Métodos de Apoio à Decisão

Problemas de rotas: o Problema do Caixeiro Viajante

João Pedro Pedroso

2023/2024

- Últimas aulas: problemas de otimização em grafos
  - partição
  - clique máxima
- Próximas aulas: problemas de determinação de rotas (*routing problems*)
  - problema do caixeiro viajante (*traveling salesman problem*)
  - algoritmos com planos de corte (*cutting plane algorithms*)
  - problema de rotas de veículos (*vehicle routing problems*)

Problemas de determinação de rotas

Routing problems

Problemas de determinação de rotas

Vamos considerar vários problemas relacionados com o roteamento de veículos, discutindo e caracterizando diferentes formulações:

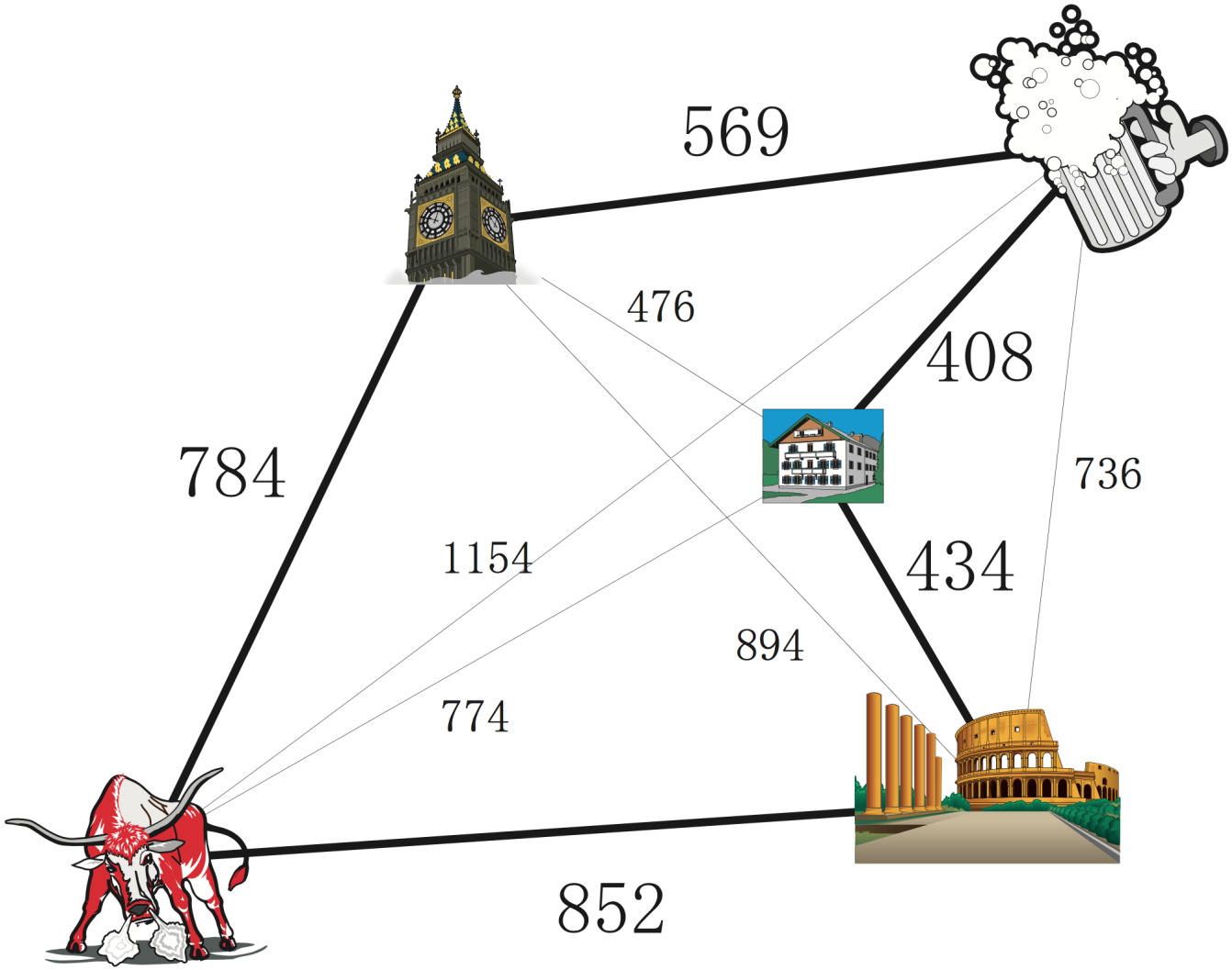
- Problema do caixeiro viajante (TSP)
  - Traveling Salesman Problem
- TSP com janelas temporais
  - TSP with time windows
- Problema de planeamento de entregas com restrições de capacidade
  - capacity-constrained delivery planning problem/vehicle routing problem

Problema do Caixeiro Viajante

Traveling Salesman Problem

Problema do Caixeiro Viajante: examplo

Representação num grafo de cidades europeias



(solução ótima: linha carregada)

Problema do Caixeiro Viajante

Está a pensar tirar férias e fazer um tour pela Europa.
Está atualmente em Zurique, Suíça (centro) e seu objetivo é visitar todas as atrações representadas.
Decidiu alugar um helicóptero, mas tem que pagar uma alta taxa de aluguer, proporcional à distância percorrida.
Depois de visitar as outras quatro cidades (Madrid, Londres, Roma, Berlim), deseja regressar a Zurique, tudo isso percorrendo a distância mais curta possível. A distância percorrida entre as cidades está representada na figura. Em que ordem deve viajar para que a distância seja minimizada?

Definições preliminares

Recorde: dado um grafo $G = (V, E)$

- um passeio é uma sequência de vértices $v_1, v_2, \ldots, v_k$ tal que $\forall i \in 1, 2, \ldots, k-1$, os vértices v_i e v_{i+1} são adjacentes
  - *i.e.*, existe uma aresta com as extremidades v_i e v_{i+1}
- um caminho é um passeio onde $v_i \neq v_j, \forall i \neq j$
  - passeio que visita cada vértice no máximo uma vez
- um ciclo é um caminho fechado, *i.e.*, um caminho com a aresta adicional $\{v_k, v_1\}$
- um tour/circuito é um ciclo que visita cada vértice $v \in V$ exatamente uma vez
  - subtour: ciclo que visita um subconjunto próprio de V
- um grafo é conexo se existe um caminho entre cada par de vértices

Preliminary definitions

Recall: given a graph $G = (V, E)$

- a walk is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $\forall i \in 1, 2, \ldots, k-1$, vertices v_i and v_{i+1} are adjacent
  - *i.e.*, there is an edge with endpoints v_i and v_{i+1}
- a path is a walk where $v_i \neq v_j, \forall i \neq j$
  - path is a walk that visits each vertex at most once
- a cycle is a closed path, *i.e.*, a path combined with the edge $\{v_k, v_1\}$
- a tour is a cycle that visits every vertex $v \in V$ exactly once
  - subtour: cycle visiting a proper subset of V
- a graph is connected if there exists a path between each pair of vertices

