An Open-source System for Travelling Salesman Optimisations

Daniel John Ellis
School of Computing, University of Portsmouth, United Kingdom

Oliver Peters
School of Computing, University of Portsmouth, United Kingdom

Shadille Samuels
School of Computing, University of Portsmouth, United Kingdom

Glanyell White
School of Computing, University of Portsmouth, United Kingdom

Rich Boakes
School of Computing, University of Portsmouth, United Kingdom

Matt Dennis
School of Computing, University of Portsmouth, United Kingdom

Jacek Kopecký
School of Computing, University of Portsmouth, United Kingdom

Abstract

This paper introduces the idea of an open-source repository for solutions to the travelling salesman problem, discusses some preliminary results that have been gathered, and describes the overall aims that this project hopes to achieve.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Combinatorial optimization

Keywords and phrases Travelling Salesman Problem, Tour Construction, Heuristic

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements We want to thank ...

1 Introduction

1.1 What the Travelling Salesman Problem is

The Travelling Salesman Problem (TSP) is one of the most widely studied problems in combinatorial optimisation [12, 13, 30]. The problem asks the following question [29, p. 52]:

> Given a list of cities, and the distances between each pair of cities, what is the shortest route that can be taken to visit each city exactly once, and then return to the starting city?

The general form of the problem was seemingly first studied in the 1930s after mathematician Karl Menger invited researchers to consider the problem from a mathematical perspective [18, p. V], however the problem had already accrued some interest prior to this. One occurrence of TSP was found in the 1850s with William Hamilton's creation of *The Traveller's Dodecahedron* which was a puzzle that invited people to try and create a complete circuit around each of the pegs on the provided dodecahedron [10, p. 35].

1.2 Applications of TSP

The applications of the Travelling Salesman Problem are widespread. Direct uses include circuit board drilling, X-ray crystallography and VLSI fabrication [11] [25]. Logistical usages

such as mapping, route planning and crane control are also included. Since the TSP is an optimization problem, for which the best solution is the shortest route, this means that improvements upon the given solutions may have knock-on effects to many applications [27]. Some applications would prefer a quicker calculation with an estimated short route, others would prefer a longer calculation to obtain a more optimal route.

This wealth of applications come from problems of the TSP's genre: Combinatorial Optimization. When a single machine has to process many jobs, one at a time, optimization occurs that may often be an instance of TSP. Branch and bound, a well-known computing method, was used contextually in the TSP first [15].

Perhaps another reason the TSP generates such interest too, is the intellectual challenge given. Much like how chess has a staggering number of possible moves, so does the TSP when a large count of cities are involved. The TSP with n cities has (n-1)!/2 possible tours. A 7397-city count contains over 1,025,000. [21]. An example of this interest can be seen in the recent 2020 breakthrough of a metric TSP. Although the efficiency subtracted is only 0.2 billionth of a trillionth of a trillionth, this was the first breakthrough in 44 years [23]. This shows how attractive the TSP is as a problem to optimize to its highest potential.

1.3 Crowdsourcing Problems

Crowdsourcing information has proved to be an effective way of gathering large quantities of data, as well as solving mathematical problems in a timely manner. An example of this lies with Terence Tao and his solution to the Erdős discrepancy[31] which was based on a conjecture made by Paul Erdős around 1932 that stated [17, p. 469]:

"Let $f(n) = \pm 1$. Is it true that for every c there is a d and an m so that $|\sum_{k=1}^{m} f(kd)| > c$?"

In a post on his blog in 2015 [32], Terence Tao shared two new papers that he had posted to arXiv, one of which was the proof for the aforementioned Erdős discrepancy problem. This proof had eluded mathematicians for some 80 years, and the proof that Tao formed was a result of his own abilities and the assistance from Polymath5 contributors [22].

