Final Year Project

Learning shortest path tours to all pubs in UK

Sabeer Bakir

Student ID: 16333886

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Deepak Ajwani

UCD School of Computer Science

University College Dublin

November 5, 2019

Table of Contents

1 Project Specification 3

2 Introduction 4

3 Related Work and Ideas 5

4 Data Considerations 9

5 Outline of Approach 10

6 Project Workplan 11

7 Summary and Conclusions 12

Abstract

Designing algorithms for NP-hard combinatorial optimization problems over graphs is a complex task. It has been shown with use of good heuristics and approximation algorithms that these problems can be solved in polynomial time. However, these solutions require problem-specific knowledge and trial-and-error. Is it possible to automate this learning process with the use of machine learning classification? In real world applications, the data is very rarely similar however the structures within may not differ as much. In this paper, we will explore a method to solve the Travelling Salesman Problem (TSP) using classification to identify edges that are contained within the tour created by the TSP. The applications of this can be applied in many areas such as, DNA Sequencing, Route Planning, Logistics Management, etc.

Chapter 1: Project Specification

Core

- Identify features of edges that can discriminate between edges in the shortest TSP tour and edges that are not.
- Take random samples of the pub crawl dataset and use a state-of-the-art heuristic on that to get the ground truth for training
- Use the identified features and the ground truth on random samples to train a classification model that learns the edges in the shortest TSP tour
- Evaluate the accuracy of the classification model

Advanced

- Compare the running time vs. optimality trade-off of the learning approach with the Concorde TSP solver (http://www.math.uwaterloo.ca/tsp/concorde.html)
- Improve the accuracy of the classification algorithm by careful feature selection

Chapter 2: **Introduction**

The Travelling salesman problem (TSP) is a combinatorial graph optimization question that is. This problem is NP-hard and has caused considerable interest by theory and algorithm design communities in the past. In the realm of computational complexity, one of Karp's 21 NP-complete problems is the Hamiltonian Cycle Problem, this is a special case of the travelling salesman problem.

The TSP was first introduced in the 1800s by the Irish mathematician Sir William Rowan Hamilton and the British mathematician Thomas Penyngton Kirkman. However, optimisation of the TSP only came about in the 1930s from Karl Menger in Vienna and Harvard. Menger defined the problem as, "Given a finite number of points, with known pairwise distances, find the shortest path connecting the points."[1]

Techniques that are used to solve these graph optimisation problems comes in three main type: exact algorithms, approximation algorithms and heuristics. Exact algorithms always find the most optimal solution however it requires exponential time which does not scale well for large inputs. Approximation algorithms offer the desirable polynomial time solution however it is hard to guarantee the optimality for certain problems. Heuristics are typically fast but require problem specific research and experimenting from the algorithm designers.

Designing algorithms for NP-hard combinatorial optimization problems is a complex task. It has been shown with use of good heuristics and approximation algorithms that these problems can be solved in polynomial time. However, these solutions require problem-specific knowledge and trial-and-error. Is it possible to automate this learning process with the use of machine learning classification? In real world applications, the data is very rarely similar however the structures within the data do not differ. In this paper, we will explore a method to solve the Travelling Salesman Problem (TSP) using classification to identify edges that are contained within the tour created by the TSP. We will see if it is possible to learn the shortest path to all pubs in the UK. The applications of this can be applied in many areas such as, DNA Sequencing, Route Planning, Logistics Management, etc.

Chapter 3: Related Work and Ideas

3.0.1 Background to Combinatorial Optimization

Combinatorial optimization is an area of research that consists of finding an optimal ordering/subset from a finite set of objects. In many of these problems, exhaustive search is not an option as these problems are NP-hard, meaning that there is no known polynomial time algorithm to solve them. There is a demand for solutions to these problems in many domains such as network design, bioinformatics, game theory and many more [2]. Is it possible to still find good solutions to these problems? Due to the popularity of these problems, it has created a surge of algorithms that approximate a solution close to the optimal solution. These methods include exact algorithms [3], approximation algorithms [4], domain specific heuristics[5] and meta-heuristics[CITATION]. The travelling salesman problem is no exception to these methods, the TSP is known as a classic problem in combinatorial optimisation.

