How do Modern Exact Algorithms (Branch and Cut) Compare to Approximate Algorithms (Nearest Neighbor with Lin-Kernighan Optimization) in Terms of Time Complexity and Accuracy When Solving the Travelling Salesman Problem?

A Computer Science Extended Essay

Word Count: 4000 words

Contents

1 Introduction

The Travelling Salesman Problem, or TSP for short, has tormented various ingenious attempts by scientists and researchers throughout the last two centuries, splitting them into two groups. First, there are those who chose to sacrifice rigor for practicality in the real world—the leaders of approximate algorithms. Then, there is a set of researchers who have "bashed on regardless" (D. L. Applegate, Bixby, Chvátal, & Cook, 2006) , despite the challenges, and as a result, created implications greater than some salesman. These are the forerunners of the exact algorithms.

1.1 Background Information

The General Travelling Salesman Problem is easy to say but hard to solve:

What is the shortest path from a given starting node to visit every node exactly once in a complete graph before returning to the starting node?

For the data sets to be researched, the files will begin with the value n— the number of nodes before listing the (x, y) coordinates of all the points in the x-y plane.
Concorde is a computer software program made by Cook et al. that incorporates the modern methods of both the approximate and the exact algorithms to solve the TSP.
More definitions are peppered throughout the paper where they make the most sense.

1.1.1 Definitions

The approaches for the TSP can get complex, and thus require solid notation and definitions established.

- A valid set of edges that visits every node except for the starting node once are denoted as a **tour**.
- The exact algorithms are algorithms that can always yield the most optimal answer. The claim of optimality is typically proven rigorously.
- In contrast, the approximate algorithms are algorithms that can return answers close to the optimality with a specified level of accuracy or a margin of error(Ryan, 2003).
- Throughout this paper the variable n will denote the number of nodes in the graph— the overall size of the problem.

1.2 Scope of Research

The specific variation of the Travelling Salesman Problem (TSP) considered in this paper will be the symmetric version. This means that, given nodes A and B, the distance from A to B is equal to the distance from B to A, making the graph undirected.

Additionally, every instance of the TSP will assume a complete graph, meaning there is an edge between every pair of nodes.

Although exact algorithms, such as dynamic programming or brute-force approaches, exist, both are impractical due to their time complexity, which is not feasible even for moderately sized instances (e.g., $n = 50$). Moreover, a detailed implementation of the efficient algorithms for the TSP would be too complex for this paper. For instance, the Concorde program used in this investigation contains over 130,000 lines of code (D. L. Applegate et al., 2006).

Therefore, the main goal of this paper is to appreciate, understand, and develop the theories behind the leading algorithms for the TSP before comparing their efficiencies in terms of accuracy and running them on various instances of the TSP to determine their performance.

1.3 Experimental Overview

The paper will first focus on developing the general ideas behind the Concorde's implementation of the Lin-Kernighan heuristics algorithm, along with its Branch and Cut Linear Programming algorithm. Then, using these algorithms, instances of the TSP will be solved, with 50 random nodes added for each consecutive dataset. The accuracy and average time taken will be analyzed to answer the question: **How do Modern Exact Algorithms (Branch and Cut) Compare to Approximate Algorithms (Nearest Neighbor with Lin-Kernighan Optimization) in Terms of Time Complexity and Accuracy When Solving the Travelling Salesman Problem?**

2 Research

2.1 Approximate Algorithm

The motivation behind approximate algorithms is clear: With so much research going into exact algorithms and "little" result back, most turn change their focus on practicability, where approximate algorithms reign supreme.

The classes of approximate algorithms are divided into three types:

1. Tour Construction Algorithms
2. Tour Improvement Algorithms

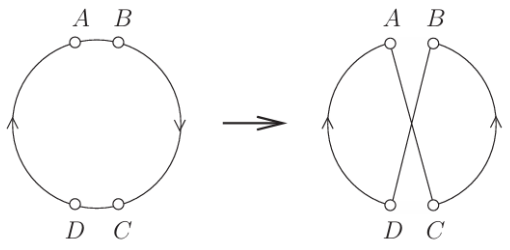


Figure 1: 2-Opt Move taken from (D. L. Applegate et al., 2006)

3. Composite Algorithms

For the Tour Construction Algorithm, the Concorde code by default utilizes the Nearest Neighbour algorithm, which essentially means greedily taking the closest unvisited node on each iteration until a tour is formed.

The *tour improvement* algorithm created by Lin Shen and Brian Kernighan will be the main focus.

2.1.1 2-opt,3-opt,...,k-opt

The main idea behind the algorithm is repeatedly replacing a set of edges in a tour with a cheaper alternative set where possible. (D. L. Applegate et al., 2006).

For instance, a tour that where there is an edge from A to B and from C to D. If the sum of the tour edges made by {A, B} and {C, D} is greater than the sum of costs of the tour edges of {A, C} and {B, D}, the tour can be improved by removing the inital first two sets of edges and replacing them with the last two. This exchange is known as a 2-opt move (Reducible, 2022). To improve the tour, this process is done continuously until it is not possible.

Lin and Kernighan took this a step further by also considering every three pairs of edges—yielding successful results.

Which motivated the pursuit the k-opt—replacing k edges to improve the tour, but the computational costs of even 4-opt deterred further investigation (D. L. Applegate et al., 2006).

2.1.2 Red and Blue Exchange

The algorithm starts by arranging the given tour into a circle— to help with the visualization, but keep in mind that the lengths of edges in Figure 2 are not meant to indicate their travel costs.

The next step is to choose a starting node, then an edge part of the tour that meets the selected node, and a non-tour edge meets the other end of the tour edge. Colour the home node and tour edge *red* and the non-tour edge *blue*, as in 2) in Figure 2. In essence, the red edges are edges that are currently part of the tour, blue if not.

The plan is to remove the red edges and add blue edges, but only if the blue edge is shorter than the red edge. Only then, the colours are swapped. This is akin to a 2-opt move.

The next step, illustrated in the (3) in Figure 2, is to paint the second tour edge red (the one the program just tried deleting), and to consider a blue alternative to the direct route home. This potential 3-opt move is saved if it yields the biggest saving so far.

