Genetic Algorithm for Process Sequencing Modelled as the Travelling Salesman Problem with Precedence Constraints

by

NORAINI MOHD RAZALI

BEng., MEng.

A thesis submitted for the degree of

Doctor of Philosophy

School of Mechanical and Manufacturing Engineering

Faculty of Engineering and Computing

Dublin City University

Supervisor

Dr. John Geraghty

August 2014

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, and that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

[illegible]

…………………………

Noraini Mohd Razali

ID No: 58114262

August 2014

Publications

This work has been disseminated through the following publications.

Conference papers and presentations:

- N. M. Razali and J. Geraghty, Biologically inspired genetic algorithm to minimize idle time of the assembly line balancing, Third World Congress on Nature and Biologically Inspired Computing (NaBIC2011), 19 – 21 Oct 2011, Salamanca University, Spain. *Conference proceedings, pp 105-110, 2011*

- N. M. Razali and J. Geraghty, Genetic Algorithm Performance with Different Selection Strategies in Solving TSP, The 2011 International Conference of Computational Intelligence and Intelligent Systems (ICCIIS11), 6 – 8 July 2011, London, United Kingdom. *Conference proceedings, volume 2, pp 1134-1139, 2011*

- N. M. Razali and J. Geraghty, Selection Strategies for Genetic Algorithm to Solve Travelling Salesman Problem, SIAM conference on Computational Science & Engineering (CSE11), 28 Feb – 4 March 2011, Grand Sierra Resort & Casino, Reno, Nevada, USA

- N. M. Razali and J. Geraghty, Performance comparison between different GA selection strategies in solving TSP instance, 24th European Conference on Operational Research, 11 – 14 July 2010, Lisbon, Portugal

Posters:

- N. M. Razali and J. Geraghty, Genetic Algorithms Performance Between Different Selection Strategy in Solving TSP, UMIES 2010 UK-Malaysia-Ireland Engineering Science Conference, 23 – 26 June 2010, Queens University, Belfast, Northern Ireland

Dedication

TO MY BELOVED SON

(MUHAMMAD IZMEER)

Acknowledgments

I am grateful to Almighty Allah for the fulfilment of my desire to finish this thesis. I would not have been able to complete this research without the aid and support of many people over the past three years. I must first express my gratitude towards my supervisor, Dr. John Geraghty, for his continuous support, guidance and understanding which contributed greatly to the successful completion of this thesis. I am deeply indebted to my colleagues Saba and Anna Rotundo for assisting me with coding in MATLAB. This research would not have been possible without the financial assistance of Skim Latihan Akademik Bumiputera (SLAB), The Ministry of Higher Education of Malaysia, and the Universiti Malaysia Pahang (UMP). I express my gratitude to those sponsors. Finally, I would like to thank my parents for their endless love and pray, though physically far away, have been with me all along in the spirit.

Abstract

This thesis addresses process sequencing subject to precedence constraints which arises as a subproblem in scheduling, planning and routing problems. The process sequencing problem can be modelled as the Travelling Salesman Problem with Precedence Constraints (TSPPC). In the general Travelling Salesman Problem (TSP) scenario, the salesman must travel from city to city; visiting each city exactly once and wishes to minimize the total distance travelled during the tour of all cities. TSPPC is similar in concept to TSP, except that it has a set of precedence constraints to be followed by the salesman. The exact methods that are used to find an optimal solution of the problem are only capable of handling small and medium sizes of instances. Genetic algorithms (GA) are heuristic optimization techniques based on the principles and mechanisms of natural evolution and can be used to solve larger instances and numerous side constraints with optimal or near-optimal solutions. However, the use of a conventional genetic algorithm procedure for TSP, with an order-based representation, might generate invalid candidate solutions when precedence constraints are involved. In this thesis, a new GA procedure is developed which includes chromosome's repairing strategy based topological sort to handle the precedence constraints and to generate only feasible solution during the evolutionary process. The procedure to select the task in sequence is based on the "earliest position" techniques. This procedure is combined with a roulette wheel selection, linear order crossover and inversion mutation. The effectiveness and the stability of the proposed GA are then evaluated against a wide range of benchmark problems and the solutions are compared with the results obtained from research results published in the relevant literature. The results from the computational experiments demonstrate that the proposed GA procedure developed in this thesis is capable to tackle large-size problem and reach for optimal solutions. The developed GA procedure improved the performance of the algorithm with less number of generations and less convergence time in achieving optimal solutions. The genetic operators used are capable to always introduce new fitter offspring and free from being trapped in a local optimum. Therefore it can be stated that the proposed genetic algorithm is efficient to solve process sequencing modelled as the travelling salesman problem with precedence constraints. This result will greatly help to solve many real world sequencing problems especially in the field of assembly line design and management.

Table of Contents

List of Tables

List of Figures

Nomenclature

i	:	city i
j	:	city j
V	:	set of node
R	:	set of arc
c_{ij}	:	cost or distance from city i to city j
x_{ij}	:	tour from city i to city j
n	:	number of cities
C	:	cost matrix
v_i	:	vertex i
v_j	:	vertex j
E	:	relative error
P_i	:	selection probability for city i
f_i	:	fitness value of city i
f_j	:	fitness value of city j
N	:	number of string
T_i	:	Task or processing time
T_{id}	:	Total idle time
m	:	number of workstation
CT	:	Cycle time
DOE	:	Design of experiment
GA	:	Genetic algorithm
TSP	:	Travelling Salesman Problem
TSPPC	:	Travelling Salesman Problem with precedence constraints
ATSP	:	Asymmetric Travelling Salesman Problem
STSP	:	Symmetric Travelling Salesman Problem
MTSP	:	Multi Travelling Salesman Problem

AGV	:	Automated Guided Vehicle
VRP	:	Vehicle Routing Problem
DARP	:	Dial-a-Ride Problem
SOP	:	Sequencing Ordering Problem
PDTSP	:	Pick-up and delivery problem
PC	:	Precedence constraints
NP-hard	:	Non-deterministic polynomial hard
ALB	:	Assembly line balancing
EOL	:	End-of-life
MOTSP	:	Multi-objective travelling salesman problem
TS	:	Topological sort
SP	:	Selective pressure
T_s	:	Tournament selection
Pos	:	Position of an individual in the population
PMX	:	Partially mapped crossover
OX	:	Order crossover
LOX	:	Linear order crossover
CX	:	Cycle crossover
IM	:	Inversion mutation
pop_size	:	Population size
ngener	:	maximum number of generation
P_c	:	Probability of crossover
P_m	:	Probability of mutation
l	:	Chromosome length
PROX	:	Combination of roulette wheel selection, LOX and IM
TSOX	:	Combination of tournament selection, LOX and IM
RBOX	:	Combination of rank based roulette wheel selection, LOX and IM

Chapter 1

Introduction

1.1 Background of the study

The travelling salesman problem (TSP) is a classical model for various process sequencing and production scheduling problems. Many production scheduling problems can be reduced to a simple concept that there is a salesman who must travel from city to city, visiting each city exactly once and returning to the home city [1]. It is possible for the salesman to select the orders of the cities visited so that the total distances travelled in his tour is as small as possible which will apparently save him time and money [2]. Obtaining a solution to the problem of a salesman visiting *n* cities while minimizing the total distance travelled is one of the most studied combinatorial optimization problems [3]. It is called combinatorial optimisation because the optimum solution consists of a certain combination of variables from the finite pool of all possible variables e.g. optimal arrangement, grouping, ordering, or selection of discrete objects in finite number [4]. The TSP problem is well known to belong to the class of Non-deterministic Polynomial hard (NP-hard) problems [5].

The fundamental concept of travelling salesman problem with precedence constraints (TSPPC) is similar to TSP, except that it has a set of precedence to be followed by the salesman [6]. When a process can only be performed after another process was performed, TSPPC model can then be utilised. The TSPPC is mimicking the real problem in life; where there is always some order to be followed in performing the jobs [6]. In the context of manufacturing, process sequencing and production scheduling problem is the most frequent application that is modelled as travelling salesman problem with precedence constraints [7]. Besides sequencing and scheduling, robot path planning for assembly or material handling can also be modelled as TSPPC in order to find the shortest route [8-10].

1.2 Motivation of the study

The study on TSPPC is interesting as their concept can be applied to solve many scheduling and routing problems both in manufacturing and service industry. In the manufacturing industry, the problems are mostly dealing with process sequencing which arises as a subproblem in scheduling, routing and process planning. The process sequencing problem may be regarded as a generalization of the TSP with precedence constraints. TSPPC is harder to solve than the general TSP because the model formulations are complex and the algorithm for solving these models are difficult to implement [6]. Since the TSPPC belongs to the class of NP-hard problem, the optimal solution to the problem cannot be obtained within a reasonable computational time for large size instance [6]. For an algorithm that cannot execute in polynomial time the term Non-deterministic Polynomial time is used, meaning that the execution time needed by the algorithm is not a polynomial function of the problem size [11]. There are many manufacturing optimization problems that are NP-hard, including vehicle routing problems, bin packing problems and scheduling problems [12].

TSPPC is difficult to solve efficiently by conventional optimization techniques when its scale is very large. The earliest research on TSPPC problem was solved using exact methods such as branch-and-bound and dynamic programming. However the exact methods that guarantee to find the optimal solution of the problem are only capable of handling small and medium size of instances [13]. In addition, the size of the instances that are practically solvable is rather limited and the computational time increases rapidly with the instance size. The memory consumption of exact algorithms can also be very large and lead to the early termination of a program. Therefore it is necessary to develop more efficient algorithms for solving TSPPC problems.

Approximate or heuristic methods such as genetic algorithms, tabu search and simulated annealing do not guarantee the optimal solution [14], but empirically they have often been shown to return high quality solutions in short computation time. The genetic algorithm (GA) has powerful performance for solving combinatorial optimization problems, especially for sequencing problems such as TSP and flow shop scheduling [3, 15-17]. However, the use of conventional GA procedure for TSP with order-based representation might generate invalid candidate solution when applying to TSPPC problem. The infeasible sequences might be produced after crossover and mutation operations. A method to handle precedence constraints should be integrated in the GA procedure in order to generate only feasible solution during the evolutionary process. Hence, a study to develop an efficient genetic algorithm to obtain feasible and optimal solution of TSPPC is needed.

1.3 Objectives of the study

The aims of this study can be summarized as follows:

1. To study the components and the procedure of genetic algorithm to solve TSP and TSPPC.
2. To propose an efficient genetic algorithm for TSPPC that promising feasible and optimal solution.
3. To compare and verify the efficiency of the proposed algorithm with the existing algorithm (i.e. Moon's algorithm) through computational experiment.
4. To apply the proposed algorithm to different sequencing problems benchmark from the literature.

1.4 Scope of the study

The study will focus on solving process sequencing in an assembly line. This problem is categorized as an NP-hard problem that can be formulated as the travelling salesman problem with precedence constraints (TSPPC). Prior to solve TSPPC, several experiments on TSP instances with different GA parameters and operators will be carried out as part of the preliminary research towards the development of the GA framework for TSPPC.

The proposed optimisation algorithm in this study deals only with the genetic algorithm method. Therefore, the performance of the proposed algorithm is not compared with other heuristic methods such as tabu search, simulated annealing, ant colony or neural network.

In this study, the result of the proposed algorithm will be compared with another algorithm that solved similar applications, which is for process sequencing problem.

The proposed algorithm is not compared with other algorithms for different applications. A data on process sequencing from the related journals will be used to test the efficiency and the stability of the proposed algorithm.

In this study, the parameters used in conducting computational experiments are based on parameters that are reported in available scientific literatures and trial runs of the model. Simple design of experiments is also performed to obtain the best combination of parameters in finding optimal solutions.

1.5 Research process flow

The study is conducted under two main stages. The first stage is to study and examine the genetic algorithm procedure that is used to solve general TSP. At this stage, various genetic operators and parameters are explored. Understanding the coding used for programming the TSP in MATLAB environment is very crucial at this stage. New proposed algorithms to solve TSP are developed which benchmark from numerous research works. Various numerical experiments are then carried out to investigate the performance of the proposed algorithm to obtain the optimal solution to TSP problem. All the experiments will be performed in MATLAB software version 2009b.

The next stage is to use the proposed algorithm in the first stage and modify the procedure according to TSPPC limitation, and also to study previous methods that are used to solve TSPPC in solving sequencing problem. In this stage, the previous related TSPPC algorithm is modelled and simulated to ensure the algorithm is working as reported in the literature. Then, the algorithm limitations are identified.

A new improved algorithm for TSPPC is developed as the proposed solution to reduce some of the limitations of the previous method. In order to verify the efficiency and the stability of the proposed method, computational experiments of different test problems are performed. The performance (i.e. number of generations and iteration time to converge on the optimal solution) of the proposed algorithm is compared with the previous developed method. The results of the optimal solution obtained from the proposed algorithm are also compared with the results reported in the relevant published paper. Figure 1.1 presents an overall picture of the research process.

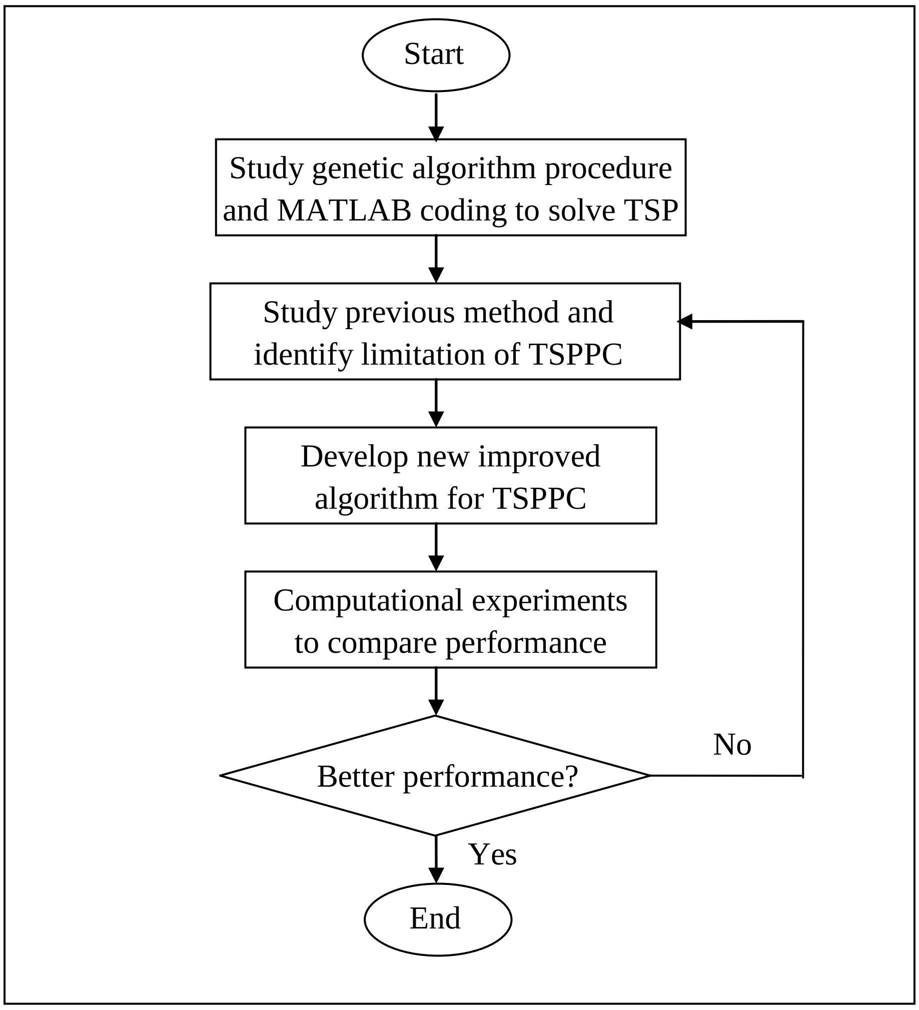

Start
Study genetic algorithm procedure and MATLAB coding to solve TSP
Study previous method and identify limitation of TSPPC
Develop new improved algorithm for TSPPC
Computational experiments to compare performance
Better performance?
No
Yes
End

Figure 1.1: Flowchart of the research process

1.6 Outline of the thesis

The thesis contains an introductory chapter which gives a brief introduction on TSP and TSPPC. The chapter also discusses the motivation of the study leading to the objectives and scope of the study. The remainder of the thesis is organized as follows;

Chapter 2: Literature review introduces the background knowledge of TSP which includes a detailed introduction of TSP, TSP related problems, TSPPC and their applications in different areas. The chapter also covers optimization techniques of exact and heuristic methods to solve TSPPC problem and gives the definition of NP-hard and the theory of computational complexity. The general knowledge of genetic algorithm operation, representation, operators, and parameters are also described in this chapter.

Chapter 3: Research methodology initially describes in detail the development of the GA procedure to solve TSP. Then this procedure is further used as a framework to solve TSPPC with some modifications in representation and genetic operators. A previous method that solved similar TSPPC problem is also benchmarked and modelled and the development process of the proposed GA to solve TSPPC is discussed in detail.

Chapter 4: Computational experiments and results provide computational experiment for TSP and TSPPC. It starts with a computational experiments objectives and set-up. The proposed algorithms are tested through various experiments and the results obtained by the proposed methods are reported and compared with the results obtained in related published paper.

Chapter 5: Discussion focuses on the observations and overall findings from the experiments.

Chapter 6: Conclusions and future work concludes and summarizes the research achievements and contributions, and finally suggest the directions for future research.

Chapter 2

Literature Review

The travelling salesman problem (TSP) concept is very easy stated, however it is hard to solve because classified as NP-hard problem. The importance of TSP and why it is hard to solve and the method to solve them will be explained in this chapter. The chapter starts with giving informal and formal description of the TSP, overview of TSP history and the complexity of TSP. The TSP related problem which is TSP with precedence constraints (TSPPC), the application and the techniques to solve them are briefly explained. Finally, a general concept of genetic algorithm operation, representation, operators, and parameters are given in more detail.

2.1 Travelling salesman problem (TSP)

TSP can be described as a salesman who starts the journey from his home city, visiting each city exactly once and then return to his home city. It is possible for the salesman to select the order of his visits so that the total of the distances travelled in his tour is as small as possible [2]. In this case, the problem is to minimise the total distance over the set of all tours or in terms of the graph theory, to find a Hamiltonian cycle of minimal length in a fully connected graph [18]. By a proper choice and scheduling of the tour, the salesman can often gain so much time to cover as many locations as possible without visiting a location twice [2]. Figure 2.1 shows an example of the simple TSP diagram.

A formal mathematical definition of the TSP is given as follows [19-20]: Let $G = (V, E)$ be a graph (directed or undirected) and F be the family of all Hamiltonian cycles (tours) in G. For each edge $e \in E$ a cost (weight) c_e is prescribed. Then the TSP is to find a tour (Hamiltonian cycle) in G such that the sum of the costs of the edges of the tour is as small as possible. Without loss of generality, assume that G is a complete graph. Let the node set $V = (1, 2, \ldots, n)$. The matrix $C = (c_{ij})_{nxn}$, is called the cost matrix (also referred to as the distance matrix or weight matrix), where the (i, j)th entry c_{ij}

corresponds to the cost of the edge joining node *i* to node *j* in *G*. The integer programming formulation for TSP is given as follows [19];

$$\text{Minimise } \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij} \qquad (2\text{-}1)$$

Subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, \ldots, n;\ i \neq j. \qquad (2\text{-}2)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, \ldots, n;\ i \neq j. \qquad (2\text{-}3)$$

$$u_i - u_j + nx_{ij} \leq n-1 \qquad i, j = 2, 3, \ldots, n;\ i \neq j. \qquad (2\text{-}4)$$

All $x_{ij} = 0$ or 1, All $u_i \geq 0$ and is a set of integers

Here variables x_{ij} represent the tour of the salesman travels from city *i* to city *j*. The cost or distance between city *i* and city *j* is denoted as c_{ij}. The objective function (2-1) is simply to minimise the total cost or distance travelled in a tour. Constraint set (2-2) and (2-3) ensures, respectively that the salesman enter and leaves each city exactly once. Constraint set (2-4) is to avoid the presence of sub-tour [19].