Another example of crowdsourcing being used in mathematics is the Zooniverse platform [5]. Created in 2009, Zooniverse was launched after the success of a two year long project, *Galaxy Zoo*, which recruited volunteers to classify galaxies. Since then, Zooniverse has enabled users to create their own projects to allow volunteers to help solve problems such as classification [3] and transcription [2], often with the aim to improve the capabilities of machine learning. Many people were interested because they wanted to be a part of something bigger than themselves. Due to the large number of contributions, projects that were expected to take decades to complete are being completed in a matter of months. In just 14 months, Galaxy Zoo 2 users helped to classify over 60 million images [1]. Since their launch, Zooniverse have almost 400 publications relating to projects that they've hosted [4].

Waze, a navigation application that uses data from users to assist give more accurate and efficient solutions, is a well-known example of crowdsourcing. As a result, it has grown in popularity as users place more trust in their knowledge of the routes, which allows them to be aware of any hazards or traffic because the data is regularly updated. Due to the recent increase in oil costs, things like updating petrol prices are also beneficial to users.

2 Existing Heuristics

2.1 Genetic algorithms

A genetic algorithm is a type of nature-inspired algorithm that was first detailed by Alan Turing in 1950 [33, p. 456]. The basic premise is that, rather than attempting to create a program that simulates how adults solve a problem, we instead simulate how a child might attempt to solve it, and then we teach the program to improve itself through simulated evolution. Research involving solving the TSP with a Genetic Algorithm has already been conducted extensively in academia [26] [19] [28]. Genetic Algorithms come in a range of methods: binary, path, adjacency, ordinal, matrix representations and more. It is possible that any of these methods mentioned could be improved in conjunction with an optimisation method.

2.2 Nearest Neighbour

The Nearest Neighbour (NN) algorithm works by making any arbitrary node i the current tail of the tour and appending the node j closest to it to the tour. j is then set as the current tail, and the process repeats until a full tour is created. In every iteration, the current tail cannot be set to a node already contained in the tour, as this would create a closed cycle, which is prohibited. A downside to this algorithm is that the worst case of the algorithm results in a tour that is much longer than the optimal tour [20]. It's even possible that the algorithm may not find a feasible tour, even when one exists.

2.2.1 Double-Ended Nearest Neighbour

The Double-Ended Nearest Neighbour (DENN) algorithm works similarly to the NN algorithm, however whilst NN grows the tour from only one end, DENN allows the tour to grow from both ends. Research has been conducted on NN, DENN and Hybrid usage in academia and is proved to be an efficient algorithm [24].

2.3 Multiple Fragment

The Multiple Fragment heuristic is a form of greedy heuristic algorithm that finds a solution to the Travelling Salesman Problem by creating multiple partial tours and then combining them together when possible, to create a final tour [8, p. 97]. The name comes from the fact that each of these partial tours are considered as *fragments* of the final tour [7, p. 390].

The algorithm works by looking through all the edges in a network in ascending order by weight, and then evaluating every edge against a set of rules before deciding whether the edge should be discarded, added to a fragment, or if it should form the beginning of a new fragment. The rules to decide this are as follows [8, p. 97]:

1. If one of the edge's nodes appears in the middle of a fragment, the edge is discarded.
2. If neither of the edge's nodes appear in any existing fragments, then a new fragment is created from the edge.
3. If one of the edge's nodes appears at the end of another fragment (**A**), and doesn't appear on the end of any other fragment, then the edge gets appended to fragment **A**.
4. If one of the edge's nodes appears at the end of a fragment (**A**) and the other node appears at the end of a fragment (**B**), then the fragments can be combined together along with the edge to form a longer fragment (**C**), providing the larger fragment **C** would not form a closed tour which doesn't include all nodes in the network.