Traveling salesman problem (TSP): definition

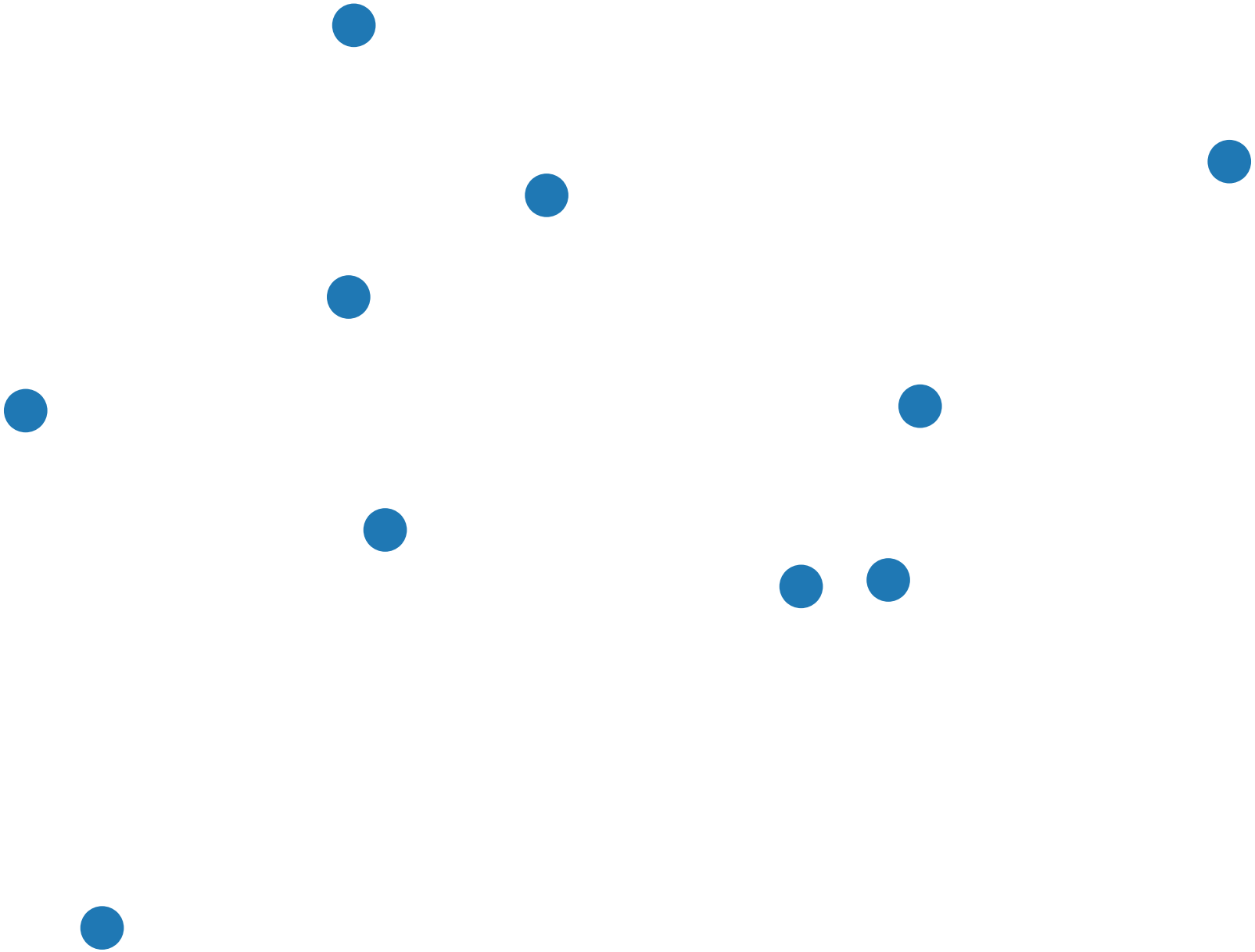
Given:

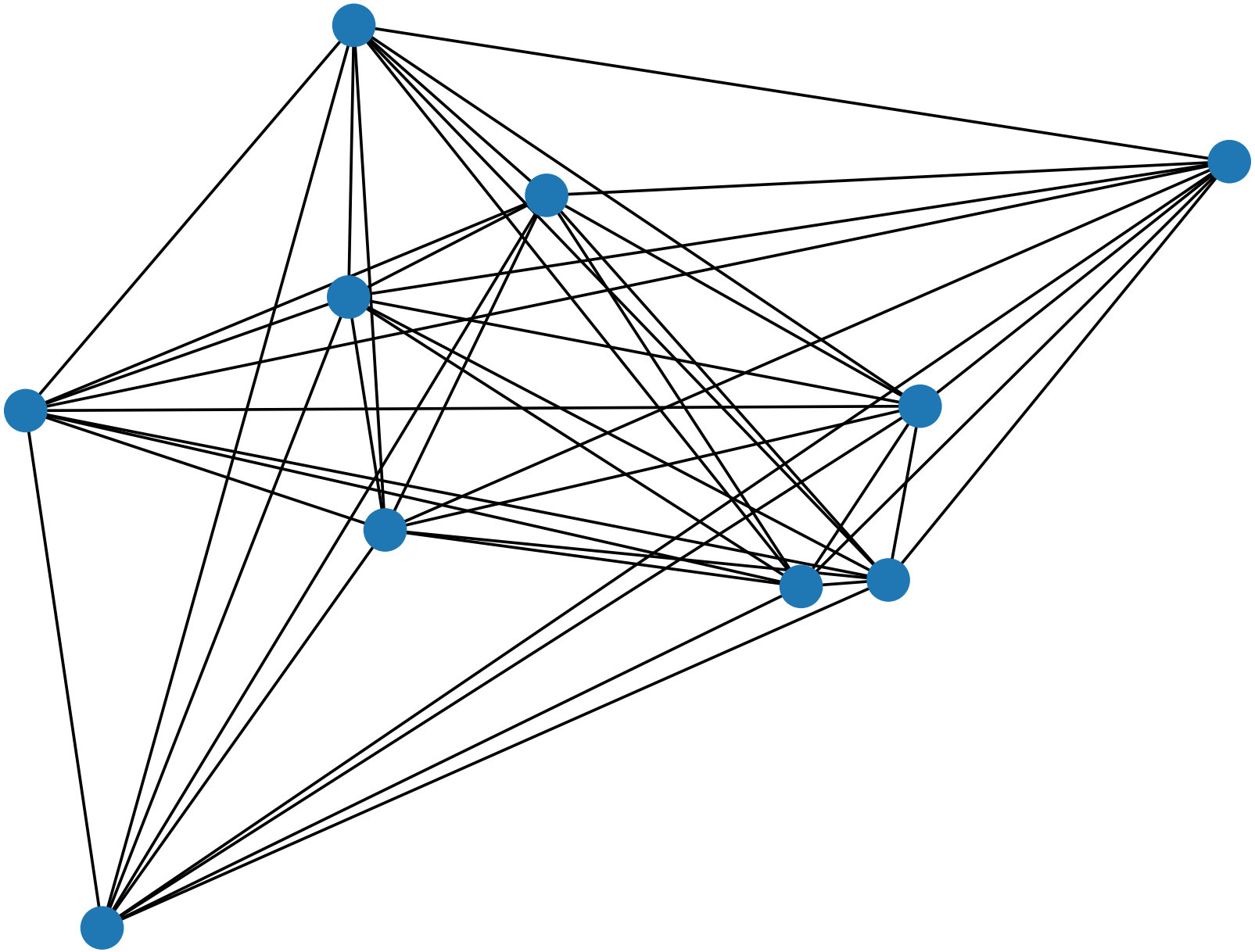
- undirected graph $G = (V, E)$ with n vertices (cities)
- function $c : E \to \mathbb{R}$ associating a distance to each edge
  - or weight, cost, travel time

