and the node positions in POS
# # note that you could have different distances, for example if using the positions directly
# # also, kepp in mind that the distances in general don't have to be symmetric!

POS= [[24, 54], [58, 10], [88, 40], [85, 60], [22, 33], [56, 27], [46, 64]]

DIST= [[0, 786, 549, 657, 331, 559, 250],
       [786, 0, 668, 979, 593, 224, 905],
       [549, 668, 0, 316, 607, 472, 467],
       [657, 979, 316, 0, 890, 769, 400],
       [331, 593, 607, 890, 0, 386, 559],
       [559, 224, 472, 769, 386, 0, 681],
       [250, 905, 467, 400, 559, 681, 0]]

#Initialize sets and parameters as empty (tuplelists and dictionaries, respectively)
N=tuplelist([])
A=tuplelist([])
c={}

#If you prefer, regular lists could be used as well in most cases
# N=[]
# A=[]

#read the lists and positions and use it to create the set of Nodes (N), set of Arcs(A),
#and the parameters of distances between nodes (c)
for i, pos_i in enumerate(POS):
    N.append(i)
    for j, pos_j in enumerate(POS):
        if j!=i:
            A.append((i,j))
            c[i,j]=DIST[i][j]
            #you could also calculate the eucliden distance between each pair of nodes:
            #c[i,j]=round(((pos_i[0]-pos_j[0])**2+(pos_i[1]-pos_j[1])**2)**0.5)


# n is the number of Nodes
n=len(N)
```

TSP – Gurobi/Python Example

```
# Create optimization model
m = Model('TSP')

# Create variables (and add coefficients of the objective function)
# variable "x_{ij}"
x = m.addVars(A, obj=c, name="x", vtype=GRB.BINARY)
u = m.addVars(N, obj=0, name="u")

# constraints (1)
m.addConstrs(
    (x.sum('*',j)==1 for j in N)
    , "arriveFromNode")

# constraints (2)
m.addConstrs(
    (x.sum(i,'*')==1 for i in N)
    , "goToNode")

# constraints (3)
m.addConstrs(
    (n*(1-x[i,j])>=u[i]-u[j]+1 for (i,j) in A if (j!=0))
    , "timeLabels")


# Compute optimal solution
#modelSense is 1 for minimization (default) or -1 for maximization)
m.setAttr("modelSense", 1)
#OutputFlag is 1 for automatic output (default) or 0 to avoid default output)
m.setParam('OutputFlag', 0)
m.optimize()

# Print solution
if m.status == GRB.Status.OPTIMAL:
    solution_OF= m.objVal
    solution_x = m.getAttr('x', x)
    solution_u = m.getAttr('x', u)
    print('\nOptimal Objective Function: %g' %solution_OF)
    print('\nOptimal path:')
    for i,j in A:
        if solution_x[i,j] > 0:
            print('%s -> %s' % (i, j))
```

Force label to be incremented

```
Optimal Objective Function: 2575

Optimal path:
0 -> 6
1 -> 5
2 -> 1
3 -> 2
4 -> 0
5 -> 4
6 -> 3
```

THANK YOU

QUESTIONS?

Andrés D. González | andres.gonzalez@ou.edu