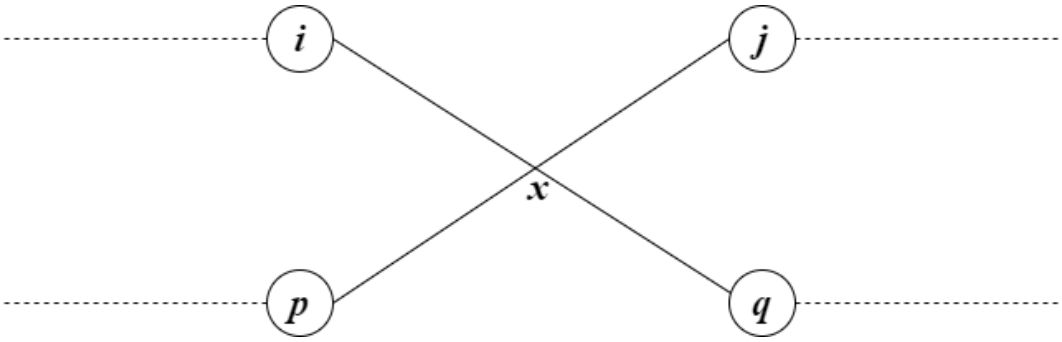

Figure 1 Pair of edges crossing

3 Existing Optimisations

3.1 2-opt

It is a well established fact that if a Hamiltonian cycle in Euclidean space has any edges which cross over each other, then the cycle is not the shortest cycle possible for the network [14, p. 78][6, p. 24]. If you consider the setup shown in figure 1, we have edges (i, q) and (p, j). If we consider the point at which they cross each other x, we can imagine two triangles $\triangle(i, x, j)$ and $\triangle(p, x, q)$, because of the triangle inequality, we know that the distance $\overrightarrow{ixj}$ is greater than or equal to the distance $\overrightarrow{ij}$, and the same is true for the distance $\overrightarrow{pxq}$ being greater than or equal to the distance $\overrightarrow{pq}$. Because of this, we know that to shorten the cycle, we should remove the edges (i, q) and (p, j) and insert the edges (i, j) and (p, q) instead.

3.1.1 Complete 2-opt

A 2-opt implementation is complete if the optimisation checks every possible valid swap that can be made. This means that a tour can be split into a pair of subtours a and b, these subtours can be recombined as

$$ab,\ a\overline{b},\ \overline{a}b, \text{ and } \overline{a}\overline{b}$$

Where $\overline{a}$ and $\overline{b}$ are the reversed subtours a and b respectively.

Each newly combined tour can then be compared to the original tour, and if a new tour is better than the original tour, then that original tour is replaced with the new tour, and the process continues.

This process of splitting the tour into subtours and evaluating them happens for every pair of subtours that can be made up of the original tour, which is to say that every pair of edges will be removed at some point and the resulting subtours will be tested.

In a symmetrical TSP, only two of these combinations need to be considered, because

$$ab = \overline{a}\overline{b}\ \&\ a\overline{b} = \overline{a}b$$

3.2 3-opt

3-opt is similar to 2-opt, as it requires the removal of three edges, and then the recombination of the three resulting subtours in unique ways. For a symmetrical TSP, there are a total of 8 unique ways to recombine the tour, including the initial configuration. If we consider the tour with three edges removed to form three subtours a, b, and c, then the tour can be recombined as:

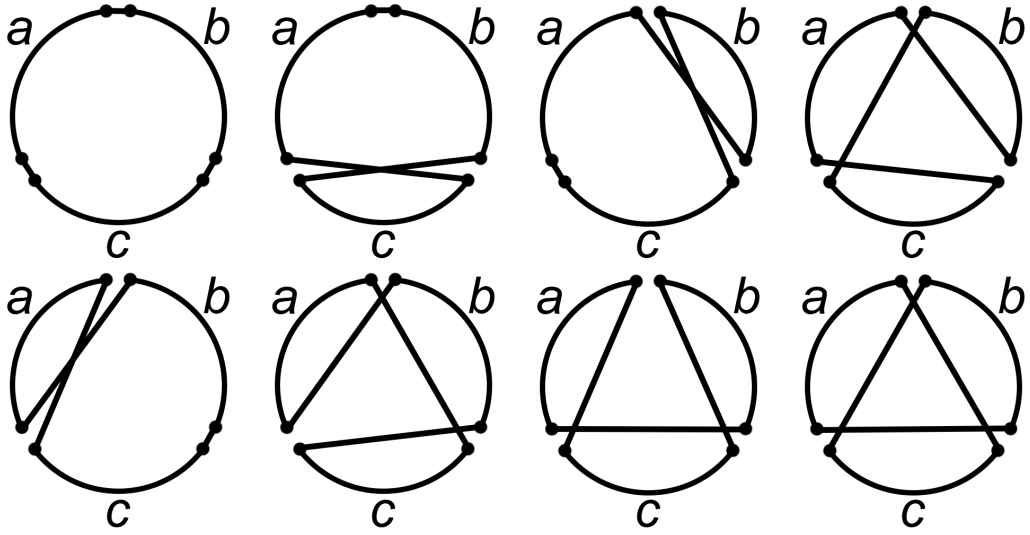

Figure 2 3-opt combinations in order from left to right, top to bottom: abc, $ab\overline{c}$, $a\overline{b}c$, $a\overline{b}\overline{c}$, $\overline{a}bc$, $\overline{a}b\overline{c}$, $\overline{a}\overline{b}c$, and $\overline{a}\overline{b}\overline{c}$

Listing 1 Node swapping algorithm.

```
For i := 0 to length(T) do
    // Assume index -1 is last element in list
    temp = T.swap(i - 1, i);
    If weight(temp) < weight(T) then
        T = temp;
end
```

abc, $ab\overline{c}$, $a\overline{b}c$, $a\overline{b}\overline{c}$, $\overline{a}bc$, $\overline{a}b\overline{c}$, $\overline{a}\overline{b}c$, and $\overline{a}\overline{b}\overline{c}$
Where $\overline{a}$, $\overline{b}$, $\overline{c}$ are the reversed subtours a, b, c respectively (Fig. 2).

3-opt should, in the same way as 2-opt, be applied to every possible trio of subtours that can be made from the original tour, and if one of the new tours is an improvement on the original tour, then the original tour is replaced with the newly found tour.

It is also worth noting, that in performing 3-opt optimisation, 2-opt is also carried out, as the tours $ab\overline{c}$, $a\overline{b}c$, and $\overline{a}bc$ are each equivalent to single 2-opt moves.

3.3 Node swapping

Perhaps the simplest method of improving a tour, is by iterating through every node of the tour and checking if swapping the node with the next one in the tour would decrease the overall weight.

As can be seen in listing 1, this algorithm is incredibly simple, and has a complexity of only $\mathcal{O}(n)$. This simple layout however yields weak results, as can be seen in our comparisons in appendix A.

3.4 Ant colony optimisation

Ant colony optimisation is an algorithm that is not unique to the travelling salesman problem. It's origins are based on real life ant colonies. These ants find the shortest path from a food

source to their nest without any visual cues [16, p. 53]. For the TSP, the pheromone trails update in two stages. First, where the ants begin their tours of the cities, starting from an initialization rule (e.g. randomly) and lastly, the pheromone is updated when all ants have finished their tour [9].

4 The Aim

Because of the many practical applications of the Travelling Salesman Problem throughout different industries, the need to develop faster and more accurate solutions is become increasingly necessary. Whilst most heuristic solutions can be applied to both the Euclidean and non-Euclidean forms of the problem with ease, many of the common optimisations used require the problem to be Euclidean for them to apply.

Our aim is to use crowdsourcing to fuel an open-source project which will combine different heuristic solutions to improve upon existing methods of solving TSP. This is because, the time and resources to dedicate to exploring one algorithm with a meaningful number of nodes, is significant. By crowdsourcing, we invite those focused on a specific algorithm, or have an abundance of computing power to generate data which we cannot. These results can then be collected and aggregated to a dataset.

4.1 The idea

The aim of this project is to develop a means to optimise heuristics for symmetric non-Euclidean TSP, initially by testing different combinations of heuristic algorithms to see if pairs of heuristics, applied correctly, can be used to gain better solutions. We will present this data such that the results are comparable. We seek to demonstrate comparing the average improvement upon the MF algorithm, although future collaboration would build upon any algorithm specified.