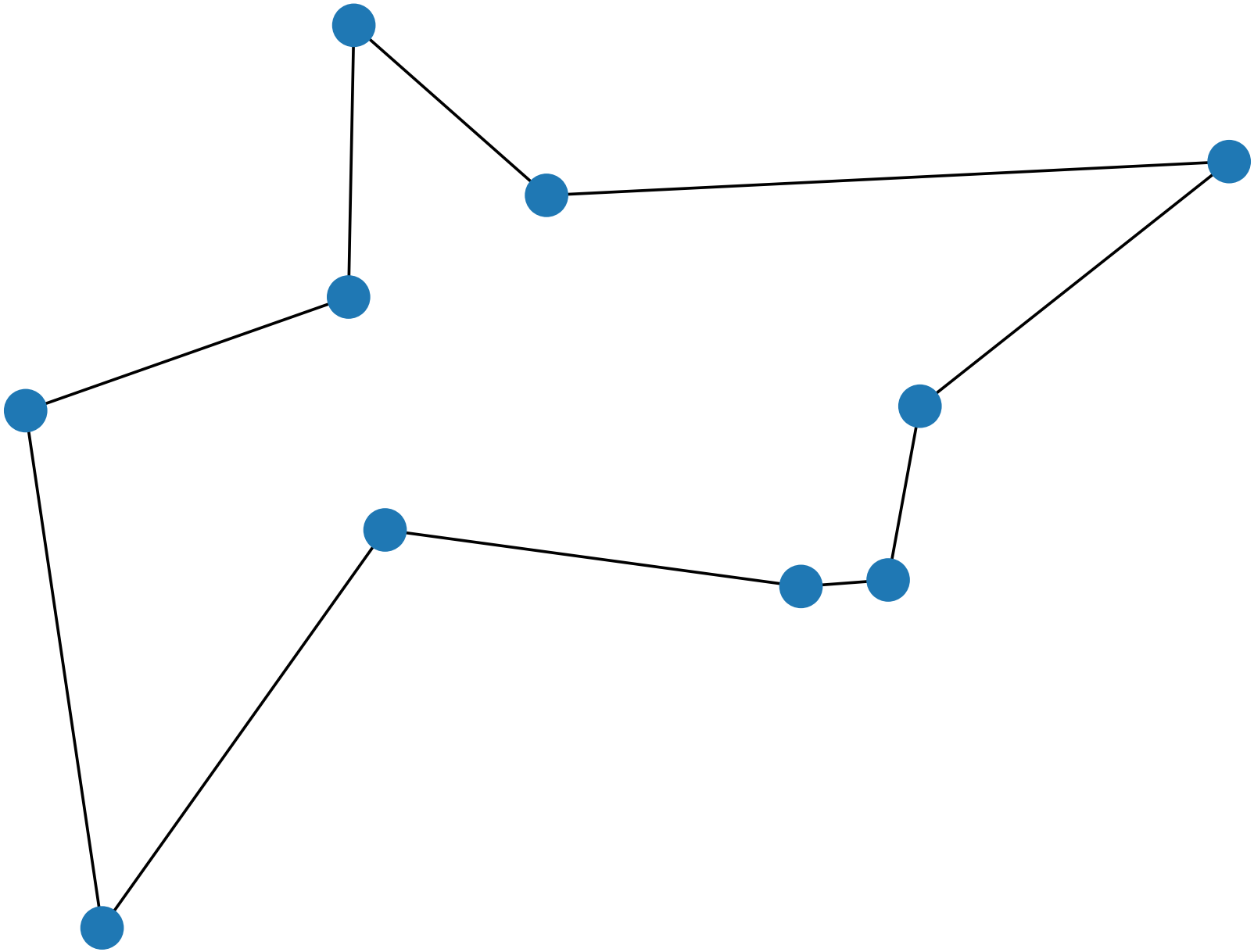
Find a tour that passes exactly once in each city and minimizes the total distance (*i.e.*, the length of the tour)

Terminology

- If problem is defined on an undirected graph
  - symmetric traveling salesman problem
  - distance any point a to another point b is the same as the distance from b to a
  - graph assumed to be complete
- If problem defined on a directed graph
  - asymmetric traveling salesman problem
  - distance for going from a point to another may be different of the returning distance
- Symmetric TSP is a special case of asymmetric TSP
  - a formulation for the asymmetric case can be applied to symmetric problems