Additional red-blue pairs are considered— continuing as long as the sum of the blue edges' cost is less than the sum of the red edges' costs. If the end of a line is reached, where it is no longer possible to add another pair of edges, then algorithm backtracks and explores alternative blue candidates at earlier levels. Eventually the process is halted, either due to time considerations or by running out of edges to consider. If the biggest saved move improves the tour, apply it, then start a new home node and repeat.

Essentially, the code considers every pair as a 2-opt algorithm would, but it can go deeper and simulates k-opt if every-time and utilizes backtracking to consider every case.

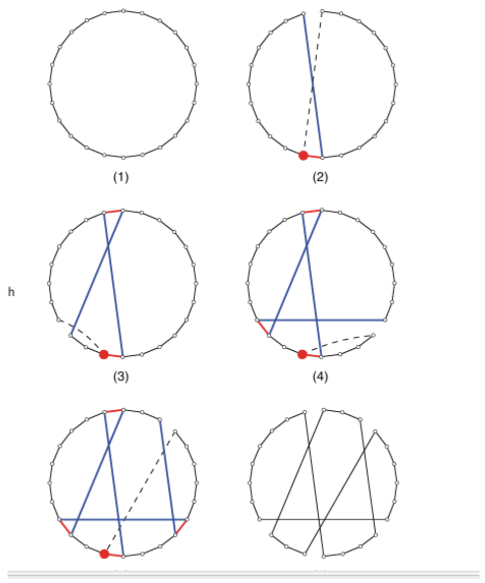


Figure 2: Lin-Kernighan search for a k-opt move. Sourced from (W. J. Cook, 2012)

2.1.3 Chained Lin-Kernighan

One can visualize the TSP tours as 2D valley of hills, in which the peaks of the hills are the local/global maximum tours. Naturally, the tour improvement algorithms from a starting tour act as hill-climbing algorithms until the top of a hill. The issue comes when it climbs a sub-optimal hill. Thus, comes

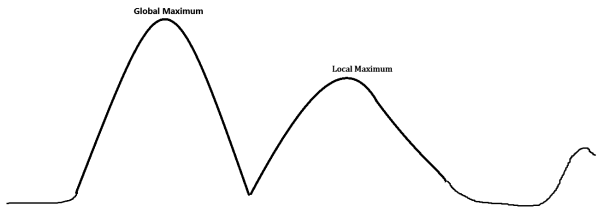


Figure 3: Hill Analogy

the proposal of *kicking* the current tour to another neighbourhood— a permanent action that allows the local-optimal search to assess different hills. A random 4-opt exchange like in 4 does this well (W. J. Cook, 2012). The Concorde program in this experiment by default will implement 1 kick.

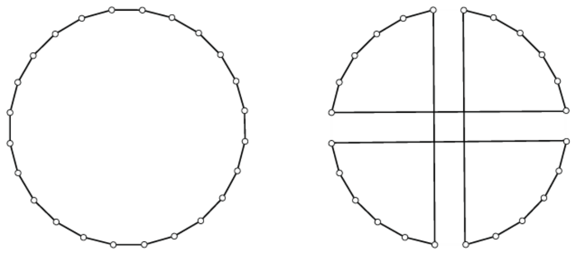


Figure 4: Random 4-opt

2.2 Linear Programming

Linear Programming or LP for short, is a tool utilized in Operational Research to solve **linear** problems—all variables have a degree of 1— that minimize or maximize a linear objective function, in which the variables are subject to a list of linear constraints which are simply valid inequalities or equalities that helps denote the specific context of the problem.
In general, the main idea of solving a linear programming (LP)problem is starting with a huge set that contains possible solutions the bounds/borders of the set defined by the initially set constraints. And slowly decrease the size of this solution set through important algorithms like the **Simplex Algorithm** to help *pivot* towards an answer.

Linear Programming is useful when there are many variables or constraints to consider— commercial LP solvers consider millions of variables to create their given function.

The general form of an LP problem is:

$$\text{maximize } c_1x_1 + c_2x_2 + ... + c_nx_n$$

$$\text{subject to}$$

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m$$

$$l_1 \leq x_1 \leq u_1$$

$$l_2 \leq x_2 \leq u_2$$

$$\vdots$$

$$l_n \leq x_n \leq u_n$$

Where there are m constraints, and n rational input/decision variables. The values of $l_1, l_2, ..., l_n$ and $u_1, u_2, ..., u_n$ can any real value to define the bounds of the variables. The "maximize" can alternatively be "minimize," and each "$\leq$" relation can alternatively be an "$=$" or a"$\geq$" relation (D. L. Applegate et al., 2006).

Throughout this paper, inequalities and constraints will be used interchangeability.

2.2.1 Simplex Algorithm

As one of the most important algorithms in the century (W. J. Cook, 2012), the simplex algorithm is an integral algorithm to solving LP Problems, and by extension the TSP.

To start, suppose one has the linear programming formulation:

$$\begin{aligned}
&\text{Maximize } z = 10x_1 + 2x_2 + 8x_3 + 9x_4 \\
&\text{Subject to} \\
&\quad 3x_1 + 5x_2 + 2x_3 + 7x_4 \leq 50 \\
&\quad 4x_1 + 3x_2 + 6x_3 + 2x_4 \leq 40 \\
&\quad 2x_1 + 7x_2 + 3x_3 + 4x_4 \leq 60 \\
&\quad x_1, x_2, x_3, x_4 \geq 0
\end{aligned}$$

The **first** step is to introduce *slack* variables.

Just like a rope, there may be slack associated with the constraints— wiggle room between the values on the right-side and left-side of the inequality. To start, define the slack of the first inequality as x_5:

$$x_5 = 50 - 3x_1 - 5x_2 - 2x_3 - 7x_4$$

, where evidently, $x_5 \geq 0$. Symmetrically for the slack variables x_6 and x_7 for next two inequalities:

$$\begin{aligned} x_5 &= 50 - 3x_1 - 5x_2 - 2x_3 - 7x_4 \\ x_6 &= 40 - 4x_1 - 3x_2 - 6x_3 - 2x_4 \\ x_7 &= 60 - 2x_1 - 7x_2 - 3x_3 - 4x_4 \\ x_5, x_6, x_7 &\geq 0 \end{aligned}$$

In summary, let this formulation be its *slack* form. x_1, x_2, x_3, x_4 are **decision** variables, and in contrast, x_5, x_6, x_7 are **slack** variables.