The aim of this project is to develop a means to improve solutions to the non-Euclidean form of TSP by combining known heuristics together in order to improve the resulting tours. As mentioned in section 1.2, in 2020 a group of computer scientists made a breakthrough with a form of the metric TSP where they found a better heuristic, which improves upon what had previously been the best heuristic for over four decades [23]. We hope that this breakthrough can reinvigorate the energy directed towards TSP such that more improvements can be found in time.

4.2 Why crowdsource

We believe that given the success of Polymath5, Zooniverse, and many others, the key to making progress with TSP is to collaborate and crowdsource in such a way that people have a platform to pitch their ideas and see how they compare to others. Having more minds solving the problem will allow for more solutions to be tested to open up the doors to the development of better heuristics.

4.3 The groundwork

We have set up a GitHub repository at https://github.com/UP940148/TSP_solutions which holds our existing work, in which we have had some initial success by combining The Nearest Neighbour, Double-Ended Nearest Neighbour, and Multiple Fragment algorithms alongside Node Swap, 2-Opt, and 3-Opt optimisation methods on our generated networks.

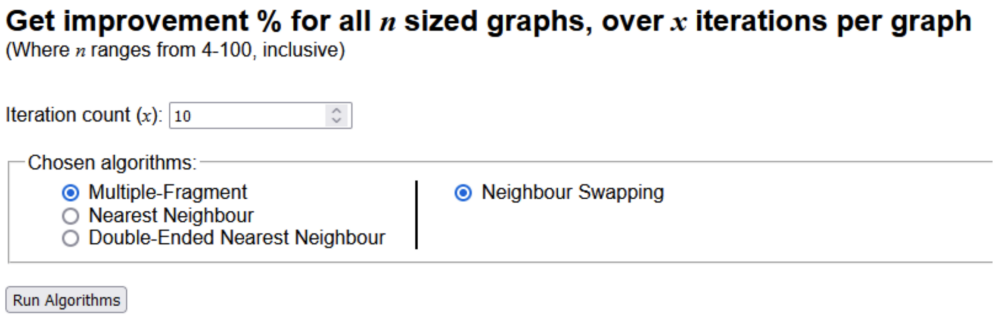

Figure 3 Initial artefact screen

All of our findings so far are available in the aforementioned repository, with the results currently hosted at https://up940148.github.io/TSP_solutions/frontend. Some of the most promising results so far include:

- In one graph of 100 nodes, Double-Ended Nearest Neighbour (DENN) with the Node Swap optimisation was improved using the Multiple Fragment (MF) algorithm before having the resulting tour improved with 3-Opt optimisation. This yielded a tour weight of 2123.29, whilst MF and 3-Opt alone yielded a weight of 2395.67 and DENN with Node Swap yielded a tour of 4602.11.
- In a graph of 90 nodes, MF with 2-Opt optimisation was improved with MF again, and then the final tour was improved with 3-Opt. This yielded a tour of weight 2607.93, whilst MF with 3-Opt yielded a weight of 3000.68 and MF with 2-Opt yielded a weight of 3572.66 when applied to the same graph.

(Tour weights have been rounded to 2 decimal places)

These results were gathered by first applying a heuristic to a network, then optimising the tour using one of the optimisation methods listed, before employing a technique similar to 2-opt, where every possible pair of subtours is considered, and the nodes that make up those subtours create a new, smaller network which is then has the second algorithm applied to it. If the second algorithm finds a way that the nodes can be re-ordered such that the subtours weight is now less than it was before, then the new ordering is used in place of the subtour and the full tour can be recombined, before being optimised with the second optimisation strategy.

5 Artefact Structure

In order to evaluate the success of different combinations of algorithms at finding improved solutions, a testing artefact was created which could run combinations of algorithms on different tours. The initial artefact created could run the chosen algorithms on x randomly generated graphs made up of n nodes where $4 \leq n \leq 100$ (Fig. 3).