Initially the travelling salesman problem was tackled using exact algorithms, searching for a complete solution. One of the first algorithms of this nature was the Held-Karp Algorithm [3]. These exact algorithms that searched the entire solution space often were only useful for very small input sizes and was usually not suitable for real world data more nodes and edges than synthetic problems.

Exact Algorithms An exact algorithm in the domain of optimization refers to a method that will always yield the most optimal solution. Such algorithms do exist for the TSP [3], this is one of the earliest adoptions of *Dynamic Programming*. The main disadvantage to this algorithm is its runtime of $\mathcal{O}(N^2 2^N)$. Due to this, algorithm designers have looked towards other methods of optimisation such as heuristics and approximation algorithms.

3.0.2 Approximation Algorithms

Algorithm designers looked towards approximating the solution such that a close to optimal solution could be obtained in substantially less time than an exact algorithm. Many approximation algorithms have been applied to the travelling salesman problem; one noteworthy method is Christofides [4]. There has been little to no improvement in approximation algorithms since.

Christofides ensures solutions that will be within a factor of 3/2 of the optimal solution. The tour is generated by constructing the minimum weight spanning tree where the weights are the distances between the nodes. Compute a perfect matching on the nodes of odd degree and combine with the spanning tree. This will result in a connected graph with each node having even degree. On one hand, this approach has the best worst-case solution of the constructive heuristics, on the other hand it's runtime is substantially longer than heuristics due to the perfect matching algorithm complexity.

3.0.3 Heuristics

Designers looked towards domain specific heuristics to solve TSPs with some prior knowledge of the problem at hand. These methods are fast and will run in polynomial time whilst yielding solutions very close to optimal. An example of such heuristics can be found in the Concorde TSP Solver [5], a very powerful program used to solve TSPs with close to optimal solutions on graphs with tens of thousands of nodes.

Constructive Heuristics This approach differs from local search heuristics where a solution is given and optimality is obtained by making local changes at various points in the solution. Constructive heuristics begin with an empty solution and attempt local optimizations at each point in the solution. In [4], algorithms such as *Nearest Neighbour (NN)*, *Greedy*, *Clarke-Wright*, and *Christofides* are applied to the TSP.

- **Nearest Neighbour (NN):** This algorithm constructs a tour from starting at some arbitrary node and builds the cycle by adding the nearest neighbour from the current node. It's runtime is $\mathcal{O}(N^2)$. This provides a decent approximation for the most optimal, on average the path length is roughly 25% longer than the most optimal path.
- **Greedy:** Often mistaken for Nearest Neighbour, it is constructed by taking the complete graph of the TSP instance, and the path starts with the shortest edge. It is further built by continuously adding the next shortest incident edge such that no node has degree greater than 2 or cycle length less than the total number of nodes. Its runtime is slightly longer than that of NN at $\mathcal{O}(N^2 logN)$ however the worst case solution is better than NN.
- **Clarke-Wright (CW):** This algorithm originally comes from a vehicle routing method designed by Clarke and Write[6]. In terms of the TSP, an arbitrary city is chosen as a central point for which the salesman would return to after each visit to another city. A *savings* metric is defined as how much shorter the tour becomes if the salesman skips returning to the central point. A greedy approach is then utilized on these savings values rather than the edge distances. Since this is a different approach of the greedy algorithm, it's runtime is similarly $\mathcal{O}(N^2 logN)$. This method has a higher best performance in comparison to the greedy algorithm however it's worse case performance is the same.

Concorde TSP Solver In the 1990s, a collection of heuristics and functions have been designed by Applegate, Bixby, Chvátal, and Cook [5]. This solver holds the most records for solving large TSP instances with minuscule loss in accuracy. This precision does come at the cost of time, many large TSP problems take over 100-years of CPU time which isn't ideal for applications that require solutions to be obtained swiftly. The applications of the Concorde include: gene-mapping[7], protein function prediction[8], vehicle routing[9], etc. According to [10], the Concorde "is widely regarded as the fastest TSP solver, for large instances, currently in existence."

