

**PATH OPTIMIZATION FOR COOPERATIVE  
MULTI-HEAD 3D PRINTING**

**CHEONG KAH JUN**

**A project report submitted in partial fulfilment of the  
requirements for the award of Bachelor of Engineering  
(Honours) Electrical and Electronic Engineering**

**Lee Kong Chian Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman**

**September 2020**

## DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :   
\_\_\_\_\_

Name : Cheong Kah Jun  
\_\_\_\_\_

ID No. : 1600422  
\_\_\_\_\_

Date : 5<sup>th</sup> September 2020  
\_\_\_\_\_

## APPROVAL FOR SUBMISSION

I certify that this project report entitled “**PATH OPTIMIZATION FOR COOPERATIVE MULTI-HEAD 3D PRINTING**” was prepared by **CHEONG KAH JUN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature

:



---

Supervisor

:

See Yuen Chark

---

Date

:

7<sup>th</sup> September 2020

---

Signature

:

---

Co-Supervisor

:

---

Date

:

---

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2020, Cheong Kah Jun. All rights reserved.

## **ACKNOWLEDGEMENTS**

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. See Yuen Chark for his invaluable advice, guidance and his enormous patience throughout the development of the research. I am deeply grateful to him for imparting his knowledge and lending his expertise to me.

In addition, I would also like to express my gratitude to my loving family and friends who had helped and given me encouragement throughout the course of the project. Without them, I would not have the strength and determination to see the whole thing through.

## **ABSTRACT**

In 3D printing, reducing the time needed to print an object is desirable. The print time of an object is largely influenced by the length of the path that the nozzle of the 3D printer takes for each layer (also called the travel distance). Hence, finding the best path for the nozzle is termed as the Layer Path Optimization Problem (LPOP). Previous authors have shown that the LPOP can be defined in terms of the Undirected Rural Postman Problem (URPP), a known problem in graph theory. Two well-known algorithms for solving a closely related graph theory problem known as the Travelling Salesman Problem (TSP) are the Ant System (AS) and Ant Colony System (ACS) algorithms. Therefore, to solve the LPOP, two algorithms are proposed, the modified AS algorithm and the modified ACS algorithm. These two proposed algorithms have been modified from the original algorithms in order to solve the URPP instead of the TSP. The performance of the two proposed algorithms is compared against Cura, which is a commonly used software for generating the nozzle path. The obtained results show that both the modified AS and ACS algorithms were able to perform better than Cura in terms of both travel distance and print time for a variety of different 3D models.

## TABLE OF CONTENTS

<b>TABLE OF CONTENTS</b>		<b>i</b>
<b>LIST OF TABLES</b>		<b>iii</b>
<b>LIST OF FIGURES</b>		<b>v</b>
<b>LIST OF SYMBOLS / ABBREVIATIONS</b>		<b>vii</b>
<b>LIST OF APPENDICES</b>		<b>viii</b>
 <b>CHAPTER</b>		
<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 General Introduction	1
	1.2 Importance of the Study	2
	1.3 Problem Statement	3
	1.4 Aim and Objectives	4
	1.5 Scope and Limitation of the Study	4
<b>2</b>	<b>LITERATURE REVIEW</b>	<b>5</b>
	2.1 Introduction	5
	2.2 Definitions and Representations of the LPOP	5
	2.2.1 Travelling Salesman Problem (TSP)	7
	2.2.2 Undirected Rural Postman Problem (URPP)	7
	2.3 Algorithms for Solving the LPOP	9
	2.3.1 Tour Construction Algorithms	9
	2.3.2 Improvement Algorithms	15
	2.3.3 Application-specific Methods	16
	2.4 Summary	20
<b>3</b>	<b>METHODOLOGY AND WORK PLAN</b>	<b>22</b>
	3.1 Introduction	22
	3.2 Cura	22
	3.3 G-code	23
	3.4 Parser Program and Conversion Back to G-code	25
	3.5 Algorithms to be Tested	29
	3.5.1 ACO Algorithm Variants	29

	3.5.2	Modifying AS and ACS Algorithms to solve the URPP	31
	3.6	Implementation of the Parser and Algorithms	34
	3.7	G-code Simulator	35
	3.8	Summary	36
<b>4</b>		<b>RESULTS AND DISCUSSION</b>	<b>37</b>
	4.1	Introduction	37
	4.1.1	Testing Process	37
	4.1.2	Explanation of Terminology Used	39
	4.2	Preliminary Parameter Testing Phase	40
	4.2.1	Setup for Varying the Number of Iterations	41
	4.2.2	Results for Varying the Number of Iterations	41
	4.2.3	Discussion for Varying the Number of Iterations	45
	4.2.4	Setup for Varying $\rho$ for Modified AS and $\rho, \varphi, q_0$ for Modified ACS	46
	4.2.5	Results for Varying $\rho$ for Modified AS and $\rho, \varphi, q_0$ for Modified ACS	47
	4.2.6	Discussion for Varying $\rho$ for Modified AS and $\rho, \varphi, q_0$ for Modified ACS	49
	4.3	Main Testing Phase	50
	4.3.1	Setup for Main Testing Phase	50
	4.3.2	Main Testing Phase Results	52
	4.3.3	Comparison of Results with State-of-the-Art	61
	4.3.4	State-of-the-art Discussion	64
	4.4	Summary	65
<b>5</b>		<b>CONCLUSIONS AND RECOMMENDATIONS</b>	<b>66</b>
	5.1	Conclusions	66
	5.2	Recommendations for Future Work	67
		<b>REFERENCES</b>	<b>68</b>
		<b>APPENDICES</b>	<b>71</b>

## LIST OF TABLES

Table 3.1: Parameters for G0 and G1	24
Table 3.2: Examples of G0 and G1 commands	25
Table 3.3: Initial G-code for the first layer	26
Table 3.4: Coordinates of each node	27
Table 3.5: Data matrix for the edges	27
Table 3.6: Elements of $V$ , $E$ and $E_r$	28
Table 3.7: Optimized G-code	29
Table 3.8: Algorithms to be tested	29
Table 4.1: Definitions of terminology used	39
Table 4.2: Default parameters for original AS	40
Table 4.3: Default parameters for original ACS	40
Table 4.4: Graphs of print time against travel distance for the modified AS and ACS algorithms when applied to the sample set of 3D models	42
Table 4.5: Graphs of travel distance against processing time for the modified AS and ACS algorithms when applied to the sample set of 3D models	43
Table 4.6: Graphs of print time against processing time for the modified AS and ACS algorithms when applied to the sample set of 3D models	44
Table 4.7: Parameters for varying the $\rho$ for the modified AS algorithm	46
Table 4.8: Parameters for varying the $\rho, \varphi, q_0$ for the modified ACS algorithm	47
Table 4.9: Parameters resulting in shortest travel distance for modified AS and ACS algorithms when applied to the sample set of 3D models	48
Table 4.10: Parameters resulting in shortest print time for modified AS and ACS algorithms when applied to the sample set of 3D models	48
Table 4.11: Standard deviations of travel distance and print time for the range of configurations tested for the AS and ACS algorithms	49

Table 4.12: Parameters for modified AS algorithm used in main testing phase	50
Table 4.13: Parameters for modified ACS algorithm used in main testing phase	50
Table 4.14: Settings for Cura used in main testing phase	51
Table 4.15: Procedure for the main testing phase	52
Table 4.16: Travel distance of G-code produced by Cura, modified AS and modified ACS in the main testing phase	53
Table 4.17: Percentage travel distance reduced by modified AS and ACS relative to Cura in the main testing phase	54
Table 4.18: Print time of G-code produced by Cura, modified AS and modified ACS in the main testing phase	56
Table 4.19: Percentage print time reduced by modified AS and ACS relative to Cura in the main testing phase	57
Table 4.20: Processing time of modified AS and ACS algorithm in the main testing phase	59
Table 4.21: Print times reported by Fok, et al. (2018) for the “Torture test” 3D model	61
Table 4.22: Comparison between percentage print time reduced for modified AS, modified ACS and Fok, et al. (2018) for the “Torture test” 3D model	62
Table 4.23: Print times reported by Fok, et al. (2019) for the “Clamp” and “Lowest poly thinker” 3D models	63
Table 4.24: Comparison between percentage print time reduced for modified AS, modified ACS and Fok, et al. (2019) for the “Clamp” and “Lowest poly thinker” 3D models	63

## LIST OF FIGURES

Figure 1.1: Print nozzle and print bed of a FFF printer	1
Figure 1.2: (a) Example of a LPOP, (b) Example of a possible solution	3
Figure 2.1: Two nodes connected by an edge	5
Figure 2.2: (a) Undirected edge, (b) Directed edge	6
Figure 2.3: $(i, j) \in E_r$ and the three replacement nodes	8
Figure 2.4: Complete graph	10
Figure 2.5: Minimum spanning tree $T$	11
Figure 2.6: Minimum-length perfect matching $M$	11
Figure 2.7: Euler tour in $M \cup T$	12
Figure 2.8: TSP tour obtained by taking shortcuts	12
Figure 2.9: (a) Initial graph, (b) After 2-opt move	15
Figure 2.10: (a) Initial graph, (b) & (c) Two possible 3-opt moves	16
Figure 2.11: Transition without retraction (A), and with retraction (B)	19
Figure 3.1: Flowchart of the process to evaluate LPOP algorithms	22
Figure 3.2: Cura user interface	23
Figure 3.3: Output of the slicer program for the first layer	26
Figure 3.4: Optimized solution	28
Figure 3.5: (a) Original URPP, (b) After transformed to TSP	31
Figure 3.6: Modified AS algorithm for solving URPP	33
Figure 3.7: Modified ACS algorithm for solving URPP	34
Figure 3.8: Screenshot from gCodeViewer	35
Figure 3.9: Screenshot from Gcode Analyser	35
Figure 4.1: Bar graph of travel distance results in the main testing phase	53

- Figure 4.2: Bar graph of percentage travel distance reduced in the main testing phase 54
- Figure 4.3: Bar graph of print time results in the main testing phase 56
- Figure 4.4: Bar graph of percentage print time reduced in the main testing phase 57
- Figure 4.5: Bar graph of processing time results in the main testing phase 59
- Figure 4.6: Scatter plot of processing time against Cura print time for modified AS and ACS algorithm in the main testing phase 60
- Figure 4.7: Bar chart for percentage print time reduced comparison with Fok, et al. (2018) for the “Torture test” 3D model 62
- Figure 4.8: Bar chart for percentage print time reduced comparison with Fok, et al. (2019) for the “Clamp” and “Lowest poly thinker” 3D models 64

**LIST OF SYMBOLS / ABBREVIATIONS**

3D	three-dimensional
CNC	computer numerical control
FFF	fused filament fabrication
FDM	fused deposition modelling
LPOP	layer path optimization problem
TSP	traveling salesman problem
URPP	undirected rural postman problem
ACO	ant colony optimization
AS	ant system
ACS	ant colony system

**LIST OF APPENDICES**

APPENDIX A: Graphs	71
APPENDIX B: Tables	72

## CHAPTER 1

### INTRODUCTION

#### 1.1 General Introduction

In the field of rapid prototyping, additive manufacturing, or more commonly known as 3D printing, is quickly gaining traction. As the name implies, the process involves the fabrication of an object layer by layer, in contrast to subtractive manufacturing such as CNC milling. This allows for greater control of the build process, and complex objects can be created that would be difficult or impossible through traditional methods.

There are a number of 3D printing technologies currently available which use different techniques. These technologies are listed below.

- Fused filament fabrication (FFF)
- Resin printing (SLA, DLP)
- Selective laser sintering (SLS)
- Inkjet printing

FFF, or otherwise termed fused deposition modelling (FDM), is currently one of the most popular methods of 3D printing. Generally, a FFF 3D printer can be visualized as having a print bed and a print nozzle, as shown in Figure 1.1. The print bed is the platform on which the object is printed on, while the print nozzle is the point at which the material, or filament is extruded from (Ganganath, et al., 2016).

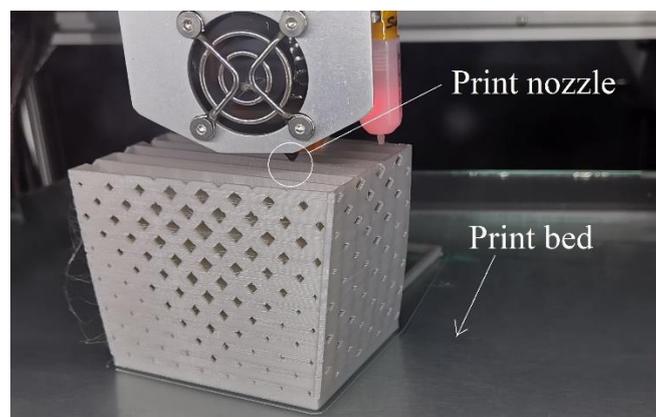


Figure 1.1: Print nozzle and print bed of a FFF printer

Usually the printer is set up such that the print bed can move in the Z-axis, while the print nozzle can move in the X and Y axes, allowing movement in all three axes. To print an object, the nozzle moves to extrude material in such a way as to construct the desired object. Once the first layer is completed, the print bed is lowered to allow printing of the next layer. This process repeats layer after layer until the print job is completed.

The set of instructions telling the 3D printer how to move and extrude material is called G-code, and is generated by a program known as a slicer. The slicer takes a 3D model and ‘slices’ it into thin layers. Each layer is made up of many print segments where material needs to be extruded to build the 3D object. The slicer then determines the path, velocity and acceleration of the nozzle as well as the amount of material extruded based on the print segments for each layer.

## **1.2 Importance of the Study**

One of the main issues with FFF is that it takes a significant amount of time to print a complex object, requiring several hours or even days (Lensgraf and Mettu, 2016). Thus, the reduction of printing time is highly desirable.

The printing speed is determined by several factors, such as:

- Dimensions and shape of the object
- Infill pattern
- Path taken by the print nozzle
- Velocity and acceleration of the print nozzle
- Mechanical limitations of the 3D printer

This report will focus on optimizing the path travelled by the print nozzle. When printing a layer, there will be certain areas or segments where material needs to be deposited in order to build the object. In other words, the print nozzle will need to travel to these areas to print the object. If the path that the nozzle needs to travel along is shortened, the time required to print the object will be reduced as well.

### 1.3 Problem Statement

The layer path optimization problem (LPOP) can be visualized mathematically as a graph, where each print segment is an edge connecting two nodes on both ends (Ganganath, et al., 2016). The task is then to find the shortest path that connects all the required edges, traversing each edge only once.

Figure 1.2 (a) shows an example of a LPOP, where three segments that need to be printed are represented as edges. Figure 1.2 (b) shows a possible solution (may not be the best solution) to the problem in Figure 1.2 (a). The dotted lines represent the transition segments, which are edges that the nozzle traverses along to travel from one print segment to another. In summary, the total length of the path travelled is the sum of all the edges, including the print segments and transition segments.

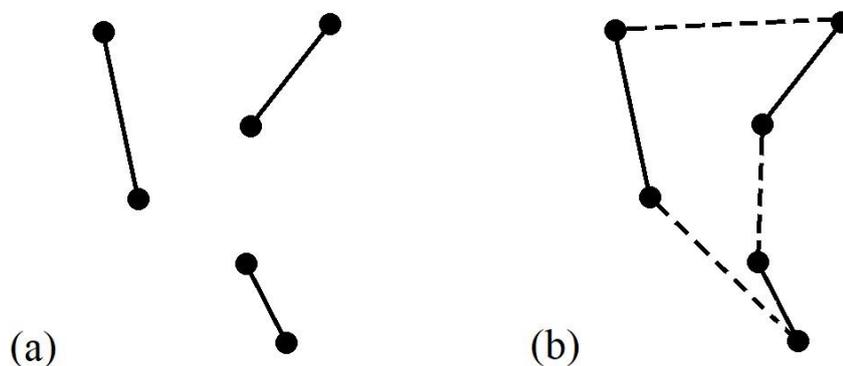


Figure 1.2: (a) Example of a LPOP, (b) Example of a possible solution

This situation is similar to the widely-known travelling salesman problem (TSP). For TSP, a graph is given, and the objective is to find the shortest path or tour that visits every node exactly once, then returns to the starting node. However, the difference between the LPOP and TSP is that the former focuses on connecting required edges, while the latter needs to connect all nodes.

According to Fok, et al. (2018), the LPOP can actually be represented as the undirected rural postman problem (URPP). The definition of the URPP is that given a set of edges  $E$  (meaning the print segments and possible transition segments), and another set of required edges  $E_r \subseteq E$  (the print segments only), the goal is to find the shortest path that travels along every edge in  $E_r$  exactly once.

#### **1.4 Aim and Objectives**

This report aims to reduce the 3D printing time by utilizing algorithms to solve the LPOP such that the path length is reduced.

The first objective is to extract the layer data from the G-code output of a slicer program using a parser program, then transform the obtained layer data into the LPOP. The next objective is to solve the LPOP using various algorithms. Finally, the last objective is to convert the obtained solution back into G-code, so that a G-code simulator can be used to analyse and compare the initial G-code and the optimized G-code.

#### **1.5 Scope and Limitation of the Study**

The report will only focus on reducing the path length based on the print segments generated by the original slicer program (Cura). It will not take into account the speed and acceleration of the nozzle when trying to optimize the path.

Retraction will also not be factored into the optimization process. The retraction setting in Cura is disabled when generating the G-code for the 3D objects.

