Applied Quantitative Logistics

Optimality, Relaxation, Bounds

Majid Sohrabi

National Research University Higher School of Economics

January 21, 2026

Solving Integer Programs

Consider the following **integer program:**

$$\begin{array}{lll} \text{(IP)} & \text{minimize} & \mathrm{f(x)} \\ & \text{subject to} & \boldsymbol{A}x \leq b \\ & & x \geq 0 \text{ and integer} \end{array}$$

Observe that it can be restated as

$$z = \min\{f(x): x \in X \subseteq Z^n\}$$

Where $X = \{x \in R^n: Ax \leq b, x \geq 0 \; and \; integer\}$, and z denotes the optimal solution value.

How can we prove that a given feasible solution x^2 is optimal?

Solving Integer Programs

We must find a **lower bound** $\underline{z} \leq z$ and an **upper bound** $\bar{z} \geq z$ such that $\underline{z} = \bar{z} = z$.

In practice, any **algorithm** must find a decreasing sequence of upper-bounds

$$\bar{z}_1 > \bar{z}_2 > \cdots > \bar{z}_s \geq z,$$

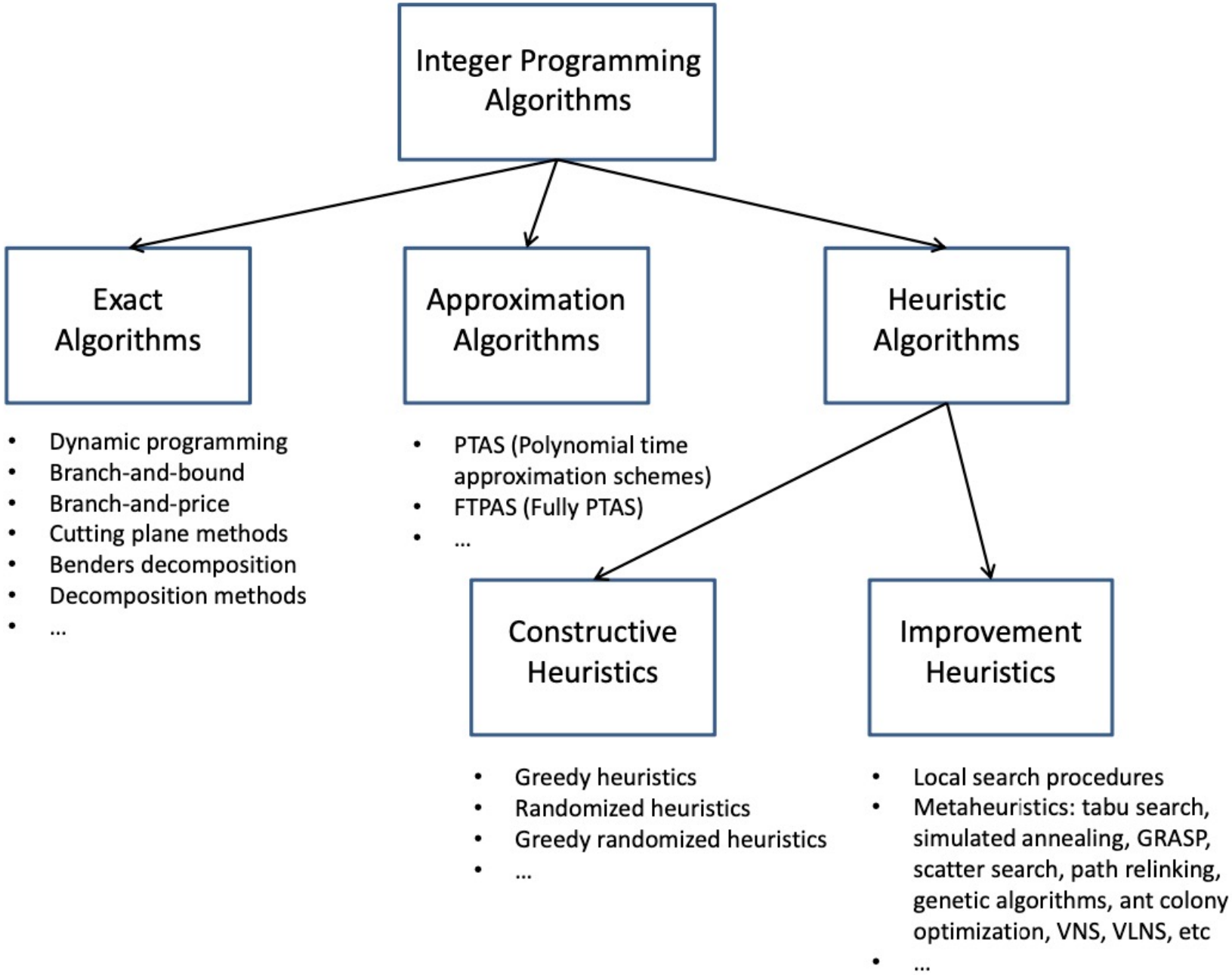
and an increasing sequence of lower bounds,