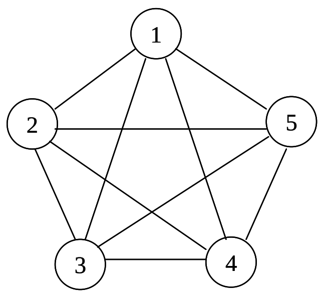


Figure 2.1: Example of undirected graph for five-city TSP

2.1.1 Overview of TSP history

The TSP has a long history and is a relatively old problem, and this history can help in the understanding of the problem and in understanding why it remains a significant problem. The TSP was documented as early as 1759 by Euler, whose interest was in solving the knight's tour problem [2]. A correct solution would have a knight visit each of the 64 squares of a chess-board exactly once on its tour. The concept of TSP was also treated by the Irish mathematician, Sir William Rowan Hamilton and by the British mathematician, Thomas Penyngton Kirkman in 1800s. Hamilton's Icosian Game was a recreational puzzle based on finding a Hamiltonian cycle. The general form of the TSP appears to be studied by Karl Menger in 1930s. Menger called TSP the "Messenger Problem", a problem encountered by postal messengers as well as by many travellers. The problem was later promoted by Hassler Whitney and Merill Flood. Flood was motivated to work on the TSP problem so as to reduce the costs for school bus routes in his district [21].

George Danzig, Ray Fulkerson and Selmer Johnson in their paper "Solution of a large-scale travelling salesman problem" proposed a novel method for solving instances of the TSP using linear programming [2, 22]. They used this technique to solve a problem containing 49 cities in the USA. Danzig, while working at the RAND Corporation, developed a technique to optimise solutions for combinatorial problems called the Simplex Algorithm. This algorithm was refined and later named the cutting-plane method. The cutting-plane method has been successfully applied to a wide range of problems in the combinatorial field [23]. During the 1960's the cutting plane method was adapted by Land and Doig to form the branch-and-bound searching technique. The branch-and-bound technique was applied to the TSP by Little *et al.* in 1963 [2]. The RAND Corporation's reputation helped to make the TSP a well known and popular problem. The TSP also became popular at that time due to the new subject of linear programming and attempts to solve combinatorial problems [24].

Since the late 1980's the Centre for Research on Parallel Computation (CRPC) at Rice University has examined the travelling salesman problem [23]. David Applegate, Robert Bixby and William Cook have examined a number of very large scale TSP problems. The problems evaluated were TSP problems in the region of 3000

– 15000 cities and were evaluated on super computers and large parallel computer systems. The technique that was implemented was the cutting-plane method [23].

As a discrete optimisation problem, the TSP is very difficult to solve optimally in polynomial time, despite its simplicity. Over the last two decades, with the increase of computing, the developments of efficient algorithms have contributed to the progress made in solving the TSP [10]. The growths of research in earlier time concentrated to solve larger instances of problems. It started with 49 cities and then increased to 24,978 cities in 2004 [25]. The milestone of the growth of instance is shown in Table 2.1. A library of TSP (TSPLIB) data sets is maintained at the University of Heidelberg by Professor Gerhard Reinelt [26]. TSPLIB is a library of sample instances for the TSP and related problems from various sources and of various types.

Table 2.1: Milestones in TSP instance solution [25]

Year	Researchers	Size of instances (cities)
1954	G. Dantzig, R. Fulkerson, and S. Johnson	49
1971	M. Held and R.M. Karp	64
1975	P.M. Camerini, L. Fratta, and F. Maffioli	67
1977	M. Grötschel	120
1980	H. Crowder and M.W. Padberg	318
1987	M. Padberg and G. Rinaldi	532
1987	M. Grötschel and O. Holland	666
1987	M. Padberg and G. Rinaldi	2,392
1994	D. Applegate, R. Bixby, V. Chvátal, W. Cook	7,397
1998	D. Applegate, R. Bixby, V. Chvátal, W. Cook	13,509
2001	D. Applegate, R. Bixby, V. Chvátal, W. Cook	15,112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, K. Helsgaun	24,978

2.1.2 The complexity of TSP

Although TSP is conceptually simple, it is difficult to obtain an optimal solution. The main difficulty of this problem is because the immense number of possible tours: (*n*-1)!/2 for symmetric *n* cities tour [27]. As the number of cities in the problem increases, the number of permutations of valid tours is also increasing e.g. for 5 cities is 12, 7 cities is 360 and for 10 cities is 181,440 possible permutations. It is possible to think of the TSP as a complete graph with *n* nodes where each edge of the graph is assigned a weight. These weights represent the distance or cost of moving from one node to another. The objective is to find a minimum distance Hamiltonian cycle of the graph [2, 28]. From a combinatorial view point one might ask how many Hamiltonian cycles must be examined in order to find a minimum cost circuit. Computing a possible tour of the graph, it is required to start at a particular node, from this node it is possible to visit any one of *n*-1 other nodes, and following the next move, any of *n*-2 other nodes, etc., the total number of circuits is therefore (*n*-1)!. However as it is possible to visit any circuit in the reverse cyclic order, then it would require (*n*-1)!/2 examinations of different circuits to compute the minimum distance Hamiltonian circuit [15]. It is this factorial growth that makes the task of solving the TSP immense even for modest *n* sized problems.

Non-deterministic Polynomial hard (NP-hard) problem

It can be said that the high attractiveness of the TSP is because it is classified as Non-deterministic Polynomial hard (NP-hard) problem, where the amount of computation required increases exponentially with the number of cities. This reason probably one of the important factors which attract researchers to study TSP [1]. In computer science, polynomial time refers to the running time of an algorithm that is the number of computation steps a computer requires to evaluate the algorithm. Polynomial time algorithms are said to be “fast”. Most familiar mathematical operations such as addition, subtraction, multiplication and division, as well as computing square roots, powers, and logarithms, can be performed in polynomial time.

An algorithm is a set of step-by-step instructions that, when executed in the order specified, will solve a certain problem. A problem is considered ‘easy’ if it can be solved by an algorithm that runs in polynomial time. On the other hand, a problem is considered ‘hard’ if it cannot be solved in polynomial time [11]. They are the basis of the computational complexity theory of problems that was developed. Polynomial algorithms are of polynomial order, for example $O(n^2)$ or $O(n^3\log(n))$, whereas exponential algorithms are of higher order, for example $O(2^n)$ or $O(n!)$. The solution times of exponential algorithms increase much more rapidly than the solution times of polynomial algorithms. The information on the speed of polynomial and exponential (non-polynomial functions) time algorithms can be referred from [11].

Non-deterministic polynomial-time (NP) defined as the class of problems that can be verified in polynomial time, in other words, the problems for which that can be checked by a polynomial time algorithm. Deterministic means that each step in a computation is predictable. It is possible to guess the solution (by some non-deterministic algorithm) and then check it, both in polynomial time [29]. If we had a machine that can guess, we would be able to find a solution in some reasonable time. Today nobody knows if some faster exact algorithm exists. Many people think that such an algorithm does not exist and so they are looking for some alternative methods, such as approximate algorithms.

2.1.3 TSP variations and applications

Several variations of the TSP that are studied in the literature have been originated from various real life or potential applications, and these variations can be reformulated as a TSP. Generally, the TSP is classified as a symmetric travelling salesman problem (STSP), asymmetric travelling salesman problem (ATSP), and multi travelling salesman problem (MTSP).

Symmetric TSP – Hamiltonian path and Euclidean distance

The most common problem in TSP is the shortest Hamiltonian path problem [2], which relate to a graph theory. Graph theory is one of the earliest problems still prominent in combinatorial mathematics [18]. A graph is a finite set of vertices which are joined by edges. A cycle in a graph is a set of vertices of the graph which is possible to move from vertex to vertex, along the edges of the graph, so that all vertices are encountered exactly once and finish where it started. If a cycle contains all the vertices of the graph, it is called Hamiltonian (from the name of the 19th century Irish mathematician, Sir William Rowan Hamilton). The TSP for a graph with specified edge lengths is the problem of finding a Hamiltonian cycle of shortest length [2, 16]. Here, the graph's vertices correspond to the cities and the graph's edges correspond to connections between cities.

An example of the Hamiltonian path problem is presented in Figure 2.2. In this figure, the node represents assembly processes, where the process A until F can be performed in any order. For example, the assembly process needs to be started and finished in station O. The engineer needs to consider all possible paths to determine the shortest route for this problem. The Hamiltonian path problem is also known as "symmetric TSP" [16] because the distance for example between A to B and B to A is equal. The distance between the two points in the below diagram can be determined by Euclidean distance formulation [28].

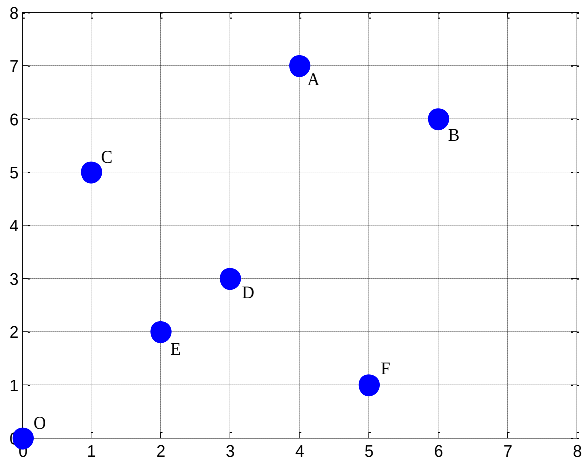


Figure 2.2: Example of Hamiltonian path problem

A direct application of the symmetric TSP is in the drilling problem of printed circuit board [30]. In order to connect a conductor on one layer with a conductor on another layer, or to position the pins of integrated circuits, holes have to be drilled through the board. The holes may be of different sizes. In order to drill two holes of different diameters consecutively, the head of the machine has to move to a toolbox and change the drilling equipment. This is quite time consuming. Thus it is clear that one has to choose some diameters, drill all holes of the same diameter, change the drill, and drill the holes of the next diameter. Thus, this drilling problem can be viewed as a series of TSP. The distance between two cities is given by the time it takes to move the drilling head from one position to the other and the aim is to minimize the travel time for the machine head.

Asymmetric TSP

The asymmetric travelling salesman problem is when the cost of travelling from node A to node B is not the same as the cost from node B to node A [16]. This can be solved in the same way as the standard TSP if certain edge weights that ensure there is a Hamiltonian cycle in the graph are applied. For example, let consider the Figure 2.2 as the nodes that need to be visited by an automated guided vehicle (AGV) to load and unload materials for work stations. Let assume the speed of AGV for moving forward and backward are different. Therefore travelling time from node A to B for moving forward is different with travelling time from B to A with moving backward. In this case, engineer need to decide the fastest travelling time to visit all nodes. Minimising the makespan on a single machine with sequence-dependent setup times is an example of application of asymmetric TSP. When setup times are dependent on the sequence, minimising makespan becomes equivalent to minimising the total setup time. If the setup times for all pairs of jobs are indifferent to their sequencing order when scheduled consecutively, the scheduling problem is equivalent to a symmetrical TSP, otherwise, it is equivalent to an asymmetrical TSP [31].

Multi-salesman TSP

The multi-salesman problem is the same as Hamiltonian problem, except that it has more than one salesman [16]. The scheduler need to decide where to send each salesman, so that every city is visited exactly once and each salesman returns to the original city [32]. For example, let consider the problem in Figure 2.2 as a layout of the manufacturing process, where two similar AGV need to serve six stations (A, B, C, D, E, F). The first AGV will start and finish the job from (0, 0) and the second AGV need to start and finish the job from (8, 8). Both of AGVs need to serve three stations each other. In this case, the AGVs need to find the shortest route to serve three stations and return to the starting point. The minimum total distance for both AGVs is considered as the best route. Another application of MTSP is for school bus routing scheduling in which the objective of the scheduling is to obtain a bus loading pattern such that the number of routes is minimized, the total distance travelled by all buses is kept at minimum, no bus is overloaded and the time required to traverse any route does not exceed a maximum allowed policy [33]. Apart from that, scheduling sequence-dependent setup times and makespan minimisation on parallel machine has been considered by Guinet [34] and this problem is similar to a vehicle routing problem which can be modelled as multi-travelling salesman problem.

2.1.4 TSP with precedence constraints (TSPPC)

Basic TSP has neither constraint nor priority given to any cities. The TSP with precedence constraints (TSPPC) is one in which a set of *n* nodes and distances for each pair of nodes are given, the problem is to find a tour from node 1 to node *n* of minimal length which takes given precedence constraints into account. In TSPPC some order of cities is given and we ought to visit cities in that order only. TSPPC differs from traditional TSP whereby in TSPPC, there is no need to return to the original city. TSPPC becomes more important because in real life problems we always have to follow some orders. An example of TSPPC is shown in Figure 2.3. Each precedence constraint requires that some node *i* have to be visited before some other node *j* [35]. In a directed graph, the vertices (circles) represent activities or tasks and the edges represent the precedence relations between activities [6]. The task dependencies deal with the relationships between giving tasks and how they affect each other. The four types of task dependencies are Finish-to-start in which predecessor task must be finished before the successor can start, Start-to-finish in which successor task can finish only after the predecessor task has started, Start-to-start in which two tasks can start simultaneously and Finish-to-finish in which two tasks must finish at the same time [36]. The TSPPC in this study is classified as Finish-to-start types of task dependencies.

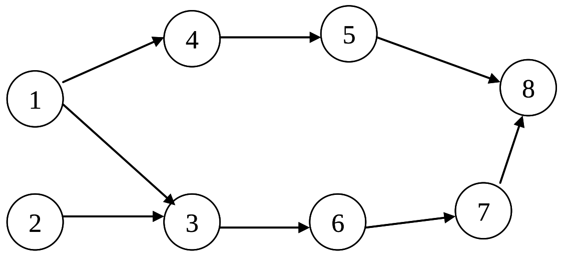


Figure 2.3: Example of TSP with precedence relationships

The term of the travelling salesman problem with precedence constraints (TSPPC) was formerly used by Kusiak and Finke in 1987 to solve machine scheduling problem using the exact method. In 2002, Moon *et al.* introduced more efficient method to solve TSPPC instances. In some researches, other terms are also used to represent TSPPC problem such as precedence constraint routing problem [37], precedence constraint travelling salesman problem (PCTSP) [35], Asymmetric TSP with precedence constraint (ATSP-PR) also referred to as sequencing ordering problem (SOP) with precedence constraint [38-39], pickup and delivery with time window and precedence constraint [40], Dial-a-ride problem (DARP), and directed vehicle routing problem (DVRP) [10].

The sequential ordering problem (SOP) with precedence constraints was first formulated by Escudero in 1988 to design heuristics for a production planning system. It consists of finding a minimum weight Hamiltonian path on a directed graph with weights on the arcs and the nodes, subject to precedence constraints among nodes [38]. Gambardella and Dorido [39] modelled the problem by considering a complete graph $P = (V, R)$ with node set V and arc set R, where nodes correspond to jobs 0,… i, …, n.. To simplify the explanation, reconsider the previous diagram in Figure 2.3. The nodes set V consist of (1, 2, 3, 4, 5, 6, 7, 8). Precedence constraints given by set R which contain [(1,3), (1,4), (2,3), (3,6), (4,5), (5,8), (6,7), (7,8)]. This set is ascertained from arc $(i, j) \in R$ if job i has to precede job j in any feasible solution. Each arc (i, j) is associated a cost t_{ij}. This cost represents the required waiting time between the ends of job i and the beginning of job j. Each of node i is associated a cost $p_i \in R$, which represents the processing time of job i [39]. Given this definition, the TSPPC can be stated as the problem of finding a job sequence subject to the precedence constraints which minimise the total makespan. There is therefore equivalent to the problem of finding a feasible Hamiltonian path with minimal cost under precedence constraints given by set R [39].

Moon [6] has used the two-commodity network flow model to formulate TSPPC. In this formulation, c_{ij} is the travel distance from vertex v_i to v_j and s is the first selected vertex in the graph. Commodity p is supplied by $(n-1)$ units at a selected starting node and used by one unit at each node that is not the starting node while commodity q is consumed by $(n-1)$ units at the starting node and supplied by one unit at the other nodes. Here n is the number of nodes or cities. Three variables are used for the two-commodity network model: x_{ij}^{p} is the quantity of commodity p from vertex v_i to v_j and x_{ij}^{q} is the quantity of commodity q from vertex v_i to v_j.

$$x_{ij} = \begin{cases} 1 & \text{if vertex } j \text{ is visited immediately after vertex } i, \\ 0 & \text{otherwise,} \end{cases}$$

The two-commodity network flow model for the TSPPC can be described as follows:

$$\text{Minimise } \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} \frac{1}{(n-1)} c_{ij} \left(x_{ij}^{p} + x_{ij}^{q} \right) \tag{2-5}$$

Subject to

$$\sum_{j=1}^{n} x_{ij}^{p} - \sum_{j=1}^{n} x_{ji}^{p} = \begin{cases} n-1 & \text{for } i = s, \\ -1 & \text{otherwise,} \end{cases} \tag{2-6}$$

$$\sum_{j=1}^{n} x_{ij}^{p} - \sum_{j=1}^{n} x_{ji}^{p} = \begin{cases} -(n-1) & \text{for } i = s, \\ +1 & \text{otherwise,} \end{cases} \tag{2-7}$$

$$\sum_{j=1}^{n} \left(x_{ij}^{p} + x_{ij}^{q} \right) = n-1 \quad \forall i, \tag{2-8}$$

$$x_{ij}^{p} + x_{ij}^{q} = (n-1) x_{ij} \quad \forall i \text{ and } j, \tag{2-9}$$

$$\sum_{j=1}^{n} x_{uj}^{p} - \sum_{j=1}^{n} x_{vj}^{p} \geq 1 \quad \text{for } (v_u \to v_v)(v_v \neq s), \tag{2-10}$$

$$x_{ij}^{p} \geq 0 \quad \forall i \text{ and } j, \tag{2-11}$$

$$x_{ij}^{q} \geq 0 \quad \forall i \text{ and } j, \tag{2-12}$$

$$x_{ij} \in \{0,1\} \quad \forall i \text{ and } j. \tag{2-13}$$

The objective function (2-5) is to obtain the total travel distance for all vertices, and the sum of commodities p and q between vertices v_i to v_j on any feasible sequence (i.e. $x_{ij} = 1$) is equal to $n-1$ (i.e. $x_{ij}^{p} + x_{ij}^{q} = n-1$). Constraints (2-6) and (2-11) are used to ensure the feasibility of flow of commodity p while constraints (2-7) and (2-12) are for feasibility of commodity q. Constraint (2-8) ensures a feasible tour, i.e. feasible sequence. Constraint (2-9) explains that, if $x_{ij} = 1$, the sum of commodities p and q between v_i and v_j be $n-1$. Constraint (2-10) is for the precedence relationship between vertices.

2.1.5 TSPPC application in manufacturing industry

The travelling salesman problem with precedence constraints has wide applications in manufacturing industry. In manufacturing, the TSPPC can be modelled to represent assembly process [6]. In the real world, the assembly process of a product has a certain order that must be followed. The assembly process cannot be completed by disobeying precedence rules. The weight in the assembly process to be modelled as TSPPC can be a transition time or travelling distance between one process to another process. Commonly, the objective of implementing TSPPC as assembly process is to find the minimum makespan, minimum travelling time or minimum travelling distance [6]. Apart from assembly process, the TSPPC can also be used to model disassembly process. The disassembly optimisation problem was formulated as a precedence-constrained asymmetric TSP by Sarin *et al.* [41]. Generally, once electronic products reach their end-of-life (EOL), they are sent to one of the EOL processes (i.e. remanufacturing, reuse, recycling or disposal) for environmental protection purpose [42]. In most of the processes, a certain level of disassembly may be necessary in order to extract components of the products. Therefore, optimal disassembly sequences are very important in order to increase the efficiency of the disassembly process and must be generated in order to help minimise the disassembly complexity and time [43]. Disassembly planning focus on disassembly sequence planning which aims to achieve a feasible disassembly sequence with minimum cost or time [44].

Another application of TSPPC in manufacturing system is for planning and scheduling purpose. The assembly planning problem is to generate the sequences for the assembly machines that transform the assembly operation from an assembly operation to an assembly product [45]. In scheduling, a finite set of jobs is processed on a finite set of machines. Each job is characterized by fixed order of operation, where it needs to process on specific machine for a specific duration before another process can be performed [8, 46]. A schedule is an assignment of operations to time slots on the machine. The objective of scheduling is to minimise the maximum completion time of jobs [8]. In this case, the processes are modelled as node and the processing time as weight. The processing order is modelled as precedence constraints in TSPPC [35]. When a proper schedule is produced, better production management is achieved. In job shop production, the demand of products varies and sometimes unpredictable [47]. Therefore, an optimisation tool that has faster response and the ability to find the optimum configuration in a shorter time is required. By generating an optimum schedule in shorter time, the time to start production can be reduced [48].

