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

MVE165
MMG630
Applied Optimization
Assignment information
May 2, 2011

Assignment 3a: 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 13.

Material for the assignment can be found at the course homepage:
`http://www.math.chalmers.se/Math/Grundutb/CTH/mve165/1011/`
Files for creating, plotting and solving TSP instances are given.

Questions 1-4 are mandatory. In addition, students aiming at grade 4, 5 or VG must answer *at least one* of questions 5 or 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 e-mailed to `anstr@chalmers.se`
at the latest on Friday 13 of May 2011.

Your shall also `(ii)` present your assignment orally at a seminar in
Week 20 (16–20 of May) 2011.

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 wishes to find the shortest route such that he only has to travel to each city once. It was first formulated as a mathematical problem in the 1930s and can be stated as:

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

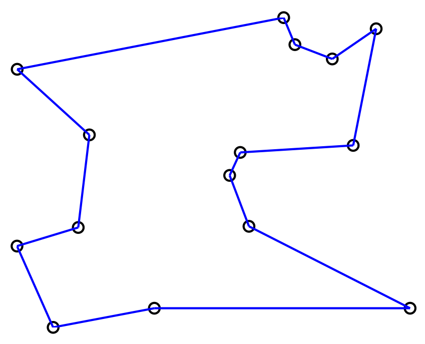

Figure 1: The shortest tour that visits all nodes exactly once.

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

Because of the computational complexity of the TSP, we have to consider solution methods that can provide us with acceptable solutions within a reasonable amount of time. Not only do we want to have a candidate for what we believe to be a short tour, but also a measure of how much we can improve our solution by searching further. To obtain a candidate solution and a quality measure of the solution, we will consider two optimization methods;

- *Heuristics:* Algorithms for finding feasible and acceptable solutions that will give upper bounds on the objective value.
- *Relaxation algorithms:* Relaxing the problem can give easier subproblems which produce lower bounds on the 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. [2]

The mathematical model

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

$$\begin{aligned}
&\text{minimize} && \sum_{(i,j)\in\mathcal{L}} c_{ij}x_{ij}, && && \text{(1a)}\\
&\text{subject to} && \sum_{j\in\mathcal{N}:(i,j)\in\mathcal{L}} x_{ij} + \sum_{j\in\mathcal{N}:(j,i)\in\mathcal{L}} x_{ji} = 2, && i \in \mathcal{N}, && \text{(1b)}\\
& && \sum_{(i,j)\in\mathcal{L}:\{i,j\}\subseteq\mathcal{S}} x_{ij} \le |\mathcal{S}| - 1, && \forall \mathcal{S} \subset \mathcal{N}, && \text{(1c)}\\
& && x_{ij} \in \{0, 1\}, && i, j \in \mathcal{N}. && \text{(1d)}
\end{aligned}$$

The objective (1a) is to minimize the distance of the tour. Constraints (1b) says that we must enter and leave each city exactly once and constraint (1c) says that no subtours are allowed. The subtour constraints can also be formulated as

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

A special case is the *metric TSP* problem in which all distances fulfill the triangle inequality, i.e., $c_{ik} + c_{kj} \ge c_{ij}$ for all $i, j, k \in \mathcal{N}$. All the problems in this assignment will be 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 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. There is an exact algorithm called the Held-Karp algorithm which utilizes dynamic programming than can solve 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, can give an upper bound on the optimal solution. Some heuristics are for example based on simple rules deciding which city the route should visit next (ex *nearest neighbour heuristic*), while other are based on solving simple subproblems to find feasible tours (ex *Christofides* and the *MST-heuristic*). See [3] and [4]

Another kind of heuristics are called probabilistic heuristics, meaning that they 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 is an algorithm that starts with a tour (given for example by some constructive heuristic) and improves it. The most used improvement heuristics for the TSP are k-opt *heuristics* and *crossing elimination.* [3], [4]

Relaxation algorithms

By relaxing some constraints and then solving the reduced problem to optimality, lower bounds on the distance of the optimal route can be obtained. The trick when relaxing a problem is to do it in such a way that the relaxed problem becomes 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 nodes without any cycles. We say that a *1-tree* defined on N nodes is a connected graph with N arcs and exactly one cycle. From this definition, we can see that the symmetric TSP can be formulated as finding the cheapest 1-tree such that all nodes has degree two, i.e., the TSP can be formulated as

$$\begin{aligned}
&\text{minimize} && \sum_{(i,j)\in\mathcal{L}} c_{ij}x_{ij}, && && \text{(2a)}\\
&\text{subject to} && \sum_{j\in\mathcal{N}:(i,j)\in\mathcal{L}} x_{ij} = 2, && i \in \mathcal{N}, && \text{(2b)}\\
& && x \text{ is a 1-tree} && && \text{(2c)}
\end{aligned}$$

If we relax the assignment constraints (2b) for all nodes except for node s, the resulting problem is the problem to find a *minimum spanning tree* on the set $\mathcal{N} \setminus \{s\}$ of nodes, and then connect node s to the tree by the two cheapest arcs. This can be done very efficiently by first finding the MST by for example Prims or Kruskals algorithm, and then connecting node s to the tree. The problem to find the least expensive 1-tree in a graph is called a 1-MST problem.