In essence, the overarching idea of the simplex method is: Turning a feasible solution with the variables $x_1, x_2, ..., x_7$ into a feasible solution with the set, $\overline{x}_1, \overline{x}_2, ..., \overline{x}_7$ which is better in that upon using all the *slack* (slack variables are set to 0), of the new set of variables:

$$10\overline{x}_1 + 2\overline{x}_2 + 8\overline{x}_3 + 9\overline{x}_4 \geq 10x_1 + 2x_2 + 8x_3 + 9x_4$$

From the original LP Problem. The above process is done repeatedly to eventually reach the answer.

To solve the above LP problem, a feasible solution is needed to improve from. A trivial solution is setting the decision variable to 0($x_1, x_2, x_3, x_4 = 0$), for the objective function, which will yield $z = 0$, and slack variables: $x_5 = 50, x_6 = 40, x_7 = 60$. To find how to increase the z, notice increasing any of the decision variables will increase the value since they are non-negative and have positive coefficients in the objective function.

Thus, to increase z, one increases x_1 (the others remain 0), but by how much?

The answer is using up all the slack in each equation with x_1 to find the *upper bound* of x_1. Observe how all variables except for x_1 is still 0: Thus, rearranging every inequality after using all the slack:

$$x_1 \leq \frac{50}{3} - 0 - 0 - 0$$
$$x_1 \leq \frac{40}{4} - 0 - 0 - 0$$
$$x_1 \leq \frac{60}{2} - 0 - 0 - 0$$

If any inequality other than the second is picked as the bound of x_1, it will result in other variables to be negative, violating the non-negative constraints so $x_1 \leq 10$. So the maximum objective function at this moment is:

$$z = 100, x_1 = 10, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 20, x_6 = 0, x_7 = 40$$

Notice how convenient having a system of equations set by the slack variables from the inequalities were in increasing the z. Also notice, how all decision variables except the one to be increased were still 0. Therefore, the new system of equations must have all decision variables to be initially 0; Notice how the decision variables were *always on the left*, and the slack on the right.
Finally, notice how the slack in the second inequality represented by x_6 is now 0 and x_1 is a positive number. Thus, without the loss of generality, the role of x_6 and x_1 switch, such that since x_1 is no longer 0, its a slack variable. And x_6 as 0 is now a decision variable. This swapping of roles is called **pivoting**.
Thus, rearranging the inequality with the switched slack and decision variable:

$$x_6 = 40 - 4x_1 - 3x_2 - 6x_3 - 2x_4$$
$$4x_1 = 40 - 3x_2 - 6x_3 - 2x_4 - x_6$$
$$x_1 = 10 - \frac{3}{4}x_2 - \frac{2}{3}x_3 - \frac{1}{2}x_4 - \frac{1}{4}x_6$$

Since x_1 is now a slack variable, it does not make sense for it to appear on the right side of the equations. Thus, substitute all the moments in which it appears, with newly formulated expression of x_1 above.

Objective:

$$\begin{aligned}
&10x_1 + 2x_2 + 8x_3 + 9x_4 \\
&10(10 - \frac{3}{4}x_2 - \frac{2}{3}x_3 - \frac{1}{2}x_4 - \frac{1}{4}x_6) + 2x_2 + 8x_3 + 9x_4 \\
&100 - \frac{11}{2}x_2 + \frac{4}{3}x_3 - 14x_4 - \frac{5}{2}x_6
\end{aligned}$$

Constraints:

$$\begin{aligned}
x_5 &= 50 - 3x_1 - 5x_2 - 2x_3 - 7x_4 \\
&= 50 - 3(10 - \frac{3}{4}x_2 - \frac{2}{3}x_3 - \frac{1}{2}x_4 - \frac{1}{4}x_6) - 5x_2 - 2x_3 - 7x_4 \\
&= 20 - \frac{29}{4}x_2 - 4x_3 - \frac{17}{2}x_4 - \frac{3}{4}x_6
\end{aligned}$$

Similarly,

$$\begin{aligned}
x_7 &= 60 - 2x_1 - 7x_2 - 3x_3 - 4x_4 \\
&= 60 - 2(10 - \frac{3}{4}x_2 - \frac{2}{3}x_3 - \frac{1}{2}x_4 - \frac{1}{4}x_6) - 7x_2 - 3x_3 - 4x_4 \\
&= 40 - \frac{17}{2}x_2 - \frac{13}{3}x_3 - 5x_4 - \frac{1}{2}x_6
\end{aligned}$$

With these, another linear programming problem is formulated.
Thus, to increase the objective function, as x_3 is the only variable in z with a non-negative coefficient, increasing it will increase the objective function. Therefore, instead of doing the entire process with x_1 this time, the pivot is done to x_3.

If one repeatedly does these pivots. Eventually z will reach a point where it cannot increase, such that every coefficient for the non-negative variables are negative. At that point, the optimal value has been found.

2.3 Connecting to the TSP

The TSP can be formulated as a linear programming problem by creating a variable general edge from i to j equal, x_{ij} whose value can be from $0 \leq x_{ij} \leq 1$. The motivation is that x_{ij} should be 1 if it is used and 0 if it is not. But later, it can be seen that it can also be the x_{ij} can yield $\frac{1}{2}$, which are to be addressed later. And letting c_{ij} represent the cost from vertices i to j, an simple objective function is:

$$\text{minimize } c_{12}x_{12} + c_{13}x_{13} + \cdots + c_{nm}x_{nm}$$

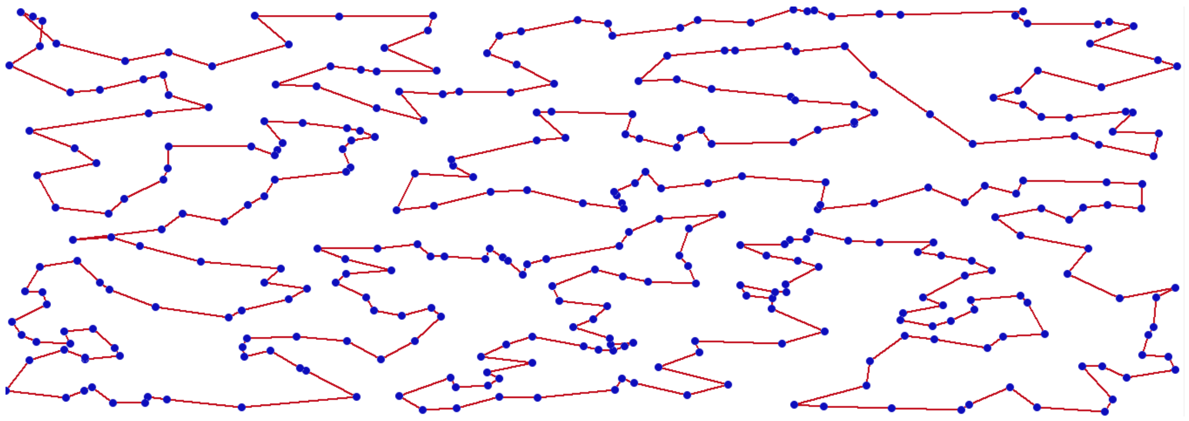


Figure 5: Tour made by Concorde

The problem now is to find a set of constraints so that to only consider tours so one can find the most optimal tour.