The reason for setting these limitations is to reduce the number of factors that need to be taken into account, thereby simplifying the optimization process. These ignored factors may be further explored in future works.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Introduction

The topic of this report is admittedly very specific and niche, and as such there is a limited amount of prior research regarding the topic. It is however not a major surprise since additive manufacturing has only gained popularity in the past decade, despite having already existed for a few decades.

Nevertheless, there has been substantial progress in the field throughout the last few years. In addition, advances made in the related fields of graph theory as well as agent and multiagent systems may prove useful in helping to solve the problem.

#### 2.2 Definitions and Representations of the LPOP

The layer path optimization problem (LPOP) is broadly defined as the best path that the nozzle of a 3D printer should take to print all the print segments of a layer. In order to discuss the problem in a formal manner, the LPOP is usually represented in terms of a graph, which is a mathematical construct originating from graph theory.

A graph,  $G$  is made up of two parts,  $V$  and  $E$ , such that  $G = (V, E)$ .  $V$  is the set of nodes or vertices, while  $E$  is the set of edges or connections. The graph can be visualized as nodes being points, and as edges being lines that link the nodes together. Figure 2.1 shows an example where nodes  $A$  and  $B$  are connected by the edge  $r$ .

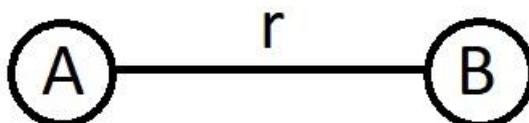


Figure 2.1: Two nodes connected by an edge

Each edge has a cost attributed to it, which is the cost incurred when travelling along it. Usually the cost of an edge is simply the distance between the nodes that it links, or in other words its length. However, it may represent other parameters, or even a combination of them.

Edges can also be divided into two types, undirected and directed edges. Take Figure 2.2 as an example. Figure 2.2 (a) has an undirected edge  $s$ , which means that travelling from node  $C$  to  $D$  and vice-versa would both be allowed. However, Figure 2.2 (b) has a directed edge  $t$ , also called an arc, which means that travel would only be limited to one direction, from node  $F$  to  $E$ .

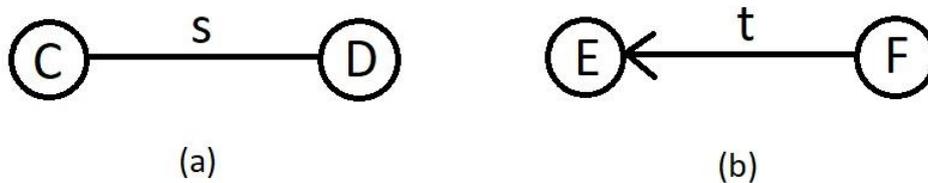


Figure 2.2: (a) Undirected edge, (b) Directed edge

Ganganath, et al. (2016) stated the LPOP problem as the following. If a print segment is defined as an undirected edge connecting two nodes, the task is to find a fast path from a predetermined start point to a predetermined end point that travels through all print segments.

However, the problem definition differs slightly from author to author. While Ganganath, et al. (2016) defines the cost of each edge as the duration taken for the nozzle to travel along it based on their model (not directly proportional to length), Fok, et al. (2016) instead defines it simply as the length of the segment.

Meanwhile, for start and end points, Fok, et al. (2016) requires that the start and end point be the same, meaning that the nozzle needs to return to the start point after visiting all the print segments. On the other hand, Ganganath, et al. (2016) allows for the start and end point to be located separately.

Since the problem definition may have a significant impact on the approach and solution to the LPOP, it is vital that the problem is clearly defined before any further progress is made.

### 2.2.1 Travelling Salesman Problem (TSP)

The travelling salesman problem (TSP) is a well-known problem in graph theory. Given an undirected graph  $G$ , the objective is to find a tour (i.e. a cycle) of minimum cost that visits each node exactly once (Held and Karp, 1970). Alternatively, if stated in layman's terms, given a set of cities (represented by nodes) and the distances between them (edges and their respective cost), the salesman needs to search for the shortest path that visits each city once, then returns to the starting city. The TSP is a NP-hard problem.

According to Ganganath, et al. (2016), the LPOP is closely related to the TSP, but differs in three ways. First, the problem focuses on linking existing edges, not nodes. Second, it does not need to return to the starting node, meaning the solution is not a cycle. Third, the goal is to reduce the total time taken to traverse the path, not the total length of the path.

However, as noted earlier, Fok, et al. (2016) reports a different definition of the LPOP that disagrees with the second and third points.

Nevertheless, due to the similarities between the LPOP and TSP, both Ganganath, et al. (2016) and Fok, et al. (2016) studied algorithms that were originally used to solve the TSP, then modified them to solve the LPOP instead. Namely, the nearest-neighbour algorithm and Christofides' algorithm.

### 2.2.2 Undirected Rural Postman Problem (URPP)

The undirected rural postman problem (URPP), a derivative of the TSP, was first formulated by Orloff (1974). An undirected graph  $G = (V, E)$  is given, where  $V$  is the set of nodes and  $E$  is the set of edges. There also exists  $E_r \subseteq E$ , where  $E_r$  is the set of required edges. The goal is then to find a tour that visits all the edges in  $E_r$  while minimizing the total cost. The URPP has been proven to be NP-hard given that  $E_r \neq E$ .

Fok, Cheng and Tse (2017) noted that the LPOP could be represented in terms of the URPP by making the print segments the required edges. Meanwhile, the possible transition segments that linked print segments would make up the remaining edges. Thus, they were able to use Frederickson's algorithm, an algorithm for solving the URPP, to solve the LPOP instead.

Later on, Fok, et al. (2018) cited a paper by Pérez-Delgado (2010) that reported the URPP could be transformed into a TSP. From the original graph  $G$

(representing the URPP), a new graph  $G' = (V', E')$  is built (in the form of the TSP).

Each required edge  $(i, j) \in E_r$  is replaced by three nodes,  $s_{ij}$  and  $s_{ji}$  which are the side nodes, and  $m_{ij}$  which is the middle node. They can be visualized as being placed at equal intervals, as shown in Figure 2.3. The new set of nodes,  $V'$  is then defined as shown in Equation 2.1.

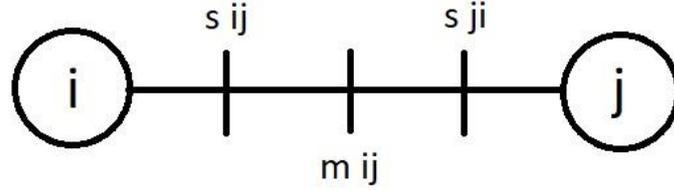


Figure 2.3:  $(i, j) \in E_r$  and the three replacement nodes

$$V' = \bigcup_{(i,j) \in E} \{s_{ij}, s_{ji}, m_{ij}\} \quad (2.1)$$

The new set of edges,  $E'$ , is thus based on the new set of nodes formed in  $V'$ . Equation 2.2 determines the cost between side nodes, where  $d(i, k)$  is the cost of the shortest path from node  $i$  to node  $k$  in the original graph  $G$ . It is important to note that there are no edges present that link any pair of side nodes directly, i.e. there is no edge linking  $s_{ij}$  and  $s_{ji}$  directly. They must pass through the middle node  $m_{ij}$ .

$$c'(s_{ij}, s_{kl}) = \begin{cases} \frac{1}{4}(c_{ij} + c_{kl}) + d(i, k) & \text{if } (i, j) \neq (k, l) \\ 0 & \text{if } (i, j) = (k, l) \end{cases} \quad (2.2)$$

Equation 2.3 then determines the cost between the middle node and another node. The cost will only be a finite value if the other node is one of the side nodes belonging to the same original edge  $(i, j)$ . This ensures that the middle node  $m_{ij}$  can only be accessed through its side nodes  $s_{ij}$  and  $s_{ji}$ .

$$c'(m_{ij}, v) = \begin{cases} \frac{1}{4}c_{ij} & \text{if } v = s_{ij} \text{ or } v = s_{ji} \\ \infty & \text{otherwise} \end{cases} \quad (2.3)$$

When the TSP is solved for the transformed graph  $G'$ , any reasonable solution will have a sequence of node triplets, where the node triplet corresponds to the sequence  $s_{ij} \rightarrow m_{ij} \rightarrow s_{ji}$  or  $s_{ji} \rightarrow m_{ij} \rightarrow s_{ij}$ . Since it is known that each node triplet has an associated edge  $(i, j)$ , the TSP solution can be converted back into a URPP solution by replacing each node triplet with its corresponding edge. If the first node of the node triplet is  $s_{ij}$ , the path is from  $i$  to  $j$ ; otherwise if it is  $s_{ji}$ , the path is from  $j$  to  $i$ .

This transformation process is a significant discovery, since it means that TSP algorithms are able to be used to solve the LPOP by transforming the problem, instead of the complicated process of modifying the algorithm. Since far more research has been done on the TSP compared to the URPP, recent advances in the TSP could hold great potential for further optimizing the LPOP.

### 2.3 Algorithms for Solving the LPOP

As demonstrated previously, the LPOP is closely related to the TSP, and can be represented in the form of the URPP. Unfortunately, since the TSP and URPP are NP-hard, this implies that the LPOP is NP-hard as well. Thus, heuristic algorithms have to be used to tackle the LPOP, which do not focus on checking all possibilities for the best solution, but instead give a good approximation that is usually within a certain percentage of the optimal solution.

#### 2.3.1 Tour Construction Algorithms

Tour construction algorithms are algorithms that generate or construct a solution to the particular graph problem (e.g. TSP or URPP).

##### 2.3.1.1 Nearest neighbour

Arguably the most intuitive heuristic for the TSP is the nearest neighbour algorithm. The algorithm can be visualized as a salesman whose rule of thumb for choosing the next destination is to select the nearest location that has not yet been visited (Johnson, 1990).

An ordering  $c_{\pi(1)}, \dots, c_{\pi(N)}$  of the nodes is constructed, with the starting location  $c_{\pi(1)}$  chosen arbitrarily. Generally,  $c_{\pi(i+1)}$  is selected as the node  $c_k$  that minimizes the cost function  $\{d(c_{\pi(i)}, c_k) : k \neq \pi(j), 1 \leq j \leq i\}$ . The solution is obtained by traversing the nodes in the order generated, then returning to  $c_{\pi(1)}$  after having visiting node  $c_{\pi(N)}$ .

Since nearest neighbour is simple to implement and is able to obtain solutions relatively quickly, it is currently used in the open-source slicer software Cura to solve the LPOP (Fok, et al., 2016). However, other methods have been shown to obtain better solutions in practice, such as Christofides' algorithm.

### 2.3.1.2 Christofides

A popular algorithm for solving the TSP is an algorithm devised by Christofides (1976), now commonly called Christofides' algorithm, The Christofides heuristic is as follows. As can be seen in Figure 2.4, the graph to be solved is assumed to be a complete graph, in which every pair of nodes is connected by a unique edge.

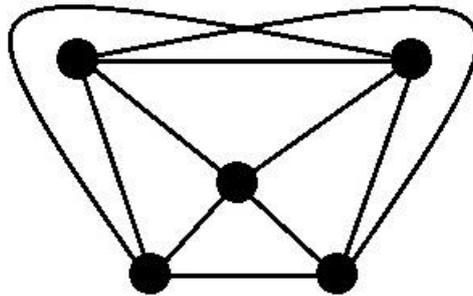
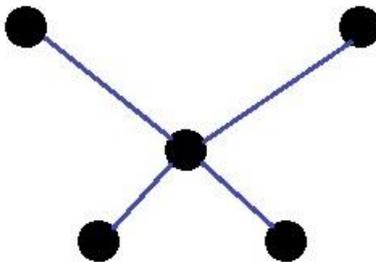
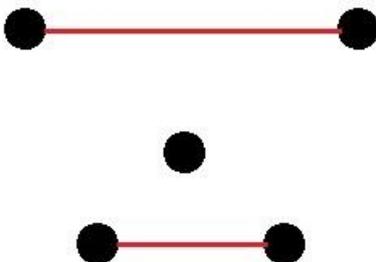


Figure 2.4: Complete graph

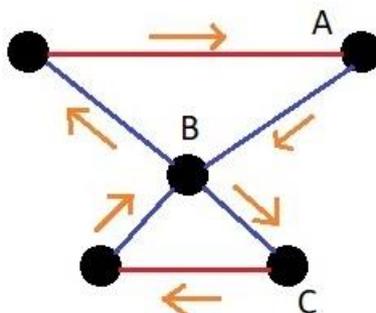
First, a minimum spanning tree  $T$  is constructed for the set of nodes, shown in Figure 2.5. This is the set of edges that connects all the nodes without forming a cycle and with the minimum total edge cost. It is noted that the cost, or length of this tree cannot be longer than the optimum TSP solution,  $OPT(I)$ , since removing an edge from the optimal tour will result in a spanning tree.

Figure 2.5: Minimum spanning tree  $T$ 

Next, a minimum-length perfect matching  $M$  is computed on the nodes of odd degree in  $T$ , where the degree of a node is the number of edges it is incident (linked) to. The resultant matching is shown in Figure 2.6. A matching is the set of edges which do not share any nodes. Subsequently, a perfect matching is a matching that includes all the nodes. It can be shown through a simple argument that by assuming the triangle inequality, the matching will not be longer than  $\text{OPT}(I)/2$ . The triangle inequality states that the shortest path to travel from node  $A$  to  $B$  is always the edge linking them directly.

Figure 2.6: Minimum-length perfect matching  $M$ 

By combining  $M$  with  $T$ , a connected multigraph is obtained in which every node has an even degree. A multigraph is a graph where edges are allowed to have the same start and end node. This graph must necessarily contain an Eulerian tour, which is a tour that traverses each edge exactly once but allows for nodes to be revisited. Figure 2.7 shows the graph of  $M \cup T$  and its Eulerian tour.

Figure 2.7: Euler tour in  $M \cup T$ 

A TSP tour (also known as a Hamiltonian tour) of equal or shorter length can be constructed from the multigraph by following the Euler tour and taking shortcuts where appropriate to avoid previously visited nodes. For example, if part of the Euler path is  $A \rightarrow B \rightarrow C$  but  $B$  had already been visited, the path  $A \rightarrow C$  is taken instead. Following the triangle inequality, this new path is equal to or shorter than the path it had replaced. However, care must be taken when choosing which shortcuts to take, as there may be several different possibilities. Figure 2.8 shows one such possible shortcut taken to get a TSP tour.

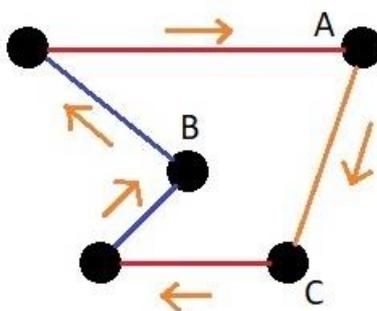


Figure 2.8: TSP tour obtained by taking shortcuts

As demonstrated previously, the worst-case solution for Christofides' algorithm is  $\text{OPT}(I) + \text{OPT}(I)/2 = 3 \text{OPT}(I)/2$ , or in other words a worst-case ratio of 1.5, which is one of the best for TSP heuristics. It also performs better than the nearest neighbour algorithm on average when solving the LPOP, as reported by Fok, et al. (2016) and Ganganath, et al. (2016).

### 2.3.1.3 Frederickson

Frederickson's algorithm was first proposed by Frederickson (1979) to solve the URPP. It shares many similarities with Christofides' algorithm for the TSP, which has already been explained previously in Section 2.3.1.2. Thus, Frederickson's algorithm will be elaborated upon more briefly.

An undirected graph  $G = (V, E)$  and  $E_r \subseteq E$ , where  $E_r$  is the set of required edges, are given. A minimum spanning tree  $T$  is then constructed to link all the edges in  $E_r$ , and the edges used in the construction form a new set  $E_T$ .

Next, a minimum cost perfect matching  $M$  is conducted on the nodes in the graph  $E_r \cup E_T$  with an odd degree. The edges added in this process produce another new set  $E_M$ .

An Eulerian tour that visits all the edges in  $E_r$  can then be easily found for the graph  $E_r \cup E_T \cup E_M$ . The tour can be further optimized by replacing consecutive edges on the tour that are not in  $E_r$  with shortcuts.

Fok, Cheng and Tse (2017) showed that by representing the LPOP as a URPP, Frederickson's algorithm could be applied to solve the LPOP. Similar to Christofides' algorithm, it was found that Frederickson's algorithm outperformed the built-in nearest neighbour algorithm in Cura when solving the LPOP.

### 2.3.1.4 Ant Colony Optimization (ACO)

The ant colony optimization (ACO) algorithm is inspired by the behaviour of ants in nature. ACO is widely used to solve the TSP, and is also capable of solving the URPP by transforming it into the TSP first (Pérez-Delgado, 2010).

In real life, ants communicate through the use of a substance called pheromone. This substance is deposited on the ground as they walk to serve as a sort of marking. Ants will tend to follow the path with more pheromone, depositing more of their own pheromone at the same time. Since pheromone evaporates over time, less visited paths will become even less likely to be visited in the future.

The version of the ACO algorithm that will be explained is taken from Fok, et al. (2018). Here,  $\tau_{i,j}$  signifies the pheromone level for an edge  $(i, j)$ , in

which  $i$  and  $j$  are the two nodes connected by that edge. Heuristic information is also taken into account through  $\eta_{i,j}$ , which is inversely proportional to the cost of the edge  $(i, j)$ .