The TSPPC also can be modelled to represent a vehicle routing problem (VRP) in manufacturing [1, 49]. This application is related to material handling system that uses automated guided vehicle (AGV) to deliver and pick up the work pieces. For this purpose, a work piece needs to be transported from one station to another station by following the precedence rule. The AGV has to find the shortest path to perform delivery and pick up of work pieces on the selected stations. It needs to find the shortest path because each time the AGV starts working, the number and location of workpieces to be delivered and picked up is different. Therefore, the AGV needs to identify destinations and the shortest path to be followed [1]. The vehicle operators require timely solutions due to their dynamic working environment. If the system can generate the shortest path in shorter time, the AGV can perform the works earlier. It gives huge benefits to the overall manufacturing system because each process can start earlier.

In many real manufacturing situations, setup operations such as cleaning or changing tools, are not only often required between jobs but they are also strongly dependent on the immediately preceding process on the same machine [6]. Sometimes, the setup times are modelled as set up costs, which are then minimised. The idea is that setup costs and setup times are related, so that minimising setup costs will lead to minimised makespan. Minimising the makespan on a single machine with sequence dependent setup times is equivalent to the travelling salesman problem [31]. The TSP is solved where the distances between cities represent the sequence dependent setup times between jobs. This result in a sequencing of jobs that yields a schedule when combined with the processing times and sequence dependent setup times. Bitran & Gilbert [50] developed a travelling salesman problem based heuristic to minimise total setup costs on parallel machines where the setup costs are not only sequence dependent but can be divided into two classes that vary in magnitude and degrees. The classes generally correspond to major setup times and minor setup times. Factory managers must decide which products to make in which periods, and the exact production sequence and production quantities, in order to minimise the sum of setup and inventory holding costs. Gilmore and Gomory [51] modelled and solves sequence-dependent setup time as a TSP. In sequence-dependent setup time, the distance between cities is the setup times between the jobs [52]. The path between cities *i* and *j* is in fact the setup time of the machine to change from the job *i* to the job *j*. Setup time can be defined as the time required to prepare the necessary resource (e.g. machines, people) to perform a task (e.g. job, operation). For example, in the printing industry, a setup time is required to prepare the machine for cleaning which depends on the color of the current and immediately following jobs. When setup times are dependent on the sequence, minimising makespan becomes equivalent to minimising the total setup time. Minimising the makespan on each single machine with sequence dependent setup time is equivalent to a Travelling Salesman Problem, which is strongly NP-hard [53].

There are also some of the examples that cannot be transformed to the TSP and TSPPC, but share some characteristics of the TSP, or in which the TSP comes up as a sub-problem [54]. Since many applications can be modelled as TSP and TSPPC, it has attracted many researchers from different field such as mathematics, operational research, artificial intelligence and Physics [55]. The TSP problem has also been used during the last years as a comparison basis for improving several optimisation techniques, such as genetic algorithm [6, 35, 56], simulated annealing [57], Tabu search [8], local search [12, 58], and ant colony [9, 39].

2.2 Optimisation techniques

Optimisation involves a mathematical formulation in order to get the ‘best’ solution for the problem [4]. The terminology ‘best’ solution implies that there is more than one solution can be obtained and the solutions are not of equal value. Optimisation tools should be used for supporting decisions rather than for making decisions, i.e. should not substitute decision-making process [59]. The input to the process to be optimised consists of variables and the process or function is known as the cost function, objective function, or fitness function; and the output is the cost or fitness. An optimisation problem is either a maximisation problem or minimisation problem with an associated set of instances. In the case of TSP, the optimisation become minimisation as the objective is to minimise the total travelling distance or minimise the cost.

2.2.1 Single & multi-objective optimisation

Most travelling salesman problems discussed in the previous sections are dealing with only one objective optimisation which is minimising the travelling distance or cost. However, many real-world decision making problems need to achieve several objectives such as maximising profits while minimising cost and maximising customer satisfaction (e.g. on time delivery) while minimising on hand inventories, minimise risks, maximise reliability, maximise performance, minimise deviations from desired levels, etc [60]. Unlike single objective optimisation, the solution of multi-objective problem is not a single point, but a family of points known as the Pareto-optimal set, Pareto front, non-dominated or non-inferior solutions [59, 61-62]. The illustration of Pareto front in multi-objective optimisation is depicted in Figure 2.4.

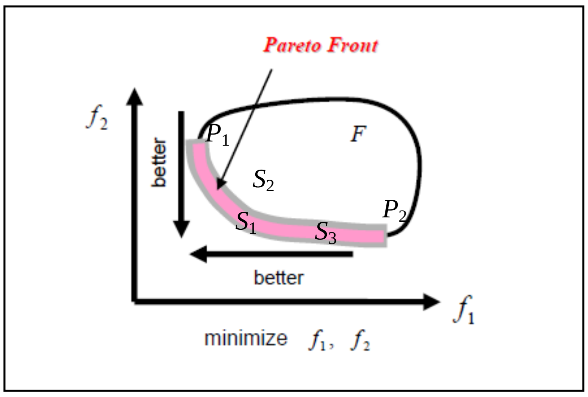


Figure 2.4: Illustration of Pareto front for a bi-objective optimisation problem [61]

In the above diagram, by assuming minimisation problem, the Pareto front is the boundary between the points P_1 and P_2 of the feasible set F. Solutions S_1 and S_3 are non-dominated Pareto optimal solutions. Solution S_2 is not Pareto-optimal as solution S_1 has simultaneously smaller values for both objectives, and solution S_1 should be accepted rather than solution S_2. Therefore the aim of multi-objective optimisation is to obtain a representative set of non-dominated solutions [61].

Gholamian [63] studied multi-objective travelling salesman problem (MOTSP) and the system has been developed for them. MOTSP solutions can be represented as a sequence of cities with related objective values. In his study, a bi-objective model is developed with opposite goals (maximum profit and minimum distance) for 8 cities problem. Multi-objective problems find applications in many areas, e.g. in production scheduling, project scheduling, production facility design, vehicle routing, supply chain management and many others [60]. As this study is only dealing with single objective optimisation which is minimisng the travelling distance, the multi-objective optimisation concept will not be discussed in more detail.

2.2.2 Exact methods

A brute force approach to solving an instance of a TSP is simply to list all the feasible solutions, evaluate their objective functions, and pick the best. However, this approach of complete enumeration is likely to be inefficient because of the vast number of possible solutions. Therefore many algorithms are developed to solve TSP efficiently. Most of the researchers state that there are two possible approaches of optimisation algorithm that can be classified as exact and heuristic (approximate) method. The exact methods will generate all possible solutions, while the heuristic methods only generate solutions according to evolution algorithm. It means that heuristic methods do not generate all solutions for the problem and do not guarantee the optimal solution.

Exact methods like branch-and-bound and dynamic programming are seen to be an effective solution of combinatorial optimisation problems and they have particular advantages and disadvantages. These methods always lead to the optimal solution [59]. On the other hand there exists a limit on problem size for exact methods. An advantage of exact methods for satisfaction problems is that they are able to show that instances cannot have solutions [12]. In the early development of the algorithm for the TSP, many researchers focusing on the exact method to solve TSP instance. In 1950s, George B. Dantzig, D. R. Fulkerson, and S. M. Johnson provided step by step application of the Dantzig-Fulkerson-Johnson for a 10-city example using linear programming. At the beginning 1960s, R. Bellman used the TSP as an example of a combinatorial problem that can be solved via dynamic programming [64]. Then, in 1970s, R.M. Karp and M. Held introduced branch-and-bound algorithm and solved 42-

city instance of Dantzig, Fulkerson and Johnson, the 57-city instance of Karg and Thomson, and 64-city random Euclidean instance [21].

The research on TSPPC which started in the early 1980s until the end of 1990s was focusing to solve the problem by using exact methods. Kusiak and Finke [65] developed a branch-and-bound algorithm for solving the single-machine scheduling problem with sequence-dependent setup times and precedence constraints. A lower bound was determined by solving a network formulation. Fischetti and Toth [66] also used branch-and-bound algorithms in which they proposed an additive approach to compute the lower bounds of TSPPC sequences. On the other hand, Savelsbergh and Sol [49] applied dynamic programming to solve dial-a-ride problem (DARP) modelled as TSPPC where a vehicle should transport a number of passengers. Each passenger should be transported from a given location to a given destination. Mingozzi *et al.* [40] presented dynamic programming strategies for the TSP with time window and precedence constraints. They developed a dynamic programming algorithm based on state space relaxation procedures for computing lower bounds to use in the branch-and-bound scheme. Fagerholt and Christiansen [67] consider a TSPPC with time window and allocation to solve the bulk ship scheduling problem using dynamic programming.

2.2.3 Heuristics methods

One of the drawbacks of exact methods for the TSP is that they usually take a long time to solve the problem or prove that no solutions exist in the case of satisfaction problems. Thus, it can only handle smaller size problems [68]. Heuristics or approximate methods were developed to find the near optimal solution for larger dimension problems within a reasonable CPU time. The term heuristics derive from the Greek *heuriskein* meaning to find or discover [69]. Heuristic algorithms have become a popular alternative to exact algorithms mainly because of their ability to handle more complex problems, larger size problems, and the numerous side constraints [13].

The most widely and successfully applied heuristic algorithms are local search algorithm. The general schemes to improve local search algorithms are called *metaheuristics* [12]. A metaheuristics is defined to be a general heuristic method which is used to guide an underlying local search algorithm towards promising regions of the search space containing high quality solutions [70]. The metaheuristics algorithm has often been inspired by analogies to naturally occurring phenomena like the physical annealing of metals or biological evolution. The most famous metaheuristics include genetic algorithm [71], simulated annealing [57], tabu search [72] and ant colony [39].

Zanakis and Evans [73] describe heuristics as simple procedures designed to provide good but not necessarily optimal solutions to difficult problems, easily and quickly. Typically, a heuristic for the TSP and VRP is categorized as either a tour construction algorithm, which involves gradually building a solution at each step, or a tour improvement algorithm, which improves upon a feasible solution [13]. The survey based on 442 articles revealed that heuristics are used more frequently in the area of production and job shop scheduling where mathematical programming solutions are rather cumbersome.

Psaraftis [37] develop *k*-interchange procedures to perform local search to solve Dial-a-Ride problem (DARP) with the objective of minimising the length of the tour travelled by a vehicle to service a number of customers from a distinct origin to a distinct destination. The DARP is a TSPPC in which the precedence constraints exist between the origin and destination of each customer on a feasible Dial-a-Ride tour. On the other hand, Escudero [74] performed a local search that uses 3 and 4-change based procedures to solve production planning system in the flexible manufacturing system (FMS).

Renaud *et al.* [10] proposed a heuristic method for the pickup and delivery problem which is formulated as the TSPPC model. The problem of pickup and delivery travelling salesman problem (PDTSP) was transformed into TSPPC by considering pickup customers as line-haul customers (i.e. one way journey from terminal to terminal) and delivery customers as back-haul customers (i.e. return journey to the original destination). Consequently, all pickup customers will be visited before visiting any delivery customers. Thus, this problem can be solved as TSPPC [10]. Basically, the heuristic algorithm that proposed by Renaud is composed of two phases. The first phase, called '*Double Insertion*' heuristic, inserts each delivery customer simultaneously with the associated pickup customer. The second phase called the '*Deletion and Re-insertion heuristic*', as an improvement procedure that uses improvement heuristic [10].

Gambardella and Dorigo [9] presented ant colony system to solve sequential ordering problem (SOP). The SOP which was first formulated by Escudero [74] can be stated as the problem of finding a job sequence that minimises the total makespan subject to the precedence constraint. The most distinctive feature ant colony optimisation is the management of pheromone trails that are used, in conjunction with the objective function, to construct new solution. Gambardella and Dorigo have designed a constructive algorithm called ACS-SOP in which a set of artificial ants builds feasible solutions to the SOP and a local search specialized for the SOP, and the resulting algorithm called the Hybrid Ant System for the SOP (HAS-SOP).

Hurink and Knust [8] were applied TSPPC to solve a scheduling problem, with the objective to determine a feasible schedule which minimises the makespan. They proposed Tabu search algorithm for scheduling a single robot in job-shop environment. In their problem, a single machine scheduling, which arises as a sub-problem in a job-shop environment was considered. The jobs additionally have to be transported between the machines by a single transport robot. Each job consists of a chain of operations which have to be processed in this order. With each operation, a dedicated machine is associated with which the operation has to be processed without pre-emption for a given duration.

Moon *et al.* [6] formulated the TSPPC as a network model. They use a topological sort (TS) technique, which is defined as an ordering of vertices in a directed graph. They proposed a new crossover for genetic algorithm (GA) which named *moon* crossover to solve TSPPC. The proposed algorithm was applied to process sequencing problem, which mainly applied to allocate assembly task in work stations [6]. They found that, the proposed algorithm came out with a better solution for the larger size problem compared to the traditional GA. Therefore, they conclude that their proposed GA is an efficient method for the TSPPC [6]. The proposed approach was then applied to solve process planning and scheduling in a multi-plant [56] with the objective to determine optimal schedule of machine assignments and operations sequences of all parts so that the makespan is minimised.

Table 2.3 shows the chronology of TSPPC related problems solved with different methods.

Table 2.2: Summary of methods to solve TSPPC

Researcher	Year	Method	Application
Moon, Kim, Choi and Seo	2002	Genetic algorithms	Process sequencing
Hurink and Knust	2002	Local search + Tabu search	Robot scheduling
Gambardella and Dorigo	2000	Local search + Ant Colony	Sequential ordering problem
Renaud and Boctor	2000	Heuristic search	Pick up & delivery
Fagerholt and Christiansen	2000	Dynamic programming	Bulk ship scheduling
Mingozzi, Bianco and Ricciardelli	1997	Dynamic programming	Dial-a-ride
Savelsberg and Sol	1995	Dynamic programming	Dial-a-ride
Fischetti and Toth	1989	branch-and-bound	Vehicle routing problem
Escudero	1988	Heuristic + Local search	Production planning in FMS
Kusiak and Finke	1987	branch-and-bound	Machine scheduling
Psaraftis	1983	Heuristic + Local search	Dial-a-ride

Evaluation of the performance of a heuristic

Silver [75] described that two main measures of performance exist to evaluate the performance of a heuristic method. First, comparing the obtain value of the objective function to the achievable by the optimal solution or some other benchmark procedure, which mentioned in Dannenbring, 1977 [76]. Then, second, the computational requirements in terms of computational effort and memory consumption for realistic sized problems. A natural question arising for approximate or heuristic algorithms is how close, in the worst case, is the returned solution to the optimum. To indicate the quality of the returned solution, the *relative error* is defined as follows [12]:

The *relative error* of a feasible solution y with respect to an instance x of an optimisation problem Π is defined as

$$E(x, y) = \frac{|opt(x) - f(y)|}{\max\{opt(x), f(y)\}} \tag{2-14}$$

The relative error is close to 0 if the feasible solution is close to the optimum. Conversely, the relative error is close to 1 if the feasible solution is far from the optimal solution.

Global and local optima

The significance of using any heuristic method is to get a global optimum (best minimum) solution instead of local optima (suboptimal minimum). A global optimum is the point in the search space with the highest fitness value while a local optimum is a point whose fitness is higher than all its near neighbours but lower than that of the global optimum [77]. By plotting the fitness for a two-dimensional search space, a fitness landscape can be obtained as illustrated in Figure 2.5. The landscape is smooth or correlated if neighbouring points in the search space have a similar fitness and it is rugged if neighbouring points have very different fitnesses. Rugged landscapes typically have large numbers of local optima. Although exhaustively evaluating the fitness of each point in the search space will always reveal the optimum, this is usually impracticable because the hugeness of the search space [77]. Thus, the essence of all the heuristic optimisation techniques is to determine the optimum point in the search space by examining only a fraction of all possible candidates as in the case of genetic algorithm.

A fitness landscape such as in Figure 2.5 is a convenient medium for displaying the GA's performance. However, fitness functions for real world problems cannot be easily represented graphically. Instead, performance graph is used. Since GA is stochastic, their performance usually varies from generation to generation. As a result, a curve showing the average performance of the entire population of chromosomes as well as a curve showing the performance of the best individual in the population is a useful way of examining the behaviour of a GA over the chosen number of generations.

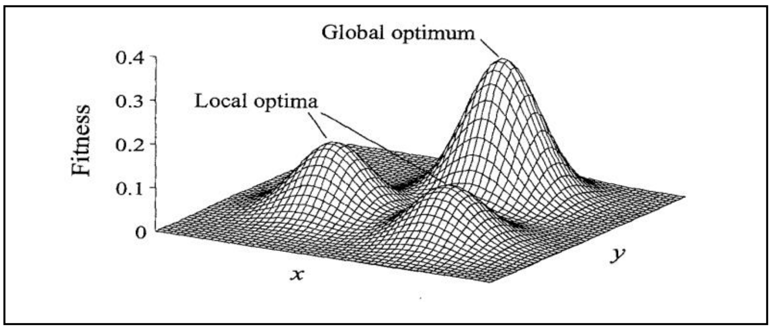


Figure 2.5: Illustration of fitness landscape [77]

2.3 Genetic algorithm operation, representation, operators and parameters

The genetic algorithm is an optimisation technique, based on natural evolution, developed by John Holland in 1975 at the University of Michigan. This technique imitates the biological evolution theory; whereby the concept of "survival of the fittest" exists. GA provides a method of searching which does not need to explore every possible solution in the feasible region to obtain a good result [47]. In nature, the fittest individuals are most likely to survive and mate, therefore the next generation should be fitter and healthier because they were bred from healthy parents. This same idea is applied to a problem by first 'guessing' solutions and then combining the fittest solutions to create a new generation of solutions which should be better than the previous generation [16].

In 1992 John Koza has used genetic algorithms to evolve programs to perform certain task. He called his method "genetic programming" [78]. Since then, many versions of evolutionary programming have appeared with varying degrees of success. The first researcher to tackle the travelling salesman problem with genetic algorithm was Brady in 1985. His example was followed by Grefenstette *et al.*, Goldberg and Lingle, Oliver *et al.*, and many others [27]. In the recent years, many researchers [3, 79-86] focused on developing more efficient GA to solve TSP related problems by further optimising the parameters, evaluating the best combination of operators and developing a mechanism to produce new fitter offspring in the standard GA procedure. Surveys of GA for TSP are compiled by Potvin [87] and Larranaga [27].

Genetic algorithm is different from other heuristic methods in several ways. The most important difference is that a GA works with a population of possible solutions at each iteration process, while other metaheuristic methods like Tabu search and simulated annealing use a single solution in their iterations [88]. GA can quickly scan vast solution set so that bad candidate solutions do not affect the end solution negatively as they are simply discarded. GA technique is robust and can deal successfully with a wide range of difficult problems. It does not require derivatives or auxiliary information. The use of an objective function to determine the quality of a solution is the only information to guide the search. GA does not guarantee to find the global optimum solution to a problem, but it is generally good at finding acceptably good solutions to problems acceptably quickly especially for a large-size instances. Two important characteristics of the genetic algorithm which are *exploitation* and *exploration* [88]; Exploitation is the ability to find good solutions quickly by make use of knowledge found at points previously visited to help find better points, whereas exploration describes the behaviour of maintaining a set of diverse individuals, which is to investigate new and unknown areas in the search space. GA is also well suited for parallel computers [89].

Despite some of the very best advantages, the problem with GA is that the genes from a few comparatively highly fit (but not optimal) individuals may rapidly come to dominate the population, causing it to converge on a local maximum. Once the population has converged, the ability of the GA to continue to search for better solutions is effectively eliminated. Crossover of almost identical chromosomes produces little that is new. Only mutation remains to explore entirely new ground, and this simply performs a slow, random search. For such cases, some of the conventional methods outperform the GA, quickly finding the minimum while the GA is still analyzing the costs of the initial population [89]. For problems that are not too difficult, other methods may find the solution faster than the GA. There is also a problem that can occur with GA known *premature convergence* which means that an individual that is fitter than others at earlier stages may dominate on the reproduction process leading to a local optimum convergence rather than a more thorough search that could lead to a global optimum [89]. GA also tends to be computationally expensive especially when using large population size. To use the GA, the solution to the problem is represented as a chromosome. Therefore, a method to encode the problem solutions as chromosomes must be well designed.

2.3.1 Genetic algorithm operation

The GA operation is based on the Darwinian principle of survival of the fittest and it implies that the 'fitter' individuals are more likely to survive and have a greater chance of passing their 'good' genetic features to the next generation [71, 90]. Figure 2.6 illustrates the basic operation of GA while the general procedure is given in Figure 2.7. In the standard or basic procedure of GA [27, 90-91], an initial population is created containing a predefined number of individuals (i.e. solutions). Each individual has an associated fitness measure, typically representing an objective value. The concept that fittest (or best) individuals in a population will produce fitter offspring is then implemented in order to reproduce the next population. Selected individuals are chosen for reproduction (by crossover and mutation) at each generation, with an appropriate crossover and mutation factor to randomly modify the genes of an individual. The algorithm identifies the individuals with the optimising fitness values, and those with lower fitness will naturally get discarded from the population. Once crossover and mutation is done, a new generation is formed and the process is repeated until some stopping criteria have been reached [92]. A comprehensive explanation on the genetic algorithm operation can also be referred to Netnevitsky [93] and Michalewicz [94].