The issue found with this initial format was that the algorithms could never be directly compared with one another as every run generated new random graphs, and so multiple algorithms could not be tested on the same graph to determine which one produced the best tour.

The artefact was revisited and a new format has been laid out. This features a JSON file containing unique tours, spanning from 5 nodes to 100 nodes, in increments of 5, with two unique tours for every graph size, totalling 40 nodes overall. Each graph is laid out with an

Listing 2 JSON structure of graph with 5 nodes.

```
{
    data: [
        [null, 377.02, 097.90, 524.67, 958.42],
        [377.02, null, 340.97, 650.37, 363.74],
        [097.90, 340.97, null, 167.97, 519.86],
        [524.67, 650.37, 167.97, null, 864.12],
        [958.42, 363.74, 519.86, 864.12, null]
    ],
    attempts: [
        {weight: 1870.75, algorithm: 'multiFrag'},
        {weight: 2238.4, algorithm: 'nearestN'},
        {...},
        ...
    ]
}
```

attribute called `data` which contains the graph's distance matrix, and an attribute `attempts` which is an array of algorithm combinations and their resulting tour weights (Listing 2).

The structure laid out has been chosen because it's believed that it can be easily modified to add new graphs, new algorithms can be implemented and included, and the structure can be recreated to solve forms of TSP beyond just symmetric, non-Euclidean.

5.1 Looking to the future

With this structure created, going forward we hope that those interested in optimising the TSP can implement their own new algorithms, use greater computing power and be able to compare and contrast results found. Not only do we think this could be used to find faster and better combinations, but we also hope that the work done over time can serve as a large dataset of TSP networks that can be used for benchmarking future progress. Those interested in this project will be able to generate meaningful data for future optimisations and algorithm comparisons.

Our current testing has been done by generating subtours in the same way that 2-Opt does, and then applying a new algorithm to the subtours to see if any improvements are made. This is not very efficient and so future work is going to be done into calculating which sections of a tour need to be improved so that they can be isolated and improved by themselves. Initial research is beginning by searching the tour for any subtours which have a weight greater than is expected of them based on the mean weight of an edge in the main tour, and using this method to recalculate subtours.

References

1 Case study: Galaxy zoo. URL: https://www.ox.ac.uk/public-affairs/media-coverage/media-guidance/case-studies/case-study-galaxy-zoo#:~:text=In%20the%2014%20months%20the%20site%20was%20up%20Galaxy%20Zoo,make%20over%2060%20million%20classifications.

2 Davy Notebooks Project. URL: https://www.zooniverse.org/projects/humphrydavy/davy-notebooks-project.

3 Dental Disease Detection. URL: https://www.zooniverse.org/projects/huhui/dental-disease-detection.

4 Publications. URL: https://www.zooniverse.org/about/publications.

5 Zooniverse. URL: https://www.zooniverse.org/.

6 David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2007.

7 John Jouis Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):390–392, 1992. doi:10.1287/ijoc.4.4.387.

8 Jon Louis Bentley. Experiments on traveling salesman heuristics. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, page 91–99, USA, 1990. Society for Industrial and Applied Mathematics.

9 Leonora Bianchi, Luca Maria Gambardella, and Marco Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In *International Conference on Parallel Problem Solving from Nature*, volume 2439, pages 883–892. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45712-7_85.

10 Norman L. Biggs, E. Keith Lloyd, and Robin J. Wilson. *Graph Theory 1736-1936*. Clarendon Press, 1996.

11 Robert G Bland and David F Shallcross. Large travelling salesman problems arising from experiments in x-ray crystallography: A preliminary report on computation. *Operations Research Letters*, 8(3):125–128, 1989. URL: https://www.sciencedirect.com/science/article/pii/0167637789900370, doi:https://doi.org/10.1016/0167-6377(89)90037-0.

12 Connor Chase, Harrison Chen, Alex Neoh, and Maximum Wilder-Smith. An evaluation of the traveling salesman problem, 2020. URL: http://hdl.handle.net/20.500.12680/8g84mp499.