If the  $k^{\text{th}}$  ant is now at node  $i$ , the probability of the ant to take the edge  $(i, j)$  is expressed in Equation 2.4.  $N_i^k$  is the set of valid nodes that are directly linked by a single edge to node  $i$  and have not yet been visited by the  $k^{\text{th}}$  ant. Meanwhile, the parameters  $\alpha$  and  $\beta$  are used for manipulating the behaviour of the ants.

$$p_{i,j}^k(t) = \frac{[\tau_{i,j}(t)]^\alpha [\eta_{i,j}]^\beta}{\sum_{l \in N_i^k} [\tau_{i,l}(t)]^\alpha [\eta_{i,l}]^\beta} \quad (2.4)$$

In order to avoid premature convergences and local optimum points, the evaporation concept is implemented into ACO. At the end of each iteration, the pheromone level of all the edges is reduced proportionally based on the factor  $\rho \in (0,1)$ . Consequently, the pheromone level  $\tau_{i,j}$  is updated at the end of the  $t^{\text{th}}$  iteration as shown in Equation 2.5.

$$\tau_{i,j}(t+1) = (1 - \rho) \cdot \tau_{i,j}(t) + \sum_{k=1}^m \Delta\tau_{i,j}^k(t) \quad (2.5)$$

In the equation,  $m$  represents the number of ants used in each iteration of the ACO. Next,  $\Delta\tau_{i,j}^k(t)$  signifies the amount of pheromone that is deposited by the  $k^{\text{th}}$  ant on edge  $(i, j)$  in the  $t^{\text{th}}$  iteration. As the process continues iteratively, the evaporation factor  $\rho$  will cause edges included in poor solutions to be eliminated in the long run.

The process is repeated until the prefixed number of iterations has been reached or the solution converges. Fok, et al. (2018) first demonstrated the effectiveness of the ACO algorithm when applied to the LPOP. However, an inherent flaw of the ACO not present in previously discussed algorithms is that due to the presence of randomness, the solution obtained may be different each time the ACO algorithm is used on the same problem.

## 2.3.2 Improvement Algorithms

Improvement algorithms use complete solutions generated by tour construction algorithms as a base to improve upon. That is to say; their job is to further optimize existing solutions.

### 2.3.2.1 $k$ -opt

$k$ -opt is part of the group of improvement algorithms known as local search algorithms. Local search algorithms work by repeatedly performing operations that shorten the length of the current solution until no operation results in an improvement, generating what is called a locally optimal tour.

Croes (1958) first proposed the 2-opt algorithm. The operation in this algorithm removes two edges from the tour, thus splitting it into two separate paths. Those paths are then reconnected in the other possible way. If the new tour is shorter than the old tour, the new connections are kept; otherwise the substitution is not performed. Figure 2.9 shows an example of this process. First, in Figure 2.9 (a), the edges  $(A, D)$  and  $(C, B)$  are removed, then in Figure 2.9 (b) the nodes are reconnected with the edges  $(A, B)$  and  $(C, D)$  to form a new cycle. Since the new tour is shorter than the old tour, the new tour is kept.

Subsequently, 3-opt replaces up to three edges, as demonstrated in Figure 2.10 (Bock, 1958; Lin, 1965). Initially, Figure 2.10 (a) shows three pairs of edges that can be removed; namely edges  $(A, F)$ ,  $(B, D)$  and  $(C, E)$ . Figure 2.10 (b) shows one way of reconnecting the nodes by using edges  $(A, B)$ ,  $(C, D)$  and  $(E, F)$ . On the other hand, Figure 2.10 (c) shows another way of joining back the nodes by using edges  $(A, B)$ ,  $(C, F)$  and  $(D, E)$ . This demonstrates that for 3-opt, it is possible to replace the edges in more than one way.

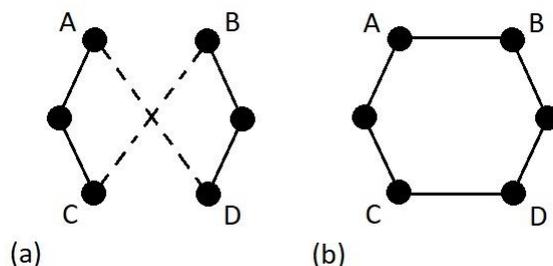


Figure 2.9: (a) Initial graph, (b) After 2-opt move

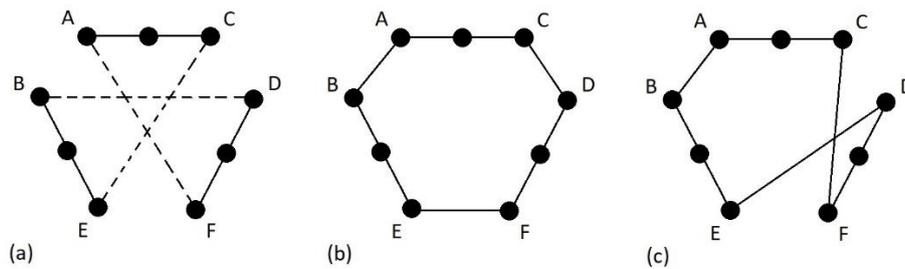


Figure 2.10: (a) Initial graph, (b) & (c) Two possible 3-opt moves

By following this line of logic, any integer value  $k$  could theoretically be used to implement  $k$ -opt, but practically only 2-opt and 3-opt are used due to the rapidly growing complexity of the operations as  $k$  increases.

Although  $k$ -opt is simple to implement, its weakness which is shared by all local search algorithms is that it may become stuck in a local minima, and is unable to search for better solutions that may exist globally. Nevertheless, several papers have shown the effectiveness of using  $k$ -opt algorithms to improve upon LPOP solutions (Ganganath, et al., 2016; Fok, Cheng and Tse, 2017; Fok, et al., 2019).

### 2.3.3 Application-specific Methods

In literature, there have been a number of proposed methods that are specifically tailored for solving the LPOP. These methods generally build on existing algorithms while exploiting the unique properties of the LPOP to achieve improved solutions.

#### 2.3.3.1 Refinement Process

The refinement process was proposed by Fok, Cheng and Tse (2017). It focuses on exploiting two characteristics of the LPOP as defined by the authors.

The first is that even though zero cost transitions are uncommon in the ordinary URPP, they are necessary in the LPOP, as curves are made of many straight print segments with zero cost transitions between them since they are connected directly. The second characteristic is that the printing nozzle does not need to return to its starting point after it has completed all the print segments on a layer.

The LPOP is formulated as an URPP, and is subsequently solved by Frederickson's algorithm. The solution obtained is then put through the proposed refinement process. To take advantage of the first characteristic, a priority list is created for the  $k$ -opt heuristic, such that transitions with higher priority will be evaluated first. Since it is very unlikely that performing  $k$ -opt on the zero cost transitions would yield an improvement, they are given lower priorities on the list.

The second part of the process changes the order of the path by removing the requirement to return to the starting node. The initial cycle is first split at the node in the cycle that is closest to the starting node, such that a chain  $C = \{(va_1, vb_1), (va_2, vb_2), \dots, (va_i, vb_i), \dots, (va_n, vb_n)\}$  is formed, where  $(va_i, vb_i)$  are the nodes of the  $i^{\text{th}}$  print segment in the chain. Then, in the  $p^{\text{th}}$  iteration of this process,  $C$  is divided into two subchains  $C_1 = \{(va_1, vb_1), \dots, (va_{p+1}, vb_{p+1})\}$  and  $C_2 = \{(va_{p+2}, vb_{p+2}), \dots, (va_n, vb_n)\}$ . If the joint cost of  $C_1$  and  $C_2$  flipped =  $\{(vb_n, va_n), \dots, (vb_{p+2}, va_{p+2})\}$  is less than  $C$ , the order of  $C$  is updated such that  $C \leftarrow \{C_1, C_2 \text{ flipped}\}$ , otherwise  $C$  remains unchanged. This process then continues for each iteration until the end of the chain.

It was reported by the authors that by using the proposed refinement process, performance was increased significantly compared to Frederickson's algorithm followed by 2-opt. The total transition length of the solution obtained was always shorter, while the post-processing time was reduced by about 34.50% on average. This paper is one of the first to demonstrate the importance of considering the unique properties of the LPOP to further increase performance.

### 2.3.3.2 Extended ACO

Fok, et al. (2018) proposed the extended ACO method to exploit a unique property of typically sliced 3D models. In the 3D printing process, the desired 3D model is first sliced by a slicer program into very thin slices, and each slice is then transformed into a layer with print segments. Due to this, adjacent layers will usually have similar structures in terms of the locations and orientations of print segments.

Before further discussion, a few terms that will be used must be made clear. Typically in ACO, the number of iterations to be executed, which is termed as  $N_{\text{ite}}$ , is defined beforehand by the user and is fixed throughout the entire optimization process. Since ACO uses a stochastic mechanism to check for improvements, it is possible that for some of the iterations, no further improvement can be found. For convenience's sake, the authors have coined the term *effective iteration* to mean an iteration in which improvements can still be found. Thus, the total number of effective iterations of a sliced model on its  $i^{\text{th}}$  layer is denoted as  $N_{\text{eff}}^i \in [0, N_{\text{ite}}]$ .

Empirical studies that were conducted by the authors on various 3D models suggested that  $N_{\text{eff}}^i$  usually approximates to  $N_{\text{eff}}^{(i-1)}$ , and that ACO is usually unable to achieve further improvement on the  $i^{\text{th}}$  layer when  $N_{\text{eff}}^i > N_{\text{eff}}^{(i-1)}$ . These properties can then be utilized to adaptively adjust the number of iterations used. However, there exist exceptional cases whereby there is a large deviation between  $N_{\text{eff}}^i$  and  $N_{\text{eff}}^{(i-1)}$ , implying that the number of iterations for each of the two layers should be different in this case to optimize the ACO process.

The extended ACO begins by checking for the condition  $N_{\text{eff}}^i > N_{\text{eff}}^{(i-1)}$  on the  $i^{\text{th}}$  layer. If the condition is met and no improvement is made on the current iteration, the process will terminate early. When the process terminates early, meaning  $N_{\text{ite}}^i < N_{\text{ite}}$ , the extra iterations are stored in a global reserve  $N_{\text{res}}$ .

However, if the condition is met and there is improvement for the current iteration, the optimization process for the  $i^{\text{th}}$  layer is allowed to continue as long as improvement is achieved for the current iteration. The number of iterations is only allowed past  $N_{\text{ite}}$  if  $N_{\text{res}} > 0$ , and every subsequent iteration will consume one iteration in  $N_{\text{res}}$ .

The idea behind such a design is to adaptively allocate the iteration numbers across the layers, such that less complex layers use up less iterations, while the surplus computational resources can be used for the more complex layers.

When the performance of extended ACO was compared to the generic ACO, Fok, et al. (2018) reported very similar build times for both the solutions,

while the extended ACO saved on average 8.20% of post-processing time. In other words, the extended ACO was able to cut down on post-processing time without comprising on the quality of the solution.

### 2.3.3.3 Detour Searching Algorithm with Modified $k$ -opt

One of the most recent approaches was proposed by Fok, et al. (2019), which exploits a particular property of 3D printing. Retraction is an important feature in FFF 3D printers. It involves retracting the filament a short distance to minimize the oozing of material when transitioning from one print segment to another.

However, as will soon be explained, retraction is not always necessary for every transition move. Transition segments can be divided into two groups, those with retraction and those without, as can be seen in Figure 2.11. Since transition A in the figure lies wholly within the model, retraction is not necessary, as the oozing will occur on the inside, and will not be visible from the outside after the model has been completed. On the other hand, a portion of transition B passes through the outside of the model, thus retraction is required to minimize the visible oozing.

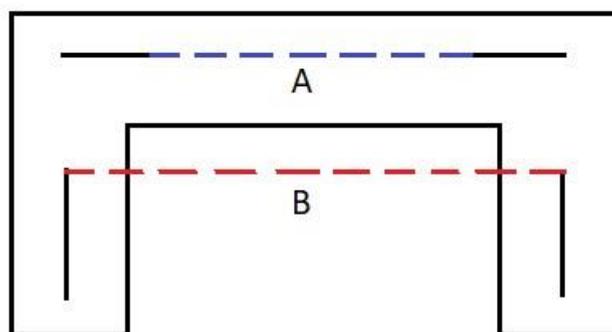


Figure 2.11: Transition without retraction (A), and with retraction (B)

As the retraction process takes up time and only minimizes the oozing but does not prevent it entirely, it is desirable to reduce the number of transition moves with retraction wherever possible.

Two algorithms were proposed by the authors to reduce the printing time. The first is a detour searching algorithm, which is a heuristic algorithm that uses

detours to replace unnecessary transitions that contain retraction. These transitions are prioritized based on their respective cost, such that higher cost transitions are considered first. As the detour searching algorithm itself is fairly complex, it will not be further elaborated upon here. Those interested in the details of the algorithm may refer to the original paper published by Fok, et al. (2019).

The second algorithm is a modified  $k$ -opt heuristic. The edges that do not have retraction are divided into two subsets of  $E$ , namely  $E_B$  and  $E_C$ .  $E_B$  contains the transitions with distance greater than zero, while  $E_C$  contains the transitions with zero distance. Based on two theorems developed by the authors, any combinations which are either made up of one transition in  $E_B$  and  $(k - 1)$  transitions in  $E_C$ , or only made up of transitions in  $E_C$  are not considered in the  $k$ -opt process, as they will yield no improvement. This idea is similar to part of the refinement process proposed by Fok, Cheng and Tse (2017) that has been discussed in Section 2.3.3.1.

Since the two proposed algorithms are improvement algorithms, they require an initial solution to improve on. In the paper, it is stated that the complete proposed method begins with Frederickson's algorithm first, followed by the detour searching algorithm, and finally the modified  $k$ -opt algorithm.

When the results of the proposed method were compared with an extensive number of other algorithms, it was found that the proposed method always obtained the shortest print time, while requiring significantly less post-processing time than most of the other algorithms which obtained similarly short print times. In terms of the final print quality, the proposed method performed better than Cura when they were compared qualitatively, due to the stark decrease in transitions with retraction.

These results make it the first paper (to the knowledge of this report's author) of a proposed method that managed to demonstrate improvement both in terms of print time and the final print quality.

## 2.4 Summary

The LPOP can be formally expressed using graph theory, and thus can be represented as an URPP, which is closely related to the well-known TSP. Since it has been shown that an URPP can be converted into a TSP, and the solution

obtained can be converted back into a URPP solution, algorithms for solving the TSP and URPP could both potentially be used to solve the LPOP.

While there exists a variety of tour construction algorithms and improvement algorithms, the approach with the most promise appears to be developing or modifying improvement algorithms that exploit the unique properties present in the LPOP. A number of papers have demonstrated that this approach yields better solutions when compared with generic algorithms, as can be observed in Section 2.3.3.

However, a noticeable gap in the literature is research into the application of some commonly-used metaheuristic algorithms such as simulated annealing and tabu search for solving the LPOP. Thus, future research into these untested algorithms is a possible path forward.

## CHAPTER 3

### METHODOLOGY AND WORK PLAN

#### 3.1 Introduction

A process must be formulated to test the algorithms and compare their performance in order to achieve the aim of this report, which is to reduce the 3D printing time by utilizing algorithms to solve the LPOP. This process is illustrated in Figure 3.1. The details of the process will be further explained in the subsequent portion of this chapter.

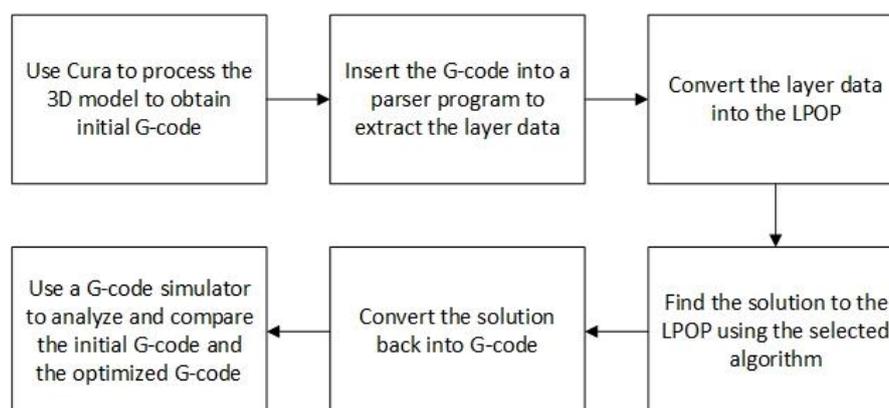


Figure 3.1: Flowchart of the process to evaluate LPOP algorithms

#### 3.2 Cura

Cura is a popular open-source slicer software developed and distributed by Ultimaker and is available to download for free on Ultimaker's website (Ultimaker, n.d.). Due to its widespread use, it is a useful baseline to compare the performance of the algorithms used.

A slicer software is a piece of software used to process a 3D model in order to obtain the corresponding G-code for that model. Countless parameters are available for tweaking within the slicer software. Such parameters include the dimensions and specifications of the 3D printer, the scaling and orientation of the 3D model, layer height, supports, etc. In general, the tuning of these parameters will affect the final appearance and quality of the print, as well as the duration of the 3D printing process. A sample of the user interface in Cura is shown in Figure 3.2.