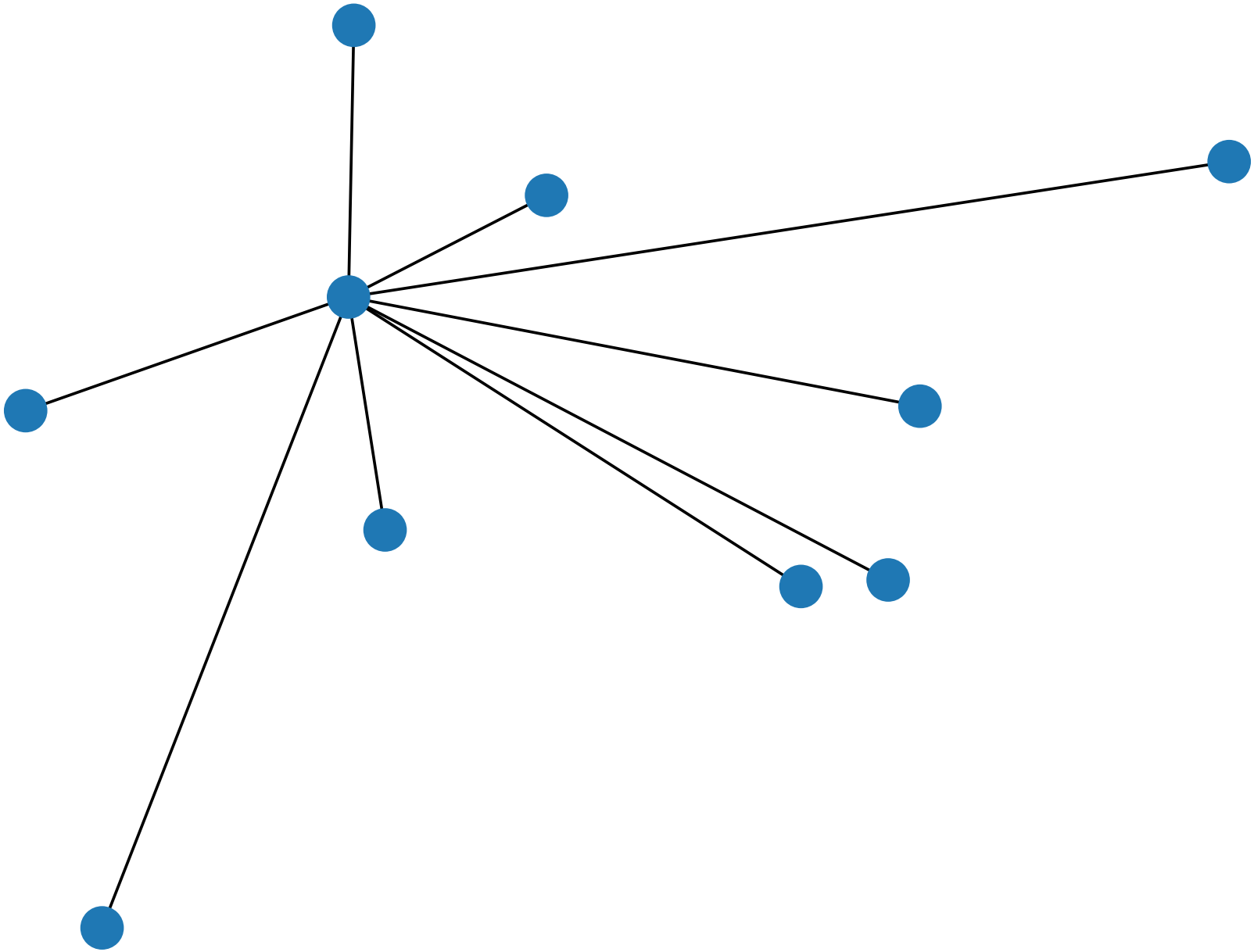


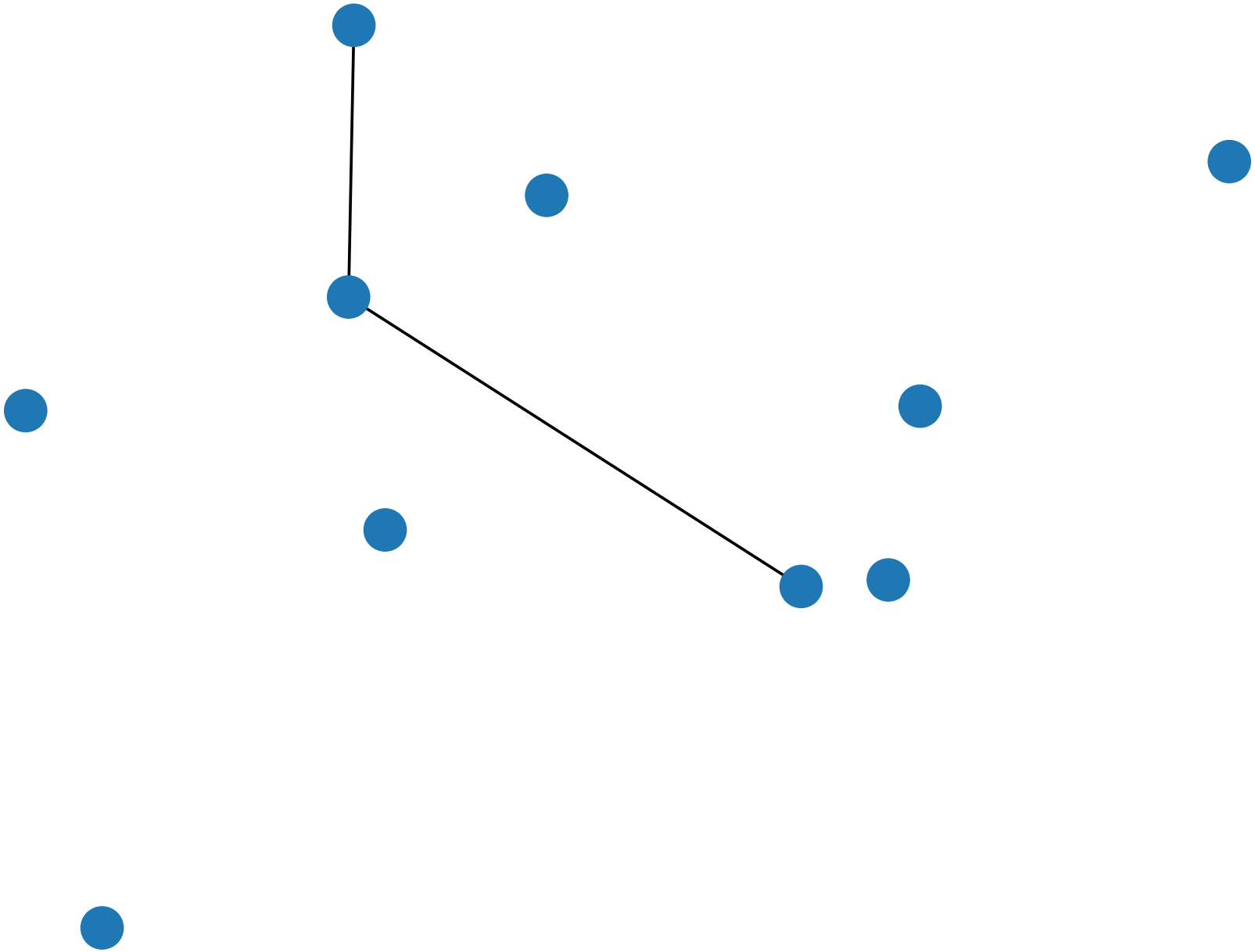
A preliminary formulation

- Data:
  - a complete graph $G = (V, E)$, defining:
    - $N \rightarrow$ number of vertices/cities
    - $\delta(\{k\}) = \{ij \in E : i = k \text{ or } j = k\} \rightarrow$ edges incident to k
  - distances $c_e, \forall e \in E$
- Variables: $x_e \rightarrow$ edges selected for the tour:
  - $x_e = 1$ if edge $e \in E$ is in the tour, 0 otherwise
- Objective: minimize cost