$$\underline{z}_1 < \underline{z}_2 < \cdots < \underline{z}_t \leq z,$$

and stop when

$$\bar{z}_s - \underline{z}_t \leq \epsilon,$$

where ϵ is a small nonnegative value.


Integer Programming Algorithms
Exact Algorithms
Approximation Algorithms
Heuristic Algorithms
Dynamic programming
Branch-and-bound
Branch-and-price
Cutting plane methods
Benders decomposition
Decomposition methods
...
PTAS (Polynomial time approximation schemes)
FTPAS (Fully PTAS)
...
Constructive Heuristics
Improvement Heuristics
Greedy heuristics
Randomized heuristics
Greedy randomized heuristics
...
Local search procedures
Metaheuristics: tabu search, simulated annealing, GRASP, scatter search, path relinking, genetic algorithms, ant colony optimization, VNS, VLNS, etc
...

Finding Upper Bounds

Observe that every feasible solution $x \in X$ provides an **upper bound** for a minimization problem (and a lower bound for a maximization problem)

For some IP problems, finding feasible solutions is trivial and the real question is how to find high-quality solutions

For other IP problems, finding feasible solutions may be very difficult

In fact, for some problems it is NP-hard to know whether there exists a feasible solution or not

We can use simple heuristics, such as greedy and local search algorithms, to obtain feasible solutions (and thus upper bounds)

We can also use more sophisticated methods, such as metaheuristics

Finding Upper Bounds

Heuristic algorithms can be classified as:

Constructive heuristics:

- Start with an empty solution
- Add one element at a time
- The goal is to construct an initial feasible solution

Improvement heuristics:

- Start with a feasible solution
- Add one element at a time
- Try to improve the solution by looking into “similar” solutions

Finding Lower Bounds

Finding **lower bounds** for a minimization problem (or upper bounds for a maximization problem) presents a different challenge

The most used approach is by relaxing a difficult IP problem to a simpler one whose optimal value is as small as z

There are two possibilities to have this property for the relaxed problem:

- Enlarge the set of feasible solutions
- Replace the objective function with a function that has the same or smaller value everywhere

Finding Lower Bounds

Proposition: Relaxations

A problem

$$\text{(RP)} \qquad z_R = \min\{f(x) : x \in T \subseteq Z^n\}$$

is a relaxation of

$$\text{(IP)} \qquad z = \min\{c(x) : x \in X \subseteq Z^n\},$$

if:

- $X \subseteq T$
- $f(x) \leq c(x)$ for all $x \in X$

Proposition: Lower bounds from relaxations

- If RP is a relaxation of IP, then $z_R \leq z$.

Finding Lower Bounds

Several relaxations of IP can be used to obtain lower bound:

- Linear programming relaxations
- Lagrangean relaxations
- Combinatorial relaxations
- Modular (or group) relaxations
- Surrogate relaxations

Linear Programming Relaxations

Definition: Linear programming relaxations

Let $P = \{x \in R_+^n : Ax \leq b\}$. For the integer program

$$\text{(IP)} \qquad z = \min\{c(x) : x \in P \cap Z^n\},$$

the **linear programming relaxation** is the linear program

$$\text{(LP)} \qquad z = \min\{c(x) : x \in P\},$$

Linear Programming Relaxations

Definition: Linear programming relaxations

Let $P = \{x \in R_+^n : Ax \leq b\}$. For the integer program

$$\text{(IP)} \qquad z = \min\{c(x) : x \in P \cap Z^n\},$$

the **linear programming relaxation** is the linear program

$$\text{(LP)} \qquad z = \min\{c(x) : x \in P\},$$

Observe that the definition of better (or stronger) formulations is related to linear programming relaxations

Proposition:

Suppose P_1 and P_2 are two formulations for the integer program $z = \min\{c(x) : x \in X \subseteq Z^n\}$ with $P_1 \subset P_2$. Then,

$$z_1^{LP} \geq z_2^{LP},$$

where z_1^{LP} and z_1^{LP} are the optimal solution values of their linear programming relaxations, respectively.