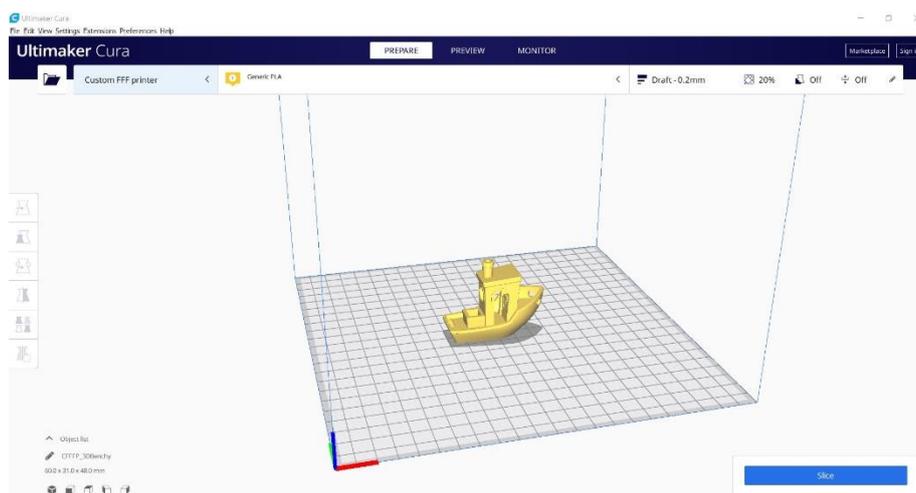


Figure 3.2: Cura user interface

For this report, Cura version 4.4 was used with the default settings, but with some minor adjustments. The infill was set to 20%, the layer height was set to 0.2 mm, and the relative extrusion mode was enabled. The first two adjustments are to create a standard for the testing procedure, while the last setting modifies the way the G-code is written to make it easier to post-process, as will be explained in Section 3.3. Meanwhile, the retraction setting is also disabled as stated in the scope and limitations of the study.

### 3.3 G-code

G-code is the language used to program the motion of computer numerical control (CNC) machines. It has long been in use for subtractive manufacturing CNC machines such as lathes and drills, but has only recently been adapted for use in additive manufacturing CNC machines, also known as 3D printers.

3D printing technology had been commercially available for many decades, but it was restricted to mostly industrial use due to the high costs associated with it. This changed circa 2010, when the RepRap project, an open-source effort to develop consumer 3D printers, gained public attention (Jones, et al., 2011). Since then, many 3D printing related open-source projects have taken off, with 3D printing start-up companies using technology and code from these open-source projects as a starting point for developing their own products.

As a result of this unusual origin of consumer 3D printing technology, most consumer-level and even some commercial-level 3D printers use modified open-source firmware. Thus, for this report, two open-source 3D printer firmware projects, namely RepRap and Marlin, will be referred to for the definition of the G-code commands (Marlin Firmware, n.d.; RepRap, n.d.). The RepRap source shows a comparison of the different definitions of the G-code according to various firmware, while Marlin is a widely-used firmware. Fortunately, the G-code definitions are generally consistent between firmware, but in the case of any discrepancy the definition from Marlin will be used.

Although there are many commands in the G-code language, only two of them will be used throughout the vast majority of the printing process, which are G0 and G1. Both of these are move commands and perform the same action, but by convention G0 means a rapid move (i.e. non-extruding) while G1 means a linear move (i.e. extruding) (Marlin Firmware, n.d.). Parameters as shown in Table 3.1 are added behind G0/G1. At least one parameter is required, but a G0/G1 command may include all of them.

Table 3.1: Parameters for G0 and G1

<b>Parameter</b>	<b>Description</b>
F<rate>	The movement rate, also known as feedrate, from the start to end point. The value set here will apply to subsequent moves that do not have this parameter.
X<pos>	The X-coordinate of the position to move to.
Y<pos>	The Y-coordinate of the position to move to.
Z<pos>	The Z-coordinate of the position to move to.
E<pos>	The length of filament extruded from the start to end point.

Some examples of G0 and G1 code are given in Table 3.2. Note that text after a semicolon means a comment. Coordinates will be discussed in the format (x, y).

Table 3.2: Examples of G0 and G1 commands

Example	Code
<b>A</b>	G0 F1200 X3.4 Y15.3 G0 X50.6 Y84
<b>B</b>	G0 F1600 G0 X23.1 Y56.7 G1 X34 Y67.1 E10.5
<b>C</b>	G0 F1600 X45.3 Y55 G1 X33.1 Y68.8 E5.4 G1 F1000 E-5 ; retraction move G0 F1600 X12 Y45

In example A, the first line tells the nozzle to move to the coordinates (3.4, 15.3) in mm at a feedrate of 1200 mm/minute. The second line then moves the nozzle to (50.6, 84). Since this line has no F parameter, it will retain the same feedrate of 1200 mm/minute.

In example B, the first line sets the feedrate of subsequent moves to 1600 mm/minute. The second line moves the nozzle to (23.1, 56.7). The third line then moves the nozzle to (34, 67.1) while extruding 10.5 mm of filament evenly between the two points. The feedrate of the extrusion itself is calculated by the firmware to ensure that an even amount of material is deposited along the path.

In example C, the first line moves the nozzle to (45.3, 55) at a feedrate of 1600 mm/minute, then the second line moves the nozzle to (33.1, 68.8) at the same feedrate while extruding 5.4 mm of filament. Next, the third line retracts 5 mm of filament at a feedrate of 1000 mm/minute, since it has a negative sign in the E parameter. This type of move is termed as a retraction move, and its purpose is briefly explained in Section 2.3.3.3. Lastly, the nozzle moves to (12, 45) at a feedrate of 1600 mm/minute.

In summary, understanding G0 and G1 instructions will be sufficient enough to construct the parser program.

### 3.4 Parser Program and Conversion Back to G-code

The job of the parser program is to extract the layer data from the initial G-code and convert it into the LPOP. As an example, say that the output given by the

slicer program for the first layer is as shown in Figure 3.3. Solid lines represent the print segments, while the dotted lines represent the transition segments.

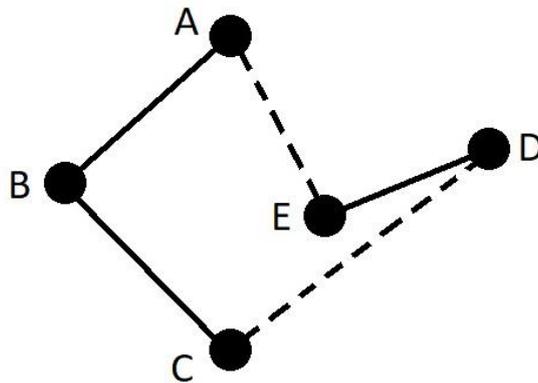


Figure 3.3: Output of the slicer program for the first layer

The corresponding G-code may be something similar to the code shown in Table 3.3, where  $\langle x_A \rangle$  and  $\langle y_A \rangle$  are the respective X-coordinate and Y-coordinate of node A. Meanwhile,  $\langle e_{AB} \rangle$  is the length of filament extruded from node A to node B and is equivalent to  $\langle e_{BA} \rangle$ . A transition move is the same as a non-extruding move.

Table 3.3: Initial G-code for the first layer

<pre>G0 X&lt;x_A&gt; Y&lt;y_A&gt; ; move to node A G1 F1500 X&lt;x_B&gt; Y&lt;y_B&gt; E&lt;e_AB&gt; ; move to node B while extruding G1 X&lt;x_C&gt; Y&lt;y_C&gt; E&lt;e_BC&gt; ; move to node C while extruding G0 F3200 X&lt;x_D&gt; Y&lt;y_D&gt; ; do a transition move to node D G1 F1200 X&lt;x_E&gt; Y&lt;y_E&gt; E&lt;e_DE&gt; ; move to node E while extruding G0 F3200 X&lt;x_A&gt; Y&lt;y_A&gt; ; do a transition move to node A</pre>
--

The parser first needs to identify all the nodes and their coordinates, which is then stored in a table as shown in Table 3.4. Using this information, the Euclidean distance is calculated for all possible pairs of nodes using the Pythagorean theorem. For example, the distance between node A and B is denoted as  $\langle d_{AB} \rangle$  or  $\langle d_{BA} \rangle$ . In other words, the distance of each possible edge is calculated.

Table 3.4: Coordinates of each node

<b>Node</b>	<b>X-coordinate</b>	<b>Y-coordinate</b>
<b>A</b>	<x <sub>A</sub> >	<y <sub>A</sub> >
<b>B</b>	<x <sub>B</sub> >	<y <sub>B</sub> >
<b>C</b>	<x <sub>C</sub> >	<y <sub>C</sub> >
<b>D</b>	<x <sub>D</sub> >	<y <sub>D</sub> >
<b>E</b>	<x <sub>E</sub> >	<y <sub>E</sub> >

Next, the feedrate and extrusion length for each edge has to be extracted from the G-code. Generally, all transition moves will have the same feedrate, so the value of the transition feedrate used in the G-code will be applied to all the other possible transition segments.

All the data regarding the edges is then formatted as a matrix, as shown in Table 3.5. The data is ordered as distance, feedrate and extrusion length. A dash symbol (-) means that the particular parameter is not applicable. For example, non-extrusion moves will not have any extrusion length.

Table 3.5: Data matrix for the edges

<b>Edge</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
<b>A</b>	0, -, -	<d <sub>AB</sub> >, 1500, <e <sub>AB</sub> >	<d <sub>AC</sub> >, 3200, -	<d <sub>AD</sub> >, 3200, -	<d <sub>AE</sub> >, 3200, -
<b>B</b>	<d <sub>AB</sub> >, 1500, <e <sub>AB</sub> >	0, -, -	<d <sub>BC</sub> >, 1500, <e <sub>BC</sub> >	<d <sub>BD</sub> >, 3200, -	<d <sub>BE</sub> >, 3200, -
<b>C</b>	<d <sub>AC</sub> >, 3200, -	<d <sub>BC</sub> >, 1500, <e <sub>BC</sub> >	0, -, -	<d <sub>CD</sub> >, 3200, -	<d <sub>CE</sub> >, 3200, -
<b>D</b>	<d <sub>AD</sub> >, 3200, -	<d <sub>BD</sub> >, 3200, -	<d <sub>CD</sub> >, 3200, -	0, -, -	<d <sub>DE</sub> >, 1200, <e <sub>DF</sub> >,
<b>E</b>	<d <sub>AE</sub> >, 3200, -	<d <sub>BE</sub> >, 3200, -	<d <sub>CE</sub> >, 3200, -	<d <sub>DE</sub> >, 1200, <e <sub>DF</sub> >,	0, -, -

Using Table 3.4 and Table 3.5, the LPOP for this layer can be represented as a URPP, as discussed in Section 2.2.2. For this example, given that  $G = (V, E)$  and  $E_r \subseteq E$ , the elements of the sets are described in Table 3.6, where  $(A, B)$  is an edge connecting node A and B.

Table 3.6: Elements of  $V, E$  and  $E_r$

Set	Elements
$V$	A, B, C, D, E
$E$	(A, B), (A, C), (A, D), (A, E), (B, C), (B, D), (B, E), (C, D), (C, E), (D, E)
$E_r$	(A, B), (B, C), (D, E)

Since the length of each edge can be obtained from Table 3.5, the URPP can now be solved by the selected algorithm. As explained in Section 2.2.2, the URPP is able to be converted into the TSP and the solution can be converted back into a URRP solution, thus TSP algorithms can be used as well by utilizing this property.

Now suppose that the optimized solution obtained by the algorithm is shown in Figure 3.4. The parser program then needs to use information from the solution in Table 3.4 and Table 3.5 to construct the corresponding G-code. An example of the optimized G-code is shown in Table 3.7. A G-code simulator can then be used to analyse and compare various parameters of the initial G-code and optimized G-code.

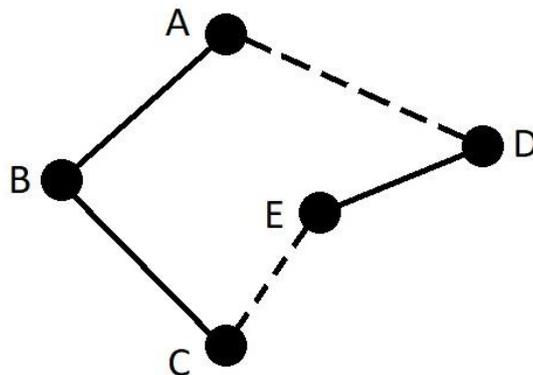


Figure 3.4: Optimized solution

Table 3.7: Optimized G-code

G0 X<x <sub>A</sub> > Y<y <sub>A</sub> >
G1 F1500 X<x <sub>B</sub> > Y<y <sub>B</sub> > E<e <sub>AB</sub> >
G1 X<x <sub>C</sub> > Y<y <sub>C</sub> > E<e <sub>BC</sub> >
G0 F3200 X<x <sub>E</sub> > Y<y <sub>E</sub> >
G1 F1200 X<x <sub>D</sub> > Y<y <sub>D</sub> > E<e <sub>DE</sub> >
G0 F3200 X<x <sub>A</sub> > Y<y <sub>A</sub> >

### 3.5 Algorithms to be Tested

Since the nearest neighbour algorithm is used in Cura as reported by Fok, et al. (2016), it will be excluded from the list of algorithms to be tested. The list of proposed algorithms to be tested is given in Table 3.8. The details of the algorithms will be elaborated in the subsequent sections.

Table 3.8: Algorithms to be tested

No.	Algorithm
1	Modified AS algorithm for solving URPP
2	Modified ACS algorithm for solving URPP

#### 3.5.1 ACO Algorithm Variants

The basis behind the ACO algorithm has been stated previously in Section 2.3.1.4. However, ACO actually refers to a family of algorithms that share the same ant-based idea, but use different techniques. The specific version of the ACO algorithm elaborated upon in Section 2.3.1.4 is the earliest version, first conceived by M. Dorigo, V. Maniezzo and A. Colorni (1996) and is known as the Ant System (AS).

Later, M. Dorigo and L. M. Gambardella (1997) proposed another ACO algorithm called the Ant Colony System (ACS). ACS differs from AS in three significant ways: the state transition rule, the local pheromone update and the global pheromone update. These differences will be discussed using the terminology used in Section 2.3.1.4.

The state transition rule is given in Equation 3.1. When the  $k$ th ant is at node  $i$ , the next node  $s$  is decided by the following rule. If  $q \leq q_0$ , where  $q$  is a

random number that is evenly distributed in  $[0...1]$  and  $q_0$  is a parameter between 0 and 1, then the node  $l$  that maximises the argument  $\{[\tau_{i,l}] \cdot [\eta_{i,l}]^\beta\}$  is chosen as the next node (called exploitation). Otherwise, if  $q > q_0$ , the same rule as in AS is used to choose the next node (called biased exploration). This state transition rule is known as the pseudo-random-proportional rule. The parameter  $q_0$  balances exploitation and biased exploration.

$$s = \begin{cases} \arg \max_{l \in N_i^k} \{[\tau_{i,l}] \cdot [\eta_{i,l}]^\beta\} & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases} \quad (3.1)$$

In AS, the path generation for each ant can be done sequentially or in parallel, since the pheromone is only updated after all the ants have completed their tours. However, in ACS the pheromone values change dynamically as the ants generate their path, so the path generation must be done in parallel. After each ant has added a new node to their path, the local pheromone update takes place according to Equation 3.2. In the equation,  $r$  is the current number of nodes in each ant's path,  $\varphi$  is the local evaporation factor, and  $\tau_0$  is the initial pheromone value. The reasoning behind the local pheromone update is to diversify the solutions obtained by evaporating a small amount of pheromone from an edge every time it is traversed by an ant, making that edge less desirable to be used by other ants later on within the same iteration. Without local updating, the ants would only explore paths very similar to the shortest previous tour.

$$\tau_{i,j}(r+1) = (1 - \varphi) \cdot \tau_{i,j}(r) + \varphi \cdot \tau_0 \quad (3.2)$$

Other than the local pheromone update, ACS also has the global pheromone update which happens at the end of every iteration  $t$ , shown in Equation 3.3. In the equation,  $\alpha$  is the global evaporation factor. Meanwhile, if the edge  $(i, j)$  is part of the globally best path, then  $\Delta\tau_{i,j}$  is the inverse of the best length (i.e. inverse length of the globally best path), otherwise  $\Delta\tau_{i,j}$  is zero. Put another way, pheromone will evaporate from all edges and only the best ant is allowed to deposit pheromone along the path it travelled.

$$\tau_{i,j}(t + 1) = (1 - \alpha) \cdot \tau_{i,j}(t) + \alpha \cdot \Delta\tau_{i,j} \quad (3.3)$$

### 3.5.2 Modifying AS and ACS Algorithms to solve the URPP

The AS and ACS algorithms were originally designed to solve the TSP. To use these algorithms to solve the URPP, previous authors such as Pérez-Delgado (2010) and Fok, et al. (2018) have used a transformation process to convert the original URPP into a TSP, and then to convert the obtained TSP solution into the URPP solution. This transformation process has been explained previously in Section 2.2.2.

In this report, a new method is proposed to solve the URPP by modifying the AS and ACS algorithms directly. In the original algorithms that solved for the TSP, the set of possible nodes that the ant could travel to next was simply  $A'$ , given that the universal set,  $U$  is the set containing all the nodes in the problem and  $A$  is the set of nodes that the ant has already visited. However, in the proposed method, the logic for determining the possible next nodes for the ant is changed.