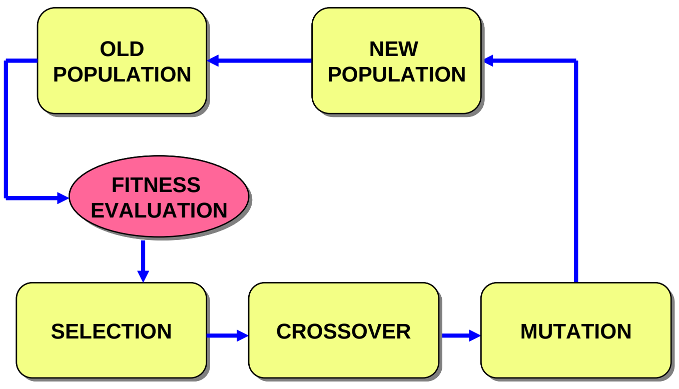


Figure 2.6: Basic operation of genetic algorithm

```
Begin GA
   Initialize population, $P$
   For generation_count < $k$ do
    Evaluate $P$ ($i$)
    Begin
            Select parents from the population
            Produce children by crossover from selected parents
            Mutate the individuals
            Increment generation_count
     End
     Return the best solution
End GA
```

Figure 2.7: General procedure of genetic algorithm [27, 90]

2.3.2 Chromosome representation

In genetic algorithm, each individual i.e. chromosome, that is a member of the population represents a potential solution to the problem [92]. A chromosome is a string of *gene* positions, where each gene position holds an *allele* value that constitutes a part of the solution to the problem. The allele value of a gene's position represents an element from a finite alphabet. Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form that a computer can process. One common approach is to encode solutions as binary strings: sequences of 1's and 0's. Another similar approach is to encode solutions as arrays of integers or decimal numbers, or to represent chromosomes as strings of letters. The virtue of all these methods is that they make it easy to define operators that cause the random changes in the selected candidates [95].

There are a number of possible chromosome representations, due to a vast variety of problem types. Larranaga [27] reviewed and compiled the different types of representations, crossover and mutation operators used in the GA. Although many types of representation have been developed, there will be only one representation type which is most commonly used for TSP will be discussed in this thesis, that is *path* representation.

Path representation

The *path* representation also called *permutation* representation [70] is probably the most natural representation of a TSP tour [27]. In this representation, the *n* cities that should be visited are put in order according to a list of *n* elements, so that if the city *i* is the *j*-th element of the list, city *i* is the *j*-th city to be visited. This representation allows a great number of crossover and mutation operators to have been developed. In TSP, there are a number of cities, where each pair of cities has a corresponding distance. The aim is to visit all the cities such that the total distance travelled will be minimised. Obviously, a solution, and therefore a chromosome which represents that solution to the TSP, can be given as an order, that is, a permutation, of the cities [96].

2.3.3 Evaluation and selection

A mechanism to select individual in population for reproduction to create new offspring or to transfer a part of the existing population to the next generation is needed. It is possible to perform the task of selection completely in a randomised fashion. This selection mechanism will eventually cause the algorithm to reach global minimum/maximum. However, using this scheme, convergence of the population will almost be impossible, and termination will take a considerably long time [3]. The selection strategy addresses on which of the chromosomes in the current generation will be used to reproduce offspring in hopes that the next generation will have even higher fitness. A number of selection techniques exist including *elitist*, *tournament*, *Roulette Wheel* and *rank-based Roulette Wheel* [97-98]. The differing selection techniques all develop solutions based on the principle of survival of the fittest. Fitter solutions are more likely to reproduce and pass on their genetic material to the next generation in the form of their offspring [90]. However, the worse members of the population still have a small probability of being selected, and this is important to ensure that the search process is global and does not simply converge to the nearest local optimum.

Elitist selection

Elitism is a general concept to favour the top individuals and to ignore the remaining ones [99]. Individuals in the population are sorted according to their fitness values. The best *n* individuals are included in the selection process and the remaining individuals are discarded. This selection method is widely used because of its speed of convergence. However, it should be used carefully, in order not to encounter premature convergence [46].

Tournament selection

In tournament selection technique, *n* individuals are selected from the larger population, and the selected individuals compete against each other. The individual with the highest fitness wins and will be included in the mating pool. The tournament selection also gives a chance for all individuals to be selected and thus it preserves diversity, although keeping diversity may degrade the convergence speed [3]. The number of individuals competing in each tournament is referred to as tournament size, commonly set to 2 (also called binary tournament). Figure 2.9 illustrates the mechanism of tournament selection. The tournament selection has several advantages which include efficient time complexity, especially if implemented in parallel, low susceptibility to takeover by dominant individuals, and no requirement for fitness scaling or sorting [3, 25].

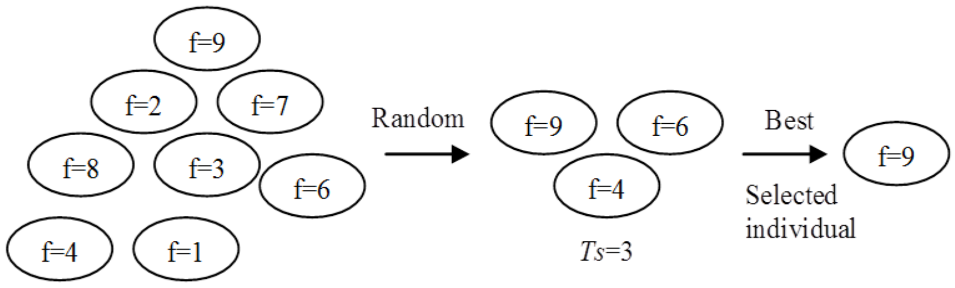


Figure 2.8: Selection method with tournament mechanism

In the above example, the tournament size, *Ts* is set to three, which mean that three chromosomes competing each other. Only the best chromosome among them is selected to reproduce. In tournament selection, larger values of tournament size lead to higher expected loss of diversity [25, 100]. The larger tournament size means that a smaller portion of the population actually contributes to genetic diversity, making the search increasingly greedy in nature. There might be two factors that lead to the loss of diversity in regular tournament selection; some individuals might not get sampled to participate in a tournament at all while other individuals might not be selected for the intermediate population because they lost a tournament.

Roulette Wheel selection

In keeping with the ideas of natural selection, it assumes that stronger individual, that is, those with higher fitness values, is more likely to mate than the weaker ones. One way to simulate this is to select parents with a probability that is directly proportional to their fitness values. This method is called the roulette wheel (or proportional Roulette Wheel) method [97]. The idea behind the roulette wheel selection technique is that each individual is given a chance to become a parent in proportion to its fitness. The chances of selecting a parent can be seen as spinning a roulette wheel with the size of the slot for each parent being proportional to its fitness. Obviously, those with the largest fitness (slot sizes) have more chance of being chosen. Consider a roulette wheel with a number of slices on it, each of which has an associated width as shown in Figure 2.10.

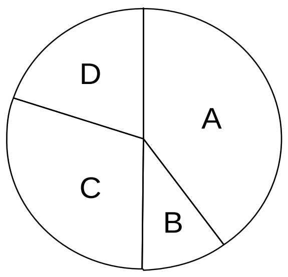


Figure 2.9: Illustration of roulette wheel selection

If a ball is put on this wheel and the wheel is rotated, the ball will finally stop on one of the slices, most probably on one of the widest ones. However, all slices have a chance, with a probability that is proportional to its width. By repeating this each time an individual needs to be chosen, the better individuals will be chosen more often than the poorer ones, thus fulfilling the requirements of survival of the fittest. The basic advantage of roulette wheel selection is that it discards none of the individuals in the population and gives a chance to all of them to be selected [101]. Therefore, diversity in the population is preserved.

Let f_1, f_2,…, f_n be fitness values of individual 1, 2,…, n. Then the selection probability, P_i for individual i is define as,

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{2-15}$$

However, Roulette Wheel selection has few major deficiencies. Outstanding individuals will introduce a bias in the beginning of the search that may cause a premature convergence and a loss of diversity. For example, if an initial population contains one or two very fit but not the best individuals and the rest of the population are not good, then these fit individuals will quickly dominate the whole population and prevent the population from exploring other potentially better individuals. Such a strong domination causes a very high loss of genetic diversity which is definitely not advantageous for the optimisation process. On the other hand, if individuals in a population have very similar fitness values, it will be very difficult for the population to move towards a better one since selection probabilities for the fit and unfit individuals are very similar.

Rank-based Roulette Wheel selection

Baker [102] proposed rank-based Roulette Wheel selection in which the probability of a chromosome being selected is based on its fitness rank relative to the entire population. Rank-based selection schemes first sort individuals in the population according to their fitness and then computes selection probabilities according to their ranks rather than fitness values. Hence rank-based selection can maintain a constant pressure in the evolutionary search where it introduces a uniform scaling across the population and is not influenced by super-individuals or the spreading of fitness values at all as in proportional selection. Rank-based selection uses a function to map the indices of individuals in the sorted list to their selection probabilities. Although this mapping function can be linear (linear ranking) or non-linear (non-linear ranking), the idea of rank-based selection remains unchanged [77]. The performance of the selection scheme depends greatly on this mapping function.

For linear rank-based selection, the biasness could be controlled through the selective pressure *SP*, such that $2.0 \geq SP \geq 1.0$ and the expected sampling rate of the best individual is *SP*, the expected sampling rate of the worst individual is 2-*SP* and the selective pressure of all other population members can be interpreted by linear interpolation of the selective pressure according to rank. Consider *n* the number of individuals in the population, *Pos* is the position of an individual in the population (least fit individual has *Pos*=1, the fittest individual *Pos*=*n*) and *SP* is the selective pressure. Instead of using the fitness value of an individual, the rank of individuals is used. The rank of an individual may be scaled linearly using the following formula [103],

$$Rank(Pos) = 2 - SP + \left(2.(SP-1).\frac{(Pos-1)}{(n-1)} \right) \tag{2-16}$$

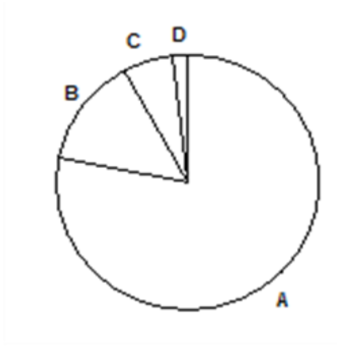
Table 2.5 contains the fitness values of the individuals for two different values of the selective pressure assuming a population of 11 individuals and a minimisation problem.

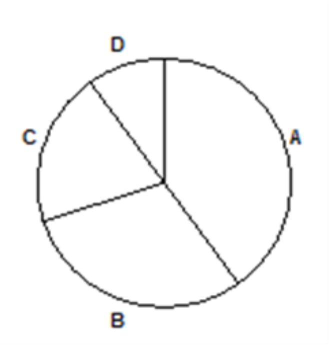
Table 2.3: Example of scaled rank with different *SP* values

Individual fitness value	Rank	Scaled rank with *SP*=2.0	Scaled rank with *SP*=1.1
1	1	2.0	1.1
3	2	1.8	1.08
4	3	1.6	1.06
7	4	1.4	1.04
8	5	1.2	1.02
9	6	1.0	1.00
10	7	0.8	0.98
15	8	0.6	0.96
20	9	0.4	0.94
30	10	0.2	0.92
95	11	0	0.9

Rank-based selection schemes can be computationally expensive because of the need to sort populations. Once selection probabilities have been assigned, sampling method using roulette wheel is required to populate the mating pool. Rank-based selection scheme helps prevent premature convergence due to “super” individuals, since the best individual always assigns the same selection probability, regardless of its fitness value [104]. However this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones. The difference between

Roulette Wheel selection with proportionate fitness and rank-based fitness is depicted in Figure 2.11.





(a) (b)

Figure 2.10: (a) proportionate fitness and (b) rank-based fitness

2.3.4 Reproduction of generations

Reproduction is the crossover of two chromosomes to produce a new offspring that has genes from both parents. In nature, although it may be much more complicated, crossover basically occurs as follows: chromosomes of both parents are randomly divided from the same gene positions into a number of segments and the corresponding segments are exchanged and copied to the chromosome of the newly created offspring. Therefore, the offspring inherit traits from the both parents [105]. In the genetic algorithm, special techniques for permutation-based chromosomes are deployed, which ensure that, when applied on two permutation-based chromosomes, the chromosomes of the resulting offspring are also valid permutations.

Crossover mechanism

Some of the most popular generic permutation-based crossover techniques in genetic algorithm are partially mapped crossover (PMX), order crossover (OX) and cycle crossover (CX) [20, 48].

Partially mapped crossover (PMX)

The partially mapped crossover was suggested by Goldberg and Lingle [106]. The main purpose of a crossover operator is to create offspring that inherits traits from both parents. PMX passes on ordering and value information from the parent tours to the offspring tours. A portion of one parent's string is mapped onto a portion of the other parent's string and the remaining information is exchanged. For example, consider chromosomes which represent TSP are given as follows: Parent 1, P_1 = (1 2 3 4 5 6 7 8) and Parent 2, P_2 = (3 7 5 1 6 8 2 4). On this two parent chromosomes, a two cut points are randomly selected. Suppose that the first cut point is selected between the third and the fourth gene, and the second one between the sixth and seventh gene.

$P_1 = 1\ 2\ 3|4\ 5\ 6|7\ 8$

$P_2 = 3\ 7\ 5|1\ 6\ 8|2\ 4$

The substrings between the cut points are called the mapping sections. In this example, the mapping section is 4-1, 5-6 and 6-8. Now the segments (mapping section) between cut points are swapped. The mapping section of the first parent is copied into the second offspring (O_2), and the mapping section of the second parent is copied into the first offspring (O_1), become;

$O_1 = * * *|1\ 6\ 8|* *$

$O_2 = * * *|4\ 5\ 6|* *$

Then we can fill further tour cities (from the original parents), for which there is no conflict;

$O_1 = *\ 2\ 3\ 1\ 6\ 8\ 7\ *$

$O_2 = 3\ 7\ *\ 4\ 5\ 6\ 2\ *$

The first $*$ in the offspring O_1 would be 1 which is the same as the first element of parent P_1. However there was a conflict (visiting city 1 twice) and hence is replaced by 4, because of the mapping 1-4. The last $*$ of offspring O_1 would be an 8, which is already present. Because of the mappings 8-6, and 6-5, it is chosen to be a 5. Hence;

$O_1 = 4\ 2\ 3\ 1\ 6\ 8\ 7\ 5$

$O_2 = 3\ 7\ 8\ 4\ 5\ 6\ 2\ 1$

The PMX crossover exploits important similarities in the value and ordering simultaneously when used with an appropriate reproductive plan [94]. The absolute positions of some strings of both parents are preserved.

Order crossover (OX)

The order crossover was proposed by Davis [107]. Besides keeping traits from parents, order crossover (OX) also takes relative order of allele values into account while performing the crossover operation [16]. The OX exploits a property of the path representation, that the order of cities (not their positions) is important. It constructs an offspring by choosing a subtour of one parent and preserving the relative order of the cities of the other parent. For example, consider the following two parent tours;

P_1 = 1 2 3 4 5 6 7 8
P_2 = 2 4 6 8 7 5 3 1

Suppose that we select a first cut point between the second and the third gene and a second one between the fifth and the sixth gene. Hence;

P_1 = 1 2|3 4 5|6 7 8
P_2 = 2 4|6 8 7|5 3 1

The offspring is created in the following way. First, the tour segments between the cut points are copied into the offspring, which gives

$O_1 = * *|3\ 4\ 5|* * *$
$O_2 = * *|6\ 8\ 7|* * *$

Next, starting from the second cut point of one parent, the rest of the city are copied in the order in which they appear in the other parent, and omitting the city that are already present. Reaching the end of the parent string, we continue from its first position. Now the sequence of the cities in the second parent from the second cut point is 5 – 3 – 1 – 2 – 4 – 6 – 8 – 7. After removal of cities 3, 4 and 5, which are already in the first offspring, we get 1 – 2 – 6 – 8 – 7. This sequence is placed in the first offspring, starting from the second cut point, which gives;

O_1 = 8 7|3 4 5|1 2 6

The sequence of the cities in the first parent from the second cut point is 6 – 7 – 8 – 1 – 2 – 3 – 4 – 5. After removal of cities 6, 8 and 7, which are already in the second offspring, we get 1 – 2 – 3 – 4 – 5. This sequence is placed in the second offspring, starting from the second cut point, which gives;

O_2 = 4 5|6 8 7|1 2 3

The OX crossover exploits a property of the path representation, that the order of cities (not their positions) is important, i.e., the two tours 6 – 7 – 8 – 1 – 2 – 3 – 4 – 5 and 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 are identical.

Linear order crossover (LOX)

Linear order crossover (LOX) is a modified version of the order crossover operator proposed by Falkenauer and Bouffix [108]. Recall that the order crossover operator treats the chromosome as a circular string, in which it wraps around from the end of the chromosome back to the beginning. This circular assumption may not play a big role in the TSP. As such, the LOX operator treats the chromosome as a linear entity. For this operator, the swap occurs in the same fashion as it occurs in the OX operator, but when sliding the parent values around to fit in the remaining open slots of the child chromosome, they are allowed to slide to the left or right. This allows the chromosome to maintain its relative ordering and at the same time preserve the beginning and ending values. In the below example, after the values are swapped, there are two open spaces in the front of the chromosome and three open spaces at the end. The algorithm then goes through Parent 1 and finds the first two values that were not part of the swap, in this example they are 5 and 4. These values are shifted left to fill the first two chromosome locations. The final three locations are filled in a similar manner.

P_1 = 3 9|5 4 6 2|7 1 8
P_2 = 7 4|3 8 9 2|1 5 6
O_1 = * *3 8 9 2 * * *
O_2 = * *5 4 6 2 * * *
O_1 = 5 4 3 8 9 2 6 7 1
O_2 = 7 3 5 4 6 2 8 9 1

Cycle crossover (CX)

The cycle crossover (CX) operator was proposed by Oliver *et al.* [107]. It attempts to create offspring in such a way that each city and its position come from one of the parents. For example, the following chromosomes are considered as the parent.

P_1 = 1 2 3 4 5 6 7 8

P_2 = 2 4 6 8 7 5 3 1

Opposite to PMX and OX, the chromosomes are not spitted to form segments to swap. The first element of the offspring is equal to be either the first element of the first parent tour or the first element of the second parent tour. Hence, the first element of the offspring has to be 1 or 2. Suppose it is chosen to be 1,

O_1 = 1 * * * * * * *

Since every city in the offspring should be taken from one of its parents (from the same position), we do not have any choice now, the next city to be considered must be city 2, as the city from the parent 2 just "below" the selected city 1. In Parent 1 this city is at position '2', thus

O_1 = 1 2 * * * * * *

Following this rule, the next cities to be included in the first offspring are 4 and 8. Note, however, that the selection of city 8 requires selection of city 1, which is already on the list, thus we have completed a cycle.

O_1 = 1 2 * 4 * * * 8

The remaining cities are filled from the other parent;

O_1 = 1 2 6 4 7 5 3 8

The same procedure also applied to Offspring 2 gives,

O_2 = 2 4 3 8 5 6 7 1

As it can be seen from the resulting offspring, each string value at each position comes from one of the parents.

Mutation mechanism

In order to avoid from getting stuck onto a local minimum and to avoid premature convergence, population diversity is required to be kept up to some extent [77]. In the genetic algorithm, this is achieved by the help of a mutation mechanism, which causes some sudden changes on the traits of individuals according to a predefined mutation probability parameter [93]. A new offspring can be achieved by different ways either by flipping, inserting, swapping or sliding the allele values at two randomly chosen gene positions.