Linear Programming Relaxations

Proposition: Infeasibility and Optimality of IP

If a relaxation RP is infeasible, the original problem IP is infeasible.

Let x^*be an optimal solution of RP. If $x^* \in X$ and $f(x^*) = c(x^*)$, then x^* is an optimal solution of IP.

Combinatorial Relaxations

If the relaxed problem is a combinatorial optimization problem, we call it a combinatorial relaxation

The relaxation is usually an easy combinatorial problem (i.e., polynomialy solvable)

Some examples:

- A combinatorial relaxation of the *TSP* is the *Assignment Problem*
- A combinatorial relaxation of the *Symmetric TSP* is the *1-tree Problem*
- A combinatorial relaxation of the *Quadratic 0-1 Problem* is a series of *Maximum Flow Problems*

Combinatorial Relaxations: Traveling Salesman Problem

It can be easily seen that the Hamiltonian tours are precisely the asignments containing no subtours.

$$z^{TSP} = \min_{T \subseteq A} \left\{ \sum_{(i,j) \in T} c_{ij} : T \text{ forms a tour} \right\}$$

and

$$z^{ASS} = \min_{T \subseteq A} \left\{ \sum_{(i,j) \in T} c_{ij} : T \text{ forms an assignment} \right\}$$

Then,

$$z^{TSP} \geq z^{ASS}$$

Combinatorial Relaxations: Symmetric TSP

When distances are symmetric (i.e., $c_{ij} = c_{ji}$), the problem is to find an undirected tour of minimum weight. An interesting combinatorial relaxation can be obtained by considering special trees.

Definition: 1-tree

A 1-tree is a subgraph consisting of two edges adjacent to node 1, plus the edges of a tree on nodes $\{2, \ldots, n\}$.

Clearly every tour is a 1-tree, and thus

$$z^{TSP} = \min_{T \subseteq E} \left\{ \sum_{e \in T} c_e : T \text{ forms a tour} \right\}$$

and

$$z^{1-tree} = \min_{T \subseteq E} \left\{ \sum_{e \in T} c_e : T \text{ forms a 1-tree} \right\}$$

Then,

$$z^{TSP} \geq z^{1-tree}$$

Combinatorial Relaxations: The Quadratic 0–1 Problem

The **Quadratic 0-1 Problem** is the following problem:

$$\max\left\{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij}x_ix_j - \sum_{j=1}^{n} p_jx_j : x \in B^n, x \neq 0\right\}$$

Consider the combinatorial relaxation

$$\max\left\{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \max\{0, q_{ij}\}\, x_ix_j - \sum_{j=1}^{n} p_jx_j : x \in B^n, x \neq 0\right\}$$

Combinatorial Relaxations: The Quadratic 0–1 Problem

The **Quadratic 0-1 Problem** is the following problem:

$$\max\left\{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij}x_ix_j - \sum_{j=1}^{n} p_jx_j : x \in B^n, x \neq 0\right\}$$

Consider the combinatorial relaxation

$$\max\left\{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \max\{0, q_{ij}\}\, x_ix_j - \sum_{j=1}^{n} p_jx_j : x \in B^n, x \neq 0\right\}$$

This relaxation can be solved as a series of **Maximum Flow Problems**

Lower Bounds: Lagrangean Relaxation

Consider the integer program:

$$
\begin{array}{lll}
 & \text{minimize} & cx \\
(IP) & \text{subject to} & Ax \leq b \\
 & & Dx \leq d \\
 & & x \in Z_+^n
\end{array}
$$

Suppose that constraints $Ax \leq b$ are "**nice**" in the sense that an integer program with just these constraints is easy

Thus, if one drops the "**complicating constraints**" $Dx \leq d$, the resulting relaxation is easier to solve than the original problem

Lower Bounds: Lagrangean Relaxation

For any value $\lambda = (\lambda_1, \ldots, \lambda_m) \geq 0$, we define the problem:

$$
\begin{array}{lll}
 & L(\lambda) = \text{minimize} & cx + \lambda(Dx - d) \\
(IP(u)) & \text{subject to} & Ax \leq b \\
 & & x \in Z_+^n
\end{array}
$$

$IP(u)$ is called a **Lagrangean relaxation** of IP with parameter λ.

