

CS 312, Winter 2006

Project #6: Monte Carlo TSP

Early Date: April 3, 2006
Due: April 5, 2006

Overview:

In this project, you will implement another strategy for solving the TSP using a probabilistic algorithm of the Monte Carlo variety.

Objectives:

- To understand the role of randomization in probabilistic algorithms.
- To understand the Monte Carlo class of probabilistic algorithms.
- To contrast a Monte Carlo algorithm with a deterministic algorithm.
- To attempt to improve on the performance of your deterministic TSP solver from Project #5.

Details:

In this project, you will implement a probabilistic algorithm to solve the TSP. Specifically, you will implement a Monte Carlo TSP solver. Refer to chapter 10 in the text in order to gain a better appreciation of the role of randomization in probabilistic algorithms and to understand the distinction among the various types of probabilistic algorithms, including Monte Carlo. This project will complement that reading in helping you understand the principles at stake.

A probabilistic algorithm chooses a course of action at random among multiple options rather than spending excessive time working out which alternative is best. A Monte Carlo algorithm is a probabilistic algorithm that is guaranteed to return a solution (or an optimal solution) with some probability p ($0 < p \leq 1$). Run a sufficient number of times, it will return the desired solution.

For your Monte Carlo TSP solver, you will need to introduce some random choice into your algorithm. In project #5, you implemented one of two search strategies (i.e., node expansion strategies):

1. create one new search node for each neighboring city remaining (for which there was not an infinity in the reduced cost matrix), or
2. select one edge and create one new search node representing the inclusion of that edge and another new search node representing the exclusion of that edge.

For this project, you may either stick with your chosen strategy or try the other. In either case, your Monte Carlo algorithm will not need to maintain a priority queue to hold alternative search states. Instead, your algorithm will randomly select one (child) node and pursue that and only that! You must also restrict the selection to the set of feasible nodes using the techniques you developed for project #5. In other words, you will want to avoid premature cycles by using the routine presented in lecture #29.

If you select strategy #2, not only will you randomize the selection of the “child” node (inclusion or exclusion), you may also randomly select the edge to be included or excluded, subject to feasibility, of course. You may consider experimenting with both kinds of randomness and their combination. The former kind (randomly selecting the next search state), however, is required. Note that a Monte Carlo algorithm for search strategy #2 (edge inclusion/exclusion) may fail to come up with a solution (a valid TSP tour). In other words, it may terminate with a set of n edges that do not complete a tour. You will want to tighten your feasibility constraints in order to help prevent this.

Note that there is good advice on p. 332 of the text regarding how to use a $\text{uniform}(0,1)$ pseudo-random number generator for $\text{uniform}(a,b)$, $\text{uniform}(i..j)$, and $\text{coinflip}()$.

Side note: Las Vegas algorithms differ from Monte Carlo algorithms in the following respect: in a Las Vegas algorithm a purported solution can be verified efficiently for correctness. If it is found to be wrong, the Las Vegas algorithm will admit that it failed rather than respond incorrectly. Note that regardless of your implementation choice (with regard to search strategy), your algorithm will not be a Las Vegas algorithm, because even when you return a solution, you don't know whether it is the optimal solution. A Las Vegas algorithm would return the desired solution or admit failure. Now, if we were seeking only a Hamiltonian cycle (ignoring edge weights), and not the least cost solution (i.e., a TSP tour), our algorithm would indeed be a Las Vegas algorithm.

To Do:

1. Use the codebase from project #5 (TSP with Branch and Bound; distribution is here: http://vv.cs.byu.edu/cs312/archives/2006/03/project_5_tsp_w.html). You will be comparing your deterministic algorithm implemented in project #5 with your probabilistic algorithm. You may want to reuse some elements of your deterministic TSP solver code in your probabilistic algorithm. There is no new code to be distributed with this project; however, if you are behind on project #5 or you submitted an incomplete implementation for partial credit, then you should speak with your instructor, who will make some accommodations for you.
2. You will run your solvers on a set of at least 10 instances of TSP. Your basic probabilistic algorithm will run to completion very quickly, so you will run it several times in a loop in order to monitor the behavior of your probabilistic algorithm. For each instance, your solver should re-run on the same instance

- until time elapses. Employ a time-out mechanism so that the loop of retries will terminate and report the best solution after 10 seconds of execution time.
3. On any given instance of the problem, the execution time and result may vary from one run to the next. You will be expected to monitor both execution time and result cost and to report your findings. Within the time permitted, you will need to measure and report the mean, maximum, and minimum execution time and result quality (TSP tour cost) across runs on those TSP instances. Also report the mode (most frequent answer) of the tour costs.
 4. Search strategy #2 (edge inclusion/exclusion) may fail to come up with a solution (a valid TSP tour). In such cases, your code should count failures encountered across the runs on each instance. On a given instance, what percentage of the runs lead to failure (a non-solution)? This is a good estimate of the probability that the algorithm fails.
 5. You will need to compare your best answer with the answers achieved by your deterministic algorithm from project #5. In each case, which algorithm leads to a better result: a deterministic B&B search or several iterations (time permitting) of a probabilistic algorithm?
 6. Write a report as described below and turn in the report.

Report:

1. Report your results in a table having the following format:
 - a. One row per instance: Your table must include at least 10 rows of results for 10 different TSP instances with one instance per row. Of those instances, when run deterministically, at least 3 must run for 10 seconds (before timing out). You must run your probabilistic algorithm as many times as possible within the 10 sec. time limit and report the following statistics. Note that iterating your algorithm and calculating statistics should be done in code rather than by hand.
 - b. One group of columns describing performance of the Monte Carlo algorithm. Note that you may want to format your table in landscape mode to accommodate all of the columns.
 - i. Mean running time (sec.)
 - ii. Max running time (sec.)
 - iii. Min running time (sec.)
 - iv. Mean cost of tour
 - v. Max cost of tour
 - vi. Min cost of tour
 - vii. Mode cost of tour
 - viii. Total iterations
 - ix. Total failed iterations

- x. Estimated probability of failure
 - c. One group of columns describing performance of the deterministic B&B algorithm
 - i. Running time (sec.)
 - ii. Cost of tour
 - d. One column indicating which algorithm found the superior answer
2. Make a clear concise statement comparing the time performance of your Monte Carlo algorithm with your deterministic Branch and Bound algorithm. Your statement may also include the impact on space performance. Be careful to make a conclusion no stronger than your results.
 3. Give your best explanation of the outcome you observed.

Revised: 3/28/2006