Yet Another 2-opt

Simple And Fast Approximation For
The Traveling Salesperson Problem

Jacob Thomsen
jakethom02@gmail.com

John Tappen
jtappen@gmail.com

Jonah Simmons
jonahksimmons@gmail.com

Professor Tony Martinez
Brigham Young University
April 18, 2023

Abstract

The 2-opt algorithm is an optimization algorithm that can be applied to any approximate solution to the Traveling Salesperson Problem. We will explore and discuss our testing and variation on the algorithm as well as possible improvements to it.

1. Introduction

The Traveling Salesperson Problem (TSP) is arguably the most well know NP-Hard problem in the world. From its formalizing in the 19th century, it seems all possible solutions have been exhausted: from optimal exponential solutions to fast heuristic approximations. We will be focusing on an approximation that uses a greedy approach to find a fast solution. Then, we will improve upon it using an optimization known as *2-opt*.

2. Greedy

Greedy algorithms solves problems with the "best now" mentality. It looks at all current options, assigns some quantifier to each, and chooses the best. For example, a greedy chess solver would protect its pieces or capture rather than sacrificing a piece for a checkmate in three moves.

2.1. Disadvantages

While greedy might seem very good, it has several flaws. As it does not take into account future decisions, it usually does not return the optimal for more complex problems, TSP being one of those. The solution from a greedy approach can be vastly worst than the optimal and, in theory, it almost random.

2.2. Advantages

Although it usually does not return an optimal, greedy approaches return good enough solutions for most problems. From the tests we ran, the greedy solutions were hard to beat by much. Greedy also usually returns a valid solution to most problems it is given.

In addition to returning a good value, it does it very fast. The main issue with guaranteed optimal approaches is that it must take into account all other possibilities to ensure the best solution; this takes an unpractical amount of time.

2.3. Time Complexity

Algorithm 1 Greedy algorithm

```
1: procedure GREEDY → PATH
2:     let current = starting city
3:     while not visited all cities do
4:         let best = closest neighbor to current that has not been visited
5:         path.add(best)
6:         current = best
7:     end while
8: end procedure
```

Given the pseudo-code, we can see the time complexity is $O(V \times E)$, where V is the number of cities and E is the number of out edges. The outer loop goes through all nodes, $O(V)$, and each iteration looks at all neighbors, $O(E)$.

2.4. Purpose

The purpose of including the greedy algorithm is to offer a benchmark and comparison to the other algorithms. It produces fine results, but can be improved with a little bit of work. Due to its inefficiencies and speed, it will be used as the starting point for our 2-opt.

3. 2-opt

Optimizations are very helpful in solving problems, especially very hard problems. One such optimization that works for TSP is *2-opt* which makes minor changes to a given solution; in our case the initial solution is greedy. The basic idea of 2-opt is swapping two edges, if it would improve the overall cost, then re-ordering the rest of the graph to make it a valid path.

This approach was chosen because of its flexibility and speed. It's flexible because it can be applied to any other approximate solution, no matter how it was solved. This "back end" optimization is especially helpful for very difficult problems, such as the TSP, and virtually has no drawbacks; whenever it is able to be used, it should b used.

3.1. Disadvantages

The issues with this approach are that it must have a valid solution to begin with and it only works for bi-directional graphs. For many real world problems, the bi-directional requirement can be a big problem and the initial solution that is chosen really impacts the output of 2-opt.

3.2. Advantages

Considering its shortcomings, 2-opt is still a very good approach. Since it is an optimization, it will always return a valid solution that is as good or better than your initial solution. In addition, 2-opt is rather fast.

3.3. Improvements

While the base algorithm is fine, there are some changes we can make to overcome some of its disadvantages. Since it is a local algorithm, or only looks at changes that are "close", it can easily miss a better solution that is further away.

To get out of these local minimums, we used a Monte Carlo approach. We would simulate a coin flip to determine if there would be a swap (if the normal conditions aren't met) or not. This randomness gives it the ability to explore a new path that might lead to an overall better solution. From testing of this approach, we determined that it did not improve the final results by any significant amount and it dramatically increased the runtime.

Since this initial Monte Carlo approach was not effective, we decided to lower the chance of a random swap down to ten percent. This was far better than the previous 50/50 and was able to overcome the localization issue with standard 2-opt.

Another key issue with 2-opt is how the initial solution strongly effects the final solution. If a random or purely convenient initial solution was chosen, it would not guarantee a good final result. Because of this, we had to be careful to pick an initial solution that was fast but still good. From trying a few different options, we settled on the best of a few greedy solutions each starting at different points. Since the greedy was quite fast but determined on starting point, running it multiple times will produce a fairly good solution without increasing the time complexity.

With these slight improvements, we were able to find very good solutions in a practical amount of time.

3.4. Time Complexity

Algorithm 2 2-opt algorithm

```
1: procedure 2OPT → TOUR
2:     let tour = initial solution
3:     while tour can be improved do
4:         for edge1, edge2 ∈ tour do
5:             if swapping the destinations improves cost then
6:                 swap them
7:                 reverse intermediate edges
8:             end if
9:         end for
10:    end while
11:    return tour
12: end procedure
```

As the pseudo-code shows, the number of iterations is unknown as the condition for the while loop is based on uncertain state. For each iteration, it must loop over all pairs of edges which is $O(E^2)$, where E is the number of edges. So the time complexity of 2-opt is $O(P \times E^2)$, where P is the unknown number of iterations.

Since 2-opt has to use an initial solution, the total time complexity must account for that as well. For our implantation, we use the best of 10 runs of our greedy algorithm, so the time complexity would be $O(V \times E + P \times E^2)$, assuming P is a constant, is equivalent to $O(V \times E + E^2)$.