$0 \leq x_{ij} \leq 1$ is one, and Julia Robinson, a pioneer in the TSP, noticed that every vertex in a tour must have a degree of 2—

$$\sum_{j=1}^{m} x_{ij} = 2$$

(W. J. Cook, 2012).

This arrangement is called the **Degree LP Relaxation** of the TSP.

Unfortunately, non-tours can still get past the constraints:

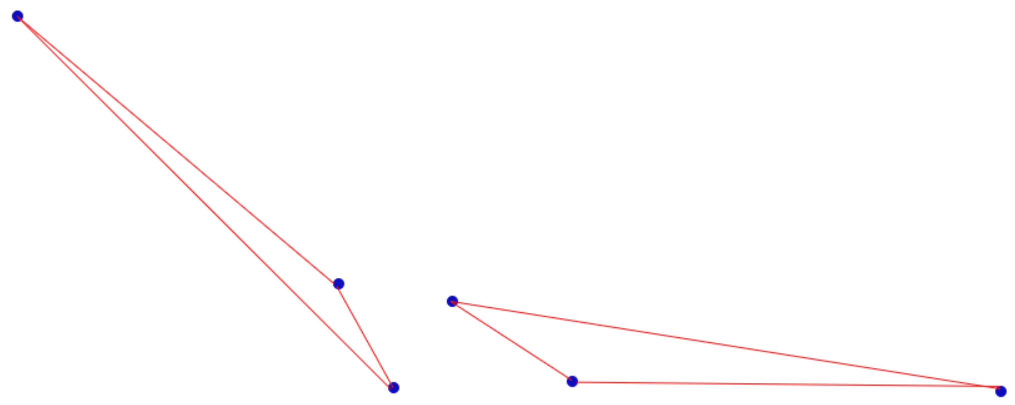


Figure 6: Subtours

These are called sub-tours, and they can be efficiently dealt with by the Cutting Plane Method.

2.4 Cutting Planes

The main problem of the linear programming approach is ensuring that only tours are in the solution set that is considered by the linear program. Indeed, if there was a way to only consider the set of edges, S that were tours through constraints, one can get the optimal tour simply through the simplex algorithm. Thus, the name of the game is to repeatedly introduce constraints that tours will always fall under and *cuts* off non-tours. It does this until it reaches a solution set such that the optimal value is a tour.

At each iteration, the algorithm aims to add a family of cuts (D. L. Applegate et al., 2006) that each addresses non-tour that slip through the original constraints. Some common families or cuts/constraints are the subtour, comb, clique tree, and blossom constraints. There are many more known and unknown types. Indeed, the current research explains only a small fraction of the fifty-one billion possible inequalities known for a ten-city instance (W. J. Cook, 2012).

2.4.1 Separation Algorithms

The heart of the cutting plane method and code used by Concorde is the Separation Problem. Concorde are basically wrappers for calling separation routines that addresses different unique families of inequalities. (W. J. Cook, 2012) .

2.5 Branch and Cut

When the iterations of the cutting plane methods become either too inefficient at cutting the problem or can no longer add constraints despite not reaching the optimal answer, inspiration from divide and conquer is used.
An analogy is the goal of finding the best needle amongst shiny needles in large haystack. Cutting planes can cut away at the haystack, but eventually it will slow down or even stop. Thus, at that point, the haystack should be split in half, a good division can reveal more needles through the new perspective created through splitting.

To be efficient, the optimal solutions in these sub problems that are less optimal than another are stopped immediately, a process called *pruning*— since it means the final answer cannot be in that subproblem. Additionally, the constraints of these divided sub-problems are shared amongst each other in a data-structure, so the same separation algorithms are called efficiently. These are known as **pool cuts**.

In essence, The Concorde TSP Solver embodies the "Bash on regardless" philosophy, using hundreds-of-thousands lines of code to address various separation algorithms and their inequalities. It iteratively applies the process: cut until no more cuts are possible, branch, and repeat—ultimately finding the optimal tour via the simplex algorithm.

2.6 Hypothesis

As an approximate algorithm, this paper hypothesizes the accuracy of the Lin-Kernighan heuristic algorithm will be extremely inaccurate compared to the Exact Linear Programming approach, espically with larger problem sizes— since the design of the algorithm does not explicitly account for optimality. And as a local-search algorithm, the larger the problem, the higher chance the algorithm can get stuck on suboptimal tours.
But in terms of computational time, although an extensive list of techniques and optimizations were applied to the exact algorithm, the burden of slow separation algorithms bottlenecks the speed of the code.

Thus, while the Lin-Kernighan heuristic (approximate algorithm) will demonstrate significantly faster computational time compared to the exact Linear Programming approach, it will simultaneously show a measurable decrease in solution accuracy as the problem size increases.

2.7 Dependent Variables

2.7.1 Time (seconds)

The main bottle-neck of most algorithms is the time it takes to run it. Thus, a good indicator of the effectiveness of the problem is the time it takes for an algorithm to run for a problem of given size.

2.7.2 Percentage Accuracy

Accuracy will be indirectly calculated by calculating the value of the tour returned by the algorithm. Through the equation:

$$\% \text{ accuracy } = \frac{\text{best tour}}{\text{current tour}} \times 100\% \tag{1}$$

This second parameter will be used to validate and assess the performance of the approximate algorithm.