As an example, say that the graph in Figure 3.5 (a) is the original URPP, where the solid lines are the required edges. When solving for any URPP, the assumption is made that each node is connected to a maximum of two required edges. Thus, each node can have zero, one, or two required edges. Nodes with zero required edges can be ignored, since the goal of the URPP is to connect all the required edges. This leaves the nodes with one or two required edges.

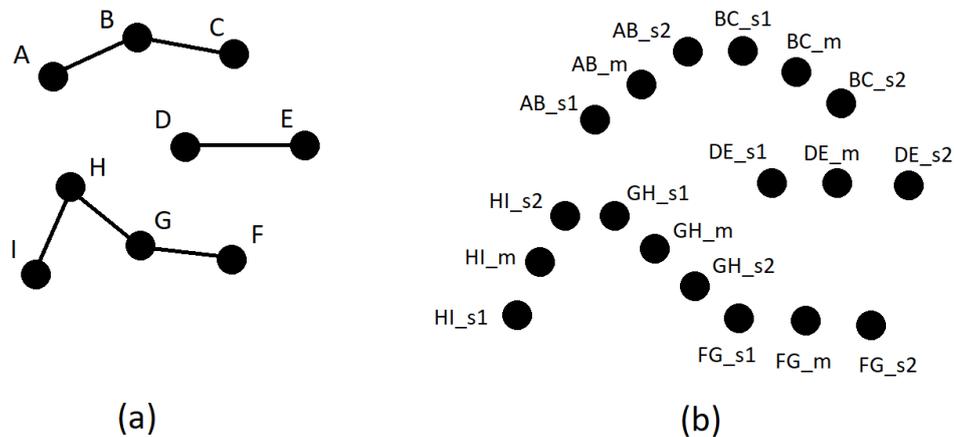


Figure 3.5: (a) Original URPP, (b) After transformed to TSP

To begin the modified AS/ACS algorithm, an ant randomly chooses a node with only one required edge. From Figure 3.5 (a), this means that the ant can start from node A, C, D, E, F, or I. Say that the ant starts at node C. Next, the ant will immediately choose node B, since it is connected to node C by a required edge. After that, the ant sees that node B is connected to both node A and C by required edges, but since node C has already been visited, the ant goes to node A. Now, at node A, it randomly chooses the next node according to the rules of the normal AS/ACS algorithm, where the set of possible next nodes are those with only one required edge that have not been visited, which is node D, E, I, and F. This process repeats until all the nodes have been visited, then the ant returns to the starting node. In other words, the ant will always follow a path made of required edges, then when it reaches the end of that path, it will randomly choose the start of another path of required edges according to the AS/ACS algorithm. This method of traversing the graph guarantees that all the required edges will be included in the final tour.

For the URPP in Figure 3.5 (a), it can be observed that if the modified AS/ACS algorithm is used as described above, the ant only has to make a random decision on which node to go next two times. On the other hand, Figure 3.5 (b) shows the graph after the URPP has been transformed to its TSP equivalent according to Section 2.2.2, where each required edge is represented as a node triplet. If the conventional AS/ACS algorithm is applied to the graph in Figure 3.5 (b), the ant would need to make a random decision on which node to go next 17 times.

Therefore, it can be clearly seen that the proposed modified AS/ACS algorithm requires significantly less processing to solve the URPP compared to using the original AS/ACS algorithm to solve the TSP equivalent of the URPP. Practically, this translates into a significant reduction in processing time. This improvement is even more apparent when the processing required to perform the initial transformation and the transformation of the obtained solution is taken into account.

Figure 3.6 illustrates the logic of the AS algorithm after it has been modified to solve the URPP, while Figure 3.7 shows the modified ACS algorithm instead.

```

1. Initialize pheromone data
2. for each iteration in all_iterations do
3.   for each ant in all_ants do
4.     starting_node ← random node with only one required edge
5.     current_node ← starting_node
6.     Append starting_node to ant_tour
7.     repeat
8.       if current_node has two required edges
9.         next_node ← the node connected by a required edge that is not
           in ant_tour
10.      if current_node has one required edge
11.        if node connected by required edge not in ant_tour
12.          next_node ← the node connected by required edge
13.        else
14.          possible_next_nodes ← nodes with only one required edge
           that are not in ant_tour
15.          next_node ← random node chosen from
           possible_next_nodes according to AS rule
16.          Append next_node to ant_tour
17.          current_node ← next_node
18.        until all nodes are in ant_tour
19.      best_tour ← globally shortest ant_tour so far
20.    Perform global pheromone update according to AS rule
21. return best_tour

```

Figure 3.6: Modified AS algorithm for solving URPP

```

1. Initialize pheromone data
2. for each iteration in all_iterations do
3.   for each ant in all_ants do
4.     starting_node[ant] ← random node with only one required edge
5.     current_node[ant] ← starting_node[ant]
6.     Append starting_node[ant] to ant_tour[ant]

```

```

7.  repeat
8.    for each ant do
9.      if current_node[ant] has two required edges
10.         next_node[ant] ← the node connected by a required edge that
            is not in ant_tour
11.      if current_node[ant] has one required edge
12.         if node connected by required edge not in ant_tour[ant]
13.            next_node[ant] ← the node connected by required edge
14.         else
15.            possible_next_nodes[ant] ← nodes with only one required
            edge that are not in ant_tour
16.            next_node[ant] ← random node chosen from
            possible_next_nodes according to ACS rule
17.            Append next_node[ant] to ant_tour[ant]
18.            current_node[ant] ← next_node[ant]
19.            Perform local pheromone update according to ACS rule
20.    until all nodes are in ant_tour for all ants
21.    best_tour ← globally shortest ant_tour so far
22.    Perform global pheromone update according to ACS rule
23. return best_tour

```

Figure 3.7: Modified ACS algorithm for solving URPP

### 3.6 Implementation of the Parser and Algorithms

The parser and algorithms to be tested were written in the Python programming language due to its ease of use and natural data-handling capabilities. Preliminary testing showed that the code for the parser and algorithm took a long time to process a single G-code file, usually around a few hours. It was expected that since Python is an interpreted language, it would have worse performance compared to compiled languages such as C and C++.

To decrease the processing time of the code, Numba, which is a Just-In-Time (JIT) compiler for Python, was integrated into the code so that it could achieve speeds similar to compiled languages. After Numba was added, the processing time of the code dropped by about 20 times, such that a G-code file could now be processed in a matter of minutes.

### 3.7 G-code Simulator

A G-code simulator is a software that virtually simulates each G-code command, allowing the user to see visualizations of each layer in detail as well as obtain useful metrics for the entire print process, such as the total distance travelled, the printing time, etc.

Two G-code simulators will be used, which are gCodeViewer and Gcode Analyser (hudbrog, 2018; syue87, 2018). The former is used to visualize the printing process for each layer since it shows the transition moves, while the latter is used to obtain useful metrics. Figure 3.8 shows the gCodeViewer user interface, while Figure 3.9 shows a screenshot from Gcode Analyser.

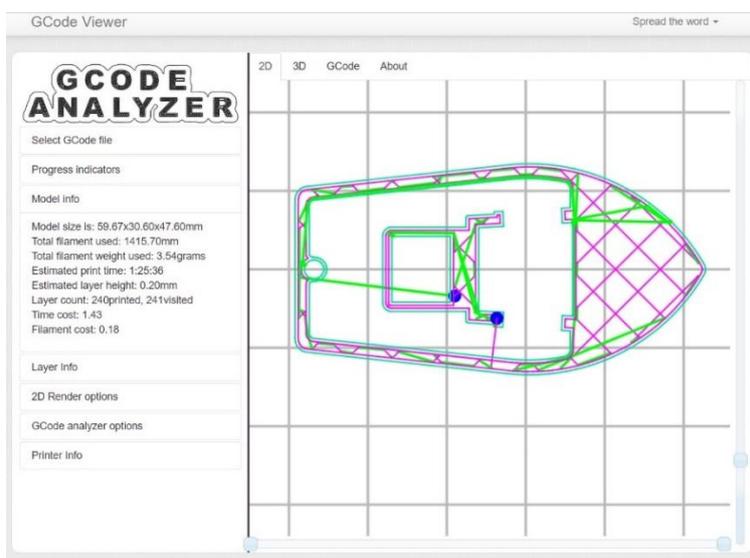


Figure 3.8: Screenshot from gCodeViewer

G-Code Statistics			
	Time	Speed	
Print Time	1:18:42	Average Speed	31.7 mm/s
Time Spent Accelerating/Decelerating	31:55 (40.6%)	Average Print Speed	32.3 mm/s
Time Spent at Target Speed	46:47 (59.4%)	Average Travel Speed	37.8 mm/s
Total Z Hop Time	0:00	Raw Filament Usage Rate	0.30 mm/s, 1.81 cm/min
Total Retract/Prime Time	4:05	Min / Max XY Feedrate	600 (10.00mm/s) / 7200 (120.00mm/s)
	Distance	Count	
Total Distance Moved	149.84 m	Number of Lines	104103
Print / Travel Move Distance	113.48 m (75.7%) / 36.35 m (24.3%)	Total Move Commands (G0 & G1)	101418
Distance Accelerating/Decelerating	54.41 m (36.3%)	Move Commands Reached Target Speed	58677 (57.9%)
Distance at Target Speed	95.44 m (63.7%)	Print Move / Travel Move	82296 (81.1%) / 18195 (17.9%)
Raw Filament Usage	1.42 m, 3.42 cm <sup>3</sup>	Z Hop Count	0
Printed Line Length per mm of Raw Filament	79.74 mm	Retraction Count	463

G-Code Statistics Per Axis				
	X	Y	Z	E
Times Move Speed Limited By Axis Max Speed	6144	5909	1	0
Total Distance Moved	96.82 m	84.85 m	0.08 m	7.44 m
Average Speed	20.5 mm/s	18.0 mm/s	0.0 mm/s	1.6 mm/s

Figure 3.9: Screenshot from Gcode Analyser

### 3.8 Summary

Before the testing can be carried out, the parser program and the part of the program that reconstructs the G-code have to be coded first. Subsequently, all the combinations of algorithms to be tested have to be written in code as well.

Once the coding is finished, a range of 3D models will be needed to serve as the sample set. Some 3D models used in prior research such as those by Fok, Cheng and Tse (2017), Fok, et al. (2018) and Fok, et al. (2019) will form part of the sample set, while the rest will be 3D models that can be easily obtained online.

The testing procedure can begin after the coding is completed and the sample set of models are chosen. The selected model is first processed through Cura to generate the initial G-code. The initial G-code is then parsed and optimized by the chosen algorithm to obtain the optimized G-code. Finally, the initial and optimized G-code are analysed by using the G-code simulators and the corresponding data is recorded. This is repeated for every algorithm in the list. After all the algorithms have been tested for the selected model, another model is chosen from the sample set of models. The process repeats until it has gone through the entire sample set.

## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Introduction

The purpose of this report is to investigate the use of improved algorithms to solve the layer path optimization problem (LPOP) in 3D printing to reduce the time needed to print an object. The LPOP is defined as finding the shortest path that the nozzle of the 3D printer should traverse in order to print the desired object.

The main hypothesis is that by reducing the path length of the nozzle, the time taken to traverse the nozzle path (i.e. time taken to print the object) will be decreased as well.

In Section 2.2.2, it was shown that the LPOP could be represented formally as the undirected rural postman problem (URPP), a known problem in graph theory. Another closely related graph theory problem is the travelling salesman problem (TSP).

Thus to solve the URPP, two algorithms that were previously created to solve the TSP, namely the ant system (AS) and ant colony system (ACS) algorithms, were modified to solve the URPP instead. The modifications were explained in Section 3.5.2.

To compare the performance of the modified AS and ACS algorithms with the widely-used 3D printing software (also known as a ‘slicer program’) Cura, a testing procedure was devised.

##### 4.1.1 Testing Process

Cura produces a file called a G-code file from a 3D model. The G-code file contains instructions on how the printer should print out the 3D model. To enable the modified AS and ACS algorithms to optimize the G-code file, a parser program was written.

The job of the parser program is to extract the relevant data from the G-code file and convert it into the URPP form, where it can be solved by the modified AS and ACS algorithms. Following that, the parser program uses the obtained solution to produce the optimized G-code.

By using the parser program, the performance of the modified AS and ACS algorithms can be compared with Cura. This is done by comparing the optimized G-code with the original G-code. A G-code simulator is used to analyse both the optimized and original G-code.

The first part of the testing procedure is the preliminary parameter testing phase in Section 4.2. The objective of this phase is to determine the best settings or parameters to be used for the modified AS and ACS algorithms.

The second part of the testing procedure is the main testing phase in Section 4.3. The best parameters for the algorithms that were determined in the first part will be used in the algorithms here. The objective of this phase is to compare the performance of the modified AS and ACS algorithms with Cura across a wide range of 3D models.

All processing and simulations were done on a computer with Intel i7-4720HQ processors, 8 GB RAM, Windows 10, Python 3.7.6, and Numba 0.48.0.

#### 4.1.2 Explanation of Terminology Used

The definitions of some of the terminology that will be used later on are explained in Table 4.1.

Table 4.1: Definitions of terminology used

Term	Definition
Cura	A widely-used 3D printing software, the standard by which the other algorithms are compared to, explained in Section 3.2
Modified AS algorithm	The algorithm that was modified from the original AS algorithm in order to solve the URPP and hence the LPOP, explained in Section 3.5.2
Modified ACS algorithm	The algorithm that was modified from the original ACS algorithm in order to solve the URPP and hence the LPOP, explained in Section 3.5.2
G-code	Instructions or code sent to the 3D printer in order to print out an object, explained in Section 3.3.
Parser program	The program that converts the original G-code into the optimized G-code using the chosen algorithm, explained in Section 3.4
Print time	Time taken for the entire object to be printed by the 3D printer
Travel distance	Distance of the path travelled by the nozzle of the 3D printer to print the entire object
Processing time	Time taken for the parser program and selected algorithm to optimize the original G-code file
Total time taken	The sum of the print time and processing time, i.e. the total time taken to optimize the G-code and print the object
Percentage travel distance reduced	The percentage that the travel distance was reduced by relative to the travel distance of Cura
Percentage print time reduced	The percentage that the print time was reduced by relative to the print time of Cura

## 4.2 Preliminary Parameter Testing Phase

The objective of this phase is to determine the best settings or parameters to be used for the modified AS and ACS algorithms.

The default parameters for the original AS algorithm are shown in Table 4.2. The number of iterations and ants are half of those used by Fok, et al. (2018), while the other parameters are obtained from M. Dorigo, V. Maniezzo and A. Coloni (1996).

Table 4.2: Default parameters for original AS

Parameter	Value
Number of iterations	50
Number of ants	50
Pheromone factor, $\alpha$	1
Heuristic factor, $\beta$	2
Global evaporation factor, $\rho$	0.1

Meanwhile, the default parameters for the original ACS algorithm are shown in Table 4.3. The number of iterations and ants are half of those used by Fok, et al. (2018), while the other parameters are obtained from M. Dorigo and L. M. Gambardella (1997).

Table 4.3: Default parameters for original ACS

Parameter	Value
Number of iterations	50
Number of ants	50
Pheromone factor, $\alpha$	1
Heuristic factor, $\beta$	2
Global evaporation factor, $\rho$	0.1
Local evaporation rate, $\varphi$	0.1
Pseudorandom parameter, $q_0$	0.9

These default parameters were used as a starting point to determine the best parameters to use for the modified AS and ACS algorithms respectively.

#### **4.2.1 Setup for Varying the Number of Iterations**

For the modified AS and ACS algorithms, the parameters in Table 4.2 and Table 4.3 were used respectively, except for the number of iterations, which was set as a variable. The number of iterations to be tested was 10, 20, 30, 40 and 50 for both algorithms.

Three different 3D models were selected as the sample set: “Benchy”, “Clamp” and “Lowest poly thinker”. The models were selected for their different unique structures. The “Clamp” and “Lowest poly thinker” models were rescaled to 50% of their original size due to excessive processing times.

For each algorithm, 3D model and number of iterations, the optimized G-code was generated five times. A G-code simulator was used to obtain the travel distance and print time for each G-code file. The mean travel distance, print time and processing time of the five trials were recorded.

According to M. Dorigo, V. Maniezzo and A. Colorni (1996), the number of iterations significantly affected the processing time. Thus, the processing time was recorded.

#### **4.2.2 Results for Varying the Number of Iterations**

The results were graphed in terms of print time against travel distance, travel distance against processing time, and print time against processing time.

### 4.2.2.1 Print Time against Travel Distance

Table 4.4 shows the graphs of print time against travel distance for the modified AS and ACS algorithms when they were applied to the “Benchy”, “Clamp”, and “Lowest poly thinker” 3D models. The numbers next to the data points indicate the number of iterations for the particular algorithm.

Table 4.4: Graphs of print time against travel distance for the modified AS and ACS algorithms when applied to the sample set of 3D models

3D model	Graph
Benchy	
Clamp	
Lowest poly thinker	

### 4.2.2.2 Travel Distance against Processing Time

Table 4.5 shows the graphs of travel distance against processing time for the modified AS and ACS algorithms when they were applied to the “Benchy”, “Clamp”, and “Lowest poly thinker” 3D models. The numbers next to the data points indicate the number of iterations for the particular algorithm.

