Combinatorial Optimization
Lab 10: Traveling Salesman Problem

Industrial Informatics Research Center
http://industrialinformatics.fel.cvut.cz/

April 20, 2020

Outline of the tutorial:

- Revision (5 minutes)
- Ways to model TSP (45 minutes)

This tutorial starts with a brief revision of the traveling salesman problem. We formalize the problem and discuss its complexity. Afterward, the problem is written as an ILP program, which was described during the lectures. Then, a new Integer Linear Programming formulation is introduced along with an approach based on lazy constraints.

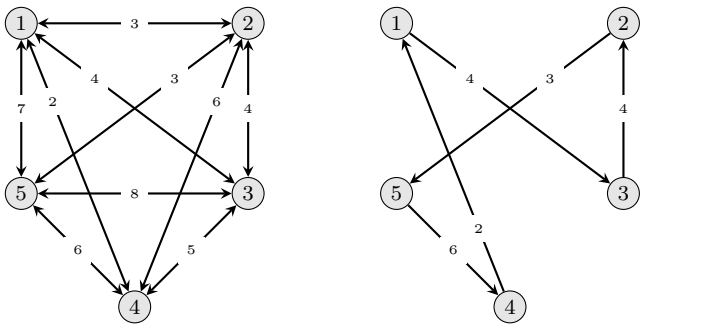
Part 1: Revision

Traveling Salesman Problem (TSP) is one of the most famous problems in combinatorial opti- mization. TSP has a wide range of applications in both theory and practice. For example, TSP is natural model for solving problems in logistics.

Input: Let us have a complete, directed graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ is a set of its nodes and $E = \{(i, j) \mid i, j \in V, i \neq j\}$ is set of its edges. We assume that $|V| \geq 3$. Each edge $e = (i, j)$ is associated with cost $c_{i,j}$. As an example, consider the graph from Figure 1.

Output: The task is to find Hamiltonian cycle H (closed path going through each node exactly once) with a minimal cost.

Complexity: The decision version of TSP (i.e., given a length L, decide whether the graph has any tour shorter than L) is $\mathcal{NP}$-complete. For general TSP, there exists no polynomial approximation algorithm (unless $\mathcal{P} = \mathcal{NP}$), but there might exist approximation algorithms for some restricted versions of TSP (e.g., for the TSP satisfying the triangle inequality).



(a) Input graph (b) Optimal solution with cost 19

Figure 1: Example of symmetric TSP.

Part 2: Modelling TSP

2.1 Model from the lecture

During the lecture, one possible model was introduced. It utilizes variables $x_{i,j}$ with the following meaning:

$$x_{i,j} = \begin{cases} 1, & \text{iff node } i \text{ immediately precedes node } j \text{ in the solution path,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Additional variable s_i is used to eliminate the subtours. It represents the "time" of arrival into node i.

$$\min \quad \sum_{(i,j)\in E} c_{i,j} \cdot x_{i,j} \tag{2}$$

$$\text{s.t.} \quad \sum_{(i,j)\in E} x_{i,j} = 1, \quad j \in V \tag{3}$$

$$\sum_{(i,j)\in E} x_{i,j} = 1, \quad i \in V \tag{4}$$

$$s_i + c_{i,j} \leq s_j + M \cdot (1 - x_{i,j}), \quad i \in V, j \in V \setminus \{1\} \tag{5}$$

$$x_{i,j} \in \{0,1\},\ s_i \in \mathbb{R} \tag{6}$$

Figure 2: ILP formulation of TSP from the lecture.

The model is easy to understand, but its performance might not be the best. Nevertheless, it might be useful, when we want to solve the TSP with time-windows.

Note that the performance could improve if we used $s_i + 1 \leq s_j + M \cdot (1 - x_{i,j})$ – big M could be smaller (e.g., $(n-1)$) and there would be no problems with negative or zero costs $c_{i,j}$.

2.2 Improved model

Now, we describe a more efficient model. We still use variable $x_{i,j}$ to represent the edges:

$$x_{i,j} = \begin{cases} 1, & \text{iff edge } (i,j) \in E \text{ belongs to the solution } H, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Now, however, we develop a different mechanism to eliminate the subtours. The idea of the model can be described as follows. The salesman sells exactly n items during his tour starting from node 1, selling exactly one item in each node.

In the following model, we use variable $y_{i,j}$, which indicates the number of items which the salesman has after leaving node i before entering node j. Because of the selling requirement (one item per node), the number of remaining items before visiting node $j \in V \setminus \{1\}$ is one more than the number of items after leaving node j (constraint (11)). When the salesman returns, it must hold that $y_{i,1} = 0$ for the immediate predecessor i of node 1 during the tour (constraint (12), modification of (11)). Constraint (13) provides an upper bound on $y_{i,j}$. If an edge (i,j) does no belong to the tour, it is forced to 0.