3.0.4 Meta-heuristics

The time taken to develop heuristics can be heavily time consuming. To remedy this learning process, meta-heuristics were implemented to augment algorithm designer's ability to distinguish the details within the problem instances. Despite this benefit of meta-heuristics, it shifts the problem to properly configuring the algorithm with the correct parameters which can vary from problem to problem. The overall goal of these algorithms is to find a global optimum within the solution space. This is different to regular heuristics that often get trapped in local optimum solutions. Some examples of these methods include: Genetic algorithms, Bee Colony Optimization, Ant Colony Optimization and Particle Swarm Optimization. These solutions introduce a new problem of interpretability of the method, many of these algorithms are considered "black boxes".

It's very difficult to analyse exactly what is happening, because of this there is no guarantee of an approximation factor.

Genetic algorithms There have been many different attempts at solving the travelling salesman problem with genetic algorithms. These methods rely heavily on mutation and crossover operators, these operators are trivial for some problems however for the TSP, a representation of the tour should be defined such that these operators can be developed. Most genetic algorithms designed for the TSP are using path representation where the cities are put into an ordered list where the first city in the list is the first city to be visited. [11]

3.0.5 Deep Learning

Deep neural networks are powerful frameworks that can take large amounts of data and process it through a series of layers to extract features from the input and produce an output according to what it's trained to learn. In the case of the travelling salesman problem, the input is the elements of the problem instance, the nodes and edges in graphs. It's expected output is edges in the travelling salesman problem that are part of the shortest tour. These neural networks automatically learn features relevant to the problem as well as the mapping from the input to the output. However, this mapping is only valid for an instance from a given domain. One of the issues with training a deep neural network is the large amount of training data required to have an accurate model. To acquire such training data, that would involve solving NP-hard problems itself. The way to combat this is to train the network on small instances that are relatively easier to collect training data for as patterns observed in smaller instances remain true for larger instances. [12] used a pointer networks to solve the travelling salesman problem, they trained on smaller instances containing 5-20 points. They observed almost perfect results for 25 points, good results for 30 points and the accuracy decreased dramatically for 40 and above. The more glaring issue with deep learning is the interpretability of the method, with large amounts of weights and biases, it becomes very difficult to analyse the solution. The method is not flexible either, adding new constraints or a change to the domain can have dramatic effects on the solution.

Deep reinforcement learning The use of an *on-policy* techniques using neural networks was applied in [13] in a framework called GCOMB. Here they explore graph embedding techniques using *Graph Convolutional Networks (GCN)* which are then fed into a neural network to learn a *Q*-function in order to predict a solution. This solution would come in the form of an unordered or ordered set of nodes, depending on the problem. The advantages of this implementation include:

- **Scalability:** The proposed framework in the above study is able to scale to networks with millions of nodes and billions of edges.
- **Generalizability:** GCOMB's framework allows itself to be applied to many combinatorial problems rather than focusing on one specific problem.

However these advantages do come with the trade-off of interpretability, it becomes difficult to narrow down how the solution came to be with such complex frameworks. We will attempt to remedy this issue in this paper with our proposed framework.

3.0.6 Machine Learning

Reinforcement learning for Combinatorial optimization Reinforcement learning (RL) is used as a natural framework for learning the evaluation function in [14]. An *off-policy* RL algorithm such as *Q*-Learning was utilized here which updates its rules on the Q-Value rather than looking at

past examples to learn. This technique was used over graph problems such as: Minimum Vertex Cut (MVC), Maximum Cut (MAXCUT), and the Travelling Salesman Problem (TSP). This type of approach lends itself to designing greedy heuristics for difficult combinatorial optimization problems. Heuristics as we know are commonly fast but in the area of optimization require certain knowledge about the underlying problem, this approach attempts to build these heuristics whilst learning about the underlying problem.

Chapter 4: Data Considerations

The data is comprised of locations of pubs within the UK scraped from https://www.pubsgalore.co.uk/, a site containing addresses of all the pubs within the UK. The gathered data will include the name of each pub, a longitude and latitude pair, and if the pub is open or closed. Upon inspection there is roughly 70000+ pubs. The data will be static and locally stored. We are using this as our primary source of data as it is regularly kept up to date and has very few missing values.