2.8 Control Variables

Control Variables	**Why are they Controlled?**	**How are they Controlled?**
The NEOS Server	To figure out the true position of modern TSP Solvers, using a strong computer was necessary. A server that is optimized, maintained, and used solely for problems removes the worry of unknown background processes skewing the data and creating inconsistent results. Much simpler to start an investigation— the server returns much more information with limited interaction compared to running it on the computer.	The experiment was conducted strictly on this one optimized and focused service, and host usage was carefully tracked.
The Previous Dataset	Adding "50" random nodes to the previous dataset provides more related results to create a trend than simply creating a completely random set of nodes.	Through the "Add Random Nodes" function in the Concorde GUI. The files can be saved as a text file (I removed the '0' that appeared after the number of nodes).

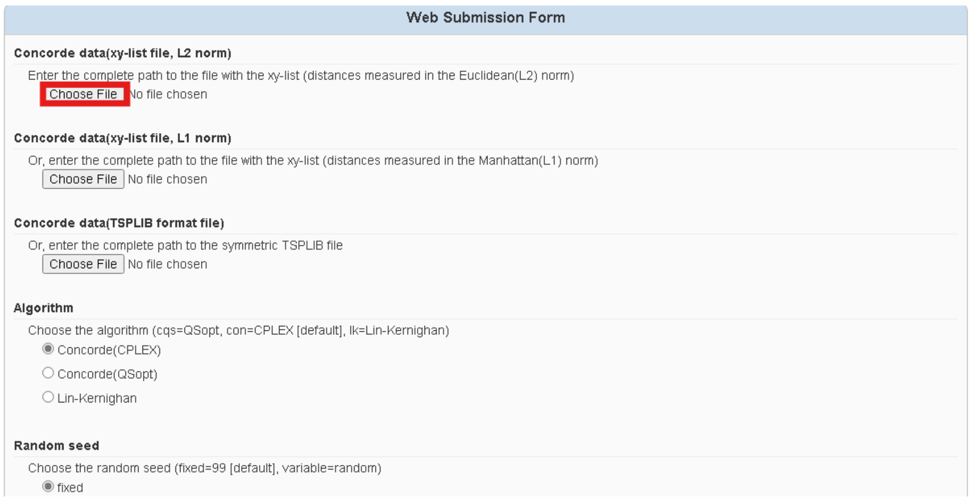
Table 1: Control Variables and Their Management

2.9 Procedure

Finally, the experiment commences.

1. Go to https://neos-server.org/neos/solvers/co:concorde/TSP.html

2. For each dataset (See Appendix 1), input the file into the Euclidean($L2$) norm.



Web Submission Form

Concorde data(xy-list file, L2 norm)

Enter the complete path to the file with the xy-list (distances measured in the Euclidean(L2) norm)

Choose File No file chosen

Concorde data(xy-list file, L1 norm)

Or, enter the complete path to the file with the xy-list (distances measured in the Manhattan(L1) norm)

Choose File No file chosen

Concorde data(TSPLIB format file)

Or, enter the complete path to the symmetric TSPLIB file

Choose File No file chosen

Algorithm

Choose the algorithm (cqs=QSopt, con=CPLEX [default], lk=Lin-Kernighan)

[● Concorde(CPLEX)] Concorde(QSopt) Lin-Kernighan

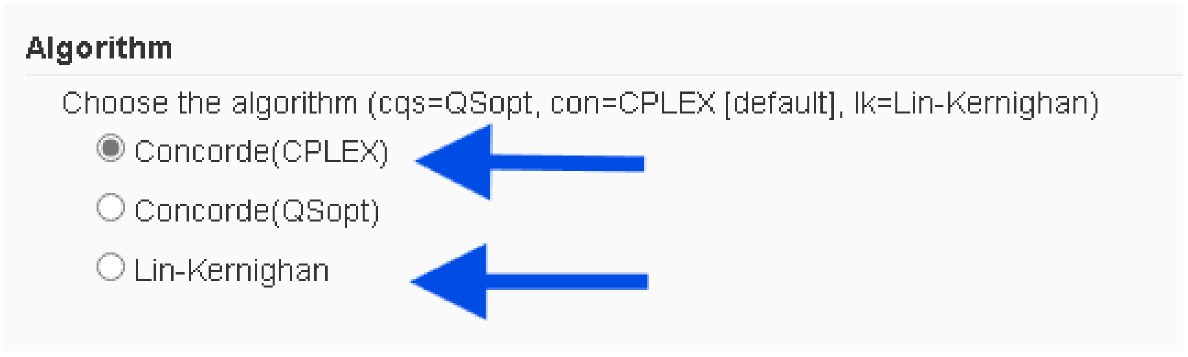
Random seed

Choose the random seed (fixed=99 [default], variable=random)

[● fixed]

Figure 7: Upload at Red

3. If measuring the exact algorithm press "Concorde(CPLEX) "; If measuring the approximate algorithm press "Lin-Kernighan"