Table 4.5: Graphs of travel distance against processing time for the modified AS and ACS algorithms when applied to the sample set of 3D models

3D model	Graph																		
Benchy	<table border="1"> <caption>Data for Benchy Graph</caption> <thead> <tr> <th>Iteration</th> <th>Modified AS (cm)</th> <th>Modified ACS (cm)</th> </tr> </thead> <tbody> <tr><td>10</td><td>13700</td><td>13670</td></tr> <tr><td>20</td><td>13655</td><td>13645</td></tr> <tr><td>30</td><td>13640</td><td>13630</td></tr> <tr><td>40</td><td>13630</td><td>13620</td></tr> <tr><td>50</td><td>13625</td><td>13610</td></tr> </tbody> </table>	Iteration	Modified AS (cm)	Modified ACS (cm)	10	13700	13670	20	13655	13645	30	13640	13630	40	13630	13620	50	13625	13610
Iteration	Modified AS (cm)	Modified ACS (cm)																	
10	13700	13670																	
20	13655	13645																	
30	13640	13630																	
40	13630	13620																	
50	13625	13610																	
Clamp	<table border="1"> <caption>Data for Clamp Graph</caption> <thead> <tr> <th>Iteration</th> <th>Modified AS (cm)</th> <th>Modified ACS (cm)</th> </tr> </thead> <tbody> <tr><td>10</td><td>14925</td><td>14915</td></tr> <tr><td>20</td><td>14895</td><td>14890</td></tr> <tr><td>30</td><td>14880</td><td>14875</td></tr> <tr><td>40</td><td>14870</td><td>14870</td></tr> <tr><td>50</td><td>14865</td><td>14860</td></tr> </tbody> </table>	Iteration	Modified AS (cm)	Modified ACS (cm)	10	14925	14915	20	14895	14890	30	14880	14875	40	14870	14870	50	14865	14860
Iteration	Modified AS (cm)	Modified ACS (cm)																	
10	14925	14915																	
20	14895	14890																	
30	14880	14875																	
40	14870	14870																	
50	14865	14860																	
Lowest poly thinker	<table border="1"> <caption>Data for Lowest poly thinker Graph</caption> <thead> <tr> <th>Iteration</th> <th>Modified AS (cm)</th> <th>Modified ACS (cm)</th> </tr> </thead> <tbody> <tr><td>10</td><td>42580</td><td>42600</td></tr> <tr><td>20</td><td>42500</td><td>42520</td></tr> <tr><td>30</td><td>42460</td><td>42480</td></tr> <tr><td>40</td><td>42450</td><td>42460</td></tr> <tr><td>50</td><td>42430</td><td>42425</td></tr> </tbody> </table>	Iteration	Modified AS (cm)	Modified ACS (cm)	10	42580	42600	20	42500	42520	30	42460	42480	40	42450	42460	50	42430	42425
Iteration	Modified AS (cm)	Modified ACS (cm)																	
10	42580	42600																	
20	42500	42520																	
30	42460	42480																	
40	42450	42460																	
50	42430	42425																	

### 4.2.2.3 Print Time against Processing Time

Table 4.6 shows the graphs of print time against processing time for the modified AS and ACS algorithms when they were applied to the “Benchy”, “Clamp”, and “Lowest poly thinker” 3D models. The numbers next to the data points indicate the number of iterations for the particular algorithm.

Table 4.6: Graphs of print time against processing time for the modified AS and ACS algorithms when applied to the sample set of 3D models

3D model	Graph
Benchy	
Clamp	
Lowest poly thinker	

### 4.2.3 Discussion for Varying the Number of Iterations

First, the print time against travel distance is analysed by referring to Table 4.4. From the graphs, it can be observed that regardless of the 3D model or algorithm, the relationship between the print time and travel distance is generally linear. This finding supports the main hypothesis that by reducing the travel distance, the print time can be reduced as well.

However, the interesting finding is that for a given travel distance, the modified AS algorithm will always have a shorter print time compared to the modified ACS algorithm. A possible explanation for this is that the modified ACS algorithm favours path construction in such a way that the nozzle has to accelerate and decelerate at a frequent rate, thus reducing the average speed and resulting in a longer print time.

Next, the travel distance against processing time is analysed by referring to Table 4.5. From the graphs, there are a few noticeable trends. For a given number of iterations, the modified ACS algorithm will always have a longer processing time compared to the modified AS algorithm, and a shorter travel distance as well. The longer processing time is to be expected, since the ACS algorithm requires more calculations per iteration.

However, for a given length of processing time, the performance of the algorithms with regards to having the shortest travel distance depends on the 3D model. For the “Benchy” model, the modified ACS is better; for the “Clamp” model, both algorithms have similar performance; while for the “Lowest poly thinker” model, the modified AS is better.

Besides that, the print time against processing time is analysed by referring to Table 4.6. It can be observed that unlike the travel distance, the modified AS algorithm will always have a shorter print time than the modified ACS algorithm regardless of whether the processing time or number of iterations is fixed.

From the graphs in Table 4.5 and Table 4.6, it can be seen that the slope of the graphs generally decreases as the processing time increases. To strike a balance between the exponential increase in processing time and linear decrease in travel distance and print time, the number of iterations for both the modified AS and modified ACS algorithms is chosen to be 30.

It should be noted that some of these observations may change in the main testing phase in Section 4.3, as tweaking the other parameters may affect the performance of both the modified AS and ACS algorithms.

#### 4.2.4 Setup for Varying $\rho$ for Modified AS and $\rho, \varphi, q_0$ for Modified ACS

For the modified AS algorithm, the parameters in Table 4.7 were used, which is similar to the default parameters in Table 4.2 except for the number of iterations and the global evaporation factor,  $\rho$ . The number of iterations was set to 30 as mentioned in Section 4.2.3, while  $\rho$  was varied in the range 0.1 to 0.9 in steps of 0.1. The testing range for  $\rho$  was chosen because it could only be set more than 0 but less than 1 (M. Dorigo, V. Maniezzo and A. Colomi, 1996). This resulted in a total of nine parameter configurations to be tested.

Table 4.7: Parameters for varying the  $\rho$  for the modified AS algorithm

Parameter	Value	Range	Step
Number of iterations	30	-	-
Number of ants	50	-	-
Pheromone factor, $\alpha$	1	-	-
Heuristic factor, $\beta$	2	-	-
Global evaporation factor, $\rho$	-	0.1 – 0.9	0.1

For the modified ACS algorithm, the parameters in Table 4.8 were used, which is similar to the default parameters in Table 4.3, except for the number of iterations, the global evaporation factor,  $\rho$ , the local evaporation rate,  $\varphi$ , and the pseudorandom parameter,  $q_0$ . The number of iterations was set to 30 as mentioned in Section 4.2.3. Meanwhile, for  $\rho$  and  $\varphi$  values of 0.1, 0.2 and 0.3 were chosen, while for  $q_0$  values of 0.7, 0.8 and 0.9 were chosen. These particular range of values were chosen to be tested so that they did not differ too greatly from the default parameters. The limited range of values was due to time constraints. This resulted in a total of  $3 \times 3 \times 3 = 27$  parameter configurations to be tested.

Table 4.8: Parameters for varying the  $\rho$ ,  $\varphi$ ,  $q_0$  for the modified ACS algorithm

Parameter	Value	Range	Step
Number of iterations	30	-	-
Number of ants	50	-	-
Pheromone factor, $\alpha$	1	-	-
Heuristic factor, $\beta$	2	-	-
Global evaporation factor, $\rho$	-	0.1 – 0.3	0.1
Local evaporation rate, $\varphi$	-	0.1 – 0.3	0.1
Pseudorandom parameter, $q_0$	-	0.7 – 0.9	0.1

Three different 3D models were selected as the sample set: “Benchy”, “Clamp” and “Lowest poly thinker”. The models were selected for their different unique structures. The “Clamp” and “Lowest poly thinker” models were rescaled to 50% of their original size due to excessive processing times.

For each algorithm, 3D model and parameter configuration, the optimized G-code was generated three times. A G-code simulator was used to obtain the travel distance and print time for each G-code file. The mean travel distance and print time of the three trials were recorded.

According to M. Dorigo, V. Maniezzo and A. Coloni (1996), the parameters that were varied did not significantly affect the processing time, thus the processing time was not recorded.

#### 4.2.5 Results for Varying $\rho$ for Modified AS and $\rho$ , $\varphi$ , $q_0$ for Modified ACS

Since varying  $\rho$ ,  $\varphi$ ,  $q_0$  does not significantly affect the processing time of the algorithms as mentioned in Section 4.2.4, the main objective of this part of the preliminary parameter testing phase is to find the combination of parameters that will result in the shortest travel distance and print time for the modified AS and ACS algorithms.

The summary of the results is shown in Table 4.9 and Table 4.10 for the travel distance and print time respectively. The raw data is available in the Appendix, from Table I to Table VI, where the values highlighted in bold are the smallest values.

Table 4.9: Parameters resulting in shortest travel distance for modified AS and ACS algorithms when applied to the sample set of 3D models

3D model	Best parameters for shortest travel distance	
	Modified AS	Modified ACS
Benchy	$\rho = 0.6$	$\rho = 0.3, \varphi = 0.1, q0 = 0.7$
Clamp	$\rho = 0.6$	$\rho = 0.3, \varphi = 0.1, q0 = 0.7$
Lowest poly thinker	$\rho = 0.5$ and $0.6$	$\rho = 0.3, \varphi = 0.1, q0 = 0.7$

Table 4.10: Parameters resulting in shortest print time for modified AS and ACS algorithms when applied to the sample set of 3D models

3D model	Best parameters for shortest print time	
	Modified AS	Modified ACS
Benchy	$\rho = 0.6$	$\rho = 0.3, \varphi = 0.1, q0 = 0.8$
Clamp	$\rho = 0.3$ and $0.6$	$\rho = 0.3, \varphi = 0.1, q0 = 0.7$
Lowest poly thinker	$\rho = 0.8$	$\rho = 0.3, \varphi = 0.1, q0 = 0.8$

Meanwhile, the standard deviations (SD) of the travel distance and print time for both the algorithms for the range of configurations tested are shown in Table 4.11. This information is useful to get a rough idea of how much the performance of the algorithms are affected by varying the selected parameters.

Table 4.11: Standard deviations of travel distance and print time for the range of configurations tested for the AS and ACS algorithms

3D model	Standard deviation			
	Modified AS		Modified ACS	
	Travel distance (cm)	Print time (s)	Travel distance (cm)	Print time (s)
Benchy	4.11	1.20	21.20	5.30
Clamp	3.27	1.15	13.12	3.61
Lowest poly thinker	7.49	1.52	41.07	8.22

#### 4.2.6 Discussion for Varying $\rho$ for Modified AS and $\rho, \varphi, q_0$ for Modified ACS

Based on Table 4.9 and Table 4.10, the overall best combination of the parameters tested for both travel distance and print time for modified AS is  $\rho = 0.6$ , while for modified ACS it is  $\rho = 0.3, \varphi = 0.1, q_0 = 0.7$ . Thus, these parameters will be used for their respective algorithms in the main testing phase in Section 4.3.

According to Table 4.11, the standard deviation of the travel distance for the modified ACS is about three to eight times greater than that for the modified AS, depending on the 3D model. The same observation can be made for the standard deviation of the print time.

This suggests that varying  $\rho, \varphi, q_0$  for modified ACS has a greater impact on performance compared to varying  $\rho$  for modified AS. Thus, future testing should investigate improving the performance of the modified ACS algorithm by widening the range of  $\rho, \varphi, q_0$  values that are tested.

### 4.3 Main Testing Phase

The objective of this phase is to compare the performance of the modified AS and ACS algorithms with Cura across a wide range of 3D models.

#### 4.3.1 Setup for Main Testing Phase

The parameters that will be used for the modified AS algorithm are shown in Table 4.12. These parameters are based on those for the original AS algorithm in Table 4.2, but they have been tweaked according to the findings in Section 4.2.3 and 4.2.4.

Table 4.12: Parameters for modified AS algorithm used in main testing phase

Parameter	Value
Number of iterations	30
Number of ants	50
Pheromone factor, $\alpha$	1
Heuristic factor, $\beta$	2
Global evaporation factor, $\rho$	0.6

Meanwhile, the parameters that will be used for the modified ACS algorithm are shown in Table 4.13. These parameters are based on those for the original ACS algorithm in Table 4.3, but they have been tweaked according to the findings in Section 4.2.3 and 4.2.4.

Table 4.13: Parameters for modified ACS algorithm used in main testing phase

Parameter	Value
Number of iterations	30
Number of ants	50
Pheromone factor, $\alpha$	1
Heuristic factor, $\beta$	2
Global evaporation factor, $\rho$	0.3
Local evaporation rate, $\varphi$	0.1
Pseudorandom parameter, $q_0$	0.7

Besides that, the settings for Cura are shown in Table 4.14, as previously mentioned in Section 3.2.

Table 4.14: Settings for Cura used in main testing phase

Setting	Value
Version	4.4
Infill	20 %
Layer height	0.2 mm
Retraction	Disabled

Six different 3D models were selected as the sample set: “Benchy”, “Clamp”, “Lowest poly thinker”, “Torture test”, “Eiffel tower” and “Spiral vase”. The models were selected for their different unique structures. They were obtained through the website Thingiverse, an online open-source repository for 3D models (<https://www.thingiverse.com/>). The “Clamp” and “Lowest poly thinker” models were rescaled to 50% of their original size due to excessive processing times.

The overall procedure is described in Table 4.15. Cura is only required to produce the initial G-code file once for each 3D model because according to Fok, et al. (2018), Cura uses the nearest-neighbour algorithm which is deterministic. Thus, it will produce the same G-code for a given 3D model no matter how many times it is repeated.

Table 4.15: Procedure for the main testing phase

Step	Process
1	For the “Benchy” 3D model, use Cura to produce the initial G-code file
2	Put the initial G-code file into the parser program so that it can be optimized by the modified AS algorithm to produce the optimized G-code file
3	Record the processing time
4	Repeat step 2 to 3 nine more times to get a total of 10 optimized G-code files for the modified AS algorithm
5	Repeat steps 2 to 4 for the modified ACS algorithm
6	Repeat steps 1 to 5 for all the models in the sample set
7	Analyse all the initial and optimized G-code files using a G-code simulator to obtain the travel distance and print time for each file

### 4.3.2 Main Testing Phase Results

The results are divided into three parts that compare Cura with the modified AS and ACS algorithms, namely in terms of travel distance, print time and processing time. The results for Cura will not have mean value or standard deviation because the G-code file is only generated once for each 3D model as explained in Section 4.3.1.

#### 4.3.2.1 Travel Distance Results

The results of the travel distance for the different methods when applied to the sample set of 3D models are tabulated in Table 4.16. Meanwhile, the graph is shown in Figure 4.1. SD means standard deviation, which is measured to gauge the consistency of the algorithm.