13 Kenneth Steiglitz Christos H. Papadimitriou. *Combinatorial Optimization: Alogrithms and Complexity*. Dover Publications, 1982.

14 William J. Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2011. doi:10.1515/9781400839599.

15 G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954. URL: http://www.jstor.org/stable/166695.

16 M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997. doi:10.1109/4235.585892.

17 Paul Erdős. *Some of my favourite unsolved problems*. Cambridge University Press, 1990. doi:10.1017/CBO9780511983917.039.

18 Federico Greco. *Traveling Salesman Problem*. InTech, 2008.

19 John Grefenstette, Rajeev Gopal, Brian Rosmaita, and Dirk Van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, volume 160, pages 160–168. Lawrence Erlbaum, 1985.

20 Gregory Gutin, Anders Yeo, and Alexey Zverovich. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp. *Discrete Applied Mathematics*, 117(1):81–86, 2002. URL: https://www.sciencedirect.com/science/article/pii/S0166218X01001950, doi:https://doi.org/10.1016/S0166-218X(01)00195-0.

21 Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. URL: https://www.sciencedirect.com/science/article/pii/S0377221799002842, doi:https://doi.org/10.1016/S0377-2217(99)00284-2.

22 Gili Kalai. The Erdős discrepancy problem has been solved by Terence Tao. URL: https://polymathprojects.org/2015/09/22/the-erdos-discrepancy-problem-has-been-solved-by-tao/.

23 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, page 32–45. Association for Computing Machinery, 2021. doi:10.1145/3406325.3451009.

24 Gözde Kizilateş and Fidan Nuriyeva. On the nearest neighbor algorithms for the traveling salesman problem. In *Advances in Computational Science, Engineering and Information Technology*, pages 111–118. Springer, 2013.

25 B Korte. Applications of combinatorial optimization. In *talk at the 13th International Mathematical Programming Symposium, Tokyo*, 1988.

26 Pedro Larranaga, Cindy M. H. Kuijpers, Roberto H. Murga, Inaki Inza, and Sejla Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial intelligence review*, 13(2):129–170, 1999.

27 John D. Litke. An improved solution to the traveling salesman problem with thousands of nodes. *Commun. ACM*, 27(12):1227–1236, dec 1984. doi:10.1145/2135.2141.

28 Jean-Yves Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63(3):337–370, 1996.

29 Alexander Schrijver. On the history of combinatorial optimization (till 1960). In K. Aardal, G.L. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 1–68. Elsevier, 2005. doi:10.1016/S0927-0507(05)12001-5.

30 Kenneth Steiglitz and Peter Weiner. Some improved algorithms for computer solution of the travelling salesman problem. In *6th Annual Allerton Conference on Circuit and Systems Theory*, 1968.

31 Terence Tao. The Erdős discrepancy problem. *Discrete Analysis*, pages 1–29. doi:10.19086/da.609.

32 Terence Tao. The logarithmically averaged Chowla and Elliott conjectures for two-point correlations; the Erdos discrepancy problem. URL: https://terrytao.wordpress.com/2015/09/18/the-logarithmically-averaged-chowla-and-elliott-conjectures-for-two-point-correlations-the-erdos-

33 Alan Mathison Turing. COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 1950. arXiv:https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf, doi:10.1093/mind/LIX.236.433.