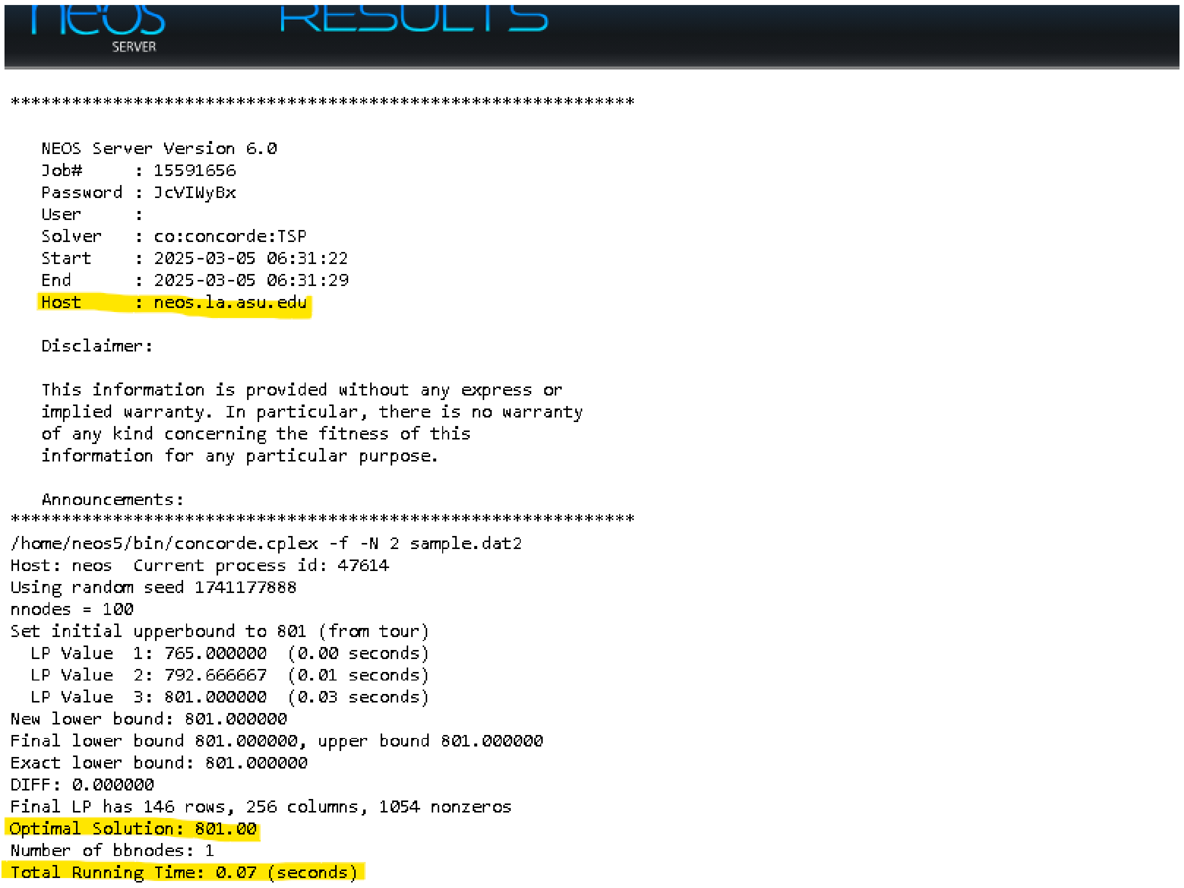
Algorithm

Choose the algorithm (cqs=QSopt, con=CPLEX [default], lk=Lin-Kernighan)

[● Concorde(CPLEX)] Concorde(QSopt) Lin-Kernighan

Figure 8: Consider only Blue Choices

4. Click Variable Seed
   (a) Or else the Lin-Kernighan values will all be the same
5. This experiment needs many measurements, to prevent spam in the inbox, put a **fake** email. Then click submit.
6. Refer to Figure 9 , this page indicates the results are finished.
   (a) Write the trial number (5 in total), the server name receiving the request, the optimal tour value, and total running time for this dataset
7. Repeat 5 trials for each dataset with both algorithms



```
*************************************************************

   NEOS Server Version 6.0
   Job#     : 15591656
   Password : JcVIWyBx
   User     :
   Solver   : co:concorde:TSP
   Start    : 2025-03-05 06:31:22
   End      : 2025-03-05 06:31:29
   Host     : neos.la.asu.edu

   Disclaimer:

   This information is provided without any express or
   implied warranty. In particular, there is no warranty
   of any kind concerning the fitness of this
   information for any particular purpose.

   Announcements:
*************************************************************
/home/neos5/bin/concorde.cplex -f -N 2 sample.dat2
Host: neos  Current process id: 47614
Using random seed 1741177888
nnodes = 100
Set initial upperbound to 801 (from tour)
  LP Value  1: 765.000000  (0.00 seconds)
  LP Value  2: 792.666667  (0.01 seconds)
  LP Value  3: 801.000000  (0.03 seconds)
New lower bound: 801.000000
Final lower bound 801.000000, upper bound 801.000000
Exact lower bound: 801.000000
DIFF: 0.000000
Final LP has 146 rows, 256 columns, 1054 nonzeros
Optimal Solution: 801.00
Number of bbnodes: 1
Total Running Time: 0.07 (seconds)
```

Figure 9: Highlighted Text is Important

3 Experimental Results

Measured using the output data received from the server, the results are in Tables 2 and 3. The mean time taken was calculated by:

$$\text{mean} = \frac{t_1 + t_2 + t_3 + t_4 + t_5}{5}$$

Problem Size	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5	Mean Time (s)
N=50	0.01	0.02	0.01	0.01	0.01	0.012
N=100	0.02	0.03	0.02	0.03	0.02	0.024
N=150	0.05	0.03	0.02	0.02	0.05	0.034
N=200	0.04	0.04	0.07	0.07	0.07	0.058
N=250	0.06	0.06	0.09	0.06	0.05	0.064
N=300	0.06	0.06	0.07	0.1	0.1	0.078
N=350	0.09	0.14	0.09	0.13	0.1	0.11

Table 2: Time Taken for Chained Lin-Kernighan Heuristic

The exact algorithm returns the values of the optima tours: Which can be used to assess the accuracy of the approximate algorithm through equation 1 (The mean distances will be used):

Problem Size	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5	Mean Time (s)
N=50	0.06	0.03	0.05	0.03	0.03	0.04
N=100	0.06	0.1	0.06	0.1	0.06	0.076
N=150	0.12	0.21	0.21	0.21	0.14	0.178
N=200	0.14	0.15	0.14	0.15	0.23	0.162
N=250	0.15	0.24	0.24	0.24	0.15	0.204
N=300	1.14	1.15	0.68	0.61	0.67	0.85
N=350	1.76	1.77	1.75	1.76	1.76	1.76

Table 3: Time Taken for Branch & Cut LP Algorithm

Problem Size	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5	Mean Distance
N=50	557	557	557	557	557	557
N=100	801	801	801	801	801	801
N=150	976	976	976	976	976	976
N=200	1086	1084	1086	1084	1086	1085.2
N=250	1180	1180	1188	1180	1180	1181.6
N=300	1292	1292	1292	1292	1292	1292
N=350	1415	1416	1415	1416	1416	1415.6

Table 4: Shortest Tour (unitless) Obtained by Lin-Kernighan algorithm

Problem Size	Optimal Tour
N=50	557
N=100	801
N=150	976
N=200	1084
N=250	1180
N=300	1292
N=350	1415

Table 5: Optimal Values From LP Algorithm

Problem Size	Percentage Accuracy (%)
N=50	100%
N=100	100%
N=150	100%
N=200	99.89%
N=250	99.86%
N=300	100%
N=350	99.96%

Table 6: Percentage accuracy calculated using Equation 1

3.1 Graphical Analysis

Utilizing the values from Tables 2 and 3:

From Figure 10, a parabolic shape is evidently seen—which indicates a time complexity $O(N^2)$. Agreeing with the extensive research papers that detail the heuristic(Kernighan & Lin, 1970).

