Chalmers University of Technology
University of Gothenburg
Mathematical Sciences
Optimization
Emil Gustavsson
Ann-Brith Strömberg

MVE165
MMG631
Linear and integer optimization
with applications
Assignment information
May 2, 2014

Assignment 3a: The traveling salesman problem

Given below is a description of the traveling salesman problem (TSP). The model together with its problem background are described in more detail in the notes of Lecture 9b.

Material for the assignment is found at the course homepage:
`http://www.math.chalmers.se/Math/Grundutb/CTH/mve165/1314/`
Files for creating, plotting and solving TSP instances are given.

Exercises 1–4 are mandatory. In addition, students aiming at grade 4, 5 or VG must perform *at least one* of exercises 5 and 6.

To pass the assignment you should (in groups of two persons) `(i)` write a **detailed report** that gives satisfactory answers and explanations to the questions. You shall also estimate the number of hours spent on this assignment and note this in your report.

The file containing your report shall be called **Name1-Name2-Ass3a.pdf**, where "Name$k$", $k = 1, 2$, is your respective family name. **Do not forget to write the authors' names also inside the report.**

The report should be

submitted in PingPong at the latest Friday 16th of May 2014.

Your shall also `(ii)` present your assignment orally at a seminar on

May 20, 22, or 23, 2014.

The seminars are scheduled via a doodle link, which will be published on the course home page. Presence is mandatory at at least one of these seminars.

Problem background

The Traveling Salesman Problem (TSP) is one of the most studied problems in optimization and computational mathematics. The name of the problem comes from the case when a salesman has a number of cities to visit, and he/she wishes to find the shortest route such that each city is visited at least once. It was first formulated as a mathematical problem in the 1930s and can be stated as (see also Figure 1):

Given a list of cities and their pairwise distances, find the shortest possible tour that visits each city exactly once.

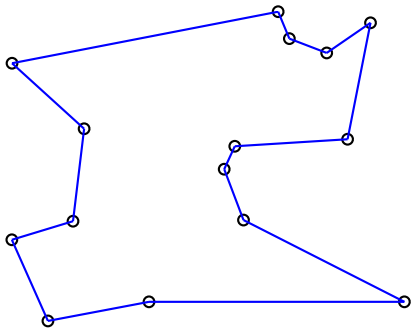

Figure 1: The shortest tour that visits all nodes exactly once.

The TSP is ane *NP-hard* problem, meaning that there does not exist any algorithm that can solve a general instance of the problem to optimality in polynomial time. In essence, an NP-hard problem is computationally difficult, and we can not expect to solve it exactly when considering larger problem instances. For a more general description of NP-hard problems; see [1].

Because of the computational complexity of the TSP, we have to consider solution methods that can provide acceptable solutions within a reasonable amount of time. Not only do we wish a candidate for a short tour, but also a measure of how much the solution can be improved by searching further. To obtain a candidate solution and an associated quality meausure, consider the following two optimization approaches:

- *Heuristics.* Algorithms for finding feasible and acceptable solutions that provide upper bounds on the objective value.
- *Relaxation algorithms.* Relaxing the problem to obtain easier–to–solve problems, which produce lower bounds on the optimal objective value.

The largest TSP instance solved to optimality is a problem with 85,900 cities, which was solved by the *Concorde code* in 2006. Another interesting example is the shortest tour visiting all the cities in Sweden (24 978 cities) which was found in 2004 and has a length of approximately 72,500 km; see [2].

