Discrete Optimization
Assignment 1

Mehdi Nadif & Viktor Hansen

October 11, 2017

Abstract

This is the first weekly assignment for the Discrete Optimization course offered at The Department of Computer Science, Uni. Copenhagen.

Theoretical part - formulation and lower bounds

1.1

Any feasible solution satisfies constraints: Consider a Hamiltonian tour $p = \pi(1) \to \pi(2) \to \ldots \to \pi(n) \to \pi(1)$ of G defined by a bijection $\pi : V = \{1, \ldots, n\} \to \{1, \ldots, n\}$. Let $E_p = \{(\pi(i), \pi(i+1)) \mid 1 \leq i \leq n-1\} \cup \{(\pi(n), \pi(1))\}$ denote the set of edges traversed by p. Then

$$\sum_{i \in V, i \neq j} x_{ij} = \sum_{i \in V, i \neq j} \mathbb{1}_{E_p}((i,j)) = 1 \quad \forall j \in V$$
$$\text{and} \quad \sum_{j \in V, j \neq i} x_{ij} = \sum_{j \in V, j \neq i} \mathbb{1}_{E_p}((i,j)) = 1 \quad \forall i \in V$$

since there is exactly one edge entering and leaving and each vertex. To see that p satisfies the subtour constraint, consider any subset $S \subset V$ s.t. $2 \leq |S| \leq n-2$. Then for each such subset S,

$$\sum_{i,j \in S} x_{ij} = \sum_{i,j \in S} \mathbb{1}_{E_p}((i,j)) \leq \sum_{i \in S} \left[\mathbb{1}_{E_p}((\pi(i), \pi(i+1)))\right] - 1 = \sum_{i \in S} (1) - 1 = |S| - 1$$

This first inequality follows since in the worst case, S forms a Hamiltonian path (i.e. subgraph of the Hamiltonian tour), and this has one edge less than the number of vertices.

Any solution satisfying constraints is feasible: To show the other direction, we need to show that an assignment satisfying the constraints correspond to a Hamiltonian tour of G, i.e. that

1. All vertices are visited exactly once (except for the first vertex)
2. There is only one cycle.

Suppose $\mathbf{x} \in \{0,1\}^{n \times n}$ is an assignment satisfying the contraints. To show 1), we note that $x_{ij} = 1$ iff. there is an edge going from vertex v_i to v_j. For each vertex v_k to be visited exactly once, we require that $\sum_{i \in V} x_{ik} = \sum_{j \in V} x_{kj} = 1$, $\forall k \in 1 \ldots n$ which is satisfied by assumption.

To show 2), assume for the sake of contradiction that the feasible solution consists of $k > 1$ subtours $p_1, \ldots, p_k$. The sets of vertices in each such subtour, p_i, denoted by $L(p_i)$, form a partition of V, so $2 \leq |L(p_i)| \leq n-2$, $1 \leq i \leq k$. Since any subtour p_i is a cycle, it has exactly $|L(p_i)|$ edges, and hence $\sum_{i,j \in L(p_i)} x_{ij} = |L(p_i)|$ which contradicts the subtour constraint. Hence there can only one cycle.

1.2

We may first note that

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$

Since 2^n is the total number of subsets of a set of size n. Since we are only looking at subsets of size between 2 and $n-2$, the number of subsets must be

$$\begin{aligned} & 2^n - \binom{n}{0} - \binom{n}{1} - \binom{n}{n-1} - \binom{n}{n} \\ & = 2^n - 1 - n - n - 1 \\ & = 2^n - 2n - 2 \end{aligned}$$

since there are as many constraints as subsets, this is also the number of constraints.

1.3

The number of constraints is equal to the number of combinations of $i \in V, j \in V \backslash \{1\}$. As $|V| = n$, there are $|V \times V \setminus \{1\}| = n(n-1)$ constraints.

1.4

Even though there are many more constraints in the subtour formulation, the variables that have to be optimized are only the x_{ij}. As the compact formulation have the additional $t_i,\ i = 1, ..., V$ variables to optimize on, branch and bound will be on more variables. Therefore, the depth of the branching tree will be deeper, and it might take more time to get to a feasible solution which could have been used as an incumbent to prune with.