The ***inversion*** mutation (flipping) operator [71, 91] randomly selects two cut points in the chromosome, and it reverses the subtour between these two cut points. Suppose that the first cut point is chosen between city 9 and city 5, and the second cut point between the sixth and seventh city. For example, consider the tour

Parent: 3 9|5 4 6 2|7 1 8

This result in

Offspring: 3 9 **2 6 4 5** 7 1 8

The ***insertion*** mutation [7, 94] operator selects a gene at random and then inserts it at a random position. Suppose that the insertion mutation operator selects city 5, removes it, and randomly inserts it after city 7. For example, consider again the tour

Parent: 3 9 **5** 4 6 2 7 1 8

Hence, the resulting offspring is

Offspring: 3 9 4 6 2 7 **5** 1 8

The ***displacement*** mutation [94] operator first selects a subtour at random. This subtour is removed from the tour and inserted in a random place. For example, consider the tour represented by

Parent: 3 9 **5 4 6** 2 7 1 8

Suppose that the tour (5 4 6) is selected. Hence, after the removal of the subtour we have (3 9 2 7 1 8), and suppose we randomly select city 7 to be the city after which the subtour is inserted. This result in

Offspring: 3 9 2 7 **5 4 6** 1 8

The ***exchange*** mutation [109] operator, also known as reciprocal exchange mutation or swapping [91, 94, 107] randomly selects two cities in the tour and exchanges them. For example, consider the tour represented as below and suppose that third and the eighth city are randomly selected.

Parent: 3 9 **5** 4 6 2 **7** 1 8

This result in

Offspring: 3 9 **7** 4 6 2 **5** 1 8

2.3.5 Genetic algorithm parameters

One of the main difficulties in building a practical GA is in choosing suitable values for parameters such as population size, crossover rate, and mutation rate. De Jong's guidelines are still widely followed which is to start with a relatively high crossover probability (0.6-0.7), relatively low mutation probability (typically set to 1/*l* for chromosomes of length *l*), and a moderately sized population (50-500) [110]. However, the selections of parameter values are very depend on the problem to be solved [77].

Population size

The population size is the number of candidate solutions in any one generation. The decision as to what is an appropriate population size has undergone significant amounts of research. Some researchers believe that genetic algorithm should be constructed with sufficiently large populations so as to enhance the search diversity [20, 111]. Others believe that small populations combined with higher numbers of generations allows for a more controlled search for optimal solutions [3, 112]. Goldberg [113] has evaluated the role of population size extensively and noted that with the increased size of the population the chances of initializing to an optimal solution are greatly improved. However this has the desirable effect of increasing the length of time for each generation to be computed. He also derived population sizes that are relatively small for short chromosomes and increase exponentially with chromosome length. Reeves [69] investigated the minimum practical population size in a GA and suggested that small populations suffice when chromosomes are binary, while coding over alphabets of higher cardinality requires larger populations. The population size has to increase exponentially with the complexity of the problem (i.e. the length of the chromosome) in order to generate best solutions. A large population is quite useful, but it demands excessive costs in terms of both memory and time.

Crossover and mutation probability

These parameters control the frequency of reproduction. Crossover probability is how often will be crossover occurred in each generation. If there is no crossover, offspring is an exact copy of parents, but this does not mean that the new generation is the same. If there is a crossover, offspring is made from parts of parents' chromosome. Mutation probability is how often will be part of chromosome mutated. If there is no mutation, the offspring is taken after crossover or copy without any changes. If mutation is performed, part of a chromosome is changed [114].

Termination criteria

Termination is the criterion by which the genetic decides whether to continue searching or to stop the search. Generally, defining stopping criteria is a hard task, like defining population size, because we have no idea of the true performance of the GA on a specific problem. If we let the algorithm continuously run, it will waste time and it will only revisit all the previous solutions. The stopping criterion of the GA should be related to the specific problem. There are a number of termination method can be used to stop the evolution process. The GA run can be terminated using different criteria such as when a best-so-far solution does not improve for a specific number of generations, a pre-determined solution quality is obtained, the population reaches a lower limit of diversity that indicate convergence, or a maximum length of CPU time is reached. The most frequently used stopping criterion is a specified number of generations [88] in which the algorithm stop once a pre-specified number of generations is reached.

- Number of generations – a termination method that stops the evolution when the user-specified maximum number of generations. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

- Evolution time – a termination method that stops the evolution when the elapsed evolution time exceeds the user-specified maximum evolution time. By default, the evolution is not stopped until the evolution of the current generation has completed, but this behaviour can be changed so that the evolution can be stopped within a generation.
- Fitness threshold – a termination that stops the evolution when the best fitness in the current population becomes less than the user-specified fitness threshold and the objective is set to minimise the fitness.
- Population convergence – a termination method that stops the evolution when the population is deemed as converged (say 95%). The population is deemed as converged when the average fitness across the current population is less than a user-specified percentage away from the best fitness of the current population.

Generational Gap

It is possible that an individual with the highest fitness value in a generation may not survive selection process. A parameter called the generation gap was defined to control the fraction of the population to be replaced in each generation. This means that a certain percentage of the population with the best fitness values is kept and preserve for the crossover process. The GA is said to have a generation gap of 1 if the reproduction replaces the entire population with a new population [115].

If the GA has been correctly implemented, the population will evolve over successive generations so that the fitness of the best and the average individual in each generation increases/decreases towards the global optimum. The term *convergence* refers to the progression towards increasing uniformity. A gene is said to have converged when 95% of the population share the same value [116].

2.4 Research hypothesis

The previous section reviewed the concept of the TSP and TSPPC and the genetic algorithm technique that can be used to solve them. However, the use of the genetic algorithm approach to solve TSP, with a traditional representation scheme might generate invalid candidate solutions when precedence constraints are involved.

To handle precedence constraints, Moon *et al.* [6] proposed new encoding scheme based on topological sort, which is defined as an ordering of vertices in a directed graph. The procedure to select the task to be placed in sequence is performed by comparing the available tasks based on higher priority. The priority of tasks is generated randomly for initial population and is used to generate the priority for the other population. The proposed encoding scheme will not generate infeasible chromosomes because it only generates priority of tasks as chromosomes.

Some of the challenges that may arise in solving TSPPC efficiently are related to parameter tuning and maintaining population diversity. Running GA requires setting a number of parameters. However, finding settings that work well on a specific problem is not a trivial task. Poor settings lead to inferior results while good settings require time-consuming trials to find. The degree of success of GA on a given problem also depends largely on their ability to balance between exploration and exploitation during the search process. Population diversity plays an important role in achieving this balance, since high diversity directs the search towards the exploration of unvisited regions of the search space whereas low diversity focuses the search on specific regions to exploit possibly good solutions. This population diversity is much related to the mechanism of crossover and mutation used in the procedure. Moon *et al.* proposed new crossover technique called *moon* crossover, which they claimed more effective than conventional order crossover. Although the Moon's algorithm ensures that the optimal solution exists, it is assumed that there will be computationally expensive to achieve because the Moon's algorithm requires comparing the higher priority before deciding to select the next task. Therefore, it takes longer time and more generations to come out with an optimal solution.

In the next chapter, a new GA procedure which integrates chromosome repairing strategy such as done by *Moon et al.* will be developed. This algorithm must be able to handle the precedence constraints and to generate only feasible solution (i.e. legal tour) during the evolutionary process. The algorithm that will be developed in this thesis is also expected to be more efficient than the algorithm developed by Moon *et al.* with improvement in the number of generations and iteration time to come out with an optimal solution. In addition, the algorithm should be able to solve larger TSPPC instances with optimal or near optimal solution.

Chapter 3

Research Methodology

This chapter describes the genetic algorithm procedure that will be used to find the optimal solution for TSP and TSPPC. The development of the procedure is carried out in two stages. The first stage is to develop the genetic algorithm procedure for TSP. Then this procedure is used as a framework to solve TSPPC with some modifications in the representation stage. The second stage is to benchmark and to model the previous method that solves similar problem to solve TSPPC. Finally, the development process of the new proposed genetic algorithm to solve TSPPC is described in detail.

3.1 Genetic algorithm procedure for TSP

In this stage three different genetic algorithms to solve TSP will be developed. The first proposed algorithm employs a combination of proportional Roulette Wheel selection mechanism for parent selection and linear order crossover with an inversion mutation for producing the offspring at every generation. This algorithm is called PROX. In contrast with PROX, the second proposed algorithm will use the different selection mechanism which is rank-based Roulette Wheel while using similar crossover and mutation operation as in PROX, thus it is called RBOX algorithm. As for the third proposed algorithm, it employs tournament selection mechanism to select individuals for the mating process in which it is called TSOX algorithm. In TSOX, again LOX and IM is used for crossover and mutation operation. The difference procedure between them is only in the process of parent selection for reproduction. The general GA procedure to solve TSP can be viewed as a flow chart given in Figure 3.1. There are three main steps involved in the GA procedure for TSP which is Initialization & Representation, Evaluation & Selection and Generation of offspring.

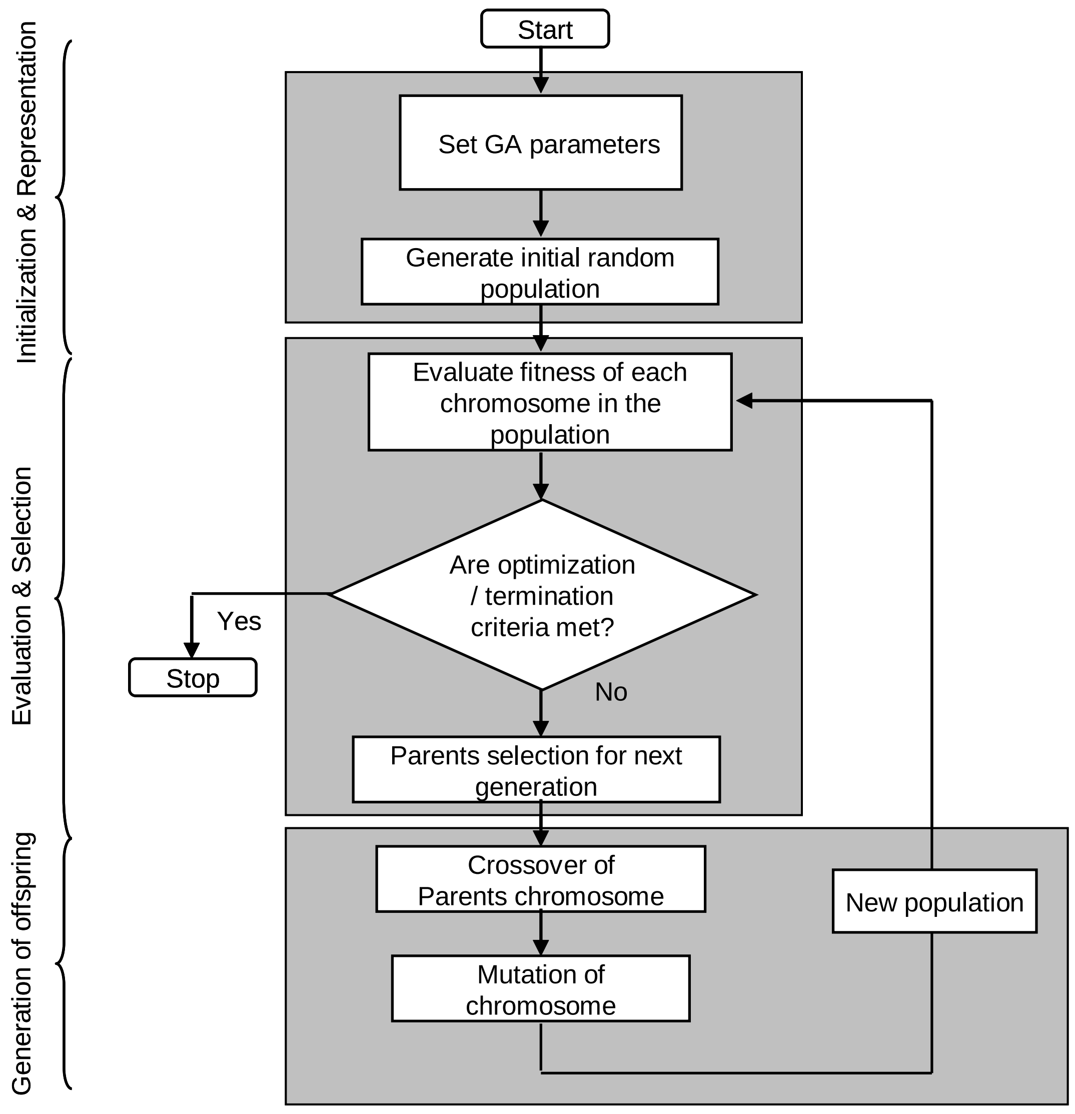

Start
Set GA parameters
Generate initial random population
Evaluate fitness of each chromosome in the population
Are optimization / termination criteria met?
Yes
Stop
No
Parents selection for next generation
Crossover of Parents chromosome
Mutation of chromosome
New population
Initialization & Representation
Evaluation & Selection
Generation of offspring

Figure 3.1: Flow chart of GA for TSP

3.1.1 PROX algorithm

In PROX algorithm, the letter ‘PR’ is used to denote proportional Roulette Wheel selection while letter ‘OX’ is to denote linear order crossover. The PROX algorithm starts with supplying important information to the GA program such as the location of the cities, distance, time or cost incurred between the cities, and the GA parameters such as maximum number of generations, population size, the probability of crossover and the probability of mutation. The algorithm will generate an initial random population of chromosomes in which *path* also called *permutation* is used as a chromosome representation. Then the algorithm continues with evaluating the fitness of each chromosome according to the fitness function, and selecting the best chromosomes as parents for reproduction according to selection mechanism.

In every generation, the offspring (i.e. new chromosomes) are produced by a combination of linear order crossover and inversion mutation mechanism with specified probability of crossover (P_c) and probability of mutation (P_m). A new set of chromosomes where the size is equal to the initial population size (*pop_size*) is evolving. The procedure is repeated until the maximum number of generations (*ngener*) is reached. The procedure of the PROX algorithms is described in Figure 3.2.

```
Procedure: PROX algorithm
Begin
   Step 1: Initialization & Representation
   Step 1.1: Set GA parameters and the problem information
   For $i \leftarrow 1$ to pop_size do
       Step 1.2: Generate random initial population ($x_1,\ldots x_N$) with $N$ strings
   End For
   While number of generation < ngener do
       Step 2: Evaluation & Selection
       Step 2.1: Calculate the fitness value of each chromosome in $i$
       Step 2.2: Select chromosomes in $i$ as parents using Roulette Wheel selection
       scheme and preserve best 10% of pop_size
       Step 3: Generation of Offspring
       Step 3.1: With probability $P_c$, select chromosomes in step 2.2 and apply linear
       order crossover
       Step 3.2: With probability $P_m$, select chromosomes in step 3.1 and apply
       inversion mutation
   End While
   Return new population of chromosomes
End procedure
```

Figure 3.2: Procedure for PROX algorithm

Step 1: Initialization & representation

The purpose of initialization step is to establish an initial chromosome population. The initial chromosome is very useful to create a new chromosome which is known as offspring in the next generation.

Step 1.1: Set GA parameters and the problem information

The problem information such as the location of the cities or the distance matrix for a particular TSP is the most important input to the GA program. The distance or cost matrix also can be automatically generated by supplying the location of the cities in the GA program. The GA parameters such as population size, the termination criteria (i.e. maximum number of generations), the probability of crossover and probability of mutation must be set earlier in the program. The decision to what values of each of the parameters to be used in the experiment is obtained by trial run or by implementing the design of experiment (DOE).

Step 1.2: Generate random initial population

In order to create an initial chromosome population, random permutation method is used. The random permutation method creates initial chromosome by generating numbers between 1 and the total of string, *N* in random sequence. Therefore, if the chromosome population size, *pop_size* is 10, it means there are 10 sets of chromosomes which consists of a number from 1 to *N* in random sequence. Here, the number of strings, *N* represents the number of cities.

Step 2: Evaluation & selection

The evaluation of the chromosome is performed by measuring the fitness of each chromosome. Selection is then performed to choose the chromosome to be re-generated for the next generation. A generation gap of 0.9 is applied in which new population will compose 90% of new chromosomes and 10% of the best old chromosome.

Step 2.1: Evaluation

Every chromosome is evaluated by calculating the fitness value using a fitness function given in (2-1).

If the distance between the cities is known for *n* cities location, we can calculate the total distance (fitness value) of each tour (chromosome) in the population. The procedure of fitness evaluation of each chromosome in the population is given in Figure 3.3.

```
Procedure: Fitness evaluation
    For p ← 1 to pop_size do
        Calculate d, the distance between the last city and the starting city (closed loop)
        For k ← 2 to n do (n is the number of genes i.e. number of cities)
            Calculate the distance between city k-1 to k and add d to get the total
            distance
        End For
        Return fitness value (i.e. total distance) for each chromosome in p
    End For
End Procedure
```

Figure 3.3: Procedure for fitness evaluation

Step 2.2: Selection

The selection process is performed by applying the proportional Roulette Wheel method. The probability of an individual to be selected is simply proportionate to its fitness value. The procedure for proportional Roulette Wheel is given in Figure 3.4.

```
Procedure: proportional Roulette Wheel selection
    While population size < pop_size do
        Generate pop_size random number (R)
        Calculate cumulative fitness, total fitness and sum of proportional fitness (Sum)
        Spin the wheel pop_size times
        If Sum < R then
            Select the first chromosome, otherwise, select jth chromosome
        End If
    End While
    Return chromosomes with fitness value proportional to the size of selected wheel
    section
End Procedure
```

Figure 3.4: Procedure for proportional roulette wheel selection

Step 3: Generation of offspring

The new chromosomes (i.e. offspring) are produced through two different mechanisms which is crossover and mutation. The two selected parent chromosomes are first mating by crossover techniques and produced two new offspring. These two new offspring is then mutated in order to further improve their genetic material.

Step 3.1: Crossover

The crossover operation required two parent chromosomes of the population to create two new chromosomes. In PROX, linear order crossover is used to create two new offspring. The procedure for linear order crossover is presented in Figure 3.5.

```
Procedure: Linear order crossover
  Begin
      $N$ = total number of strings
      Select two chromosomes; $Pa = a_1,a_2,\ldots a_N$; $Pb = b_1,b_2,\ldots b_N$
      Get two crossover points randomly between 1 to $N$; swap_sect
      Offspring 1; osp1 = swap_sect in Pb
      Offspring 2; osp2 = swap_sect in Pa
      While length of osp1 $\neq N$
              osp1 = <osp1, the remaining unselected string from Pa>
              osp2 = <osp2, the remaining unselected string from Pb>
      End While
End Procedure
```

Figure 3.5: Procedure for linear order crossover

As an example, assume that the parent chromosomes are *Pa* = [3 9 5 4 6 2 7 1 8] and *Pb* = [7 4 3 8 9 2 1 5 6]. The crossover point is then selected randomly between second and third string and between sixth and seventh string. The selected sections are then swapped. Thus, the *swap_sect* are *osp1*= [3 8 9 2] and *osp2* = [5 4 6 2]. Then the remaining unselected string form *Pa* is filled in Offspring 1, which finally produce *osp1* = [5 4 3 8 9 2 6 7 1]. Similarly the procedure is applied to Offspring 2 by selecting the remaining string from *Pb*. Therefore, *osp2* = [7 3 5 4 6 2 8 9 1].

Step 3.2: Mutation

Mutation operation is performed in a single chromosome to create a single new chromosome. Here, inversion mutation is applied to an individual after going through a crossover process. The inversion mutation procedure starts by selecting two cut points randomly in the chromosome. Then, the section of these genes is reversed (flip left to right) to create a new chromosome. For example the chromosome [7 3 **5 4 6 2 8** 9 1] is mutated becoming [7 3 8 2 6 4 5 9 1].

3.1.2 RBOX algorithm

In RBOX algorithm, the letter ‘RB’ is used to denote rank-based Roulette Wheel selection while letter ‘OX’ is used for linear order crossover. RBOX algorithm uses a similar procedure as in PROX. In RBOX, rank-based selection is used instead of proportional selection. The procedure of rank-based Roulette Wheel selection is presented in Figure 3.6.

```
Procedure: rank-based Roulette Wheel selection
   While population size < pop_size do
       Sort population according to rank (from highest to lowest value)
       Assign fitnesses to the individuals according to linear rank function
       Generate pop_size random number (R)
       Calculate cumulative fitness, total fitness and sum of proportional fitness (Sum)
       Spin the wheel pop_size times
       If Sum < R then
         Select the first chromosome, otherwise, select jth chromosome
       End If
   End While
   Return chromosomes with fitness value proportional to the size of selected
   wheel section
End Procedure
```

Figure 3.6: Procedure for rank-based Roulette Wheel selection

3.1.3 TSOX algorithm

In TSOX the tournament selection is applied in the selection stage in which letter ‘TS’ is used to denote tournament selection and letter ‘OX’ is for linear order crossover. The procedure of tournament selection is presented in Figure 3.7.

```
Procedure: Tournament selection
   While population size < pop_size do
       Set the tournament size, Ts
       Pick Ts random individuals from the population
       From those Ts individuals, pick one with the best fitness
       If the same individual chosen as both parents, discard the second one
   End While
   Return chromosome with the best fitness among Ts chromosomes
End Procedure
```

Figure 3.7: Procedure for tournament selection

Table 3.1 shows a comparison between PROX, RBOX and TSOX algorithm. As mentioned before, the difference between them is only in the selection strategy used for reproduction. However, each procedure requires different parameters setting for different size of problems.