There is plenty of cleaning and pre-processing to be completed on the data. Converting the points to an edge list of a complete graph such that each point has an edge to every other point, calculating the weights of each edge (Euclidian distance between points), creating features for the edges and generating the ground truth for each edge.

We will be using the Concorde TSP Solver [5] to build a tour from our dataset. This will provide us with the edges that are in the solution set and will help create our target for each instance in the original data. All the data used in this paper will be made publicly available. Any data used in this project was fairly gathered and raises no ethical concerns.

Chapter 5: Outline of Approach

In this paper, we will attempt to use a simple binary classification model to learn whether an edge is part of the TSP tour or not. The advantage to this approach is the interpretability and scalability of this model over more complex frameworks such has deep learning.

We will be training several classifiers such as:

- Random Forest: Each tree within the forest will be trained on different types of graph structures (i.e. big cities, islands, rural areas, etc.). Majority voting from these trees will make calculate whether an edge is contained within the tour.
- Naïve Bayes
- Linear Regression

The other classifiers will be used to compare the performance of the Random Forest algorithm. Start with small subsets of the data (i.e. 1000 points), as the algorithm improves, we will increase the input size until we are able to fully classify the entire dataset. We hope to prune the dataset and remove as many edges that are guaranteed not to be contained within the tour.

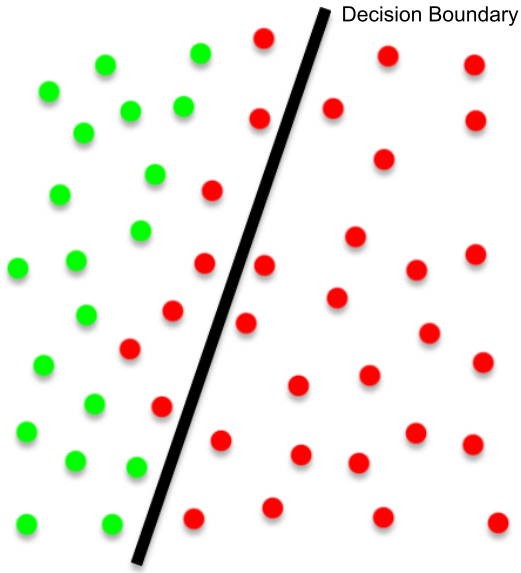

Figure 5.1: Decision boundary created by classifier

As seen in 5.1, these classifiers will create a decision boundary on the edges. Using repeated pruning, we can limit our data to edges that are part of the solution with few misclassified edges. After reducing the problem space, We can then execute an approximation algorithm on this smaller dataset in turn providing close to an approximate solution in significantly less time.

Chapter 6: **Project Workplan**

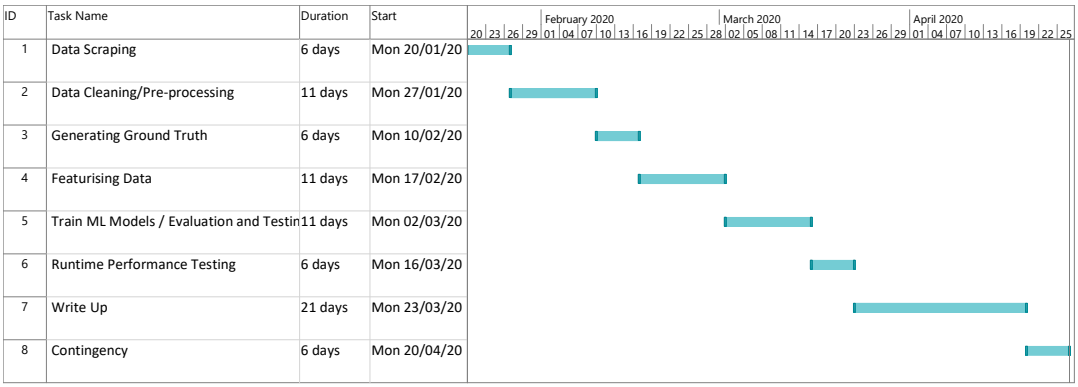

Figure 6.1: Project Workplan

- **Data Scraping:**

  Writing the script to collect the data.

