Combinatorial Algorithms
Lab No. 5
Traveling Salesman Problem

Industrial Informatics Department
http://industrialinformatics.cz/

March 20, 2024

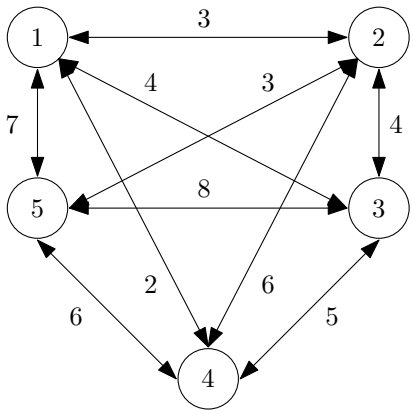
Abstract

In this lab we review various ways how to solve the famous Traveling Salesman Problem. A new Integer Linear Programming formulation is introduced along with an approach based on lazy constraints. As a homework, you will solve a problem of image unshredding.

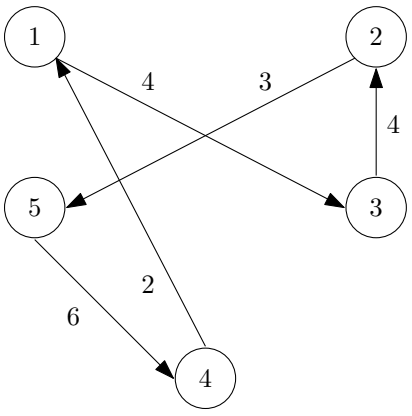
1 Traveling Salesman problem

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

Let $G = (V, E)$ be a complete, directed graph where $V = \{1, \ldots, n\}$ is the set of nodes and $E = \{(i, j) \mid i, j \in V, i \neq j\}$ is the set of edges. We assume $n \geq 3$. Let each edge $(i, j) \in E$ be associated with cost $c_{i,j}$. The task is to find a closed path of minimal cost going through each node of G exactly once Hamiltonian circuit of a minimal cost). As an example, consider a graph from Fig. 1a; the corresponding optimal solution is depicted in Fig. 1b.



(a) Instance. To simplify the illustration, edges (i, j), (j, i) are drawn only once using a bidirectional edge $\{i, j\}$.



(b) Optimal solution with cost 19.

Figure 1: TSP example with symmetric costs.

On lectures, you were introduced to the following Integer Linear Programming (ILP) model of TSP

$$\min \quad \sum_{(i,j)\in E} c_{i,j} \cdot x_{i,j} \tag{1}$$

$$\text{s.t.} \quad \sum_{(i,j)\in E} x_{i,j} = 1, \quad j \in V \tag{2}$$

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

$$s_i + c_{i,j} \leq s_j + M \cdot (1 - x_{i,j}), \quad i \in V, j \in V \setminus \{1\} \tag{4}$$

$$x_{i,j} \in \{0,1\},\ s_{i,j} \in \mathbb{R} \tag{5}$$

Figure 2: ILP formulation of TSP from the lecture.

where $x_{i,j}$ is a binary variable such that $x_{i,j} = 1$ iff node i in the circuit just before node j. s_i is a variable representing the "time" of arrival into the node and is used for eliminating subtours.

Although this model is straightforward, we show a more efficient one below. Nevertheless, the model in Fig. 2 is still useful if we consider TSP with time-windows (time interval in which the node has to be visited).

1.1 More efficient ILP formulation for TSP

More efficient ILP model is shown in Fig. 3 where the decision variable $x_{i,j}$ indicates whether the edge $(i,j) \in E$ belongs to the circuit or not.

$$\min \quad \sum_{(i,j)\in E} c_{i,j} \cdot x_{i,j} \tag{6}$$

$$\text{s.t.} \quad \sum_{(i,j)\in E} x_{i,j} = 1, \quad j \in V \tag{7}$$

$$\sum_{(i,j)\in E} x_{i,j} = 1, \quad i \in V \tag{8}$$

$$\sum_{(i,j)\in E} y_{i,j} - 1 = \sum_{(j,k)\in E} y_{j,k}, \quad j \in V \setminus \{1\} \tag{9}$$

$$\sum_{(i,1)\in E} y_{i,1} + (n-1) = \sum_{(1,k)\in E} y_{1,k} \tag{10}$$

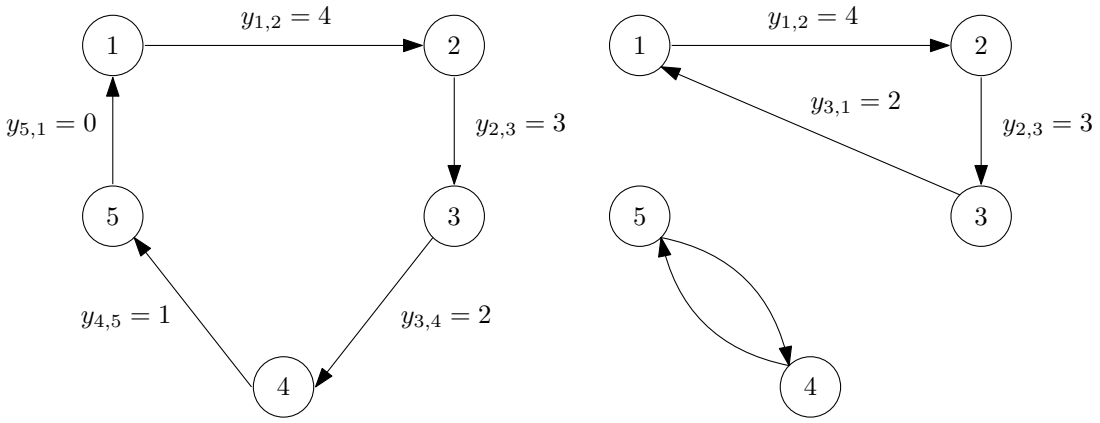
$$y_{i,j} \leq (n-1) \cdot x_{i,j}, \quad (i,j) \in E \tag{11}$$