Table 3.1: Comparison between PROX, RBOX and TSOX algorithm

Operators	**PROX**	**RBOX**	**TSOX**
Representation	Path/permutation	Path/permutation	Path/permutation
Selection method	Proportional Roulette Wheel	Rank-based Roulette Wheel with selection pressure, *SP*=1.1	Tournament with tournament size, *Ts*=2
Crossover method	Linear order	Linear order	Linear order
Mutation Method	Inversion	Inversion	Inversion
Generation gap (replacement strategy)	0.9	0.9	0.9

3.2 Design of experiment for GA parameter setting

The value of each GA parameter can be obtained through the design of experiments. A screening design is normally performed at the beginning of an investigation when the experimenter wants to characterise a process. In this case, characterising means to determine the main factors and investigate the changes of the response by varying each factor. Due to its characteristic of identifying significant main effects, rather than interaction effects, screening designs are often used to analyze designs with a large number of input factors. However, from the literature survey, there are four main factors (input variables) which usually affecting the quality of the solution (output variable). They are population size, the probability of crossover, probability of mutation and the maximum number of generations. To simplify and reduce the experimentation process and time, the usual practice is to first perform a trial run experiment based on the combination of these four factors. For more precise results, full factorial design of experiments with 2 or 3 levels can then be used. However, in this thesis, statistical analysis such as analysis of variance is not being discussed. This is because the main concern of the study is not optimising the GA parameters; the DOE table is only used to assist the experiments.

3.3 GA procedure for TSP with precedence constraints

In the preceding section, three different GA procedures have been developed for solving TSP. The three GA procedures were PROX, RBOX and TSOX. The proposed procedures should be able to solve any size of TSP instance. The proposed procedures could also be used to solve TSP with precedence constraint (TSPPC). However, modification in representation stage should be made in order to be practical for TSPPC applications. Repair operation such as done by Moon *et al.* [6] to encounter precedence constraint in process sequencing and Pongcharoen *et al.* [117] for scheduling production problem is needed to ensure all chromosomes in the population are valid tours which do not violated the precedence constraint added to the tours. In the next stage, the topological sort techniques which can be used to repair the infeasible chromosomes generated during the evolution process will be first reviewed. The topological sort technique has been used by Moon *et al.* in their work. Then the overall GA procedure to solve TSPPC developed by Moon *et al.* is critically reviewed. Figure 3.8 illustrates the GA process flow with the repair process denoted by “Route repair” which will be added in the GA procedure developed in the first stage for solving TSPPC.

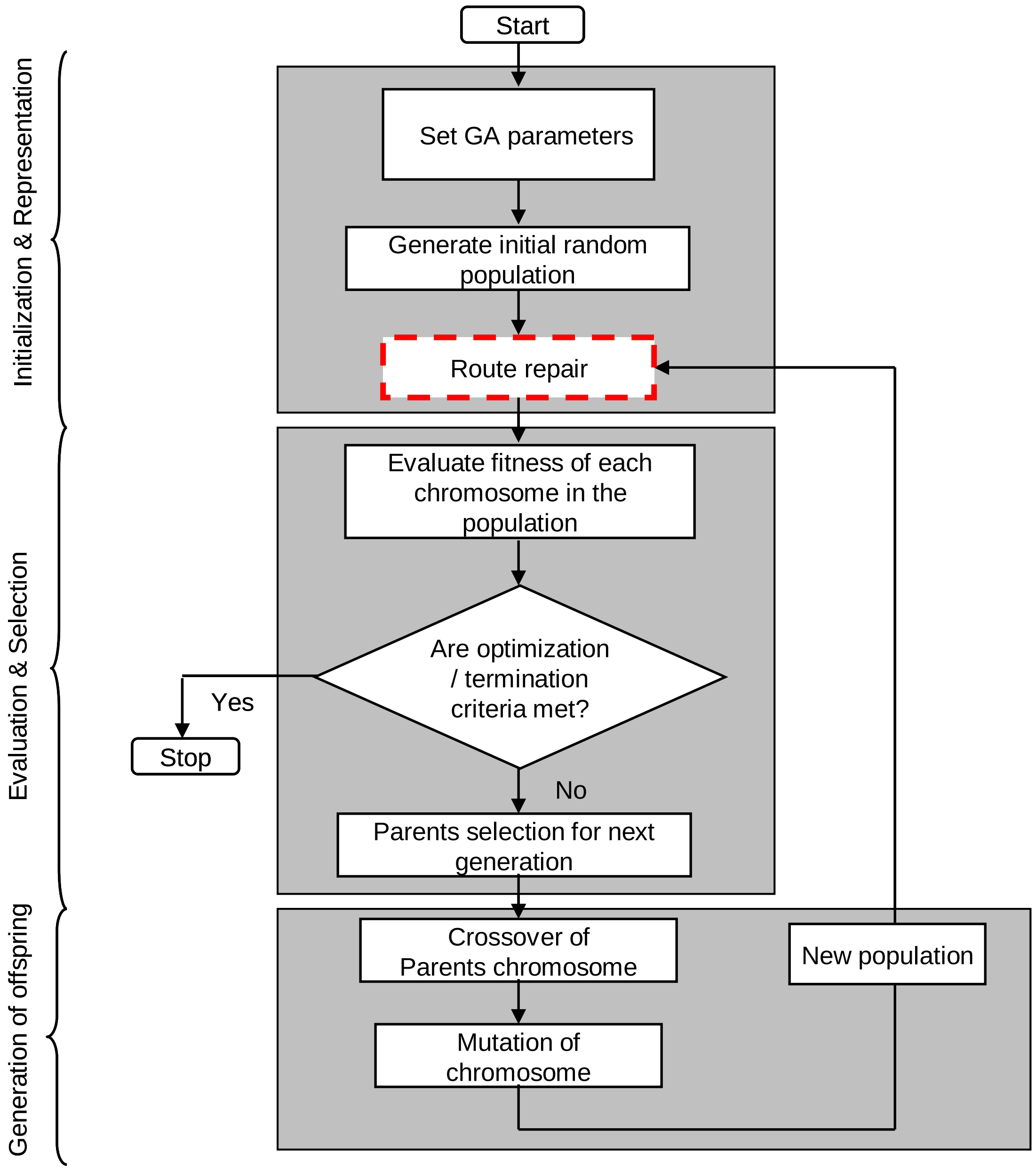

Start
Set GA parameters
Generate initial random population
Route repair
Evaluate fitness of each chromosome in the population
Are optimization / termination criteria met?
Yes
Stop
No
Parents selection for next generation
Crossover of Parents chromosome
Mutation of chromosome
New population
Initialization & Representation
Evaluation & Selection
Generation of offspring

Figure 3.8: Flowchart of GA for TSPPC

3.3.1 Route repair using topological sort technique

Since the chromosomes in the initial population are all randomly generated, the possibility of which generated chromosomes are infeasible due to the breaking of one or more precedence constraints is greater. The procedure of repairing chromosome in the initial population as well as after crossover and mutation operation is necessary before going through the evaluation process. The approach to overcome generating invalid sequence is based on a topological sort (TS), which allows the GA to generate only valid solutions in each generation. Every precedence graph has at least one topological sort. The topological sort is a node ordering in a directed graph such that if there is a path from a node v_i and a node v_j, then v_j appears after v_i in the ordering [6]. In a directed graph, the nodes represent tasks and the edges represent the precedence relations between tasks. More than a single sequence of tasks can be derived from a directed graph using the topological sort technique.

The procedure to sort nodes consists of selecting and storing any node that has no incoming edges. Then the nodes and all the edges leading out from the node are removed from the graph. Thus the path (v_i, v_j) in the directed graph shows that node v_i must be executed or scheduled before node v_j. If there is more than one node that has no incoming edges, a few ways can be performed in order to select the node such as by random selection, comparison of the lower number of nodes and comparison of higher number of nodes. Then, the edges that start from the selected nodes are removed. This procedure is repeated until all nodes are selected. An example of directed graph to represent the precedence constraints are illustrated in Figure 3.9 and the travelling time between nodes is represented by Table 3.2.

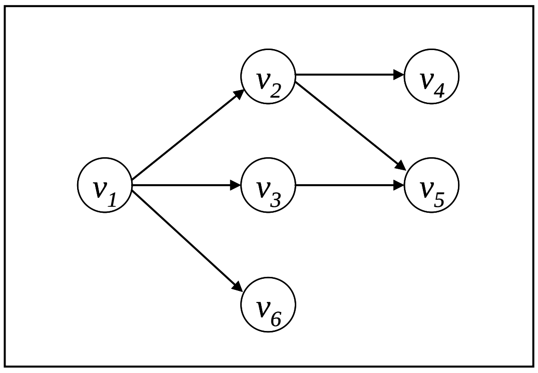


Figure 3.9: Example of directed graph

Table 3.2: Traveling time between nodes

Nodes	v_1	v_2	v_3	v_4	v_5	v_6
v_1	-	7	5	6	10	9
v_2	7	-	14	6	10	8
v_3	5	14	-	16	16	10
v_4	6	6	16	-	10	6
v_5	10	10	16	10	-	12
v_6	9	8	10	6	12	-

As an example, let the directed graph in Figure 3.9 represent an assembly process of a product. The vertex v_1 to v_6 represent the process number. The travelling time between one process to another process is given in Table 3.2. For example, the travelling time from v_1 to v_2 is 7 seconds. The objective of this problem is to minimise the travelling time for this assembly process. An optimum assembly sequence must be obtained from the problem. The sequence is considered complete if the sequence visits all the vertices (processes) and is considered feasible if the sequence does not violate the precedence constraints.

A topological sort technique can be used to obtain all the feasible paths in a directed graph. Nevertheless, more than a single sequence of vertices can be derived from a directed graph using the topological sort technique. In order to find a feasible path from directed graph, first, create a list of the start node. The start node is the nodes that have no incoming edges. After that, the start node is inserted into a *queue*. Then during the selection process, a selected node is removed from *the queue* and stored in *sequence*. Then, the edges that start from selected nodes are removed. This procedure is repeated until all nodes are selected. A procedure to generate a feasible path from a directed graph using a topological sort is described in Figure 3.10, and the example of step by step repairing technique is illustrated in Figure 3.11 to Figure 3.16.

```
Procedure: Feasible path generation
  input: directed graph
  While (any vertex remains) do
        if every vertex has a predecessor, then the network is infeasible: stop
        else pick a vertex v randomly with no predecessors
    queue ← v;
    delete v and all edges leading out of v from the directed graph;
  End While
End Procedure
```

Figure 3.10: Feasible path generation algorithm

For the above example, there are a few feasible sequences that can be generated from the given graph. For the first step, the start node for this problem is v_1, because it is the only node without incoming edge. Therefore, this node is selected to be stored in *sequence* as shown in Figure 3.11.

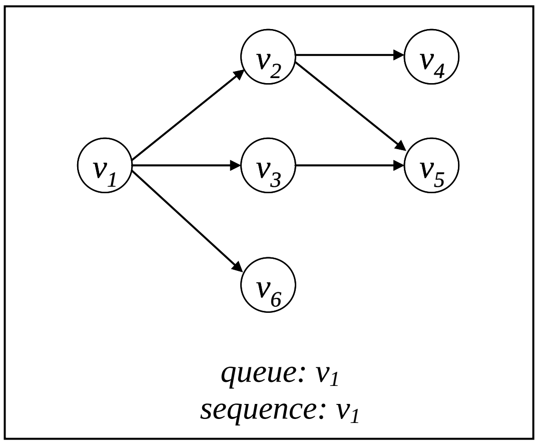


Figure 3.11: Topological sort – first step

Then, the incoming edges on v_2, v_3 and v_6 from v_1 are removed. As a result, v_2, v_3 and v_6 have no incoming edges. So, these nodes are selected to be stored in *a queue*. Selection of a node to be stored in *sequence* must be performed by one of the methods that were described earlier. In this example, selection of nodes is performed randomly from *queue* because it can derive many feasible sequences from the problem above.

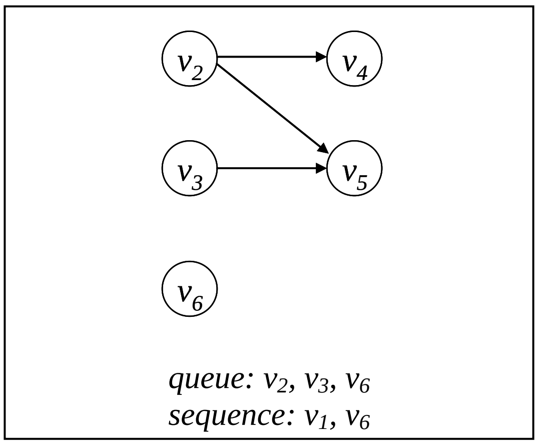


Figure 3.12: Topological sort – second step

Referring to Figure 3.12, v_6 is randomly selected to be stored in *sequence*. Therefore node v_6 is deleted as shown in Figure 3.13. The other two nodes without incoming edges which are v_2 and v_3 are stored in *the queue*. Next, v_3 is randomly selected as the next *sequence*.

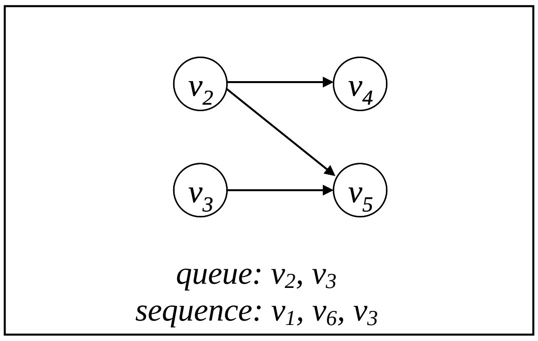


Figure 3.13: Topological sort – third step

When v_3 was selected, this node and outgoing edge are removed from directed graph. Therefore the remaining node without incoming edge is only v_2 and this node is selected as the next node in sequence.

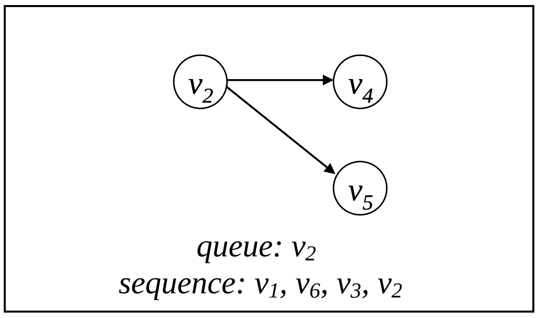


Figure 3.14: Topological sort – fourth step

After v_2 and the outgoing edges from v_2 were removed, the available nodes for selection are v_4 and v_5.

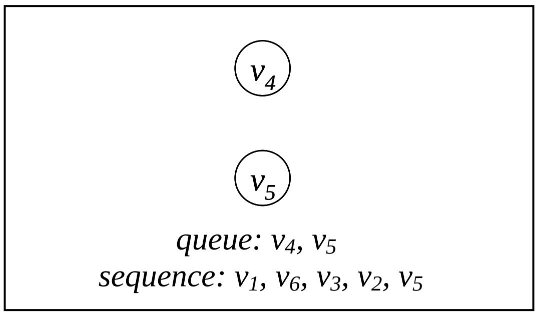


Figure 3.15: Topological sort – fifth step

By using the random selection procedure, v_5 is selected and stored in the next position of *sequence* as presented in Figure 3.15. Finally, v_4 is selected to complete the selection as can be seen in Figure 3.16.

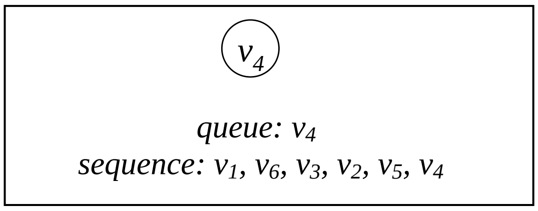


Figure 3.16: Topological sort – sixth step

From this procedure the final feasible sequence is (v_1, v_6, v_3, v_2, v_5, v_4). Because the selection is random, another feasible sequence can also be derived from the directed graph such as (v_1, v_3, v_6, v_2, v_4, v_5). However, notice that the total assembly times for both sequences are different. In the first sequence, the total assembly time is 9 + 10 + 14 + 10 + 10 = 53 seconds whereas for the second sequence, the total assembly time is 5 + 10 + 8 + 6 + 10 = 39 seconds. In this case, the second sequence is better than the first sequence because it minimizes assembly time.

3.3.2 Review of Moon's procedure to solve TSPPC

Due to the existence of precedence constraints among tasks, an arbitrary permutation may yield an infeasible order. In Moon's work, an encoding scheme for genetic algorithm based on topological sort was proposed. Moon has also introduced a new efficient crossover operator in producing offspring. The procedure of the Moon's algorithm can be divided into three main steps which are Initialization & Representation, Evaluation & Selection, and Generation of offspring. The procedure and the flowchart of Moon's algorithm implementation can be generated as in Figure 3.17 and Figure 3.18 respectively.

```
Procedure: Moon's algorithm
Begin

   Step 1: Initialization & Representation
   Step 1.1: Set GA parameters and the problem information
   For i ←1 to pop_size do
       Step 1.2: Generate random permutation of priority ($x_1$,…$x_N$) with N strings
   End For

  While number of generation < ngener do
   While population < pop_size do
    While length of chromosome < N do
       Step 1.3 Route repair
       Step 1.3.1: Check and store available task without incoming edge in available
                   set
       Step 1.3.2: Compare the priority of available task
       Step 1.3.3: Select and store task with highest priority in updated sequence
       Step 1.3.4: Remove edge from selected task
    End While
   End While

       Step 2: Evaluation & Selection
       Step 2.1: Calculate fitness value of each chromosome in i
       Step 2.2: Select chromosomes in i as parents using Roulette Wheel selection
                 scheme
       Step 3: Generation of Offspring
       Step 3.1: With probability $P_c$, select chromosomes in step 2.2 and apply moon
                 crossover
       Step 3.2: With probability $P_m$, select chromosomes in step 3.1 and apply
                 Exchange mutation

  End While
End procedure
```

Figure 3.17: Procedure of Moon's algorithm

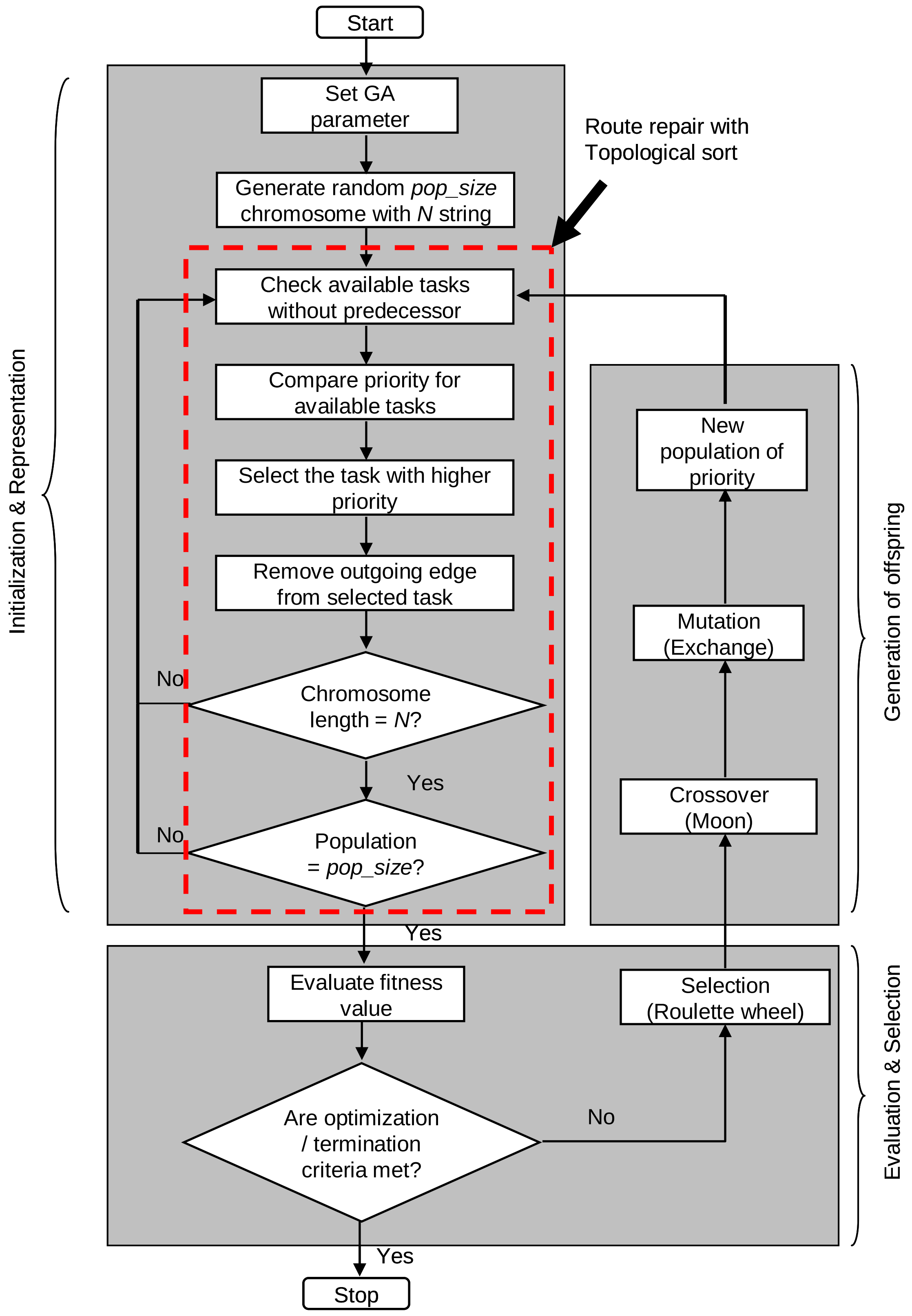

Start
Set GA parameter
Generate random pop_size chromosome with N string
Route repair with Topological sort
Check available tasks without predecessor
Compare priority for available tasks
Select the task with higher priority
Remove outgoing edge from selected task
Chromosome length = N?
No
Yes
Population = pop_size?
No
Yes
Evaluate fitness value
Are optimization / termination criteria met?
No
Yes
Stop
Selection (Roulette wheel)
Crossover (Moon)
Mutation (Exchange)
New population of priority
Initialization & Representation
Generation of offspring
Evaluation & Selection

Figure 3.18: Flowchart of Moon's algorithm

Step 1: Initialization & representation

Step 1.1: Set GA parameters

The GA parameters such as population size, maximum number of generations, the probability of crossover and probability of mutation are set earlier in the program.