Table 4.16: Travel distance of G-code produced by Cura, modified AS and modified ACS in the main testing phase

3D model	Cura	Modified AS		Modified ACS	
	Travel distance (cm)	Mean travel distance (cm)	Travel distance SD (cm)	Mean travel distance (cm)	Travel distance SD (cm)
Benchy	15288	13640	3.27	13597	2.74
Clamp	19487	14879	2.83	14840	2.80
Lowest poly thinker	45406	42466	5.87	42367	4.22
Torture test	25499	23295	12.44	23176	7.07
Eiffel tower	16452	15070	8.00	14901	4.81
Spiral vase	30240	25570	2.77	25562	2.16

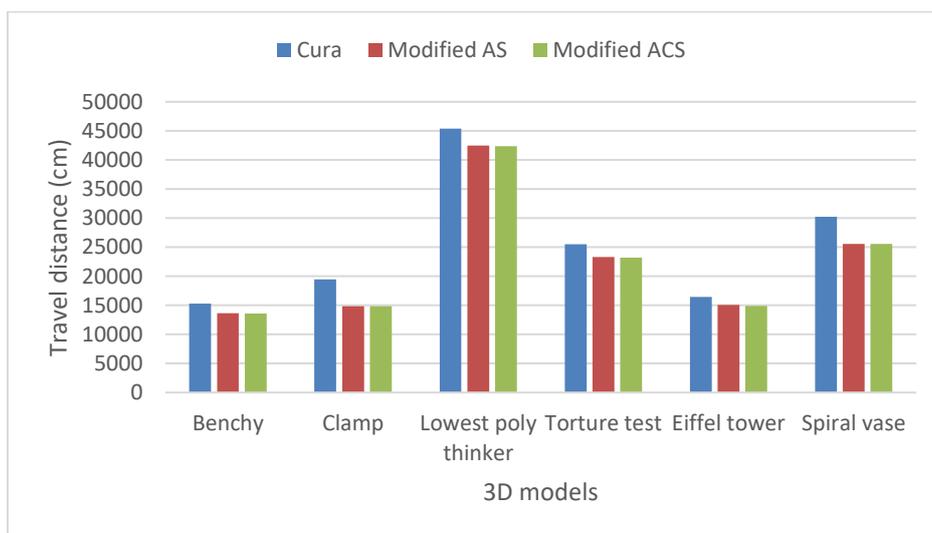


Figure 4.1: Bar graph of travel distance results in the main testing phase

The percentage travel distance reduced by the modified AS and ACS algorithms relative to Cura are tabulated in Table 4.17 and graphed in Figure

4.2. The percentage travel distance reduced is obtained by using data from Table 4.16 and Equation 4.1.

$$\begin{aligned} & \text{Percentage travel distance reduced} \\ &= \frac{\text{Cura travel distance} - \text{Algorithm travel distance}}{\text{Cura travel distance}} \quad (4.1) \\ & \times 100\% \end{aligned}$$

Table 4.17: Percentage travel distance reduced by modified AS and ACS relative to Cura in the main testing phase

3D model	Mean percentage travel distance reduced (%)	
	Modified AS	Modified ACS
Benchy	10.78	11.06
Clamp	23.65	23.85
Lowest poly thinker	6.47	6.69
Torture test	8.64	9.11
Eiffel tower	8.40	9.43
Spiral vase	15.44	15.47

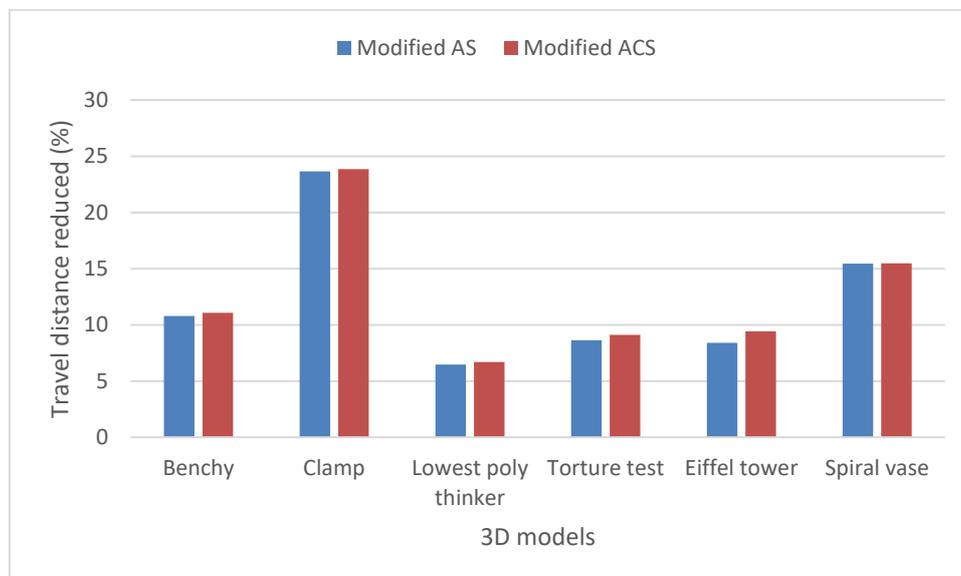


Figure 4.2: Bar graph of percentage travel distance reduced in the main testing phase

#### **4.3.2.2 Travel Distance Discussion**

Based on Figure 4.1, the modified AS and ACS algorithms were able to achieve shorter travel distances compared to Cura for every 3D model in the sample set. This demonstrates that both the algorithms were able to optimize the print path better than Cura in terms of travel distance.

Taking a closer look at Table 4.16, the modified ACS was able to achieve shorter travel distances compared to the modified AS for every 3D model in the sample set. In summary, in terms of the travel distance, the modified ACS performed the best, followed by the modified AS and lastly Cura.

According to Table 4.16, for both the algorithms, the standard deviations of the travel distance are three to four orders of magnitude smaller than their respective mean values. Since the amount of deviation from the mean is negligible, thus both the modified AS and ACS algorithms are able to produce consistent results in terms of travel distance for a given 3D model.

Besides that, the degree of improvement achieved by the algorithms compared to Cura in terms of travel distance is not constant, as can be seen in Figure 4.2. The modified AS algorithm showed a percentage reduction ranging from 6.47% to 23.65%, while for the modified ACS algorithm it ranged from 6.69% to 23.85%. This suggests that the amount of improvement that can be achieved for the travel distance is dependent on the unique structure of each 3D model.

#### **4.3.2.3 Print Time Results**

The results of the travel distance for the different methods when applied to the sample set of 3D models are tabulated in Table 4.18. Meanwhile, the graph is shown in Figure 4.3. SD means standard deviation, which is measured to gauge the consistency of the algorithm.

Table 4.18: Print time of G-code produced by Cura, modified AS and modified ACS in the main testing phase

3D model	Cura	Modified AS		Modified ACS	
	Print time (s)	Mean print time (s)	Print time SD (s)	Mean print time (s)	Print time SD (s)
Benchy	4612	4206	0.95	4203	0.71
Clamp	6408	5171	1.49	5167	0.82
Lowest poly thinker	11099	10520	1.85	10516	0.966
Torture test	7404	6853	1.66	6844	1.93
Eiffel tower	6999	6032	1.77	6014	1.81
Spiral vase	9841	8328	0.63	8326	0.82

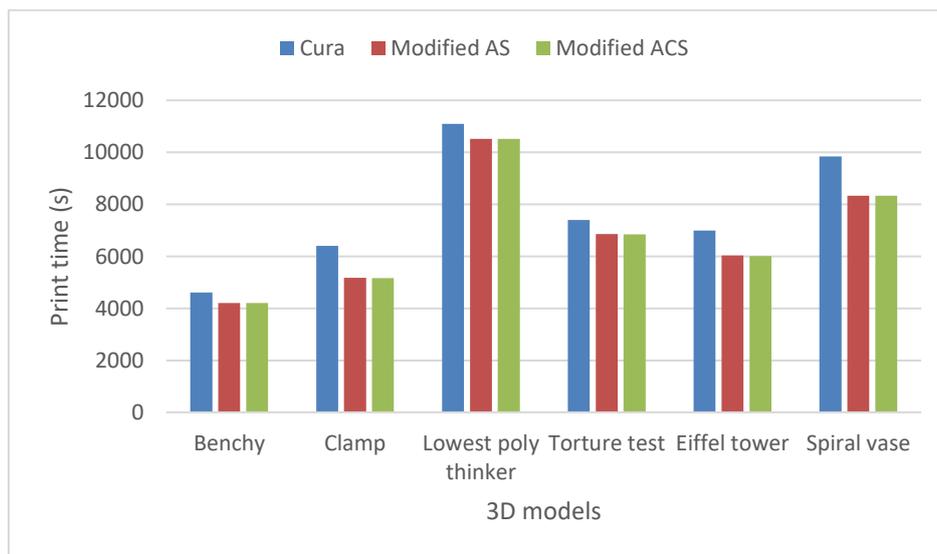


Figure 4.3: Bar graph of print time results in the main testing phase

The percentage print time reduced by the modified AS and ACS algorithms relative to Cura are tabulated in Table 4.19 and graphed in Figure

4.4. The percentage print time reduced is obtained by using data from Table 4.18 and Equation 4.2.

$$\begin{aligned} & \text{Percentage print time reduced} \\ &= \frac{\text{Cura print time} - \text{Algorithm print time}}{\text{Cura print time}} \times 100\% \quad (4.2) \end{aligned}$$

Table 4.19: Percentage print time reduced by modified AS and ACS relative to Cura in the main testing phase

3D model	Mean percentage print time reduced (%)	
	Modified AS	Modified ACS
Benchy	8.80	8.87
Clamp	19.30	19.37
Lowest poly thinker	5.22	5.25
Torture test	7.44	7.56
Eiffel tower	13.82	14.07
Spiral vase	15.37	15.39

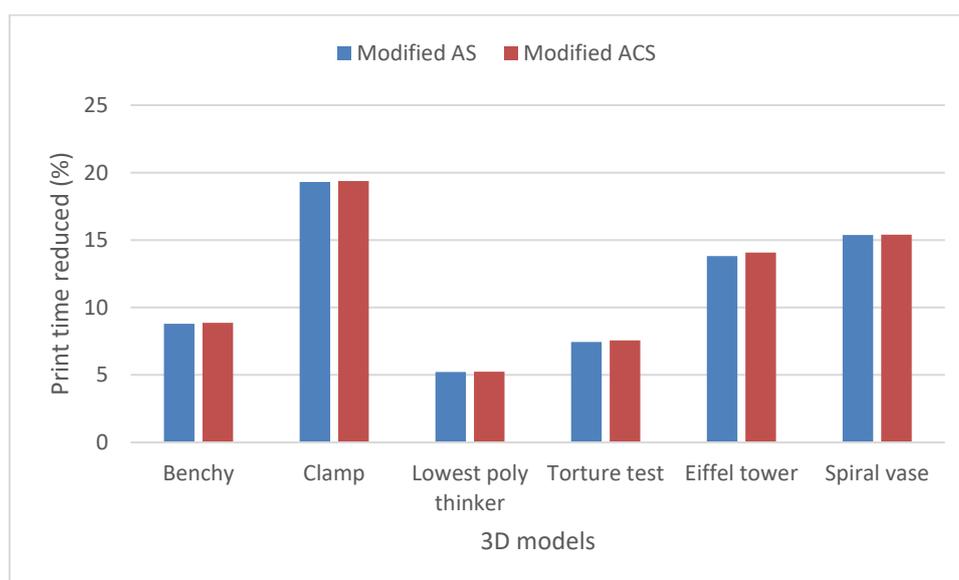


Figure 4.4: Bar graph of percentage print time reduced in the main testing phase

#### 4.3.2.4 Print Time Discussion

Based on Figure 4.3, the modified AS and ACS algorithms were able to achieve shorter print time compared to Cura for every 3D model in the sample set. This

demonstrates that both the algorithms were able to optimize the print path better than Cura in terms of print time.

Taking a closer look at Table 4.18, the modified ACS was able to achieve shorter print times compared to the modified AS for every 3D model in the sample set. In summary, in terms of the print time, the modified ACS performed the best, followed by the modified AS and lastly Cura.

According to Table 4.18, for both the algorithms, the standard deviations of the print times are three to four orders of magnitude smaller than their respective mean values. Since the amount of deviation from the mean is negligible, thus both the modified AS and ACS algorithms are able to produce consistent results in terms of print time for a given 3D model.

Besides that, the degree of improvement achieved by the algorithms compared to Cura in terms of travel distance is not constant, as can be seen in Figure 4.4. The modified AS algorithm showed a percentage reduction ranging from 5.22% to 19.30%, while for the modified ACS algorithm it ranged from 5.25% to 19.37%. This suggests that the amount of improvement that can be achieved for the print time is dependent on the unique structure of each 3D model.

Another important point to note is that the results and discussion regarding the print time are very similar to that for the travel distance. This indicates that there is a close relationship between the travel distance and print time.

#### **4.3.2.5 Processing Time Results**

The results of the processing time for the modified AS and ACS algorithms when applied to the sample set of different 3D models are tabulated in Table 4.20 and graphed in Figure 4.5.

Table 4.20: Processing time of modified AS and ACS algorithm in the main testing phase

3D model	Mean processing time (s)	
	Modified AS	Modified ACS
Benchy	355	398
Clamp	320	316
Lowest poly thinker	320	504
Torture test	748	932
Eiffel tower	797	1190
Spiral vase	979	681

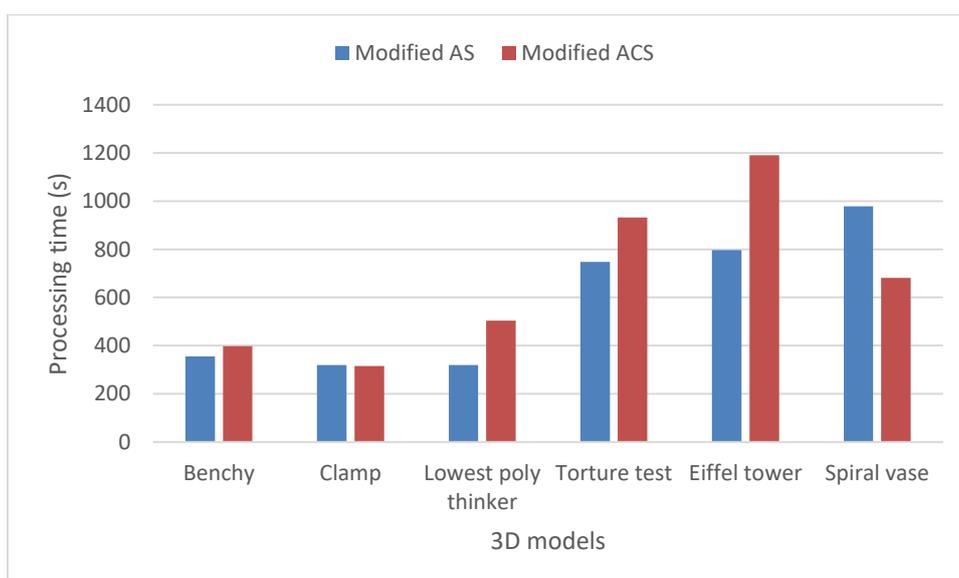


Figure 4.5: Bar graph of processing time results in the main testing phase

Figure 4.6 shows the scatter plot of the processing time against the print time of Cura for the sample set of 3D models. It was plotted using data from Table 4.18 and Table 4.20. The graph was plotted to investigate if there was any relationship between the Cura print time and the processing time required by the modified AS and ACS algorithms, or in other words whether the processing time could be predicted using the Cura print time.

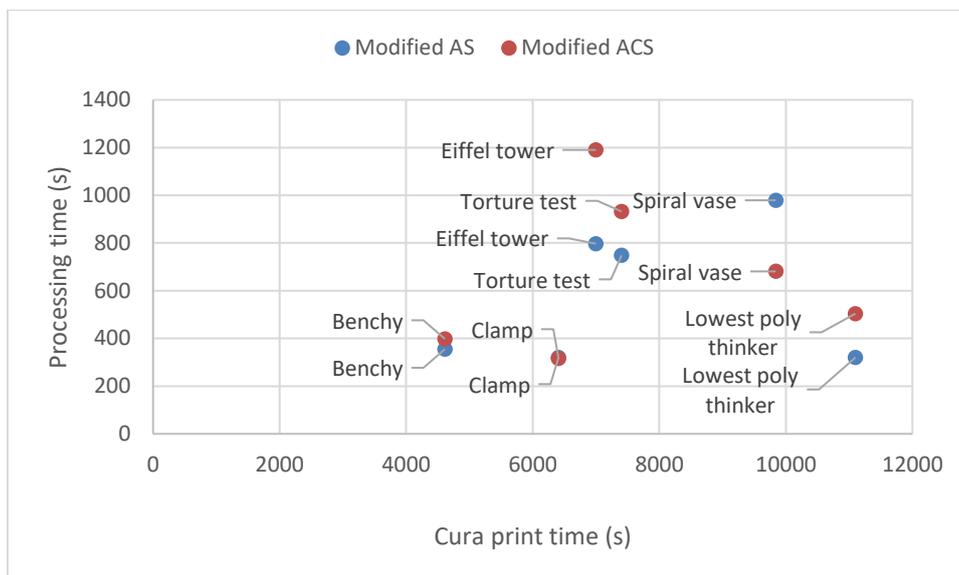


Figure 4.6: Scatter plot of processing time against Cura print time for modified AS and ACS algorithm in the main testing phase

#### 4.3.2.6 Processing Time Discussion

Since the current method of implementing the algorithms requires Cura to generate the initial G-code file, the processing time of Cura could be treated as zero. However, the processing time of both the modified AS and ACS algorithm are rather significant, as shown in Figure 4.5. The processing time is generally in the order of hundreds of seconds. This issue needs to be overcome for the algorithms to have practical use. One possible solution is to use GPU acceleration to decrease the processing time.

To compare the two algorithms in terms of processing time, Table 4.20 is referred to. Based on the table, the modified ACS algorithm usually has significantly longer processing times compared to the modified AS algorithm, sometimes up to hundreds of seconds more.

The only outlier for the sample set tested is the “Spiral vase” 3D model, in which the processing time for the modified ACS is only 681 seconds, which is much lower than the 979 seconds for the modified AS. A possible explanation is that the structure of the “Spiral vase” 3D model is easier to be optimized by the modified ACS compared to the modified AS. Testing on a wider range of 3D models for the sample set may clarify if the modified ACS will sometimes have shorter processing times than the modified AS, or if it is unique to the “Spiral vase” model only.

An attempt also made was made to investigate the relationship between the print time obtained by Cura and the required processing time of the modified AS and ACS algorithms. However, as can be observed from Figure 4.6, there does not appear to be any clear relationship between the two variables. A possible hypothesis is that the processing time is influenced by the size and complexity of the structure of the 3D model. This hypothesis may be investigated further in future research.

### 4.3.3 Comparison of Results with State-of-the-Art

The results obtained in this report are compared with results obtained by two previously published papers authored by Fok, et al. (2018) and Fok, et al. (2019).

The percentage print time reduced metric will be used in the comparison. For this report, this metric is tabulated in Table 4.19. For the other papers, the metric will be calculated by using Equation 4.2 together with their reported Cura print time and algorithm print time. The metrics are then compared for the same 3D model.

Since the Cura print time reported by this report and the other papers are different, this comparison method is used to ensure fairness by comparing the reduction in print time relative to the respectively reported Cura print times.

#### 4.3.3.1 Comparison with Fok, et al. (2018)

The print times of Cura and the extended ant colony optimization (ACO) algorithm as reported by Fok, et al. (2018) for the “Torture test” 3D model are tabulated in Table 4.21. ACO refers to a family of algorithms that the original AS and ACS algorithms belong to. The extended ACO algorithm was explained in Section 2.3.3.2.