A Comparisons

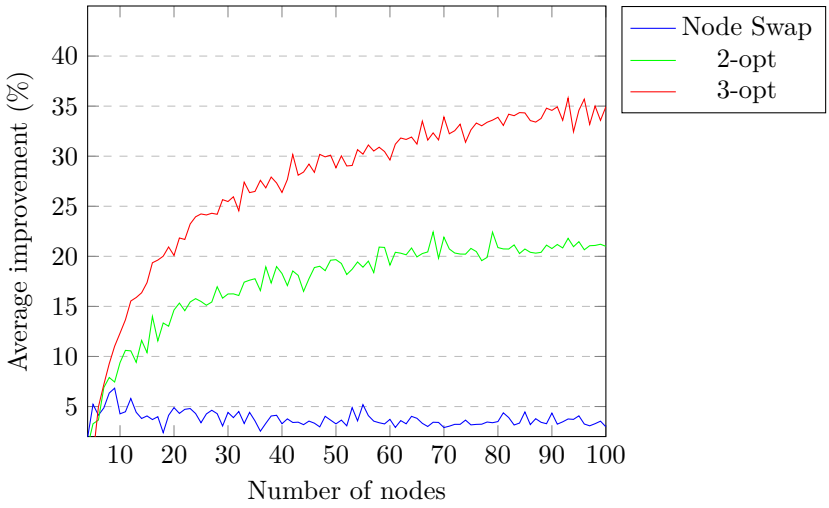

Figure 4 Average tour improvement of MF algorithm after different optimisations.

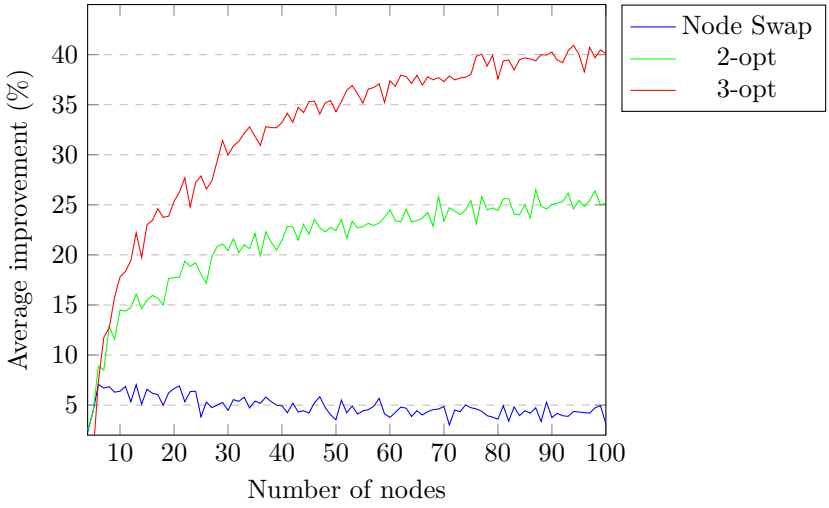

Figure 5 Average tour improvement of NN algorithm after different optimisations.

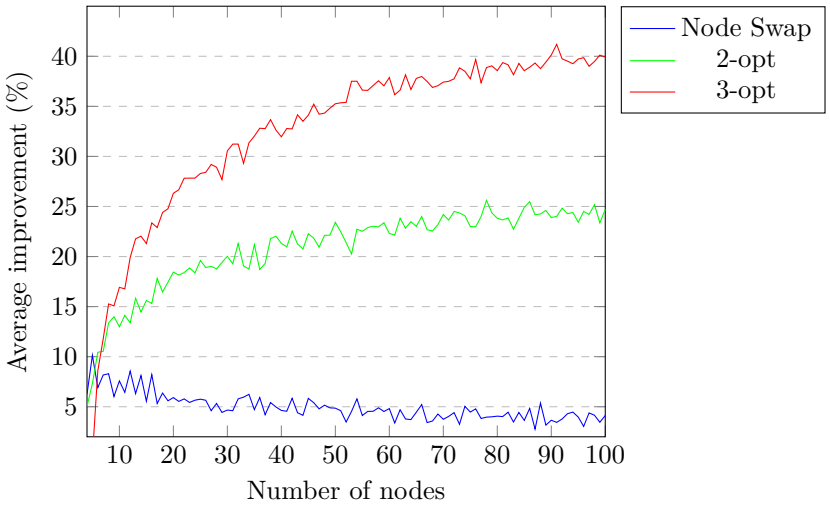

Figure 6 Average tour improvement of DENN algorithm after different optimisations.