$$\text{minimize} \quad \sum_{e \in E} c_e x_e$$

- *What are the constraints?*
  - degree constraints

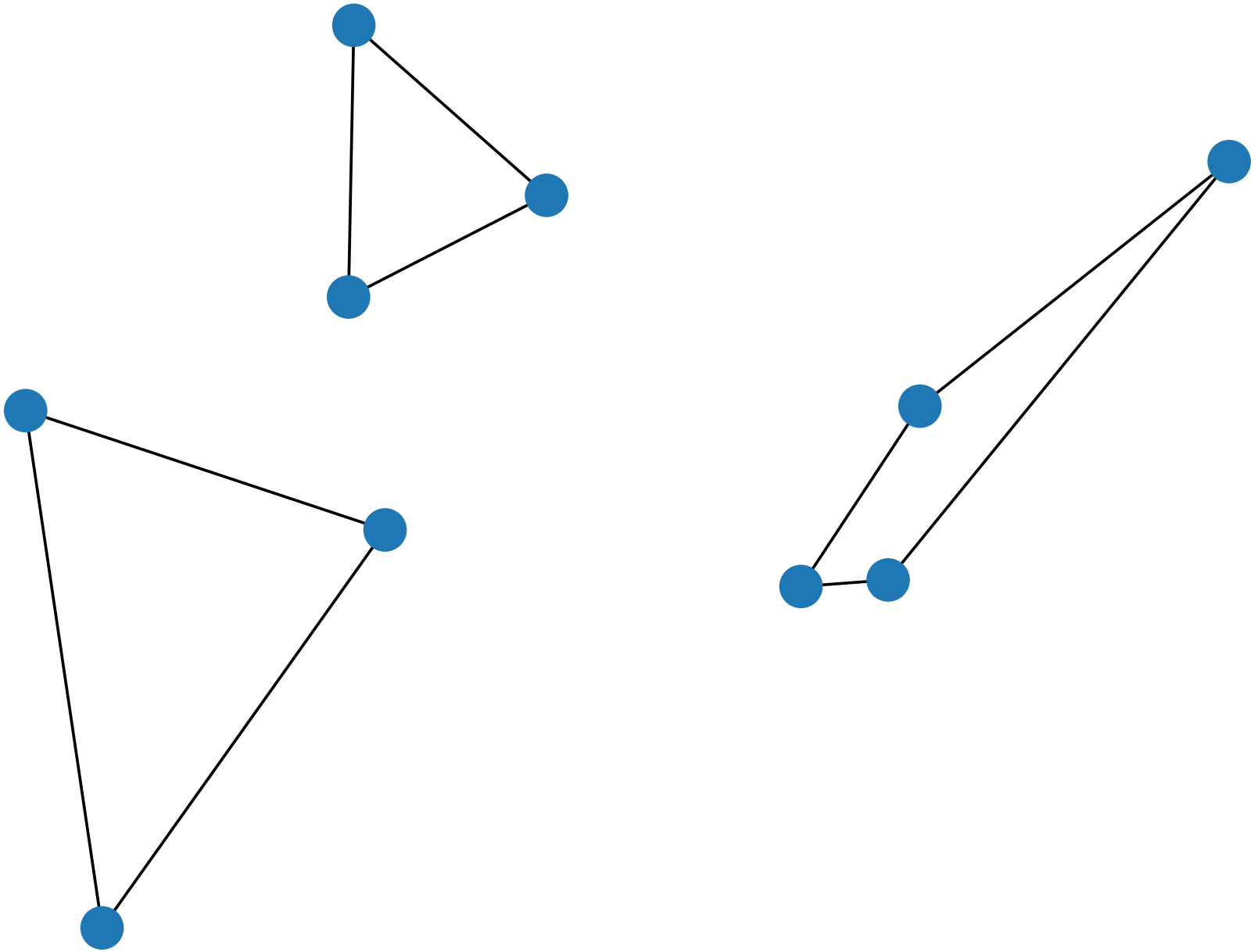




Preliminary formulation

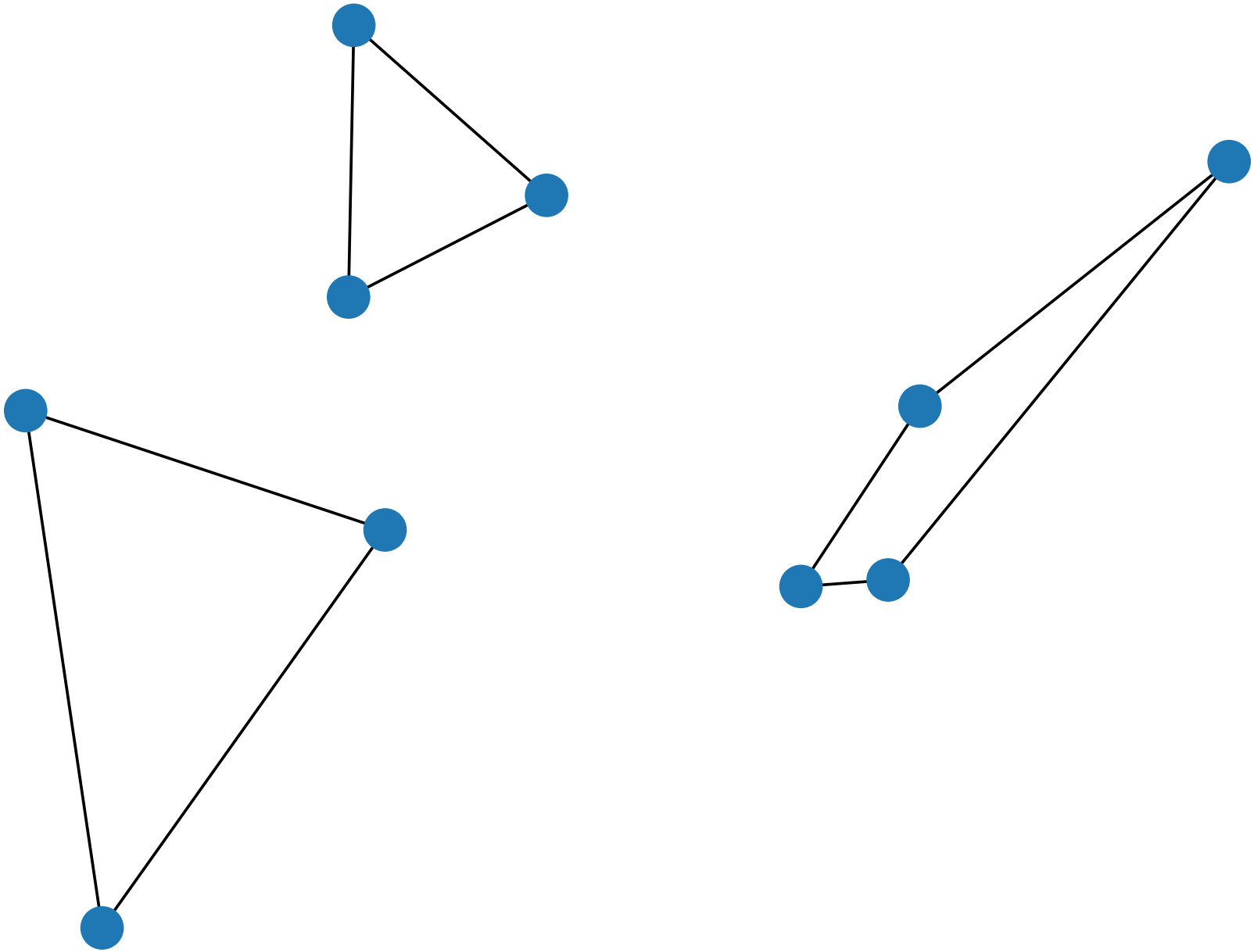
- Data:
  - a complete graph $G = (V, E)$, defining:
    - $N \rightarrow$ number of vertices/cities
    - $\delta(\{k\}) = \{ij \in E : i = k \text{ or } j = k\} \rightarrow$ edges incident to k
  - distances $c_e, \forall e \in E$
- Variables: $x_e \rightarrow$ edges selected for the tour:
  - $x_e = 1$ if edge $e \in E$ is in the tour, 0 otherwise