Step 1.2: Generate random initial population

The chromosomes as many as the population size are generated randomly. Each chromosome is represented as a string of integers. Each digit of the string means the priority of the gene and ranges between one to the number of genes. Therefore, for every chromosome, there are *N* strings which represent the priority for each node. The maximum number of generations for computational experiment is also set at this stage. The maximum number of generations determines the termination of the iteration. When the number of generations is equal to the maximum number of generations, the program will be terminated immediately.

Step 1.3: Route repair

Route repair in the Moon's algorithm is based on topological sort. The topological sort procedure in the Moon's algorithm is detail explained in Step 1.3.1 until Step 1.3.4.

Step 1.3.1: Checking available tasks

In order to derive a unique sequence from precedence graph, a priority assignment technique to assign a differentiated priority to each vertex is used by Moon. Moon uses a priority rule to select a node (task) when two or more of them have no preceding tasks. This is accomplished by applying an order-based genetic algorithm in which the chromosome represents the priority list. Thus, the chromosome has *N* genes, each one holding a different value between 1 and *N*, representing one of the *N* tasks or nodes from the graph in Figure 3.9. Table 3.3 shows an example of a chromosome that can be used to optimise the priorities for the tasks shown in the graph of Figure 3.9.

Table 3.3: Priority representation

Vertex	v_1	v_2	v_3	v_4	v_5	v_6
Priority	5	2	6	4	1	3

In the above table, the second row represents the selection priority in case the available task has multiple vertices with no incoming edges. The priority for each vertex is generated at random within [1, N] exclusively, where N is the number of vertices or processes. As an example to generate a feasible path from the representation scheme, the precedence graph in Figure 3.9 is considered. In the graph, the first node to be selected is v_1, since this is the only node with no predecessor. Then, v_1 is stored in the queue and v_1 and the edges v_1-v_2, v_1-v_3 and v_1-v_6 are removed. As a result of removing the edges, the processes that have no predecessor are now v_2, v_3 and v_6. These vertices (v_2, v_3 and v_6) are then moved into the queue as *available set* as shown in Table 3.4.

Step 1.3.2: Comparing priority of available task

Now, there are three vertices in the available set which are v_2, v_3 and v_6. The comparisons of priority for these vertices are performed by referring to Table 3.3. Based on this table, the priority factor for v_2, v_3 and v_6 are 2, 6 and 3 respectively.

Step 1.3.3: Selecting task with higher priority

Vertex v_3 is selected as the next position in sequence since its priority is higher than v_2 and v_6. Thus, the *updated sequence* is now (v_1, v_3) as appeared in Table 3.4.

Step 1.3.4: Removing edge

As a result of selecting v_3, the outgoing edge from v_3, which is v_3-v_5 is removed from directed graph. Now the available sets left are v_2 and v_6, and v_6 is selected as its priority is higher than v_2. Therefore the new *updated sequence* is now consists of (v_1, v_3, v_6).

Repeating Step 1.3.1 to Step 1.3.4, a final feasible path is obtained. In this case, the final feasible path is (v_1, v_3, v_6, v_2, v_4, v_5) which is uniquely obtained from the priority string of [5 2 6 4 1 3]. Table 3.4 summarizes the chronological sequence of selection to create a feasible path for the given priority. Table 3.5 and 3.6 are the example of the 12 random initial chromosomes and chromosomes after repairing process, respectively.

Table 3.4: Selection of tasks using ‘priority’ technique

Path : [v_1, v_2, v_3, v_4, v_5, v_6] Priority : [5 2 6 4 1 3]	
available set	*updated sequence*
v_1	[v_1]
v_2, v_3, v_6	[v_1, v_3]
v_2, v_6	[v_1, v_3, v_6]
v_2	[v_1, v_3, v_6, v_2]
v_4, v_5	[v_1, v_3, v_6, v_2, v_4]
v_5	[v_1, v_3, v_6, v_2, v_4, v_5]

Table 3.5: Chromosomes (i.e. priority) in the initial population

5	2	6	4	1	3
5	2	6	1	4	3
4	3	2	6	1	5
4	2	3	6	1	5
1	4	6	5	3	2
5	4	1	3	2	6
3	5	2	4	1	6
2	5	1	6	3	4
1	6	4	5	2	3
2	6	5	1	3	4
5	1	6	3	4	2
4	1	3	6	5	2

Table 3.6: Chromosomes after repairing process

1	3	6	2	4	5
1	3	6	2	5	4
1	6	2	4	3	5
1	6	3	2	4	5
1	3	2	4	5	6
1	6	2	4	3	5
1	6	2	4	3	5
1	2	4	6	3	5
1	2	4	3	6	5
1	2	3	6	5	4
1	3	6	2	5	4
1	3	6	2	4	5

Step 2: Evaluation & selection

Step 2.1: Calculate fitness value

The fitness value for each chromosome in the population is calculated based on the objective function. In this case, the objective function is similar with TSP, with the objective to minimize the total traveling distance; however, the calculation will exclude the distance between the last city to the start city (i.e. salesman does not return to the starting city).

Step 2.2: Parent selection

Moon also uses a Roulette Wheel selection technique in order to select parents for crossover.

Step 3: Generation of offspring

Step 3.1: Crossover

A new crossover operator was proposed by Moon. The proposed crossover is called the ***moon*** crossover because it is very similar to the change of the moon such as waxing moon – half moon – gibbous – full moon. For example, the sub-tour can be compared to the waxing moon or half moon. The procedure of the moon crossover operator is described in Figure 3.19.

```
Procedure: moon crossover
  Begin
        Initialization: $osp \leftarrow$ null, $k \leftarrow 0$
        Select two random chromosomes $p_a$ and $p_b$, $p_a = g_1\ g_2\ g_3 \ldots g_J$ and $p_b = q_1\ q_2$
        $q_3 \ldots q_J$
        Select two genes from $p_a$ at random
        $osp \leftarrow$ the substring between $g_i$ and $g_j$ selected from $p_a$
        if the length of $osp = J$ then end
        else $sub\_p_b \leftarrow$ the remaining substring results from the deleting genes which are
        already selected from $p_a$
        end if
        while (length of $osp \neq J$) do
              if $i = 1$ then $i = J+1$;
                    $i \leftarrow i - 1$;
                    $k \leftarrow k + 1$, $k = 1, 2, \ldots$, length of $sub\_p_b$;
                    if $g_i \neq q_k$, then $osp \leftarrow <osp, g_i, q_k>$;
                    else $g_i = q_k$, then $osp \leftarrow <osp, g_i>$;
              else if $j = J$ then
                    $i \leftarrow i - 1$;
                    $k \leftarrow k + 1$, $k = 1, 2, \ldots$, length of $sub\_p_b$;
                    if $g_i \neq q_k$, then $osp \leftarrow <q_k, g_i, osp>$;
                    else $g_i = q_k$, then $osp \leftarrow <g_i, osp>$;
              else
                    $i \leftarrow i - 1$;
                    $k \leftarrow k + 1$, $k = 1, 2, \ldots$, length of $sub\_p_b$;
                    if $g_i \neq q_k$, then $osp \leftarrow <g_i, osp, q_k>$;
                    else $g_i = q_k$, then $osp \leftarrow <g_i, osp>$;
              end if
        end while
end procedure
```

Figure 3.19: Procedure of moon crossover [6]

In Figure 3.19, *osp* is the first offspring that is generated from crossover. The second offspring that is generated is denoted by sub_p_b. The variable g_i refers to a string in the first parent chromosome while q_i refers to string in the second parent.

Suppose that two chromosomes are p_a = [7 4 8 2 3 5 1 6] and p_b = [3 6 1 4 8 2 5 7]. First, we select the substring from p_a at random. In this example, the substring is selected as *osp* = [4 8]. Then, we can obtain sub_p_b = [3 6 1 2 5 7] from p_b.

Next, $g_2 = 4$ and $q_1 = 3$ because $i \leftarrow 3 - 1$ and $k \leftarrow 0 + 1$. In this case, the number 3 refers to the third position in the first offspring, *osp*. Therefore, the number on second position from p_a and the number on the first position from p_b are selected, which are 4 and 3. The number 4 already exist in *osp*, so we cannot add these cities into the first offspring, *osp*. Therefore the first offspring becomes *osp* = [4 8 3].

In the same way, add g_1, q_2, and the first offspring becomes *osp* = [7 4 8 3 6]. Now the next string is g_8 = 6 and q_3 = 1, and the *osp* = [7 4 8 3 6 1]. The next iteration comes out with g_7 = 1 and q_4 = 4. Both values already exist in *osp*, thus the values are ignored. The value of g_6 = 5 and q_5 = 8 is also ignored because they already exist in *osp*. Then, g_5 = 3 and q_6 = 2 have produced *osp* = [7 4 8 3 6 1 2]. Finally, q_7 is picked and the offspring becomes *osp* = [5 7 4 8 3 6 1 2]. For the second offspring, the selected *osp* in earlier iteration which is [4 8] is combined with *sub_pb*. Then it produced the second offspring as [3 6 1 2 5 7 4 8].

Step 3.2: Mutation

The exchange (swap) mutation operator is used by Moon. The exchange mutation scheme is selecting two genes in a chromosome at random and swapping between them. For example, the parent chromosome is given as [7 4 8 2 3 5 1 6]. Then two strings are selected randomly, which are 8 and 1. The selected strings are then swapped each other that produces new offspring as [7 4 1 2 3 5 8 6].

3.4 Proposed GA procedure for TSPPC

The proposed GA procedure for TSPPC is a modified PROX algorithm which integrates topological sort techniques in the procedure in order to obtain feasible solution subject to precedence constraints. In TSPPC, the precedence constraints require that certain nodes must precede certain other nodes in any feasible directed tour. For this reason, the use of conventional genetic algorithm procedure for TSP, with an order-based representation, might generate invalid candidate solutions. To overcome this problem, Moon's work is benchmarked which incorporated the topological sort technique in the GA procedure to handle the constraints and to generate only the feasible solution during the evolutionary process. The proposed TSPPC procedure maintains the main steps which are Initialization & Representation, Evaluation & Selection and Generation of offspring as in the GA procedure for general TSP. The only difference is in the representation stage. The chromosomes in the initial population as well as the offspring chromosomes created from the reproduction process need to be repaired before going through the evaluation process. The procedure and the flowchart of the proposed algorithm are presented in Figure 3.20 and Figure 3.21 respectively.

```
Procedure: Proposed algorithm
Begin

   Step 1: Initialization & Representation
   Step 1.1: Set GA parameters and the problem information
   For i ←1 to pop_size do
       Step 1.2: Generate random permutation of sequence ($x_1,\ldots x_N$) with N strings
   End For

  While number of generation < ngener do
   While population < pop_size do
    While length of chromosome < N do
       Step 1.3 Route repair
       Step 1.3.1: Check and store available task without incoming edge in available set
       Step 1.3.2: Select and store task in earlier position in updated sequence
       Step 1.3.3: Remove edge from selected task
    End While
   End While

       Step 2: Evaluation & Selection
       Step 2.1: Calculate fitness value of each chromosome in i
       Step 2.2: Select chromosomes in i as parents using Roulette Wheel selection scheme
       Step 3: Generation of Offspring
       Step 3.1: With probability $P_c$, select chromosomes in step 2.2 and apply linear order crossover
       Step 3.2: With probability $P_m$, select chromosomes in step 3.1 and apply inversion mutation

  End While
End procedure
```

Figure 3.20: Procedure of the proposed GA for TSPPC

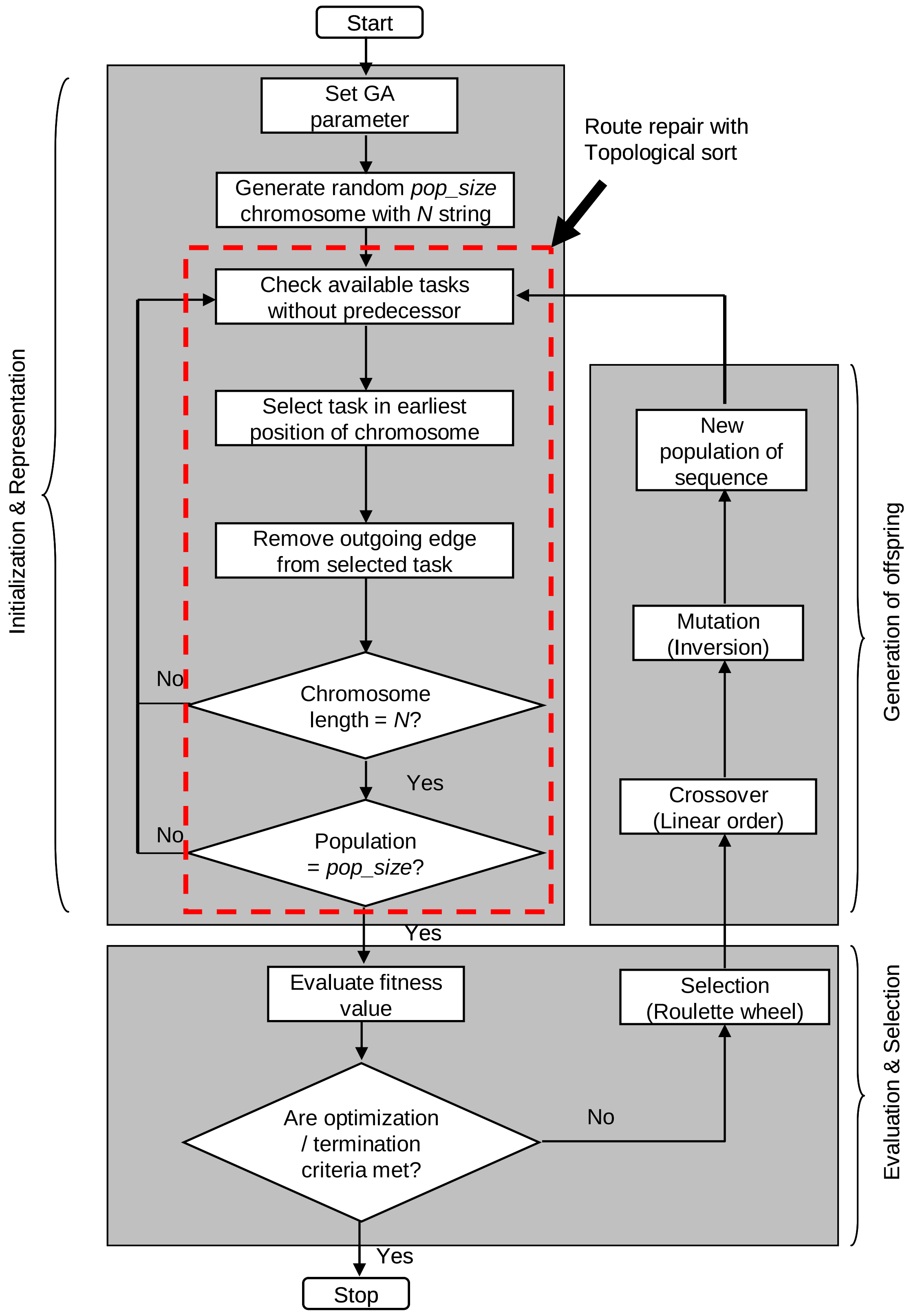

Start
Set GA parameter
Generate random pop_size chromosome with N string
Route repair with Topological sort
Check available tasks without predecessor
Select task in earliest position of chromosome
Remove outgoing edge from selected task
Chromosome length = N?
No
Yes
Population = pop_size?
No
Yes
Evaluate fitness value
Are optimization / termination criteria met?
No
Yes
Stop
Selection (Roulette wheel)
Crossover (Linear order)
Mutation (Inversion)
New population of sequence
Initialization & Representation
Generation of offspring
Evaluation & Selection

Figure 3.21: Flowchart of the proposed GA for TSPPC

Step 1: Initialization & representation

Step 1.2: Generate random initial population

For initial population, the random permutation method is used to generate chromosomes. The integer from 1 to *N*, which is the number of tasks, is generated in random sequence. The number of chromosome generated is depending on the size of population, *pop_size*. These sequences normally did not satisfy the precedence constraint. Therefore, the infeasible chromosomes must be repaired using the topological sort technique.

Step 1.3: Route repair

Route repair in the proposed algorithm for TSPPC is based on the topological sort technique and is explained in Step 1.3.1 through Step 1.3.4.

Step 1.3.1: Check available task

Initially, a chromosome is generated randomly and may not feasible. For example the chromosome structure represented as [4 1 3 6 5 2] in Figure 3.22 is infeasible because it did not satisfy the precedence constraint. In order to repair the chromosome become feasible solution, tasks without predecessor are selected and store in *available set*. In this example, task 1 is the only task without predecessor and therefore task 1 is selected and being stored in sequence. Then the outgoing edges of task 1, which is task 2, 3 and 6, are removed. As a result, the new *available set* consist of task 2, 3 and 6 as displayed in Table 3.7.

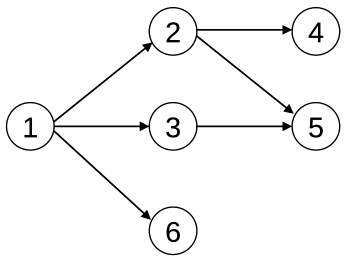


Figure 3.22: Precedence diagram

Step 1.3.2: Select task in earliest position on chromosome

In the proposed GA for TSPPC, the selection of task to be stored in sequence is based on the "earliest position" found in the chromosome. By referring to the *available set* [2 3 6], task number 3 is firstly found in the chromosome [4 1 3 6 5 2]. Therefore, task 3 is selected as the second string to be stored in sequence and the *updated sequence* is now consists of [1 3].

Step 1.3.3: Remove edges from selected task

When task 3 is selected to be stored in sequence, the outgoing edge of this task should be removed. Therefore, the edge 3 $\rightarrow$ 5 is removed, and the new available set is consisting of [2 6]. Again, based on 'earliest position' selection of task approach, task 6 is first appeared before task 2 in the chromosome [4 1 3 6 5 2] and therefore task 6 is selected to be placed in *updated sequence*. The selection procedure is repeated until the length of the sequence is equal to *N*. The final feasible path that is generated from this approach is [1 3 6 2 4 5]. Table 3.7 shows the selection of task based on "earliest position" in generated sequence. An example of initial population with *pop_size*=12 and repaired chromosomes for this population are shown in Table 3.8 and Table 3.9, respectively.