Similarly, one may note that bounding the compact formulation by an LP relaxation yields weak bounds as the x_{ij} can have very low values without breaking constraints. This is observed in a slightly weaker formulation, where we first change the constraints so that

$$\begin{aligned} & t_j \geq t_i + 1 - n(1 - x_{ij}), \quad i \in V, j \in V \backslash \{1\} \\ \Leftrightarrow\ & 1 - \frac{t_i - t_j + 1}{n} \geq x_{ij} \end{aligned}$$

if we formulate every combination of $i \in V, j \in V \backslash 1$ as the set A, we may create the weaker

bound

$$\begin{aligned}
&\sum_{(i,j)\in A} 1 - \frac{t_i - t_j + 1}{n} && \geq \sum_{(i,j)\in A} x_{ij} \\
\Leftrightarrow &|A| - \sum_{(i,j)\in A} \frac{t_i - t_j + 1}{n} && \geq \sum_{(i,j)\in A} x_{ij} \\
\Rightarrow &|A| - \sum_{(i,j)\in A} \frac{1}{n} \geq |A| - \sum_{(i,j)\in A} \frac{t_i - t_j + 1}{n} \geq \sum_{(i,j)\in A} x_{ij}
\end{aligned}$$

which means that the bound on the compact formulation is

$$|A| \left(1 - \sum_{(i,j)\in A} \frac{1}{n}\right) \geq \sum_{(i,j)\in A} x_{ij}$$

1.5

Consider a minimum-cost Hamiltonian tour $\mathcal{H}$, a minimum spanning tree $\mathcal{M}$ in G and a leaf-node v_1 in $\mathcal{M}$. Since v_1 is on $\mathcal{H}$, it has exactly two incident edges $e_{\min}$ and $e_{\max}$. From $\mathcal{H}$, we remove the most costly of the two edges incident to v_1, denoted by $e_{\max}$. Removing this edge, we now have a tree, and therefore

$$\text{cost}(\mathcal{M}) \leq \text{cost}(\mathcal{H}) - \text{cost}(e_{\max})$$

since $\mathcal{M}$ is the tree with lowest possible cost. By definition, we must have that $e_{\max} \geq e_{\min}$ since $e_{\max}$ was the costlier among the two edges. Therefore

$$\text{cost}(\mathcal{M}) + \text{cost}(e_{\min}) \leq \text{cost}(\mathcal{M}) + \text{cost}(e_{\max}) \leq \text{cost}(\mathcal{H})$$

and hence the MST with the addition of cheapest edge to a leaf is a lower bound for $\text{cost}(\mathcal{H})$

Implementation part - branch-and-bound

2.1

There are several upper bounds that can be implemented for this TSP problem, and the ones that we have attempted to implement for this problem were:

1. **maximumSpanningTree**: First, construct a spanning tree of maximum cost using Kruskal, but chosing the maximum edge at each iteration. Then select the remaining most expensive edge to create a cycle. This should bound the optimal Hamiltonian cycle from above, from the proof in subsection 1.5.

	maxSpan	2-selecter	2-approximation
Instance 1	31.59	31.08	8.90
Instance 2	43.02	38.71	30.83
Instance 3	54.38	51.82	44.86

Figure 1: The upper bounds on the root node problems of the three test instances using three of the above mentioned four upper bound heuristics.

	Branch and bound		CPLEX	
	Time	Nodes visited	Time	Nodes Visited
Instance 1				
Instance 2				
Instance 3				

2. **two selecter**: For each $v \in V$, select the two most expensive edges adjacent to it (assure that edges that are forced to be included in the branch is chosen, and excluded are ignored). Take the sum of all these edges and divide by two to get an upper bound. The division is necessary, since every edge is in two terms of the summation.
3. **Christofides algorithm**: a more complex algorithm, that
4. **2-approximation**: Creates a minimum spanning tree of the graph, and then computes the size of an euler tour on this minimum spanning tree. Slightly weaker than Christofides with a worst case of twice the optimal solution.

As can be seen in Fig. 1, the results of each of these heuristics have greatly varying results. Clearly the 2-approximation has the strongest bound on the root node, but it proved quite complex to implement this algorithm so that it worked with the branching, as assuring that some edge would be included in the euclidean tour was quite difficult. Also, it seems to be a bound that only gets worse as you decrease the domain of the graph. This is not the case for the **maximumSpanningTree** or the **two selector** which both get better at each branching since they always chose worst case for a graph.

Unfortunately, none of these upper bounds improved the number of nodes visited from only pruning by lower bounds that are higher than the currently best feasible solution.

2.2

2.3