- **Data Cleaning/Pre-processing:**

  Cleaning the data and reformatting it into an edge list rather than a list of coordinates.

- **Generating Ground Truth:**

  Finding a close to optimal solution through other methods to generate the *target* column in the data.

- **Characterizing Data:**

  Finding features that will capture details in the edges to aid in the classification process.

- **Train Machine Learning Models/Evaluation and Testing:**

  Training, paramaterizing, evaluating and testing the machine learning models.

- **Write Up:**

  Experimental write up.

- **Contingency:**

  I've allowed roughly one week for contingency for problems that will arise while carrying out this project. I believe this is an adequate amount of time tackle the upcoming problems.

Chapter 7: **Summary and Conclusions**

Acknowledgements

In your Acknowledgements section, give credit to all the people who helped you in your project.

Bibliography

1. Menger, K. Das botenproblem. *Ergebnisse eines Mathematischen Kolloquiums 2,* 11–12 (1932).
2. *Combinatorial Optimization and Applications* (eds Du, D.-Z., Hu, X. & Pardalos, P. M.) https://doi.org/10.1007/978-3-642-02026-1 (Springer Berlin Heidelberg, 2009).
3. Bellman, R. Dynamic Programming Treatment of the Travelling Salesman Problem. *J. ACM* **9,** 61–63. ISSN: 0004-5411. http://doi.acm.org/10.1145/321105.321111 (Jan. 1962).
4. Johnson, D. S. & McGeoch, L. A. The traveling salesman problem: A case study in local optimization (eds Aarts, E. H. L. & Lenstra, J. K.) (1997).
5. Applegate, D. L. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)* ISBN: 0691129932. https://www.xarg.org/ref/a/0691129932/ (Princeton University Press, Feb. 2007).
6. Clarke, G. & Wright, J. W. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Oper. Res.* **12,** 568–581. ISSN: 0030-364X. http://dx.doi.org/10.1287/opre.12.4.568 (Aug. 1964).
7. Hitte, C. *et al.* Comparison of MultiMap and TSP/CONCORDE for Constructing Radiation Hybrid Maps. *Journal of Heredity* **94,** 9–13. ISSN: 0022-1503. https://doi.org/10.1093/jhered/esg012 (Jan. 2003).
8. Johnson, O. & Liu, J. A traveling salesman approach for predicting protein functions. *Source Code for Biology and Medicine* **1,** 3. ISSN: 1751-0473. https://doi.org/10.1186/1751-0473-1-3 (2006).
9. Applegate, D., Cook, W., Dash, S. & Rohe, A. Solution of a Min-Max Vehicle Routing Problem. *INFORMS Journal on Computing* **14,** 132–143. https://doi.org/10.1287/ijoc.14.2.132.118 (2002).
10. Mulder, S. A. & Wunsch, D. C. Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. *Neural Networks* **16.** Advances in Neural Networks Research: IJCNN '03, 827–832. ISSN: 0893-6080. http://www.sciencedirect.com/science/article/pii/S0893608003001308 (2003).
11. Larrañaga, P., Kuijpers, C., Murga, R., Inza, I. & Dizdarevic, S. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review* **13,** 129–170. ISSN: 1573-7462. https://doi.org/10.1023/A:1006529012972 (Apr. 1999).
12. Vinyals, O., Fortunato, M. & Jaitly, N. in *Advances in Neural Information Processing Systems 28* (eds Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M. & Garnett, R.) 2692–2700 (Curran Associates, Inc., 2015). http://papers.nips.cc/paper/5866-pointer-networks.pdf.
13. Mittal, A., Dhawan, A., Medya, S., Ranu, S. & Singh, A. Learning Heuristics over Large Graphs via Deep Reinforcement Learning (Mar. 2019).
14. Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B. & Song, L. Learning Combinatorial Optimization Algorithms over Graphs. *CoRR* **abs/1704.01665.** arXiv: 1704.01665. http://arxiv.org/abs/1704.01665 (2017).