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 1, 2019

Table of Contents

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**

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 [2], 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.

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 [3], 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[4]. 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.
- **Christofides:** The Christofides algorithm is an approximation algorithm that 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 the other constructive heuristics due to the perfect matching algorithm complexity.

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[6], protein function prediction[7], vehicle routing[8], etc. According to [9], the Concorde "is widely regarded as the fastest TSP solver, for large instances, currently in existence."

Reinforcement learning for Combinatorial optimization Reinforcement learning (RL) is used as a natural framework for learning the evaluation function in [10]. 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.

Deep reinforcement learning The use of an *on-policy* technique using neural networks was applied in [11] 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.

Meta-heuristics The 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.

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.

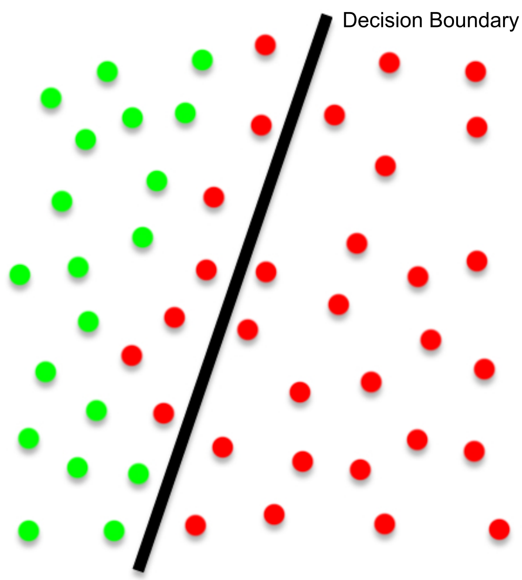
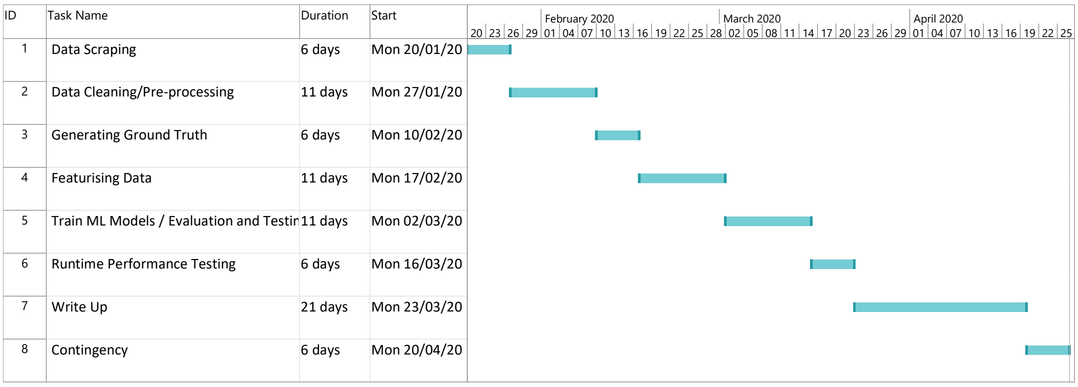


Figure 5.1: Decision boundary created by classifier

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**



ID	Task Name	Duration	Start
1	Data Scraping	6 days	Mon 20/01/20
2	Data Cleaning/Pre-processing	11 days	Mon 27/01/20
3	Generating Ground Truth	6 days	Mon 10/02/20
4	Featurising Data	11 days	Mon 17/02/20
5	Train ML Models / Evaluation and Testir	11 days	Mon 02/03/20
6	Runtime Performance Testing	6 days	Mon 16/03/20
7	Write Up	21 days	Mon 23/03/20
8	Contingency	6 days	Mon 20/04/20

Figure 6.1: Project Workplan

- **Data Scraping:**
- **Data Cleaning/Pre-processing:**
- **Generating Ground Truth:**
- **Featurising Data:**
- **Train Machine Learning Models/Evaluation and Testing:**
- **Write Up:**
- **Contingency:**

Chapter 7: **Summary and Conclusions**

In this section you will sum up your report, draw some conclusions about your work so far, and make some general observations about the work to come. You may also use this opportunity to express points of view, or make factual claims, that are more pertinent here than in other sections of the report. If your project raises some ethical concerns, for example about how data or users are treated, then address them here in a thoughtful manner.

Regarding this document, here are some concluding points that you should keep in mind when writing your own. You may use screenshots in your report, but do not overfill your report with them, or with figures of any kind. Make sure that figures earn their keep, and are not just present as space fillers or as eye candy. If you use diagrams or figures from other people's work, including the web, be sure to cite the creator in the corresponding caption. All things being equal, it is better to construct your own figures than to copy and paste those of others. In any case, always make sure that your images are readable, do not suffer from pixelation or aliasing effects, and that each is clearly numbered, captioned and meaningfully referenced in the main body of the text.

Ensure that there is a cohesive argument expressed in the text of the report and that it is not simply a bag of diagrams, screenshots and wishful thinking. Every report should tell a story, so know what story you want to tell. When you include images, make sure they are readable and truly add to the discussion.

Make sure your language is professional throughout, and steer a course between pompous and colloquial. Maintain authorial distance and do not overuse "me," "I" and "our." Your are writing for a professional audience who will judge you on the quality of your prose, so use a grammar and a spelling checker.

Use LaTeX if you wish – this is recommended if you plan to use mathematical formulae in your report, but in any case, keep the general spacing and font/style you find here (Single or 1.5 spacing, 12 pt. font for text, etc.). Be sure to submit a PDF (never a .DOC or .DOCX file) as your report. If you prepare your report in MS Word, as this document has been, save it as a PDF before you submit it. Overall it should be about 18 – 20 pages, including figures, front matter and references, A significant portion of the report will be textual, with approx.. five or six thousand words. Do not rely on images or other filler to write your report for you. The dates and means of submission will be communicated to you separately.

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. 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).
3. 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).
4. 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).
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. 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).
7. 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).
8. 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).
9. 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).
10. 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).
11. Mittal, A., Dhawan, A., Medya, S., Ranu, S. & Singh, A. Learning Heuristics over Large Graphs via Deep Reinforcement Learning (Mar. 2019).