Proposition:

Problem $IP(u)$ is a relaxation of the problem IP for all $\lambda \geq 0$.

Therefore, $L(\lambda)$ provides a lower bound on the optimal solution of IP.

Upper Bounds: Greedy and Local Search Heuristics

Greedy Heuristics

- The greedy heuristics are the simplest class of constructive algorithms. They start from scratch (an empty solution) and construct a feasible solution by choosing at each step the item with the best immediate reward.

Local Search Heuristics

- The local search heuristics start with a feasible solution, and they try to improve it by looking at solutions that are close to it. For that, we define a neighborhood of solutions close to the current solution. Then the best solution in the neighborhood is identified. If it is better than the current solution, it replaces it, and the procedure is repeated. Otherwise, the solution is locally optimal (with respect to the neighborhood) and the heuristic terminates.

Upper Bounds: Local Search Heuristics

"Local search algorithms try to find high quality solutions by searching through the solution space. More precisely, these algorithms start with an initial solution and then iteratively generate a new solution that is "near" to the current solution. A neighbourhood function defines for a given solution the solutions that are near to it. Solutions of many combinatorial optimization problems can be represented by discrete structures such as sequences, permutations, graphs, and partitions. Local search algorithms typically use these representations by defining neighbourhood functions in terms of local rearrangements, such as swapping, moving, and/or replacing items, that may be applied to a representation to obtain a neighbouring solution."

Michiels, W., Aarts, E., Korst, J. *Theoretical Aspects of Local Search*, Springer, 2007.

Upper Bounds: Local Search Heuristics

Consider the following combinatorial optimization problem

$$\min_{S \subseteq N} \{c(S) : g(S) = 0\}$$

Ingredients of local search methods:

1. An **initial solution** S
2. A **local search neighborhood** $N(S)$ for each solution $S \subseteq N$
3. A **goal function** $f(S)$ which can be either

$$f(S) = \begin{cases} c(S) & \text{if } S \text{ is feasible;} \\ \infty & \text{otherwise} \end{cases}$$

Or a composite function

$$f(S) = c(S) + \alpha g(S)$$

Upper Bounds: Local Search Heuristics

Local Search Heuristic:

Choose an initial solution S
$terminate \leftarrow$ **false**
while($terminate =$ **false**) **do**
    Search for a solution $S^{'} \in N(S)$ with $f(S^{'}) < f(S)$
    if$(f(S^{'}) \geq f(S), \forall S^{'} \in Q(S))$ **then**
        $terminate \leftarrow$ **true**
    else
        Set $S = S^{'}$
    end if
end while

The output of the local search heuristic is a local optimal solution with respect to the neighborhood $N(S)$

Upper Bounds: Local Search Heuristics

Appropriate choices of neighborhoods depend on the problem structure.

Some very simple ones are:

- Add element to S

$$N_A(S) = \left\{ S' : S' = S \cup \{j\}, j \in N \setminus S \right\}$$

- Remove element to S

$$N_R(S) = \left\{ S' : S' = S \setminus \{i\}, i \in S \right\}$$

- Add/remove element to S

$$N_{AR}(S) = \left\{ S' : S' = S \cup \{j\} \setminus \{i\}, i \in S, j \in N \setminus S \right\}$$

Upper Bounds: Metaheuristics

Question: How do we escape from a local optimal solution, and thus potentially do better than a local search heuristic?

This is the question addressed by **metaheuristic** methods.

Metaheuristic methods differ in the way they avoid local optimality in the search for a better solution (hopefully the optimal one).

Upper Bounds: Metaheuristics

Some of the most well-know and successful metaheuristic methods are:

- Genetic and Evolutionary Algorithm (GA and EA) [1]
- Particle Swarm Optimization (PSO) [2]
- Simulated Annealing (SA) [3]
- Ant Colony Optimization (ACO) [4]
- Tabu Search (TS) [5]
- Memetic Algorithm (MA) [6]
- **Genetic Engineering Algorithm (GEA)** [7]

Upper Bounds: Metaheuristics

[1] J. Holland. "*Adaptation in natural and artificial systems*". In: Ann Arbor: University of Michi-gan Press (1975).

[2] J. Kennedy and R. Eberhart. "*Particle swarm optimization*". In: Proceedings of ICNN'95-international conference on neural networks. Vol. 4. ieee. 1995, pp. 1942–1948.

[3] Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). *Optimization by simulated annealing*. science, 220(4598), 671-680.