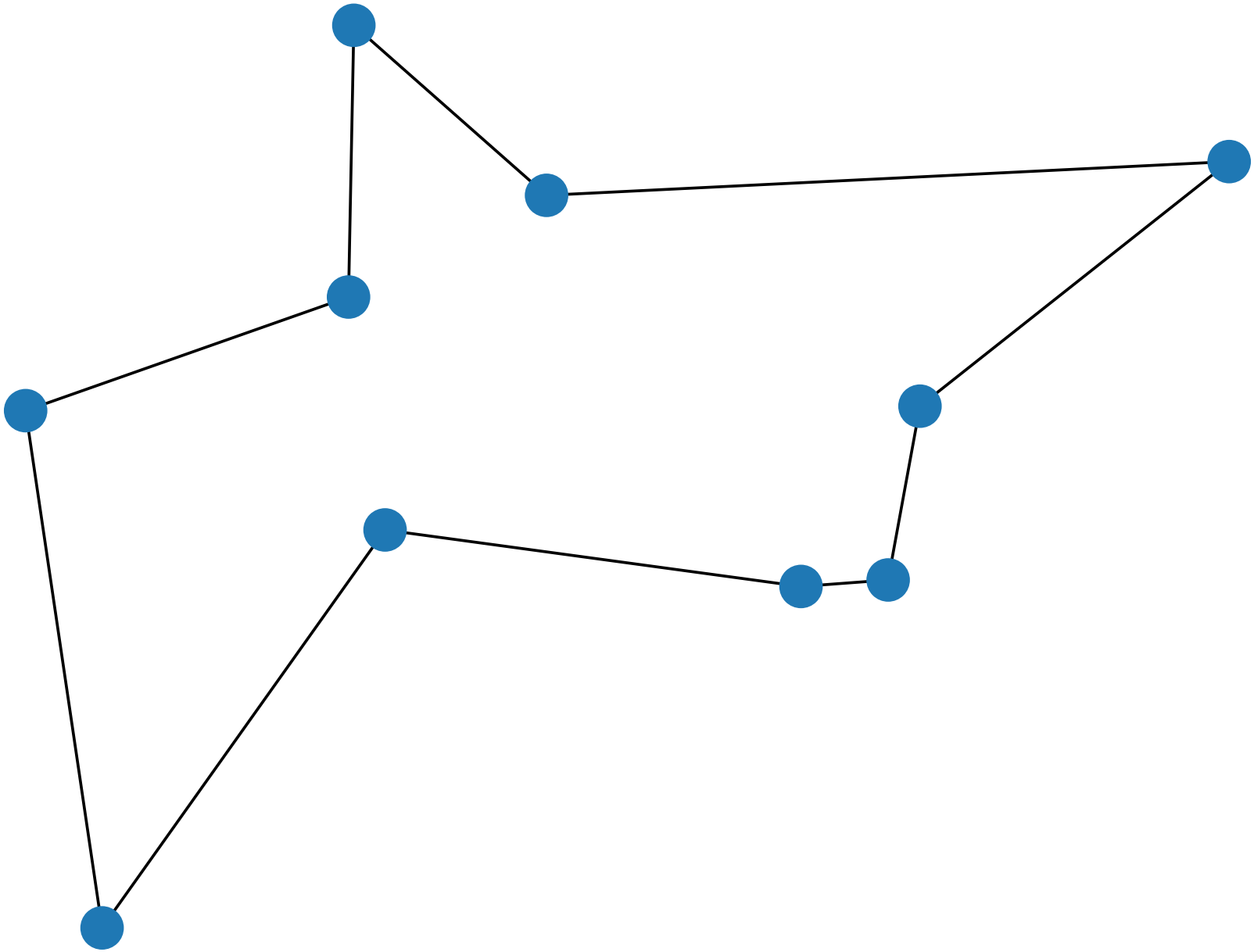
$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c_e x_e \\ \text{subject to} \quad & \sum_{e \in \delta(\{i\})} x_e = 2 \qquad \forall i \in V \end{aligned}$$



Formulation for the symmetric case

- Model by Dantzig-Fulkerson-Johnson (1954)
- "Preliminary model" + subtour elimination constraints
- Cutting plane algorithm
  - solve a relaxed version (initially, only with *degree constraints*)
  - identify subtours
    - cycles with less than N vertices, where $N = |V|$
  - add constraints for removing these subtours
  - repeat, until there are *no more subtours*





TSP: Subtour elimination formulation: notation

- For a subset S of vertices:
  - $E(S) \rightarrow$ set of edges whose endpoints are both included in S
  - $\delta(S) \rightarrow$ set of edges such that *one of the endpoints is included in S and the other is not*
- In order to have a traveling route:
  - number of selected edges connected to each vertex must be two
  - salesman must pass through all the cities
  - $\rightarrow$ any tour which does not visit all the vertices in set V must be prohibited
- One possibility for ensuring this: require that for any proper subset $S \subset V$ with cardinality $|S| \geq 2$, the number of selected edges whose *endpoints are both in S* is at most $|S| - 1$

TSP: Subtour elimination formulation: notation

$$\text{minimize} \quad \sum_{e \in E} c_e x_e$$

$$\text{subject to} \quad \sum_{e \in \delta(\{i\})} x_e = 2 \qquad \forall i \in V$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad \forall S \subset V, 2 \leq |S| \leq |V| - 2$$

$$x_e \in \{0, 1\} \qquad \forall e \in E$$

- First constraint: degree constraint
- Second constraint: subtour elimination inequality
  - it excludes partial tours, *i.e.*, cycles which pass through a proper subset of vertices
  - valid cycles must pass through all of the vertices

TSP: cutset inequality

- Another possibility for subtour elimination constraints
- For each proper subset of constraints, there must be at least two outgoing edges
- Consider the degree constraint of an individual vertex:

$$\sum_{e \in \delta(\{i\})} x_e = 2$$

- For a given subset S of vertices, let us
  - sum the double both sides of the subtour elimination inequality
  - subtract the degree constraint for each vertex $i \in S$
  - we obtain the cutset inequality
    - for the traveling salesman problem: has the same strength as the subtour elimination inequality

$$\sum_{e \in \delta(S)} x_e \geq 2, \qquad \forall S \subset V, |S| \geq 2.$$

TSP: Subtour elimination formulation: cutting plane method

- The number of subsets of a set increases exponentially with the size of the set
- Hence, for any moderate size instance the number of subtour elimination constraints/cutset constraints is extremely large
- We cannot afford solving the complete model $\rightarrow$ resort to the so-called cutting plane method:
  - subtour elimination constraints are added as necessary

Definition: separation problem

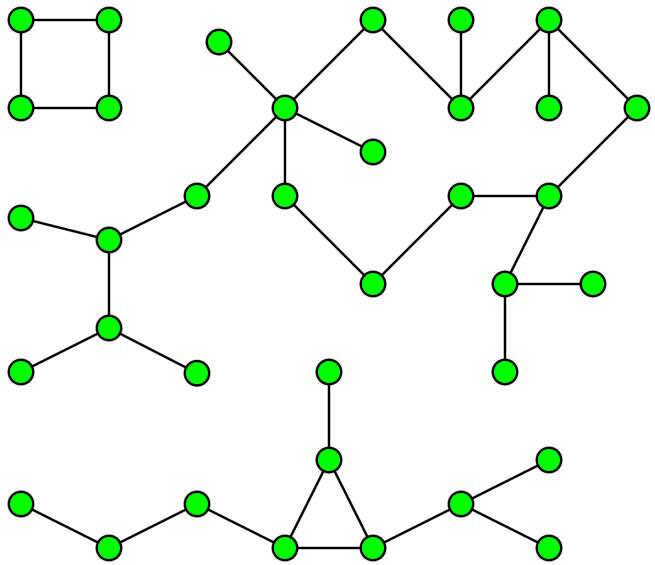
- Let $\bar{x} \rightarrow$ solution of the linear relaxation of the problem using only a subset of constraints
  - $\bar{x}$ may be fractional, not necessarily 0 or 1
- Problem of finding a constraint of the original problem that is not satisfied by $\bar{x}$: separation problem
- To design a cutting plane method $\rightarrow$ we need an efficient algorithm for the separation problem
- There are several possibilities; we'll check if there is only one connected component in the solution edges