Where as Figure 10 grew steadily, Figure 11 has a sharp turn at around $N = 250$ revealing that even modern LP TSP solvers still struggle with time complexity. This sharp-turn implies an exponential time function—supporting the consensus in research that the worse case for the algorithm runs in

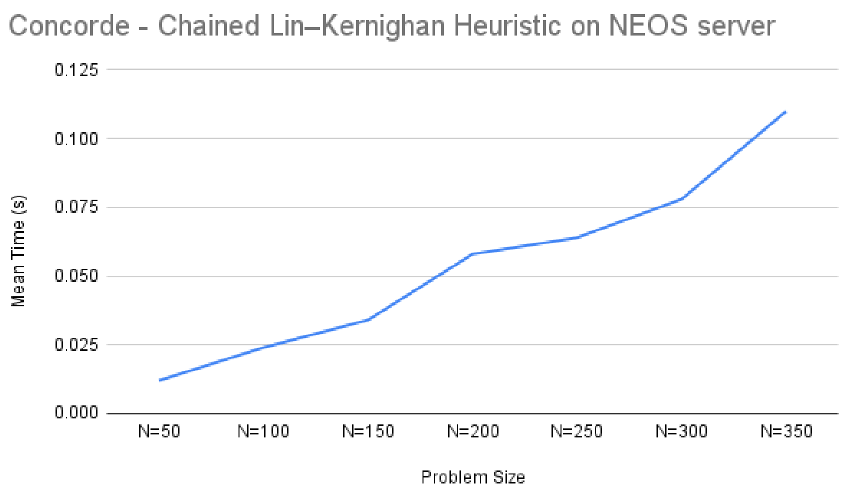


Figure 10: Graph of Table 3

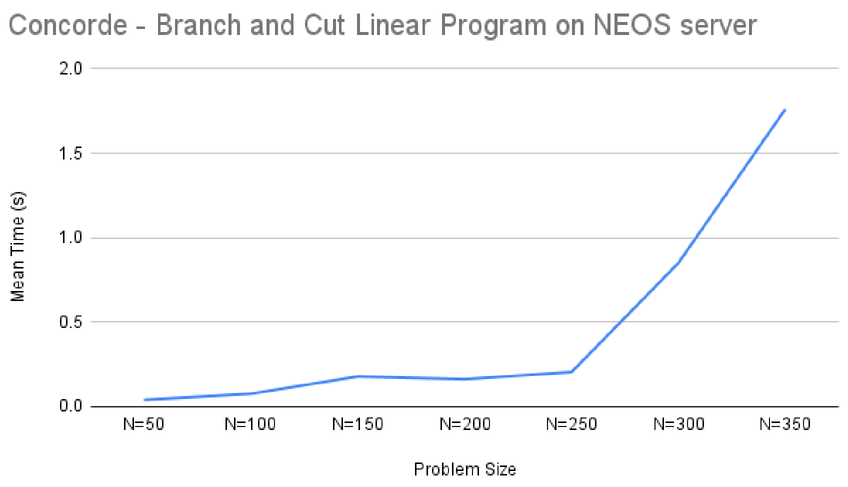


Figure 11: Graph of Table 2

non-polynomial time (D. L. Applegate et al., 2006).

Trivially, the exact algorithm had only one output: the optimal tour, and to compare to the Lin-Kernighan results on referring to Table 6, as the problem size gets higher, the percentage accuracy gets slightly get lower. Though, as a very powerful heuristic, the Lin-Kernighan algorithm would need much larger problem sizes to create larger degrees of error.

In essence, although approximate algorithms gave up the optimality for speed, as the moderately-sized instances of the TSP demonstrate, there is little-to-no loss for this approach.

In contrast, the same time it took the approximate algorithm to run the $n = 350$ dataset, it took the exact algorithm to run the $n = 100$ problem. Furthermore, the algorithm started to sharply increase in time between $n = 250$ and $n = 350$, likely because the rapid-increase of the amount of constraints and variables that the LP solution had to consider. In essence, where as a local-search tour improvement algorithm like the Lin-Kernighan considered the entire solution set in smart way through *kicking*, the Branch & Cut solution considers the entire problem at all times. Which intuitively, is not an efficient approach.

Interestingly, the times in Table 4, creating suboptimal answers were always to the same value. For example for $N = 350$, the suboptimal result was always 1416. These results are often very close to the optimal, and thus these instances appear to be 1 kick away to optimality.

4 Conclusion

4.1 Hypothesis Revisited

In essence, although approximate algorithms do not guarantee optimality, there is enough statistical evidence through the experiments such that even for moderately large cases, the approximate algorithm approach provides extremely optimal values.

From this experiment, it is evident that there is outstanding work in the field of exact algorithms because this experiment reveals the approximate counterpart to be much more efficient in terms of speed and accuracy.

And as predicted, the exact algorithm slows down considerably as the problem sizes got bigger.

So in essence, the question becomes what merit is there in continuing the approach in the exact algorithms?

4.2 Why Choose Exact Algorithms?

Despite the results of the experiment, qualitatively, the implications of the exact algorithms are important.

Through achieving the goal of understanding and appreciating both methods, it became evident that the pursuit of the exact algorithm for the Travelling Salesman Problem is a beacon of discovery. For instance, linear programming, cutting planes, and branching all originated from the pursuit of the salesman. These general optimization tools that have emerged from this field now run the world: such as making business decisions with millions of variables or optimizing supply chains, transportation networks, and even computational resource allocation.

Sources of Error	**Effect and Importance**	**Improvements**
Systematic Error in the Implementation	Without having access to the Concorde software run by the server, changes to the original source code are not accounted for. Both algorithms have personalization like the number of kicks and size of local cuts, which can lead to vastly different results. These numbers set by the server were never known.	The parameters can create huge changes thus one could potentially reverse engineer their way to determine the exact specifications. Communicate with the maintainers of the server.
Lack of Control of What Servers Ran the Concorde Code to Process the Input	There were three servers that ran my code: Athene, Karatos, and Neos. Although these the servers run by NEOS have similar hardware (NEOS Guide, n.d.), the slight differences in specs noticeably skewed the operation times—where even a fraction of a second mattered.	Utilize larger problem sizes. As the problem sizes get larger, the slight hardware differences matter less and less.