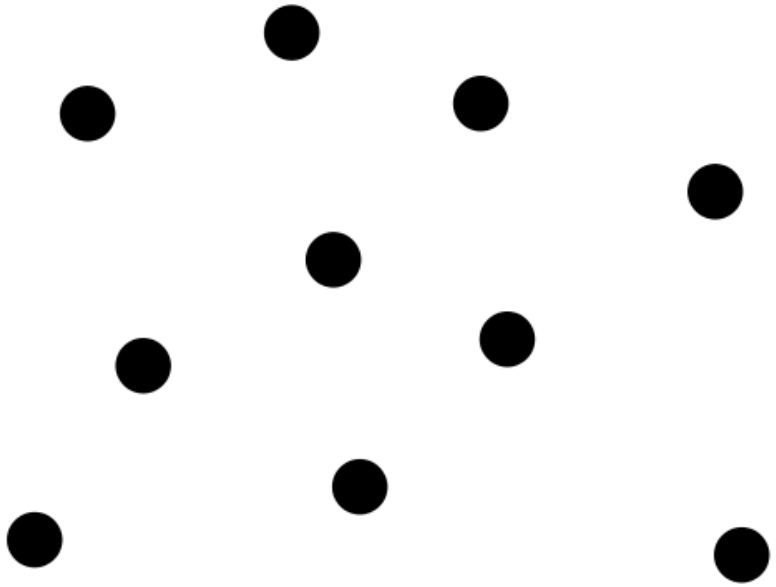
[4] M. Dorigo, V. Maniezzo, and A. Colorni. "*Ant colony System: Optimization by a colony of cooperating agents*". In: IEEE Transactions on Systems, Man and Cybernetics Part B (1997), pp. 29–41.

[5] Glover, F., & McMillan, C. (1986). *The general employee scheduling problem. An integration of MS and AI*. Computers & operations research, 13(5), 563-573.

[6] Moscato, P. (1989). *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*. Caltech concurrent computation program, C3P Report, 826(1989), 37.

[7] M. Sohrabi, A. M. Fathollahi-Fard, and V. A. Gromov. "***Genetic engineering algorithm (GEA): an efficient metaheuristic algorithm for solving combinatorial optimization problems***". In: Automation and Remote Control 85.3 (2024), pp. 252–262.

Example: The Traveling Salesman Problem



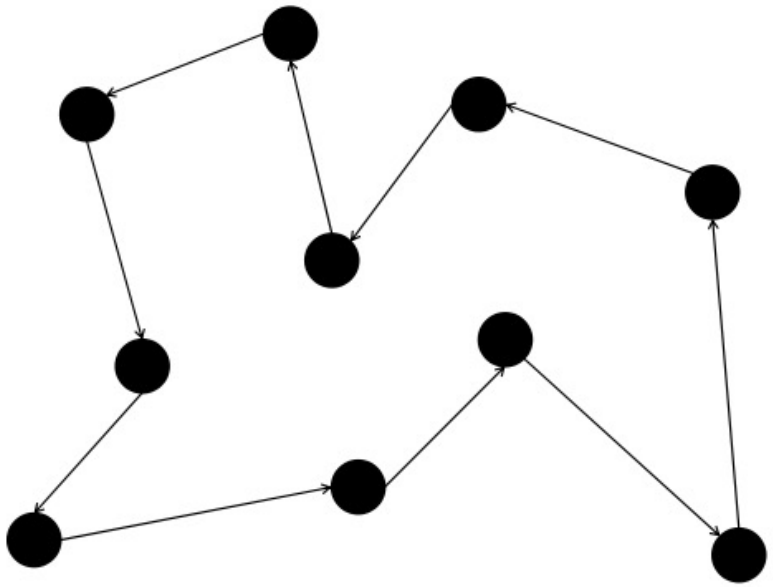
Input:

- N: set of cities ($|N| = n$)
- c_{ij}: travel time between i and j

The traveling salesman problem (TSP):

- A salesman must visit each city exactly once and then return to his starting point
- Objective: Minimize the total travel time of the tour

Example: The Traveling Salesman Problem



Input:

- N: set of cities ($|N| = n$)
- c_{ij}: travel time between i and j

The traveling salesman problem (TSP):

- A salesman must visit each city exactly once and then return to his starting point
- Objective: Minimize the total travel time of the tour

Example: The Traveling Salesman Problem

Routing variables:

