Monte Carlo 2-opt

Fast And Powerful 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 rank to each, and chooses the best. For example, a greedy chess solver would capture a pawn rather than sacrificing a knight for a checkmate in three moves.

2.1. Advantages

Greedy approaches usually return good enough solutions for most problems. From the tests we ran, the greedy solutions were hard to beat by much.

In addition to returning a good value, it does it very fast. It does not require any knowledge of the future and does not care about the past. Because of these traits, it means that greedy approaches are usually computationally cheap, especially compared to other options. 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.

2.2. Disadvantages

While greedy might seem ideal, it has several flaws. As it does not take into account future decisions, it usually is not guaranteed to return the optimal for most problems, TSP being one of those. The solution from a greedy approach can be vastly worst than the optimal and sometimes is almost random.

2.3. Time Complexity

Algorithm 1 Greedy algorithm

```
procedure GREEDY → PATH
    let current = starting city
    while not visited all cities do
        let best = closest neighbor to current that has not been visited
        path.add(best)
        current = best
    end while
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 to get the nearest unvisited neighbor, $O(E)$.

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 fast but imperfect nature, it will be used as the starting point for our 2-opt.

3. 2-opt

Optimizations are very helpful in solving 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 as long as it's fast, has few drawbacks; whenever it is able to be used, it should be 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, this requirement can be a big problem. In addition to this requirement, it is also highly limited by its initial solution.

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 whether or not there would be a swap (if the normal conditions aren't met). This randomness gives it the ability to explore a new path that might lead to an overall better solution.

From testing of this coin flip approach, we determined that it did not improve the final results by any significant amount and it dramatically increased the runtime. Since this did not return a result in a reasonable amount of time, we decided to remove it.

The other 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 overall 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

```
procedure 2OPT → TOUR
    let tour = initial solution
    while tour can be improved do
        for edge1, edge2 ∈ tour do
            if swapping the destinations improves cost then
                swap them
                reverse intermediate edges
            end if
        end for
    end while
    return tour
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 until the tour cannot be improved anymore.

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 $O(V \times E + E^2)$.

4. Results

W.I.P.

Random	Greedy	Branch and Bound	2 Opt

Number of cities	Time	Length	Time	Length	Percent of rand	Time	Length	Percent of greedy	Time	Length	Percent of greedy

5. Future Work

For further attempts, there are some considerations. The main concern is escaping local minimums with 2-opt. From our testing, this would require a more sophisticated approach than Monte Carlo. One possible approach would be running 2-opt on multiple initial solutions that might not be immediately as desirable, but might have a lower final solution.

Another direction would be to use an entirely different solver for initial solutions than greedy. There are many approximations to this problem, each with their respective pros and cons. Much additional testing could discover the best future course of action for either of these shortcomings.