Implementation: ampl model

```
set V;
param c {i in V, j in V: i < j};
param n;   # number of cutting planes
set S {1..n} within V;

var x {V, V} binary;

minimize z: sum {i in V, j in V: i < j} c[i,j] * x[i,j];

subject to
Degree {i in V}:
    sum {j in V: j < i} x[j,i] +
    sum {j in V: j > i} x[i,j] = 2;
Sep {k in 1..n}:
    sum {i in S[k], j in S[k]: i < j}
        x[i,j] <= card(S[k]) - 1;
```

Implementation: subtour elimination

- To find subtours:
  - find connected components for the graph consisting of the edges for which x_e is positive
  - convenient method in Python/NetworkX
  - "easy" problem



- Connected graph: if there is a path between any pair of its vertices
- Connected component: a maximal connected subgraph
  - a connected subgraph such that no other connected subgraph strictly contains it
- To decompose solution into connected components: *next slide*

Implementation: finding connected components in current solution

- Input:
  - set E of edges in current solution
    - used to find a subtour elimination constraint corresponding to connected components $S \subset V$
  - ampl model object
    - used to add newly found subtour elimination constraints
- Output: `True` if some constraint was added, `False` otherwise

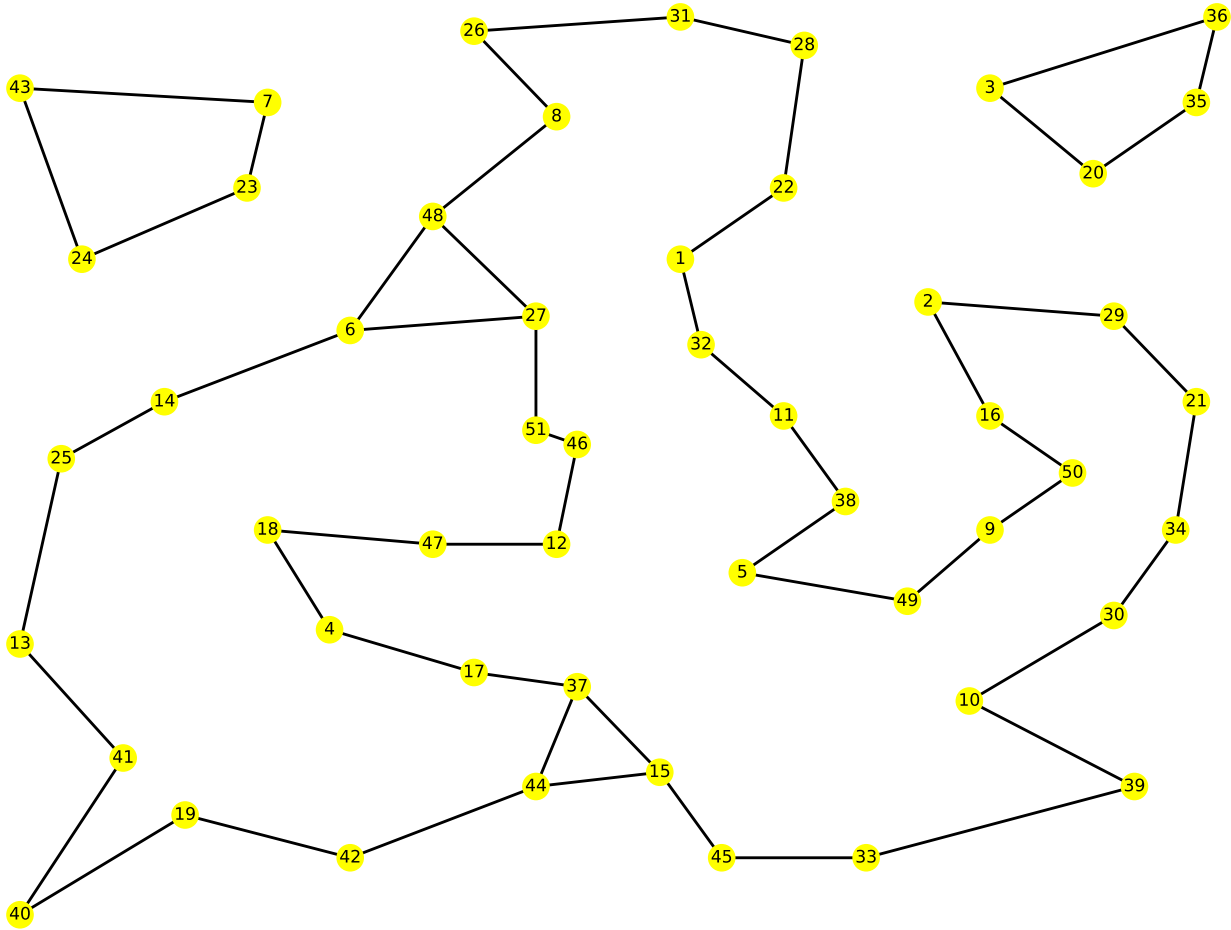
```
def addcut(ampl, edges):
    G = networkx.Graph()
    G.add_edges_from(edges)
    Components = list(networkx.connected_components(G))
    if len(Components) == 1:
        return False
    n = int(ampl.param['n'].value() + .5)
    for S in Components:
        n += 1
        ampl.param['n'] = n
        ampl.set['S'][n] = S
    return True
```

Implementation: solving the TSP

```
def solve_tsp(V, c):
    from amplpy import AMPL, Environment, DataFrame
    ampl = AMPL()
    ampl.option['solver'] = 'gurobi'
    ampl.option['relax_integrality'] = True
    ampl.read("tsp.mod")
    ampl.set['V'] = V
    ampl.param['c'] = c
    ampl.param['n'] = 0

    while True:
        ampl.solve()
        z = ampl.obj['z']
        x = ampl.var['x']
        edges = [(i,j) for i in V for j in V if i < j and x[i,j].value() > EPS]
        if addcut(ampl, edges) == False:
            if not ampl.option['relax_integrality']:
                break
            print("making model integer")
            ampl.option['relax_integrality'] = False
    return z, edges
```

Notes

- We switch the variables to `integer` just at the end
- It would possible to add constraints on the continuous variables to avoid the possible shapes of continuous solutions



Cutting plane and branch-and-cut methods

- Cutting plane method: originally applied to the TSP in 1954
  - George Dantzig $\rightarrow$ one of the founders of linear optimization
  - Ray Fulkerson
  - Selmer Johnson