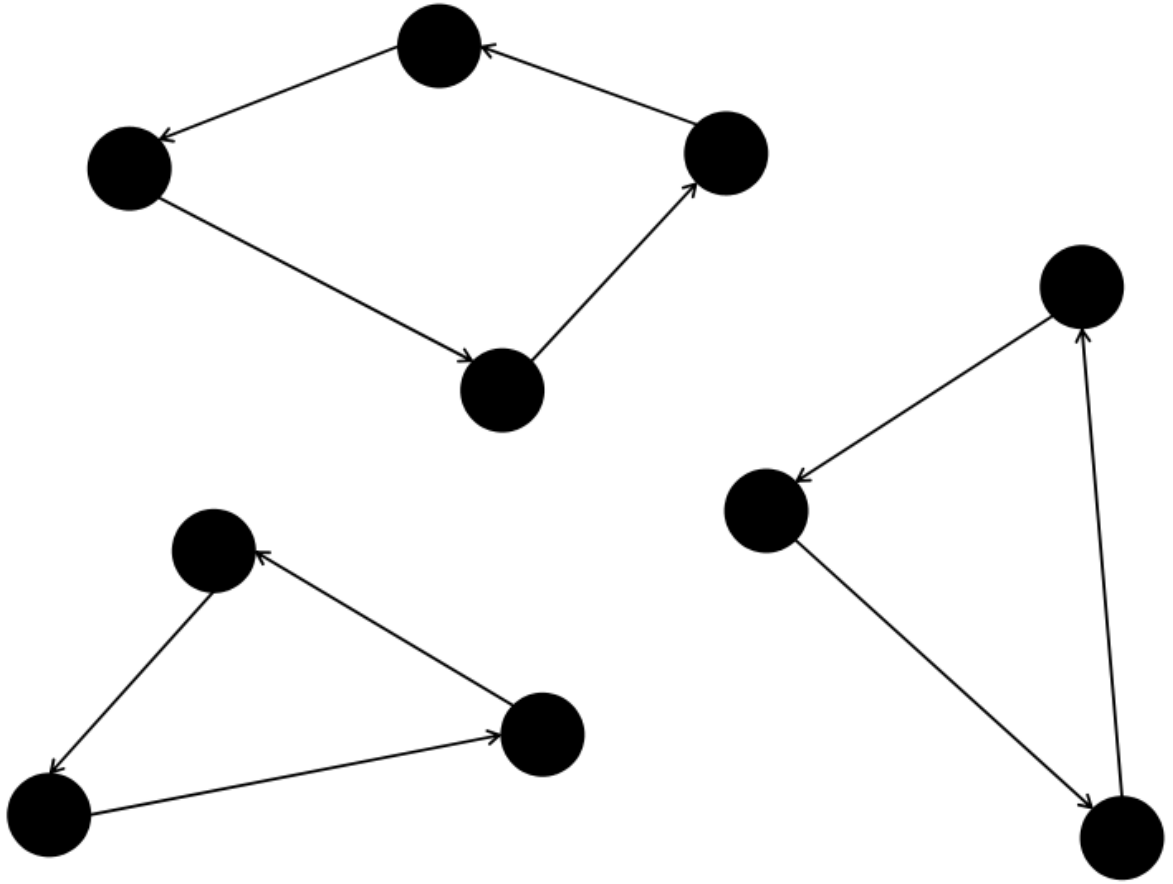
$$x_{ij} = \begin{cases} 1, & if\ the\ salesman\ goes\ directly\ from\ city\ i\ to\ city\ j; \\ 0, & otherwise \end{cases}$$

Using these variables, we could model the TSP as:

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij} & \\ \text{subject to} \quad & \sum_{j \in N: j \neq i} x_{ij} = 1 & j \in N \\ & \sum_{i \in N: i \neq j} x_{ij} = 1 & i \in N \\ & x_{ij} \in \{0, 1\} & i \in N, j \in N \end{aligned}$$

Is this enough to model the problem?

Example: The Traveling Salesman Problem



We have to eliminate subtours = obtain one single connected component!

Example: The Traveling Salesman Problem

To achieve this we need additional constraints that guarantee connectivity.

For that, we have at least two options:

Cut-set constraints:

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \qquad \forall S \subset N, S \neq \emptyset$$

Subtour elimination constraints:

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset N, 2 \leq |S| \leq n - 1$$

Which formulation is better? The one that uses cut-set inequalities or the one that uses subtour elimination constraints?

Thank you!