Table 7: Sources of Error, Their Effects, and Possible Improvements

4.3 Evaluation

At last, after evaluating the data acquired in this investigation, it has been concluded that the approximate algorithm quantitatively, is much more efficient the exact algorithm in moderately sized problems. But this simply cannot be said without considerations of the implications improving the exact algorithm would bring.

5 Appendix

5.1 Data

The datasets are much too big to be put in this paper, and thus are present at https://github.com/CodingMayus/extended-essay/tree/main/datasets

References

Alon. (2022). *Christofides algorithm: Finding a solution for the tsp problem.* Retrieved from https://alon.kr/posts/christofides (Accessed: 2024-11-29)

Applegate, D., Bixby, R., Chvátal, V., & Cook, W. J. (n.d.). *Concorde tsp solver.* https://www.math.uwaterloo.ca/tsp/concorde/index.html. (Accessed: 2025-03-05)

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2006). *The traveling salesman problem: A computational study.* Princeton University Press.

Chvátal, V. (1983). *Linear programming.* W. H. Freeman and Company.

Cook, B. (2021). *Math encounters: "optimal tours: The traveling salesman problem".* YouTube video. Retrieved from https://www.youtube.com/watch?v=tChnXG6ulyE (Accessed: 2025-03-05)

Cook, W. J. (n.d.). *Traveling salesman problem .diy.* https://www.math.uwaterloo.ca/tsp/app/diy.html. (Accessed: 2025-03-05)

Cook, W. J. (2012). *In pursuit of the traveling salesman: Mathematics at the limits of computation.* Princeton and Oxford: Princeton University Press.

Czyzyk, J., Mesnier, M. P., & More, J. J. (1998). The neos server. *IEEE Journal on Computational Science and Engineering*, *5*(3), 68–75.

Demaine, E., Devadas, S., & Lynch, N. (2015). *6.046j / 18.410j design and analysis of algorithms: Lecture 15 - linear programming.* https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/927a49adf7b3c4a6de8c31393509552c_MIT6_046JS15_lec15.pdf. (Accessed: 2025-03-05)

Devadas, S. (2015). *15. linear programming: Lp, reductions, simplex.* YouTube video. Retrieved from https://www.youtube.com/watch?v=WwMz2fJwUCg (Accessed: 2025-03-05)

Dolan, E. D. (2001). *The neos server 4.0 administrative guide* (Technical Memorandum No. ANL/MCS-TM-250). Mathematics and Computer Science Division, Argonne National Laboratory.

GeeksforGeeks. (2024). *Time complexity and space complexity.* Retrieved from https://www.geeksforgeeks.org/time-complexity-and-space-complexity/ (Accessed: 2025-03-05)

Greenwade, G. D. (1993). The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, *14*(3), 342–351.

Gropp, W., & Moré, J. J. (1997). Optimization environments and the neos server. In M. D. Buhman & A. Iserles (Eds.), *Approximation theory and optimization* (p. 167 - 182). Cambridge University Press.

Gupta, N., & Sharma, S. (2020, May). Literature review on travelling salesman problem. *ResearchGate*. Retrieved from https://www.researchgate.net/publication/341371861_Literature

_Review_on_Travelling_Salesman_Problem doi: 10.13140/RG.2.2.30585.90727

Helsgaun, K. (1999). An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational*, *Volume Number*(Issue Number), 1–25. Retrieved from https://sci-hub.se/https://doi.org/10.1016/S0377-2217(99)00284-2 doi: 10.1016/S0377-2217(99)00284-2

Helsgaun, K. (2000a). *An effective implementation of the lin-kernighan traveling salesman heuristic* (Tech. Rep.). Roskilde University. Retrieved from http://akira.ruc.dk/~keld/research/LKH/LKH-1.3/DOC/LKH_REPORT.pdf

Helsgaun, K. (2000b). An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*(1), 106–130. Retrieved from http://akira.ruc.dk/~keld/research/LKH/LKH-1.3/DOC/LKH_REPORT.pdf doi: 10.1016/S0377-2217(99)00284-2

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

Keller, B. (n.d.). *Branch-and-bound algorithm for the traveling salesman problem.* https://www.math.cmu.edu/~bkell/21257-2014f/tsp.pdf. (Accessed: 2025-03-05)

Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, *49*(2), 291-307. doi: 10.1002/j.1538-7305.1970.tb01770.x

Krymgand, A. (2023). The christofides algorithm. *https://alon.kr/*.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, *59*(2), 231–247. doi: 10.1016/0377-2217(92)90138-Y

LibreTexts. (2020). *Combinatorics: Basics of graph theory.* Author. Retrieved from https://math.libretexts.org/Bookshelves/Combinatorics_and_Discrete_Mathematics/Combinatorics_(Morris)/03%3A_Graph_Theory/11%3A_Basics_of_Graph_Theory/11.02%3A_Basic_Definitions_Terminology_and_Notation (Accessed: 2024-11-29)

NEOS Guide. (n.d.). *Neos server faq.* Retrieved from https://neos-guide.org/users-guide/faq/ (Accessed: 2025-03-05)

of Cambridge, U. (2018). *The travelling salesman problem (tsp).* https://www.cl.cam.ac.uk/teaching/1718/AdvAlgo/tsp.pdf. (Accessed: 2024-11-29)

Ralphs, T. (n.d.). *Computational integer programming lecture 8: Branch and bound.* https://coral.ise.lehigh.edu/~ted/files/computational-mip/lectures/Lecture8.pdf. (Accessed: 2025-03-05)

Reducible. (2022). *The traveling salesman problem: When good enough beats perfect.* YouTube video. Retrieved from https://www.youtube.com/watch?v=GiDsjIBOVoA (Accessed: 2025-03-05)

Ryan, C. (2003). Computer algorithms. In R. A. Meyers (Ed.), *Encyclopedia of physical science and technology (third edition)* (Third Edition ed., p. 507-523). New York: Academic Press. Retrieved from https://www.sciencedirect.com/science/article/pii/B0122274105008401 doi: https://doi.org/10.1016/B0-12-227410-5/00840-1

S, T. (n.d.). *The art of linear programming.* YouTube video. Retrieved from https://www.youtube.com/watch?v=E72DWgKP_1Y (Accessed: 2025-03-05)