The mathematical model

Consider a set $\mathcal{N} = \{1, \ldots, N\}$ of cities (nodes), where the distance between city i and j is c_{ij} for $i, j \in \mathcal{N}$. In this assignment, we consider the *symmetric TSP*, in which the distance between two cities does not depend on the direction, i.e., $c_{ij} = c_{ji}$. For the symmetric case, a model of TSP can be simplified by considering *undirected links* between the cities, meaning that we do not distinguish between the links (i, j) and (j, i). Let $\mathcal{L} \subseteq \mathcal{N} \times \mathcal{N}$ denote the undirected links in the graph; note that the links (i, j) and (j, i) are represented by one un-directed link in the set $\mathcal{L}$. When implementing the model, let $\mathcal{L} = \{(i, j) : i, j \in \mathcal{N}, i < j\}$. Introduce the binary variables x_{ij}, where $x_{ij} = 1$ if the salesman travels from city i directly to city j, and $x_{ij} = 0$ otherwise. The objective is to find a tour of minimum total distance such that each city is visited exactly once. This can be formulated as an integer linear program:

$$\text{minimize} \quad \sum_{(i,j)\in\mathcal{L}} c_{ij} x_{ij}, \tag{1a}$$

$$\text{subject to} \quad \sum_{j\in\mathcal{N}:(i,j)\in\mathcal{L}} x_{ij} + \sum_{j\in\mathcal{N}:(j,i)\in\mathcal{L}} x_{ji} = 2, \qquad i \in \mathcal{N}, \tag{1b}$$

$$\sum_{(i,j)\in\mathcal{L}:\{i,j\}\subseteq\mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1, \quad \mathcal{S} \subset \mathcal{N}, \tag{1c}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j \in \mathcal{N}. \tag{1d}$$

The objective (1a) is to minimize the distance of the tour. The constraints (1b) mean that each city must be entered and left exactly once. The constraints (1c) mean that no subtours are allowed. The subtour constraints can also be formulated as

$$\sum_{(i,j)\in\mathcal{L}:\ i\in\mathcal{S}, j\in\mathcal{N}\setminus\mathcal{S}} x_{ij} + \sum_{(j,i)\in\mathcal{L}:\ i\in\mathcal{S}, j\in\mathcal{N}\setminus\mathcal{S}} x_{ji} \geq 2, \quad \mathcal{S} \subset \mathcal{N}. \tag{1e}$$

A special case is the *metric TSP* problem in which all distances fulfill the triangle inequality, i.e., $c_{ik} + c_{kj} \geq c_{ij}$ for all $i, j, k \in \mathcal{N}$. All the problems in this assignment are metric TSP problems.

Algorithms

The most direct solution technique would be to try all possible tours and then choose the one with lowest cost. A brute force technique like this would require a running time which lies within a polynomial factor of $\mathcal{O}(n!)$, and hence would be computationally intractable even for problems with only 20 cities. The exact *Held–Karp* algorithm, which utilizes dynamic programming, solves the problem in time $\mathcal{O}(n^2 2^n)$.

Constructive heuristics

There are many specially designed constructive heuristics for the TSP which—by finding a feasible tour—yields an upper bound on the optimal objective value. Some heuristics are based on simple rules deciding which city the

route should visit next—e.g., the *nearest neighbour heuristic*—, while others are based on solving simple subproblems to find feasible tours—e.g. the *Christofides* and the *MST-heuristic*). See also [3, 4].

Probabilistic heuristics involve some stochastic rules for choosing the tour. Some of the most known probabilistic heuristics are *genetic algorithms*, *simmulated annealing* and *ant colony optimization*.

Improvement heuristics

An improvement heuristic starts from a tour (given, e.g., by a constructive heuristic) and improves it iteratively. The most used improvement heuristics for the TSP are the k-opt *heuristics* and the *crossing elimination*; see [3, 4].

Relaxation algorithms

By relaxing some constraints and solving the resulting problem to optimality, lower bounds on the distance of the optimal route are obtained. The trick when relaxing a problem is that the relaxed problem should be easy to solve. In this assignment, we will consider the *1-tree Lagrangian relaxation algorithm*.