Table 4.21: Print times reported by Fok, et al. (2018) for the “Torture test” 3D model

3D model	Print time (s)	
	Cura	Extended ACO
Torture test	9951.90	8835.07

Using Equation 4.2, the percentage print time reduced was calculated. The obtained result was then tabulated together with the results from modified AS and ACS in Table 4.22. The data is represented as a bar chart in Figure 4.7.

Table 4.22: Comparison between percentage print time reduced for modified AS, modified ACS and Fok, et al. (2018) for the “Torture test” 3D model

3D model	Percentage print time reduced (%)		
	This report		Fok, et al. (2018)
	Modified AS	Modified ACS	Extended ACO
Torture test	7.44	7.56	11.22

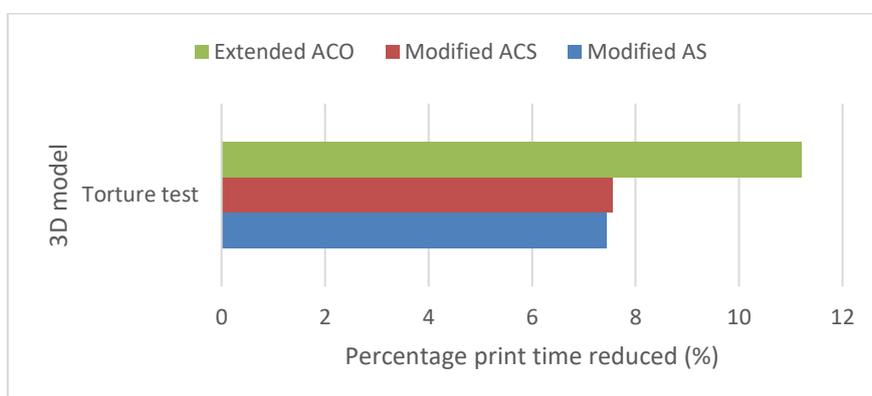


Figure 4.7: Bar chart for percentage print time reduced comparison with Fok, et al. (2018) for the “Torture test” 3D model

#### 4.3.3.2 Comparison with Fok, et al. (2019)

The print times of Cura, the Frederickson (Fred) algorithm, ACO algorithm and detour searching algorithm as reported by Fok, et al. (2019) for the “Clamp” and “Lowest poly thinker” 3D models are tabulated in Table 4.23. The Fred algorithm was explained in Section 2.3.1.3. Meanwhile, the detour searching algorithm was proposed by Fok, et al. (2019) and was explained in Section 2.3.3.3.

Table 4.23: Print times reported by Fok, et al. (2019) for the “Clamp” and “Lowest poly thinker” 3D models

3D model	Print time (s)			
	Cura	Fred	ACO	Detour searching
Clamp	8762	7983	7055	6538
Lowest poly thinker	6408	5857	5502	5475

Using Equation 4.2, the percentage print time reduced was calculated. The obtained result was then tabulated together with the results from modified AS and ACS in Table 4.24. The data is also represented as a bar chart in Figure 4.8.

Table 4.24: Comparison between percentage print time reduced for modified AS, modified ACS and Fok, et al. (2019) for the “Clamp” and “Lowest poly thinker” 3D models

3D model	Percentage print time reduced (%)				
	This report		Fok, et al. (2019)		
	Modified AS	Modified ACS	Fred	ACO	Detour searching
Clamp	19.30	19.37	8.89	19.48	25.38
Lowest poly thinker	5.22	5.25	8.60	14.14	14.56

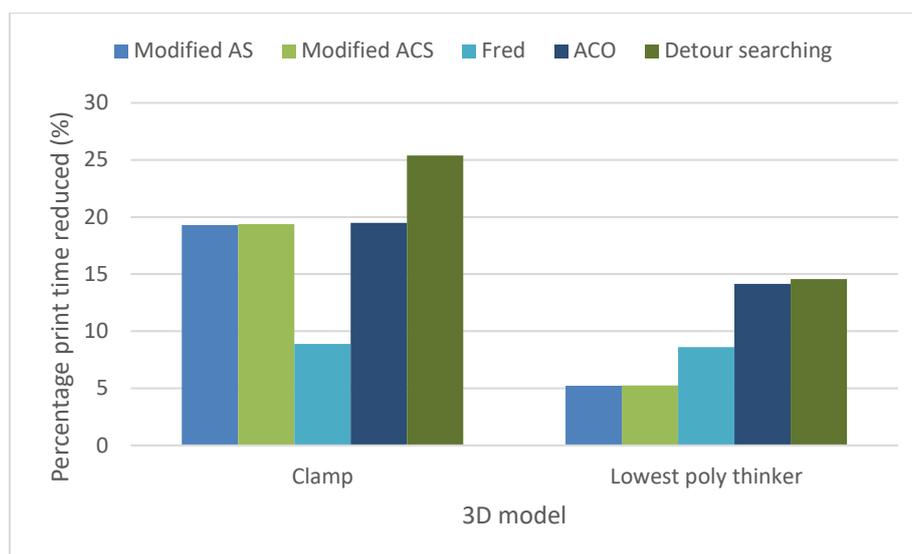


Figure 4.8: Bar chart for percentage print time reduced comparison with Fok, et al. (2019) for the “Clamp” and “Lowest poly thinker” 3D models

#### 4.3.4 State-of-the-art Discussion

When the modified AS and ACS algorithms were compared with the extended ACO algorithm proposed by Fok, et al. (2018) for the “Torture test” 3D model, the extended ACO algorithm performed better than both the modified AS and ACS algorithms in terms of print time reduced relative to Cura, as shown in Figure 4.7.

Besides that, the modified AS and ACS algorithms were also compared with the Fred algorithm, ACO algorithm and detour searching algorithm proposed by Fok, et al. (2019) for the “Clamp” and “Lowest poly thinker” 3D models. The results in Figure 4.8 showed that for the “Clamp” 3D model, both the modified AS and ACS algorithms performed better than the Fred algorithm, similar to the ACO algorithm, and worse than the detour searching algorithms. However, for the “Lowest poly thinker” 3D model, both the modified AS and ACS algorithms performed worse than the other algorithms.

It is important to note that there were many significant differences in the experimental setup for this report and the two previous works by Fok, et al. (2018) and Fok, et al. (2019), such as the earlier works using an outdated version of Cura and enabling the retraction setting. This could help partially explain the generally worse performance of the modified AS and ACS algorithms when compared to the previous works.

#### 4.4 Summary

The preliminary parameter testing phase showed that  $\rho = 0.6$  was the most suitable for the modified AS algorithm, while  $\rho = 0.3, \varphi = 0.1, q_0 = 0.7$  was the most suitable for the modified ACS algorithm in order to achieve the shortest travel distance and print time.

In the testing phase, the modified AS and ACS algorithms always achieved shorter travel distance and print time compared to Cura for the sample set of 3D models tested. The solutions obtained by both the modified AS and ACS algorithms were also fairly consistent throughout the ten trials for each 3D models, with the standard deviation for print time not exceeding two seconds.

Although modified ACS was always able to achieve better solutions in terms of travel distance and print time compared to the modified AS, it performed worse overall when the processing time was factored in.

The performance of the modified AS and ACS algorithms in terms of percentage print time reduced relative to Cura was generally worse when compared to methods analysed by Fok, et al. (2018) and Fok, et al. (2019). However, the worse performance could be partially attributed to the difference in the experimental setup.

## CHAPTER 5

### CONCLUSIONS AND RECOMMENDATIONS

#### 5.1 Conclusions

In this report, a method for reducing the printing time by using various algorithms to reduce the print distance is presented.

A parser program was designed and coded in order to extract the relevant information from a G-code file such that it could be optimized by the selected algorithm, and then reorganized to produce the optimized output G-code file. Two algorithms, the AS and ACS algorithms which were originally designed to solve the TSP, were modified to solve the URPP representation of the LPOP.

Preliminary testing was conducted to determine the most suitable parameters for the algorithms. Then in the main testing phase, the list of algorithms was used to optimize a sample set of 3D models and the mean processing time was recorded. The output G-code files were then analysed by simulator software to obtain the print time and print distance.

By analysing the data collected, it was shown that the modified AS and modified ACS algorithms managed to reduce the travel distance and print time of the sample set of 3D models significantly compared to Cura. However when the processing time of the algorithms was taken into account, the time reduction was less, but it was still an improvement over Cura on average.

The modified ACS had shorter print time and print distance than the modified AS, but longer overall time taken when the processing time was factored in. Thus, the modified AS algorithm is better in practice.

When compared with the methods proposed by Fok, et al. (2018) and Fok, et al. (2019), the modified AS and ACS algorithms performed worse in terms of reducing the print time of Cura. However, the difference in performance could be due to the difference in experimental setup.

## 5.2 Recommendations for Future Work

To improve the performance of the algorithms at reducing the print time, the travel speed and acceleration of the nozzle could be factored into the algorithms instead of merely the travel distance. The retraction factor could also be included in future research.

Since there are limitations to improving the print time by optimizing the software alone, another solution could be to improve both the hardware and software together. A proposal would be to develop a multi-head 3D printer and the corresponding algorithm to control the multiple print heads simultaneously, then compare its performance with conventional single-head 3D printers.

To decrease the processing times for the modified AS and ACS algorithms, GPU acceleration could be implemented in the algorithm code to reduce the processing time.

A more thorough study to compare the performance of the original AS and ACS algorithms with transformation method and the modified AS and ACS algorithms at solving the URPP could be carried out. If the study shows that the modified AS and ACS have better performance, they could be applied to other real-world problems that can be represented as the URPP, not just the LPOP.

## REFERENCES

- Bock, F., 1958. An algorithm for solving travelling-salesman and related network optimization problems. In: INST OPERATIONS RESEARCH MANAGEMENT SCIENCES 901 ELKRIDGE LANDING RD, STE ... *Operations Research*, p. 897–897.
- Christofides, N., 1976. *Worst-case analysis of a new heuristic for the travelling salesman problem*.
- Croes, G. A., 1958. A method for solving traveling-salesman problems. *Operations Research*, 6(6), pp. 791–812.
- Fok, K. Y., Cheng, C. T., Ganganath, N., Lu, H. H. and Tse, C. K., 2018. Accelerating 3D Printing Process Using an Extended Ant Colony Optimization Algorithm. In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5.
- Fok, K. Y., Cheng, C. T. and Tse, C. K., 2017. A refinement process for nozzle path planning in 3D printing. In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4.
- Fok, K. Y., Ganganath, N., Cheng, C. T., Lu, H. H. and Tse, C. K., 2019. A Nozzle Path Planner for 3D Printing Applications. *IEEE Transactions on Industrial Informatics*, p. 1–1. <http://dx.doi.org/10.1109/TII.2019.2962241>.
- Fok, K. Y., Ganganath, N., Cheng, C. T. and Tse, C. K., 2016. A 3D printing path optimizer based on Christofides algorithm. In: *2016 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pp. 1–2.
- Frederickson, G. N., 1979. Approximation Algorithms for Some Postman Problems. *J. ACM*, [e-journal] 26(3), pp. 538–554. <http://dx.doi.org/10.1145/322139.322150>.
- Ganganath, N., Cheng, C. T., Fok, K. Y. and Tse, C. K., 2016. Trajectory planning for 3D printing: A revisit to traveling salesman problem. In: *Proceedings - 2016 the 2nd International Conference on Control, Automation and Robotics, ICCAR 2016*. Department of Electronic and Information Engineering, Hong Kong Polytechnic University, pp. 287–290.

- Held, M. and Karp, R. M., 1970. The Traveling-Salesman Problem and Minimum Spanning Trees. *Operations Research*, 18(6), p. 1138–1138. <<http://search.ebscohost.com/login.aspx?direct=true&db=edsjsr&AN=edsjsr.169411&site=eds-live&scope=site>>.
- hudbrog, 2018. *gCodeViewer*. [online] Available at: <<http://gcode.ws/>> [Accessed 15-Apr-20].
- Johnson, D. S., 1990. Local optimization and the Traveling Salesman Problem. In: M. S. Paterson. *Automata, Languages and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 446–461.
- Jones, R., Haufe, P., Sells, E., Iravani, P., Olliver, V., Palmer, C. and Bowyer, A., 2011. RepRap – the replicating rapid prototyper. *Robotica*, [e-journal] 29(1), pp. 177–191. <http://dx.doi.org/10.1017/S026357471000069X>.
- Lensgraf, S. and Mettu, R. R., 2016. Beyond layers: A 3D-aware toolpath algorithm for fused filament fabrication. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3625–3631.
- Lin, S., 1965. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10), pp. 2245–2269.
- M. Dorigo and L. M. Gambardella, 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), pp. 53–66.
- M. Dorigo, V. Maniezzo and A. Coloni, 1996. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1), pp. 29–41.
- Marlin Firmware, n.d. *G-code Index*. [online] Available at: <<https://marlinfw.org/meta/gcode/>> [Accessed 14-Apr-20].
- Orloff, C. S., 1974. A fundamental problem in vehicle routing. *Networks*, [e-journal] 4(1), pp. 35–64. <http://dx.doi.org/10.1002/net.3230040105>.
- Pérez-Delgado, M. L., 2010. *Solving an Arc-routing problem using artificial ants with a graph transformation*. [e-book].

<<https://www.scopus.com/inward/record.uri?eid=2-s2.0-84982902984&partnerID=40&md5=f65dff9a00a92d750474e69b55ec7f3e>>.

RepRap, n.d. G-code. Available at: [Online; accessed 14-April-2020 ].

<<https://reprap.org/mediawiki/index.php?title=G-code&oldid=187649>>.

syue87, 2018. *Gcode Analyser*. [online] Available at:

<<https://www.gcodeanalyser.com/>> [Accessed 15-Apr-20].

Ultimaker, n.d. *Cura*. [online] Available at:

<<https://ultimaker.com/software/ultimaker-cura>> [Accessed 14-Apr-20].

## APPENDICES

### APPENDIX A: Graphs

## APPENDIX B: Tables

Table I: Average print distance and print time of “Benchy” model for different parameters of the modified AS algorithm

<b>Benchy</b>									
Global evap rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Distance (cm)	13639	13641	13638	13643	13649	<b>13635</b>	13639	13646	13644
Time (s)	4206	4207	4205	4207	4208	<b>4204</b>	4205	4205	4206

Table II: Average print distance and print time of “Benchy” model for different parameters of the modified ACS algorithm

<b>Benchy</b>									
Global evap rate	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Local evap rate	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3
q0	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9
Distance (cm)	13655	13632	13625	13671	13640	13625	13678	13639	13628
Time (s)	4217	4212	4209	4219	4213	4209	4223	4211	4210
Global evap rate	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Local evap rate	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3
q0	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9
Distance (cm)	13616	13603	13614	13627	13616	13607	13644	13622	13615
Time (s)	4205	4204	4205	4210	4208	4205	4214	4210	4208
Global evap rate	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Local evap rate	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3
q0	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9
Distance (cm)	<b>13593</b>	13595	13604	13612	13598	13603	13624	13607	13605
Time (s)	4203	<b>4200</b>	4202	4207	4204	4204	4210	4205	4203

Table III: Average print distance and print time of “Clamp” model for different parameters of the modified AS algorithm

<b>Clamp</b>									
Global evap rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Distance (cm)	14885	14880	14879	14875	14878	<b>14874</b>	14882	14879	14882
Time (s)	5171	5171	<b>5169</b>	5171	5170	<b>5169</b>	5171	5171	5173

Table IV: Average print distance and print time of “Clamp” model for different parameters of the modified ACS algorithm

<b>Clamp</b>									
Global evap rate	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Local evap rate	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3
q0	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9
Distance (cm)	14876	14869	14877	14886	14873	14877	14888	14878	14878
Time (s)	5176	5175	5175	5177	5176	5176	5181	5177	5177
Global evap rate	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Local evap rate	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3
q0	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9
Distance (cm)	14852	14847	14859	14859	14857	14865	14863	14857	14871
Time (s)	5171	5169	5172	5171	5173	5172	5172	5172	5172
Global evap rate	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Local evap rate	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3
q0	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9
Distance (cm)	<b>14837</b>	14845	14864	14845	14849	14858	14854	14850	14861
Time (s)	<b>5165</b>	5166	5169	5168	5170	5172	5171	5169	5172

Table V: Average print distance and print time of “Lowest poly thinker”  
model for different parameters of the modified AS algorithm

Lowest poly thinker									
Global evap rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Distance (cm)	42464	42470	42474	42472	<b>42453</b>	<b>42453</b>	42463	42462	42472
Time (s)	10520	10520	10522	10521	10521	10518	10519	<b>10517</b>	10521

Table VI: Average print distance and print time of “Lowest poly thinker”  
model for different parameters of the modified ACS algorithm

Lowest poly thinker									
Global evap	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Local evap	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3
q0	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9
Distance (cm)	42479	42456	42469	42511	42476	42484	42521	42490	42487
Time (s)	10536	10533	10530	10546	10535	10532	10548	10537	10535
Global evap	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Local evap	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3
q0	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9
Distance (cm)	42400	42397	42430	42436	42418	42449	42453	42427	42436
Time (s)	10522	10520	10525	10531	10524	10528	10537	10527	10526
Global evap	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Local evap	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3
q0	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9
Distance (cm)	<b>42365</b>	42378	42417	42390	42392	42419	42397	42397	42426
Time (s)	10515	<b>10514</b>	10522	10521	10519	10524	10528	10522	10525