$$x_{i,j} \in \{0,1\},\ y_{i,j} \in \mathbb{Z}_{\geq 0} \tag{12}$$

Figure 3: More efficient ILP formulation of TSP.

One thing to mention about the model is the elimination of the subtours, see constraints (9)–(11). The idea is that we assume the salesman sells n items during his journey starting from node 1 and he is required to sell exactly one item in each node. Let $y_{i,j}$ be the number of items which the salesman has after leaving node i and before entering node j. Due to the selling requirement, the number of remaining items before visiting node $j \in V \setminus \{1\}$ is one more than the number of remaining items after leaving node j (see constraint (9)). Notice that when the salesman return back to its starting node 1, it must hold that $y_{i,1} = 0$, where i is the immediate predecessor to node 1 during the salesman tour. Thus constraint (9) is modified for this special case as shown in constraint (10). Constraint (11) denotes that the upper bound of $y_{i,j}$ is $n-1$ while

the lower bound is 0. It is obvious that when the edge $(i, j) \in E$ does not belong to the salesman tour, then $y_{i,j}$ equals 0. In Fig. 4a we show an example of a feasible solution whereas in Fig. 4b we show an example of an infeasible solution with multiple subtours that violates constraint (10).



(a) Feasible solution. (b) Infeasible solution.

Figure 4: Feasible and infeasible solutions to model in Fig. 3.

1.2 Lazy Constraints

Since combinatorial optimization problems are studied for decades already, it is natural that there exist a lot of advanced optimization approaches that result in reduced computation time for certain problem types. One of the conceptually easiest approach is called *lazy constraints* [1]. The basic idea is to initially formulate the problem only with the most essential constraints, omitting those that are only rarely violated. These other constraints are checked and added one-by-one to the model only if the current solution violates any of them. In other words, some constraints are generated in a lazy fashion, i.e. the constraint is added to the model only if the solution violates it.

The application of lazy constraints to TSP is really straightforward. Let model TSP as follows

$$\min \quad \sum_{(i,j)\in E} c_{i,j} \cdot x_{i,j} \tag{13}$$

$$\text{s.t.} \quad \sum_{(i,j)\in E} x_{i,j} = 1, \quad j \in V \tag{14}$$

$$\sum_{(i,j)\in E} x_{i,j} = 1, \quad i \in V \tag{15}$$

$$\sum_{i,j\in S: i\neq j} x_{i,j} \leq |S| - 1, \quad S \subset V, S \neq \emptyset \tag{16}$$

$$x_{i,j} \in \{0, 1\} \tag{17}$$

Figure 5: ILP formulation of TSP, suitable for lazy constraints.

Notice that model in Fig. 5 is the same as in Fig. 3 with exception that constraints (9)-(11) were replaced with constraints (16) which eliminate subtours. It requires that in each proper non-empty subset of vertices S there are at most $|S| - 1$ edges between them. Indeed, having a subset of vertices with $|S|$ edges if $S \neq V$ means that there are subtours in the solution, for instance if $S = \{1, 2, 3\}$ as in Fig. 4b then $\sum_{i,j\in S: i\neq j} x_{i,j} = 3 > |S| - 1$.

The problem with model in Fig.5 is that there are exponential number of subsets S, e.g. $n = 10$ results in $2^n - 2 = 1022$ constraints of (16).

However, we may generate Constraint (16) in a lazy manner. In the beginning, the problem is formulated first without these constraints, i.e. only with Constraints (14) and (15). Then each time when the new integer solution is found during branching, we find a cycle in this solution; let S be the nodes in this cycle. If $|S| < n$, i.e. the solution contains a subtour, the corresponding constraint (16) is added to the model. Adding this constraint excludes this solution from the search space in the further run. The constraints are added to the model until the solution is the cycle of length n.

1.3 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 6 – 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.

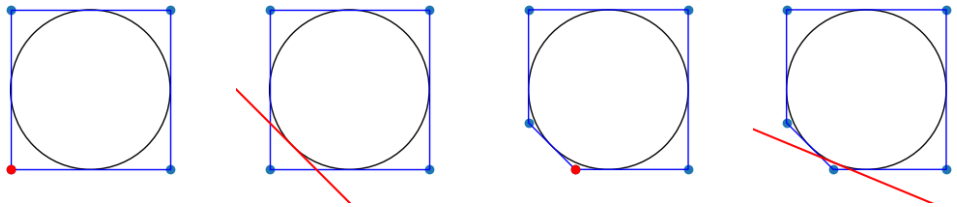


Figure 6: Using an ILP to optimizer over a circle.

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.

1.4 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 HW2)!

References

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. The traveling salesman problem: a computational study, 2011.