The Algorithm

Monte Carlo Popularity with 2-Opt

Presented as an efficient alternative to finding acceptable solutions to

The Travelling Salesman Problem

Submitted to

Dr. Tony Martinez

For

CS 312: Algorithm Analysis

Brigham Young University

Provo, Utah

December 11, 2014

By

Tommy Jensen, Sean Lane, and Epper Marshall

# Abstract

The Monte Carlo Popularity with 2-Opt algorithm (MCP2Opt) is a hybrid of algorithm methodologies attempting to combine efficiency and accuracy in solving TSP. In this paper, we will analyze the paradigm behind Popularity, how a Monte Carlo modification makes it reliably productive, and how the 2-Opt heuristic (not created by the authors) helps to make answers more accurate.

# I Introduction

This paper introduces an algorithm that was originally created by the authors as well as work done by others. An explanation of the Nearest Neighbor algorithm (known in class as “Greedy”) will set the stage for the Popularity algorithm’s paradigm as a greedy algorithm. Then, Popularity’s polynomial efficiency and its unique solution-finding efficiency when used with the Monte Carlo algorithm design will be discussed. Additionally, the author’s caching implementation of 2-opt will be shown to have great running time improvements over naively-implemented 2-opt without sacrificing any of the gains made by the brute force approach. Finally, the empirical analysis of MCP2Opt’s performance against Greedy and Branch and Bound will conclude this paper.

# II Greedy (Nearest Neighbor)

The Greedy algorithm (often known as “Nearest Neighbor”) is a commonly used approximate solver for TSP because it is easy to implement and gets an answer very quickly. However, it has the drawbacks associated with other local-search algorithms; namely, that it can get stuck in local minima and never find an even close to optimal solution. In fact, city layouts exist that Greedy will find uniquely worst possible solutions for! Nevertheless, the authors found it useful as a source of inspiration.

### Implementation

The authors implemented Greedy according to its well-known pseudocode:

1. Pick a random starting city
2. Find the shortest edge that goes to an unvisited city, and travel it
3. Repeat Step 2 until the original starting city is reached, unless no routes are available, then continue
4. If a dead-end was found, start over at Step 1 with a different start city

Using this pseudocode, Greedy becomes an excellent base-line for comparison for other TSP solving algorithms.

# III 2-Opt

The 2-opt algorithm is a simple local search used to compare the reverse sub sequences of a path. First you measure the original path and then reverse the nodes/cities between the two points and measure the same distance. If the second sequence is smaller the original path is updated, now with that subsequence updated and the overall path length reduced. Because this compares subsequences this is only used to optimize a path not solve a problem by itself.

This is fairly straight forward algorithm but it is a brute force algorithm so it adds time exponentially. Our original algorithm would take O (n^3) time, going through every node on the path as the starting and ending node, and then checking a subsequence length and reverse sequence length.

We added a cache though where if the original subsequence was not reversed we wouldn’t need to recalculate the next subsequence (previous sequence + 1). With the cache we keep the current path length and add the distance to the new end point, until a swap occurs. This way each iteration only occasionally needs to recalculate a subsequence, reducing the time complexity to O (n^2).

Even though the time is exponential, it does find path lengths 12-25% shorter than the greedy algorithm. The results are fairly steady and tend to average about 17% shorter paths than other solutions.