A Solution to the Traveling Salesman Problem

1 Problem description

We have a set $\mathcal{I} = \{1, \cdots, I\}$ of I nodes and a set of inter-node distances $d(i,j) \geq 0$, $i \neq j \in \mathcal{I}$, giving the distance from i to j. Notice that $d(i,j) \neq d(j,i)$ in general. A path is an ordered set $p = \{i_1, \cdots, i_N\}$ of nodes and its length is

$$l(p) = \sum_{n=1}^{N-1} d(i_n, i_{n+1}).$$

A tour for the problem $(\mathcal{I}, a, b)$ is a path visiting all nodes in $\mathcal{I}$ starting from a and finishing in b. We want to find the tour of shortest length.

2 Algorithm

Finding the exact solution to the above problem is computationally impossible for more than a few tens of nodes. So we aim to an approximate solution. The advantage of the proposed solution is that it becomes more accurate as available computational resources increase.

Notation 1. We use $\mathcal{I} \setminus \mathcal{J}$ to denote the set of points in $\mathcal{I}$ which are not in $\mathcal{J}$. We use p_{end} to denote the last node in p. We use $p \frown q$ to denote the path resulting from concatenating paths p and q. We also use

$$\nu(\mathcal{I}, p) = \arg\min_{i \in \mathcal{I} \setminus p} d(p_{\text{end}}, i),$$

to denote the node in $\mathcal{I} \setminus p$ which is closest to the last node in p.

2.1 The main iterations

We propose an iterative algorithm. At each iteration, the algorithm has a best candidate solution up to that iteration. So the algorithm can be stopped at any time to bound the computations. The essential idea of the algorithm is to build a tree of all possible paths. More precisely, a branch of this tree is associated to a path from the starting node a to certain other nodes. The tree is built by adding a new branch at each iteration. This is done by adding one extra node to the path associated to a given existing branch. After creating the new branch, we compute a lower bound for the length of all possible tours starting with this new branch. If this lower bound is bigger than the length of the current algorithm's best candidate tour solution, the new branch is not added to the tree. Otherwise, we compute a new candidate tour solution including the new branch. If this candidate solution is shorter than the current algorithm's best candidate, then the latter is replaced by the former. Then, all branches whose lower bounds are smaller than the length of the new candidate solution are pruned from the tree.

We summarize below a possible sequence of steps forming the algorithm.

- Initialization:
  1. Initialize the current best candidate tour $T_\star$ for the problem $(\mathcal{I}, a, b)$ using the method described in Section 2.2.
  2. Let $p^{(0)} = \{a\}$ be the initial path.
  3. Compute the lower bound $\lambda^{(0)}$ of the length of any possible tour for the problem $(\mathcal{I}, a, b)$ using the method described in Section 2.3.
  4. Initialize the current tree by $\mathcal{P} = \left\{\left(p^{(0)}, \lambda^{(0)}\right)\right\}$ (i.e., a branch in $\mathcal{P}$ is identified with a pair (p, λ) where p is a path and λ is a lower bound of all tour starting with p).

- Main loop: at iteration $k \in \mathbb{N}$, do:
  1. Choose the branch $(p_\natural, \lambda_\natural)$ having the smallest bound $\lambda_\natural$ within the tree $\mathcal{P}$.
  2. If all candidate nodes to extend $p_\natural$ have already been considered, remove $(p_\natural, \lambda_\natural)$ from $\mathcal{P}$ and go back to 1.
  3. Create a new path $p^{(k)}$ by adding to path $p_\natural$ it closest node form the remaining ones, i.e.,
  $$p^{(k)} = p_\natural \frown \{\nu(\mathcal{I}, p_\natural)\}. \tag{1}$$
  4. Compute the lower bound $\lambda^{(k)} = l\left(p^{(k)}\right) + \mu$, where μ is a lower bound on the length of all tours for $\left(\mathcal{I} \setminus \left(p^{(k)} \setminus p^{(k)}_{\text{end}}\right), p^{(k)}_{\text{end}}, b\right)$ computed using the method described in Section 2.3.
  5. If $\lambda^{(k)} \geq l(T_\star)$, go to 1, i.e., we ignore the newly created path if its lower bound is bigger than the length of the current best candidate solution.
  6. Compute a candidate tour $T^{(k)} = p^{(k)} \frown q$, where q is a tour for $\left(\mathcal{I} \setminus \left(p^{(k)} \setminus p^{(k)}_{\text{end}}\right), p^{(k)}_{\text{end}}, b\right)$ computed using the method described in Section 2.2.
  7. If $l\left(T^{(k)}\right) < l(T_\star)$, then
     (a) $T_\star = T^{(k)}$, i.e., the new candidate is better than the current best, so it becomes the current best.
     (b) Remove all branches (p, λ) form the branch set $\mathcal{P}$, for which $\lambda \geq l(T_\star)$. (i.e., prune all branches whose lower bounds are bigger than the length of the new best candidate solution).
  8. $\mathcal{P} = \mathcal{P} \cup \left\{\left(p^{(k)}, \lambda^{(k)}\right)\right\}$, i.e., add the new branch to the tree.
- If the loop is interrupted due to timeout, return the candidate tour $T_\star$.

2.2 Computing a candidate solution

Here we are given a set $\mathcal{J}$ of nodes and $a, b \in \mathcal{J}$. We need to find some tour starting from a and finishing in b, that is not too long.

A tour can be found using a greedy algorithm. More precisely, we start from path $p^{(0)} = \{a\}$. At step k we make

$$p^{(k+1)} = p^{(k-1)} \frown \left\{\nu\left(\mathcal{J} \setminus \{b\}, p^{(k-1)}\right)\right\}.$$

In words, at each iteration k we add the node in $\mathcal{J} \setminus \{b\}$ which is closest to the last node in $p^{(k-1)}$. To force the path to end at b, in the final step we do

$$p^{(k+1)} = p^{(k-1)} \frown \{b\}.$$

2.3 Computing a lower bound on the tour length

Here we are given a set $\mathcal{J}$ of nodes and $a \in \mathcal{J}$. We need to find a lower bound of the length of all tours on $\mathcal{J}$.

A lower bound μ for all tours can be found by solving the following linear program

$$\mu = \min_{x_{i,j}} \sum_{\substack{i,j \in \mathcal{J} \\ i \neq j}} x_{i,j} d(i,j), \tag{2}$$

$$\text{subject to} \quad 0 \leq x_{i,j} \leq 1, \forall i, j, \tag{3}$$

$$\sum_{j=1}^{J} x_{i,j} = \begin{cases} 1, & \forall i \in \{a, b\}, \\ 2, & \forall i \notin \{a, b\}. \end{cases} \forall i. \tag{4}$$

The intuitive explanation for the above equations is this:

- Each edge length $d(i,j)$ has an associated weight $x_{i,j}$. We have $x_{i,j} = 1$ if edge $d_{i,j}$ is in the tour and $x_{i,j} = 0$ otherwise. However, a linear program we cannot guarantee that $x_{i,j} \in \{0, 1\}$. So we add the extra condition (3) to help us. Recall that here we are aiming for a lower bound. So we only need to consider necessary conditions for a solution.
- Equation (4) means that the sum of edges associated to each node is 2 if the node is intermediate and 1 if it is a starting or ending node.
- Equation (2) says that the linear program aims to minimize the length of the tour.