In figure 4, we show an example of a feasible solution, next to the example of an infeasible solution (with multiple subtours) that violates constraint (12).

$$\min \quad \sum_{(i,j)\in E} c_{i,j} \cdot x_{i,j} \tag{8}$$

$$\text{s.t.} \quad \sum_{(i,j)\in E} x_{i,j} = 1, \quad j \in V \tag{9}$$

$$\sum_{(i,j)\in E} x_{i,j} = 1, \quad i \in V \tag{10}$$

$$\sum_{(i,j)\in E} y_{i,j} - 1 = \sum_{(j,k)\in E} y_{j,k}, \quad j \in V \setminus \{1\} \tag{11}$$

$$\sum_{(i,1)\in E} y_{i,1} + (n-1) = \sum_{(1,k)\in E} y_{1,k} \tag{12}$$

$$y_{i,j} \leq (n-1) \cdot x_{i,j}, \quad (i,j) \in E \tag{13}$$

$$x_{i,j} \in \{0,1\},\ y_{i,j} \in \mathbb{Z}_{\geq 0} \tag{14}$$

Figure 3: Different ILP formulation of TSP.

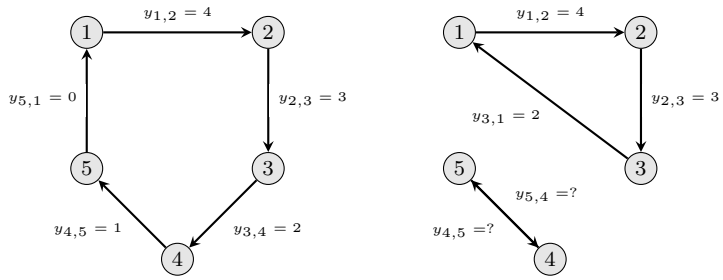


Figure 4: Feasible and infeasible solution

2.3 Going further

We can see that both models we designed so far are similar. They both contain 2 types of constraints – **degree constraints** (enforcing degree of each node to be 2) and **subtour elimination** constraints (forbidding subtours). Degree constraints were basically the same in both models, but we had some problems designing the subtour elimination constraints. What else can be done to eliminate possible subtours?

Trivially, we could list the subtours one by one, and for each of them we could add a constraint forbidding it. It can be seen that each subtour S has exactly $|S|$ edges (the number of its nodes). To forbid subtour S, we just need to forbid $|S|$ and more edges:

$$\sum_{(i,j)\in S} x_{i,j} \leq |S| - 1, \ \forall S \subset V. \tag{15}$$

Note that for $S = V$ we cannot apply the constraint as we would not allow the existence of Hamiltonian cycle itself.

Problem: What is the problem with this approach? By eliminating each and every subtour, we would generate exponentially many constraints. Well, that is not good. But do we really need all of them?

2.3.1 Sidestep: Optimizing over a circle

Imagine that someone would want you to find an optimal solution to the ILP problem over a circle. Can you do it? Well, a circle is not a polyhedron as you would need an infinite amount of constraints to describe it. But even though you cannot model the circle exactly, you may be able to do at least something. A circle could be approximated by a convex polygon with many sides. But is it necessary to generate them all?

See Figure 5 – we started from a very simple approximation (by square). Then we found an optimal solution to this simplified problem. As the solution did not lie on the boundary of our circle, it was not an optimal solution to the problem. Therefore we generated a **cut**. A cut is a separating hyperplane, which separated the previously found solution point from the rest of the set over which we optimize.

In this case, a cut can be found easily. How? We just take a vector going from the center of the circle to the found point as a normal to the separating hyperplane, which will be exactly one radius r away from the center (in the direction of the found point). By generating additional cuts, we are able to obtain the desired precision. You see that not that many cuts were needed to obtain a relatively precise solution. Also, the cust are concentrated only in the region of our interest.

For some combinatorial problems, however, it might not be that simple to find a cut. Note that we always wish to have polynomial algorithms for cut-finding; otherwise, it would not help much, as typically, there might be the exponential number of cuts necessary to solve the problem optimally.

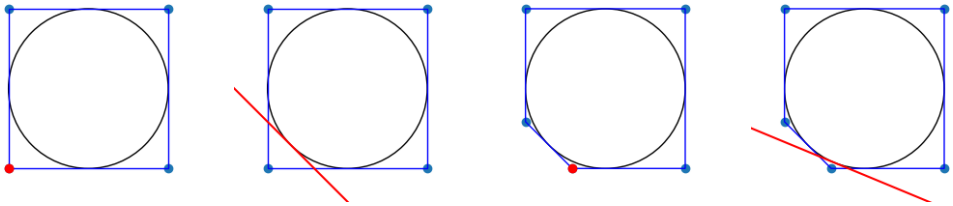


Figure 5: Using an ILP to optimizer over a circle.

The described approach is called **lazy constraints generation**. We generate the constraints dynamically. The idea is to start with a simple approximation of the original optimization space and improve it only when it is necessary.

2.4 Application of the lazy constraints on TSP

Back to our TSP. The algorithm would be very simple now – just start with a model, which does not have any subtour elimination constraints. Solve it, and if the solution contains a subtour, forbid it and continue.

Standard solvers (Gurobi) support the lazy constraints generation – it is integrated to the branch-and-bound procedure – when the solution is found, a defined callback can be called and the additional constraints can be generated. **Important:** it is necessary to set the `LazyConstraints` parameter properly.

To conclude

The take-out message is that often the first working model might not be the best one. Even while solving hard problems, new ideas can sometimes lead to significant improvements. You may experiment with the proposed models – you will see that the model using the lazy constraints outperforms the other two significantly.

Now that we have a powerful model for TSP, it is time to apply it (see the assignment of HW4)!