A *tree* defined on a set of nodes is a graph that connects all the nodes without forming any cycles. A *1-tree* defined on N nodes is a connected graph with N arcs and exactly one cycle. Using this definition, the symmetric TSP can be formulated as to find the cheapest 1-tree such that all the nodes has degree two, i.e., the TSP can be to formulated as to

$$\text{minimize} \qquad \sum_{(i,j)\in\mathcal{L}} c_{ij}x_{ij}, \tag{2a}$$

$$\text{subject to} \qquad \sum_{j\in\mathcal{N}:(i,j)\in\mathcal{L}} x_{ij} = 2, \qquad i \in \mathcal{N}, \tag{2b}$$

$$x \text{ forms is a 1-tree.} \tag{2c}$$

Relaxing the assignment constraints (2b) for all the nodes except node s, results in the problem to find a *minimum spanning tree* on the set $\mathcal{N} \setminus \{s\}$ of nodes, and then connect node s to this tree by the two cheapest arcs. This can be efficiently done by first finding an MST using, e.g., Prim's or Kruskal's algorithm, and then connecting node s to the tree. The problem to find the cheapest 1-tree in a graph is called a 1-MST problem.

The 1-tree relaxation of the TSP yields, however, quite poor lower bounds. Therefore, we will Lagrangian relax the assignment constraints and consider the relaxed problem

$$q(\boldsymbol{\pi}) := \min \quad \sum_{(i,j)\in\mathcal{L}} c_{ij}x_{ij} + \sum_{i\in\mathcal{N}} \pi_i \left(2 - \sum_{i\in\mathcal{N}:\,(i,j)\in\mathcal{L}} x_{ij}\right), \tag{3a}$$

$$\text{subject to} \qquad x \text{ is a 1-tree with root node } s \tag{3b}$$

where $q : \mathbb{R}^N \mapsto \mathbb{R}$ is called the *dual objective function*. For any value of $\boldsymbol{\pi} \in \mathbb{R}^N$, the dual objective function can easily be evaluated by finding an MST over the set $\mathcal{N} \setminus \{s\}$, and then connecting node s to this tree by the two cheapest arcs. Note that the objective function in (3a) can be expressed as

$$\sum_{(i,j)\in\mathcal{L}} \widetilde{c}_{ij}x_{ij} + 2\sum_{i\in\mathcal{N}} \pi_i,$$

where $\widetilde{c}_{ij} := c_{ij} - \pi_i - \pi_j$ denotes the *reduced cost*. A high (low) value of the multiplier π_i makes node i (un-)attractive in the 1-MST problem (3).

The dual objective value yields a lower bound on the optimal objective value, and the goal of a relaxation algorithm is to find an as good lower bound as possible, i.e., to find $q^* := \max_{\boldsymbol{\pi} \in \mathbb{R}^N} \{q(\boldsymbol{\pi})\}$. To achieve this, we will utilize a so-called *subgradient algorithm*. In each iteration k of the algorithm, new multiplier values $\boldsymbol{\pi}^{k+1}$ are computed from $\boldsymbol{\pi}^k$, such that the new iterates are closer to the optimal solution $\boldsymbol{\pi}^*$. The updating utilizes the subgradient

$$h_i^k = 2 - \sum_{i \in \mathcal{N}:\, (i,j) \in \mathcal{N}} x_{ij}^k, \qquad i \in \mathcal{N},$$

where x_{ij}^k is the solution obtained from solving the 1-MST problem (3) for $\boldsymbol{\pi} = \boldsymbol{\pi}^k$. The multipliers are then updated according to

$$\boldsymbol{\pi}^{k+1} := \boldsymbol{\pi}^k + \alpha^k \boldsymbol{h}^k, \qquad k = 0, 1, \ldots, \tag{4}$$

where the steplengths α^k are defined by

$$\alpha^k := \xi^k \frac{q^* - q(\boldsymbol{\pi}^k)}{\|\boldsymbol{h}^k\|^2}, \qquad 0 < \varepsilon_1 \leq \xi^k \leq \varepsilon_2 < 2, \qquad k = 0, 1, \ldots. \tag{5}$$

Since q^* is unknown, the value of the best primal feasible solution obtained so far is used in (5). One common choice is to start the sequence $\{\xi^k\}$ with $\xi^0 = 2$ and reduce ξ^k by a factor of 2 whenever the value of $q(\boldsymbol{\pi}^k)$ has failed to increase in a prespecified number of iterations.