However, the 1-tree relaxation to the TSP gives fairly poor lower bounds. Therefore, we will instead Lagrangian relax the assignment constraints and consider the relaxed problem

$$\begin{aligned}
q(\boldsymbol{\pi}) = &\text{minimize} && \sum_{(i,j)\in\mathcal{L}} c_{ij}x_{ij} + \sum_{i\in\mathcal{N}} \pi_i \left(2 - \sum_{i\in\mathcal{N}:\,(i,j)\in\mathcal{L}} x_{ij} \right), && \text{(3a)}\\
&\text{subject to} && x \text{ is a 1-tree with root node } s && \text{(3b)}
\end{aligned}$$

where $q(\boldsymbol{\pi})$ is called the dual objective function. For any $\boldsymbol{\pi}$, the dual objective function can easily be evaluated by finding a MST on a set $\mathcal{N} \setminus \{s\}$, and then connecting node s to the tree by the two cheapest arcs. We note that the objective function (3a) can be written as

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

where $\widetilde{c}_{ij} = c_{ij} - \pi_i - \pi_j$ are called the *reduced costs*. We see that the Lagrangian multipliers π_i makes a node attractive (unattractive) if the value of the multiplier is high (low) in the 1-MST problem (3).

The dual objective value always gives a lower bound on objective function, and the goal of a relaxation algorithm is to find an as good lower bound as possible, i.e., we are interested in finding $q^* = \max_{\boldsymbol{\pi}\in\mathbb{R}^N} q(\boldsymbol{\pi})$. To achieve this, we will utilize a subgradient algorithm. In each iteration k of the algorithm, new Lagrangian multipliers $\boldsymbol{\pi}^{k+1}$ are computed from $\boldsymbol{\pi}^k$, such that we get closer to the optimal solution $\boldsymbol{\pi}^*$. The updating is done by considering the subgradient

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

for $i \in \mathcal{N}$, where x_{ij}^k is the solution obtained from solving the 1-MST problem $q(\boldsymbol{\pi}^k)$. We then update the Lagrangian multipliers according to

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

where the steplength α^k is defined by

$$\alpha^k = \xi^k \frac{q^* - q(\boldsymbol{\pi}^k)}{\|\boldsymbol{h}^k\|}, \quad 0 < \xi^k < 2.$$

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

To have a measure on how good the lower bound is in each iteration, one should construct a heuristic that finds a feasible tour based on the solution found in each evaluation of the dual objective function. A short version of the algorithm can thus be formulated as

Step 0: Let $k = 0$ and choose $\boldsymbol{\pi}^0$.
Step 1: Given $\boldsymbol{\pi}^k$, solve (3).
Step 2: Construct a primal feasible solution. Use the solution obtained from solving the dual problem and make adjustments such that it becomes feasible.
Step 3: If the duality gap is small, then stop.
Step 4: Compute $\boldsymbol{\pi}^{k+1}$ from (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 problems `TSP_small` and `TSP_medium` with CPLEX. What is the length of the shortest tour in each instance? Now relax (one at a time) constraints (1b), (1c) and (1d), and resolve the problems. (When you relax constraints (1b), use (1e) as subtour constraints). Which of the relaxations gives the best lower bound? Plot the solutions with and without relaxations by using the `plot_TSP.m`-file in `MATLAB`. Describe what you see. Give some comments 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. Explain how it works.

   (b) Use the heuristic on all the test problems. What is the length of the tours found by the heuristic? Can you (by visually inspecting the solutions) determine if the solutions can be improved?

   (c) What can you say about the optimal objective values for the test problems? What kind of guarantee can your heuristic give? Why? Try to construct a TSP problem where your heuristic would perform as bad as possible. The problem 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 problems.

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

   (b) Try the improvement heuristic by taking the tours found by the constructive heuristic for all test problems. How much can 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) Use it on all the test problems. What can you say about the optimal objective values?

   (c) Show in a graph how the gap between the lower and upper bound improves in each iteration. Because some of the problems are large, you should set a limit on how many iterations you allow.

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

6. Describe *either* how a Branch and Bound algorithm *or* a Cutting Plane algorithm could be used for solving the traveling salesman problem. References can be found at [3], [4] and [5]. You do not need to implement it.

References

[1] The P versus NP page.
http://www.win.tue.nl/∼gwoegi/P-versus-NP.htm

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

[3] Optimization. *Jan Lundgren, Mikael Rönnqvist, Peter Värnbrand* 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
http://www.tsp.gatech.edu/methods/dfj/index.html