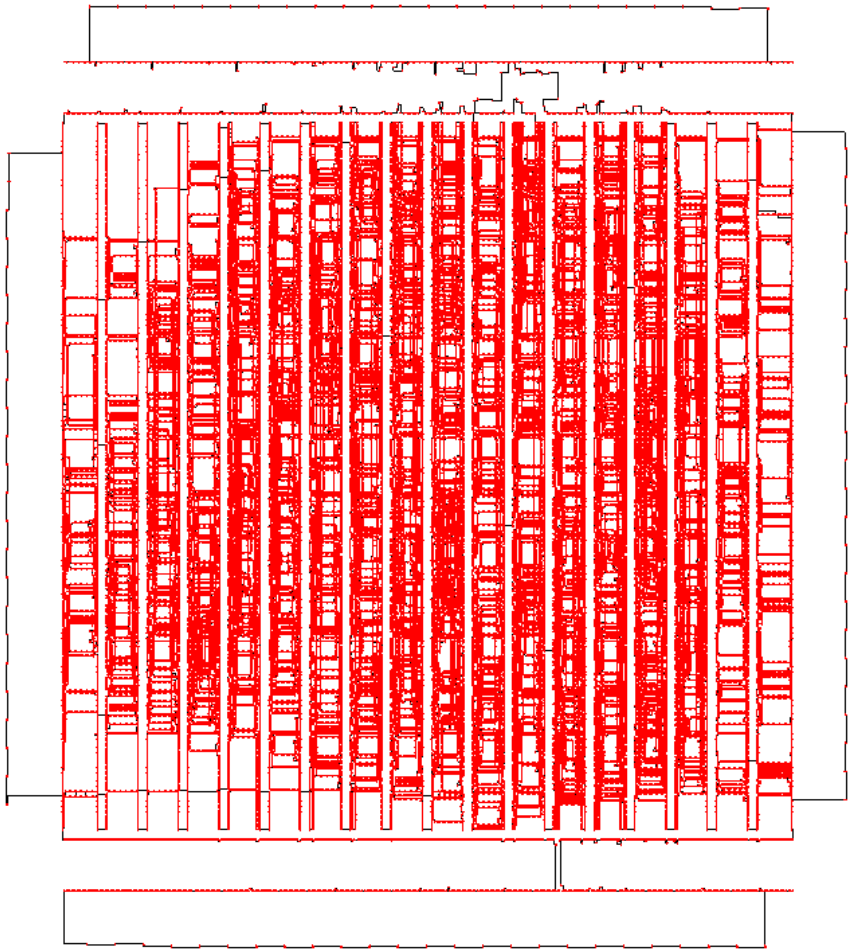
Modeling tip

When the number of constraints is very large, use the cutting plane method or the branch-and-cut method.

- In formulations in which the number of constraints becomes enormous, it is necessary to use the cutting plane method for finding only the necessary constraints
- An example is the method for subtour elimination in the traveling salesman problem

(Symmetric) TSP sizes:

As of 2020, largest instances solved to optimality:
85,900 Locations in a VLSI Application *(Solved in 2006)*



Asymmetric TSP

Miller-Tucker-Zemlin (potential) formulation

- Consider an asymmetric traveling salesman problem
- We will now see a formulation with a number of constraints of polynomial order

  Given:
  - *directed graph* $G = (V, A)$, *where*
    - $V \rightarrow$ *set of vertices*
    - $A \rightarrow$ *a set of (directed) arcs*
  - *function* $c : A \rightarrow \mathbb{R}$ *associating a distance to each arc*

  Find a tour which passes exactly once in each city and minimizes the total distance

Miller-Tucker-Zemlin formulation: notation

- Variables:
  - (binary) $x_{ij} = 1$ if visiting vertex j next to i, 0 otherwise
  - (real) $u_i \rightarrow$ represents the visiting order of vertex i
- Interpretation: u_1 is the potential at each vertex
  - starting point: $u_1 = 0$
- When visiting j next to vertex i:
  - force the potential of j to be $u_j = u_i + 1$
  - do this for any vertex except 1
  - possible values for u_i are $1, 2, \ldots ..., n-1$
    - $n \rightarrow$ number or vertices

MTZ formulation for the asymmetric traveling salesman problem

$$\text{minimize} \quad \sum_{i \neq j} c_{ij} x_{ij}$$

$$\text{subject to} \quad \sum_{j:j \neq i} x_{ij} = 1 \qquad i = 1, \ldots, n$$

$$\sum_{j:j \neq i} x_{ji} = 1 \qquad i = 1, \ldots, n$$

$$u_i + 1 - (n-1)(1 - x_{ij}) \leq u_j \qquad i = 1, \ldots, n, \quad j = 2, \ldots, n : i \neq j$$

$$0 \leq u_i \leq n - 1 \qquad i = 1, \ldots, n$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \neq j$$

Remarks

- First and second constraints: assignment constraints $\rightarrow$ ensure that each vertex is incident to one outgoing arc and one incoming arc.
- Third constraint: Miller-Tucker-Zemlin or potential constraint $\rightarrow$ define the order in which each vertex i is visited on a tour.
  - $u_i \rightarrow$ potential at vertex i
  - forcing $u_j = u_i + 1$ only when $x_{ij} = 1$ in
    $$u_i + 1 - (n-1)(1 - x_{ij}) \leq u_j$$
    - coefficient $n-1$ of term $(1 - x_{ij}) \rightarrow$ *"Big M" constraint*
- Fourth constraint: upper and lower bounds of the potential.
- This formulation is much weaker than subtour elimination constraints

AMPL model

```
param n;
set V := {1..n};
param c {i in V, j in V};

var x {i in V, j in V: i != j} binary;
var u {V} >= 0, <= n-1;

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

subject to
OutDegree {i in V}:
    sum {j in V: j != i} x[i,j] = 1;
InDegree {i in V}:
    sum {j in V: j != i} x[j,i] = 1;
Potential {i in 1..n, j in 2..n : i != j}:
    u[i] + 1 - (n-1) * (1-x[i,j]) <= u[j];
```

Traveling Salesman Problem with Time Windows

Traveling Salesman Problem with Time Windows

- TSP variant
- time window: period for visiting each city of the TSP
- applications: scheduling, routing, transportation, ...

One-index Potential Formulation

- Based on the TSP model given by Miller-Tucker-Zemlin
- Key differences:
  - for each city, time to depart is $t_i, \quad i = 1, \ldots, n$
  - lower and upper time limits: $e_i \leq t_i \leq \ell_i, \forall i = 1, \ldots, n$
  - if we arrive at point i earlier than time e_i, we can wait
- Basis: potential (Miller-Tucker-Zemlin) constraint for ATSP
- When visiting j next to i ($x_{ij} = 1$):

  $$t_i + c_{ij} - M(1 - x_{ij}) \leq t_j \qquad \forall i, j : j \neq 1, i \neq j$$

  - $M \rightarrow$ constant representing a large number.
  - movement time c_{ij} assumed to be positive
  - smaller value of $M \rightarrow$ stronger constraint
    - when $x_{ij} = 0 \rightarrow M \geq t_i + c_{ij} - t_j$
    - for being feasible, since $t_i \leq \ell_i$ and $t_j \geq e_j \rightarrow M \geq \ell_i + c_{ij} - e_j$
    - on the other hand, $M > 0$, obtaining:
      $t_i + c_{ij} - [\ell_i + c_{ij} - e_j]^+(1 - x_{ij}) \leq t_j \quad \forall i, j : j \neq 1, i \neq j$
- Potential + upper and lower limit constraints can be further enhanced by lifting

Formulation

$$\text{minimize} \quad \sum_{i \neq j} c_{ij} x_{ij}$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{j:j \neq i} x_{ij} = 1 && \forall i = 1, \ldots, n \\
& \sum_{j:j \neq i} x_{ji} = 1 && \forall i = 1, \ldots, n \\
& t_i + c_{ij} - [\ell_i + c_{ij} - e_j]^+ (1 - x_{ij}) \leq t_j && \forall i, j : j \neq 1, i \neq j \\
& e_i \leq t_i \leq \ell_i && \forall i = 1, \ldots, n \\
& x_{ij} \in \{0, 1\} && \forall i, j : i \neq j
\end{aligned}$$

Next classes:

- formulation for the asymmetric TSP
- formulation for a more general vehicle routing problem
- cutting plane algorithm for general integer optimization