A measure of the quality of the lower bound on the optimal objective value can be found by constructing a heuristic that finds a feasible tour based on the solution found in the evaluation of the dual objective function (3). A short version of the algorithm can thus be formulated as follows:

Step 0: Let $k = 0$ and choose $\boldsymbol{\pi}^0 \in \mathbb{R}^N$.
Step 1: Given $\boldsymbol{\pi}^k$, solve (3).
Step 2: Construct a feasible tour. Start from the solution obtained from solving the dual problem and make adjustments such that it becomes feasible.
Step 3: If the duality gap is small enough, stop.
Step 4: Compute $\boldsymbol{\pi}^{k+1}$ according to (4), let $k := k + 1$, and go to Step 1.

Exercises to perform and questions to answer

1. Use the TSP model implemented in `TSP.mod` and solve the instances `TSP_small` and `TSP_medium` with CPLEX. What is the length of the shortest tour in each of the instances? Now, relax (one set at a time) the constraints (1b), (1c), and (1d), and resolve the instances. (When relaxing the constraints (1b), use (1e) as the subtour constraints). Which of the relaxations yield(s) the best lower bound? Plot the solutions for the instances with and without relaxations using the file `plot_TSP.m` in `MATLAB`. Describe what you see. Comment on the computation times.

2. (a) Construct a deterministic, constructive heuristic. You are more than welcome to use your own heuristic or other heuristics not mentioned in this assignment description. Explain how it works.

(b) Apply the heuristic to all the test instances. What are the lengths of the tours found by the heuristic? Can you (by visual inspection) determine whether the solutions can be improved?

(c) What can you say about the optimal objective values for the test instances? What kind of performance guarantee can your heuristic provide? Why? Try to construct an instance for which your heuristic performs as bad as possible; it does not have to be metric.

(d) Analyze how the computation time of your heuristic depends on the number of cities. Use the file `create_TSP.m` to create TSP instances.

3. (a) Construct an improvement heuristic that starts from a feasible tour and then improves it. Explain how it works.

   (b) Test the improvement heuristic by, for each of the test instances, starting from the tour found by the constructive heuristic. How much does the heuristic improve the initial tours?

4. (a) Implement the 1-tree Lagrangian relaxation algorithm. Note that you need to construct a primal feasibility heuristic that is used in each iteration.

   (b) Apply your algorithm to all the test instances. What can you say about the optimal objective values?

   (c) Show in a graph how the gap between the lower and upper bounds improves in each iteration. Since some of the instances are large you should set a limit on the allowed number of iterations.

5. Implement a probabilistic heuristic for the TSP and explain how it works. Try it on all the test instances and analyze its performance. Can the heuristic provide any performance guarantees on the solutions found?

6. Describe how *either* a branch–and–bound *or* a cutting plane algorithm can be utilized for solving the traveling salesman problem (you don't have to implement it). References are found in [3, 4, 5].

References

[1] The P versus NP page. `www.win.tue.nl/~gwoegi/P-versus-NP.htm`

[2] Concorde TSP Solver. `www.tsp.gatech.edu/concorde.html`

[3] Optimization. *Jan Lundgren, Mikael Rönnqvist, Peter Värbrand*. Studentlitteratur AB, Lund, 2010.

[4] The Traveling Salesman Problem: An overview of exact and approximate algorithms. *Gilbert Laporte*, European Journal of Operational Research 59 (1992) 231–247, North-Holland.

[5] Solving a TSP—Cutting Plane.
`www.tsp.gatech.edu/methods/dfj/index.html`