Table 3.7: Selection of tasks using the "earliest position" technique

Chromosome: 4 1 3 6 5 2	
available set	*updated sequence*
1	[1]
2, 3, 6	[1 3]
2, 6	[1 3 6]
2	[1 3 6 2]
4, 5	[1 3 6 2 4]
5	[1 3 6 2 4 5]

Table 3.8: Chromosomes in the initial population

4	1	3	6	5	2
4	1	3	5	6	2
6	3	2	4	1	5
4	2	3	6	1	5
1	4	6	5	3	2
5	4	1	3	2	6
3	5	2	4	1	6
2	5	1	6	3	4
1	6	4	5	2	3
2	6	5	1	3	4
5	1	6	3	4	2
5	1	6	3	4	2

Table 3.9: Chromosomes after repairing process

1	3	6	2	4	5
1	3	6	2	4	5
1	6	3	2	4	5
1	2	4	3	6	5
1	6	3	2	4	5
1	3	2	5	4	6
1	3	2	5	4	6
1	2	6	3	5	4
1	6	2	4	3	5
1	2	6	3	5	4
1	6	3	2	5	4
1	6	3	2	5	4

Step 2: Evaluation & selection

Step 2.1: Calculate fitness value

The fitness value of each chromosome in the population is evaluated using the fitness function in Equation (2-1). This equation is still valid for TSPPC with excluding the distance of returning to the starting city.

```
Procedure: Fitness evaluation
    For $p$ ← 1 to $pop\_size$ do
        $d$ = 0 (salesman does not return to the starting city)
        For $k$ ← 2 to $n$ do ($n$ is the number of genes i.e. number of city)
            Calculate the distance between city $k$-1 to $k$ and add $d$ to get the total
            distance
        End For
        Return fitness value (i.e. total distance) for each chromosome in $p$
    End For
End Procedure
```

Step 2.2: Parent selection

The Roulette Wheel selection is used to select parent chromosomes to be re-generated for the next chromosome. By using the proportional Roulette Wheel, all individuals are given a chance to be selected and the chances of the fitter individual to be selected as a parent for crossover are higher.

Step 3: Generation of offspring

Step 3.1: Crossover

Linear order crossover is used to generate two new offspring. This operator is the most frequently used for the crossover operation when the chromosome representation is ordinal [118]. This crossover operator can preserve both the relative positions between genes and the absolute positions relative to the extremities of parents as much as possible.

Step 3.2: Mutation

Mutation operation based on inversion (flip) as described in the GA procedure for TSP is applied in the chromosome after crossover process.

3.5 Analysis of algorithms to solve TSPPC

The previous sections present details of two different algorithms which are Moon’s algorithm and the proposed algorithm. This section analyses the differences between algorithms which make it unique compared to each other. All algorithms consist of three main steps, which are Initialization & Representation, Evaluation & Selection and Generation of offspring. In the first step, the main different is chromosome definition. In the proposed algorithm, the chromosome is defined as a sequence of task, while in the Moon’s algorithm chromosome is defined as priority factor. However, initial chromosomes are created in a similar way, which is a random permutation method. In both algorithms, all chromosomes need to be repaired since the chances to generate feasible chromosome from random permutation is very low.

In the Moon’s algorithm, the task selection is based on the highest priority. The task with the highest priority in *available set* is selected to be in the next position in the task sequence. The use of priority factor in selecting task actually affects the time to generate the best solution. It is because the GA operators do not directly generate the solution, but it generates the permutation of priority. By generating the priority, the chances of chromosome to change and achieve an optimal solution are slightly slow. For the proposed algorithm, the selection of task is based on the earliest position found in the chromosome. Both algorithms still maintain genetic character in the task sequence because selection is performed based on an original chromosome as described earlier. However the Moon’s algorithm required an additional step to compare all available tasks and its priority before decided to select a task. Another difference is in the third step which is Generation of offspring. In the Moon’s algorithm, moon crossover is used, while for the proposed algorithm, the traditional linear order crossover operator that widely used in the conventional genetic algorithm is adapted. The mutation technique used is also different which is exchange (swap) mutation in Moon’s algorithm while inversion (flip) mutation is used for the proposed algorithm. Table 3.10 summarizes the differences between the two algorithms while Figure 3.23 compares the two different approaches presented in the study. The differences are highlighted with yellow colours for Moon’s approach and green colours for the proposed approach.

Table 3.10: Summary of Moon and the proposed algorithm

Operators	**Moon**	**Proposed**
Representation	priority factor as chromosome	sequence of task as chromosome
Task selection	Based on the highest priority	Based on the earliest position found on chromosome
Parent selection	Proportional Roulette Wheel	Proportional Roulette Wheel
Crossover method	moon crossover	Linear order crossover
Mutation Method	Exchange (swap) mutation	Inversion (flip) mutation
Generation gap	0.9	0.9

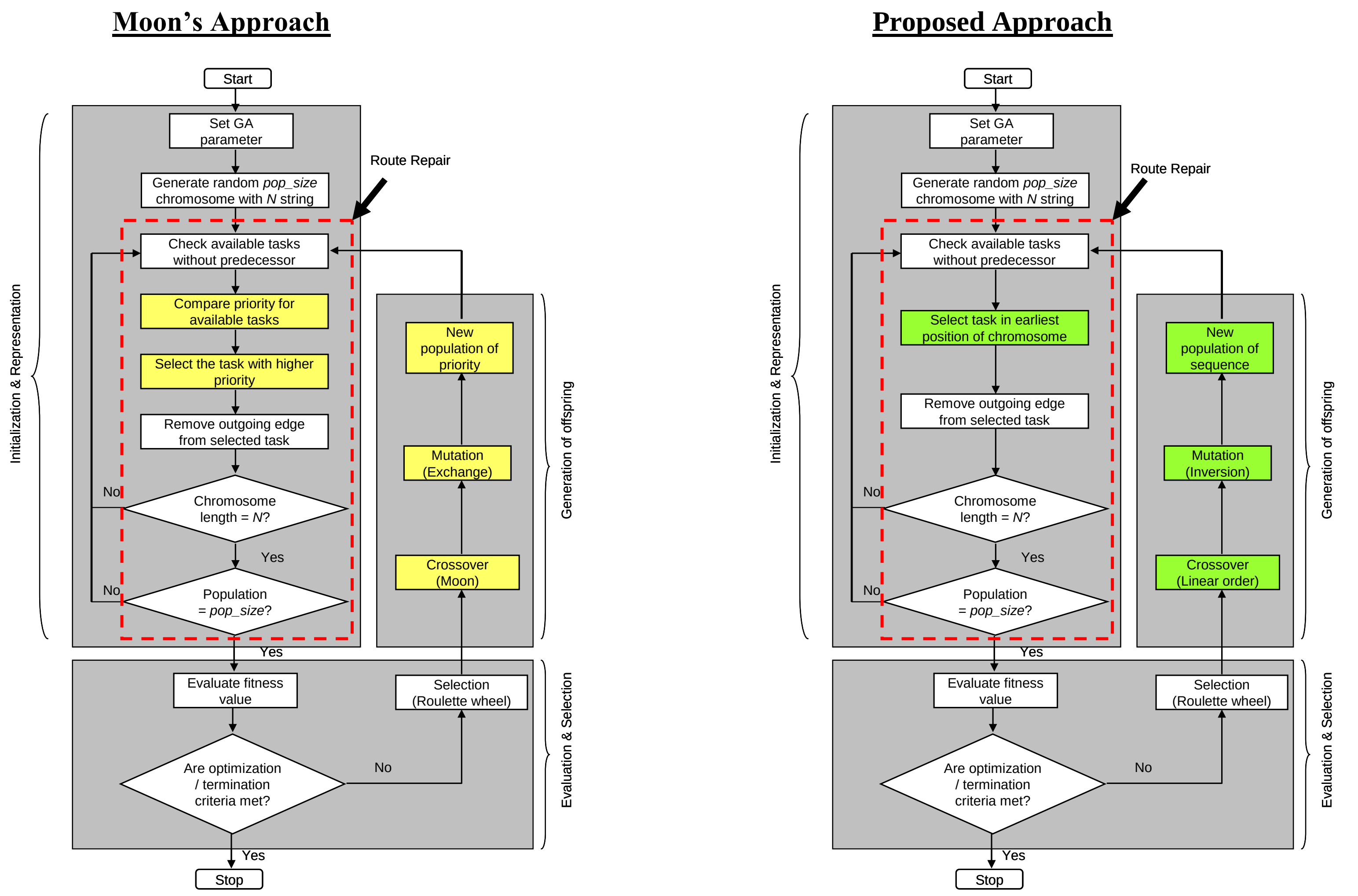

Moon's Approach
Start
Set GA parameter
Generate random *pop_size* chromosome with *N* string
Route Repair
Check available tasks without predecessor
Compare priority for available tasks
Select the task with higher priority
Remove outgoing edge from selected task
Chromosome length = *N*?
No
Yes
Population = *pop_size*?
No
Yes
New population of priority
Mutation (Exchange)
Crossover (Moon)
Evaluate fitness value
Selection (Roulette wheel)
Are optimization / termination criteria met?
No
Yes
Stop
Initialization & Representation
Generation of offspring
Evaluation & Selection
Proposed Approach
Start
Set GA parameter
Generate random *pop_size* chromosome with *N* string
Route Repair
Check available tasks without predecessor
Select task in earliest position of chromosome
Remove outgoing edge from selected task
Chromosome length = *N*?
No
Yes
Population = *pop_size*?
No
Yes
New population of sequence
Mutation (Inversion)
Crossover (Linear order)
Evaluate fitness value
Selection (Roulette wheel)
Are optimization / termination criteria met?
No
Yes
Stop
Initialization & Representation
Generation of offspring
Evaluation & Selection

Figure 3.23: Comparisons of Moon's Approach and Proposed Approach

Chapter 4

Computational Experiments & Results

This chapter deals with computational experiments and results for TSP and TSPPC. The experiments consist of two parts which start off with TSP followed by TSPPC. The procedure of GA to solve TSP and TSPPC described in the previous chapter are used to obtain the optimal solution of several TSP and TSPPC instances. In the first part of the chapter, the known optimal solution problems taken from TSPLIB are used in the TSP experiments. The TSPPC experiments are then carried out in the second part of the chapter where the problems with known optimal solution as well as randomly generated problems are tested. To confirm that the proposed GA used in this thesis is a stable and robust approach, few TSPPC application examples benchmarked from relevant published papers are included in the experiments. The results of the experiments are further discussed in the next chapter.

4.1 Experimental set-up and assumptions

The GA parameter such as population size (*pop_size*), the probability of crossover (P_c), probability of mutation (P_m) and the maximum number of generations (*ngener*) used in the experiment are obtained from trial run and from a simple design of experiment (DOE). For all experiments, the population size used is set large enough to ensure it doesn't stuck at a local optimum. The crossover rate is set relatively high which is ranging from 0.5 to 0.9 while mutation rate is set relatively low ranging from 0.01 to 0.2. For known optimal solution problems, the choice of parameters setting are by trial and error in order to minimise the computation time, while for randomly generated problems, the DOE table is used to assist the experimentation. In the genetic algorithm, termination criterion is a must. For all experiments in this thesis, termination is performed when number of generation reached the maximum number of generations. The maximum number of generations (*ngener*) is set earlier in the program code. The computer numerical experiment set up is given as follows;

- Computer hardware is fixed. Computational experiments will be performed on DELL with Intel Core 2 Duo 2.0 GHz CPU and 2.0 GB of RAM.
- The algorithms are coded in MATLAB version 2009b.

4.2 Objectives of the experiment

In general, there are three objectives to be achieved in the experiments. The two main objectives are to get the optimal solution for TSP/TSPPC and to obtain the number of generations to come out with the optimal solution.

Optimal solution

The optimal solution can be the minimum value for minimisation problem or the maximum value of maximisation problem. In this computational experiment, the optimal solution will be the minimum distance/cost/time and the value is calculated from the objective function.

Number of generations to come out with an optimal solution

This figure indicates how many generations are required for the algorithm to achieve optimal solution for specific case study. For this objective, a smaller number indicates better algorithm, because it can produce optimal solutions with less number of generations.

Iteration time to generate optimal solution

The iteration time is an elapsed time between iteration. This is the time needed for an algorithm to complete one generation. In this thesis, the iteration time to generate optimal solution is called convergence time while the iteration time to complete the generations is called completion time. Theoretically, for larger maximum number of generations, longer completion time will be acquired. The algorithm that completes all generations in the fastest time does not represent the most efficient algorithm, until the optimal solution is achieved. An efficient algorithm will come out with the fastest iteration time to converge to the optimal solution. In actual application, users do not care the number of generations to achieve an optimal solution as long as the algorithm can produce optimal solutions in the shortest time. The time to compute for optimal solution will also increase when using large population size. The iteration time also very depend on the CPU used because different hardware condition will gives different performance, and therefore the same test machine is used for the whole experiments to reduce the variability.

The third objective is not so critical in this computational experiment because the main target of this study is not developing the most efficient algorithm for TSP/TSPPC, and therefore no critical comparison is made to the algorithms that successfully developed by the other researchers. Besides, direct comparison of computation time is impossible due to different hardware and software used by the other researchers.

4.3 Computational experiments for TSP

This section will concentrate on computational experiments that use the GA approach discussed in Chapter 3 to obtain the optimal solution for TSP instances.

4.3.1 Complete enumeration for five-city problem

Homaifar [119] states that one approach which would certainly find the optimal solution of any TSP is the application of exhaustive enumeration and evaluation. The procedure consists of generating all possible tours and evaluating their corresponding tour length. The tour with the smallest length is selected as the best, which is guaranteed to be optimal.

The complete enumeration procedure is carried out for five-city problem by listing all possible tours and manually calculating the total distance of all possible tours. The objective is to obtain the minimum distance to complete the tour. The results of a complete enumeration will be used as a comparison basis in a later computational experiment and to build a confidence level that the GA performs well as an optimisation technique in finding optimal solutions. The location for 5-city problem is given in the 2-d coordinate system as depicted in the Figure 4.1.

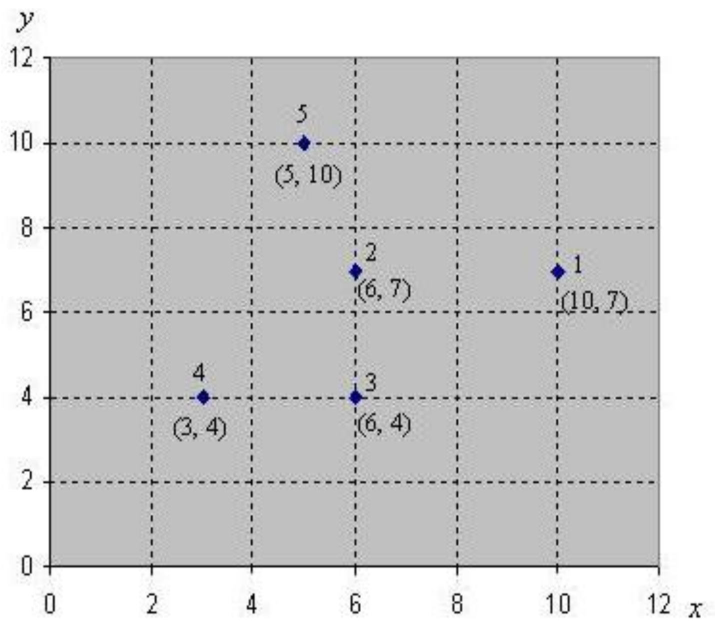


Figure 4.1: Location for 5-city problem

The total permutation of 5 cities for symmetrical tour is (*n*-1)!/2, which gives (4*3*2*1)/2. Therefore 12 possible solutions or tours are identified. The 12 possible candidate solutions are as below;

1.	1-2-3-5-4 = 4-5-3-2-1 (symmetrical tour)	7.	1-3-4-2-5
2.	1-2-4-3-5	8.	1-3-2-5-4
3.	1-2-5-3-4	9.	1-3-2-4-5
4.	1-2-3-4-5	10.	1-3-5-2-4
5.	1-2-5-4-3	11.	1-4-2-3-5
6.	1-2-4-5-3	12.	1-4-3-2-5

The distance, *d* is calculated using the Euclidean distance described in (2-5). Therefore, distance calculation for tour 1-2-3-5-4-1 (close path) is;

$$\begin{aligned} d &= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} + \ldots. \\ &\quad + \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2} \\ &= \sqrt{(6-10)^2 + (7-7)^2} + \sqrt{(6-6)^2 + (4-7)^2} + \sqrt{(5-6)^2 + (10-4)^2} \\ &\quad + \sqrt{(3-5)^2 + (4-10)^2} + \sqrt{(10-3)^2 + (7-4)^2} \\ &\quad \sqrt{16} + \sqrt{9} + \sqrt{37} + \sqrt{40} + \sqrt{58} = 27.0231 \end{aligned}$$

The result of the distance for all permutations is presented in table 4.1. It shows that the minimum distance for five-city problem with complete enumeration is 21.2359 and the shortest tour is 1-3-4-2-5-1.

Table 4.1: Result for five-city problem with complete enumeration solution

Tour	Total distance	Tour	Total distance
1-2-3-5-4-1	27.0231	**1-3-4-2-5-1**	**21.2359**
1-2-4-3-5-1	23.1564	1-3-2-5-4-1	25.1026
1-2-5-3-4-1	23.8608	1-3-2-4-5-1	24.3981
1-2-3-4-5-1	22.1555	1-3-5-2-4-1	26.1035
1-2-5-4-3-1	21.4868	1-4-2-3-5-1	26.7721
1-2-4-5-3-1	25.6500	1-4-3-2-5-1	22.6090

4.3.2 Solution for five-city with PROX algorithm

The five-city problem that's already been solved by the complete enumeration procedure will be tested in PROX algorithm. The PROX algorithm employs Roulette Wheel selection rule to select individuals for reproduction while the linear order crossover and inversion mutation are employed to produce individuals for the next generation. The location of the cities is similar as in Figure 4.1 while the distance matrix which is the distance between the cities generated by the algorithm is given in Table 4.2. The genetic parameters used in the experiment are as follows; *pop_size*=10, P_c=0. 5, P_m=0. 1 and *ngener*=20.

Table 4.2: Distance matrix for five-city problem

City	1	2	3	4	5
1	0	4.0000	5.0000	7.6158	5.8310
2	4.0000	0	3.0000	4.2426	3.1623
3	5.0000	3.0000	0	3.0000	6.0828
4	7.6158	4.2426	3.000	0	6.3246
5	5.8310	3.1623	6.0828	6.3246	0

The results for five-city problem produced by PROX algorithm are shown in Figure 4.2. The first diagram shows the optimal tour (1-3-4-2-5-1) that the salesman must travel with the minimum distance of 21.2359. The second diagram is a graph of best and average distance found in each generation. It shows that the algorithm is converged at the optimal solution as early as generation 1. With the small size of population, high probability of crossover and a small probability of mutation, the minimum distance found by GA is 21.2359 within 0.0075 seconds. The result obtained from PROX algorithm is exactly the same as calculated by complete enumeration. This proves that genetic algorithm based PROX is able to solve TSP in finding optimal solutions with the fastest time.

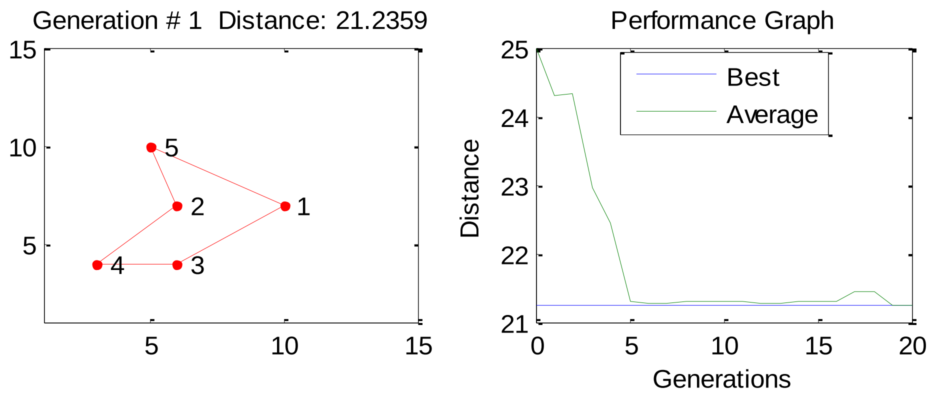
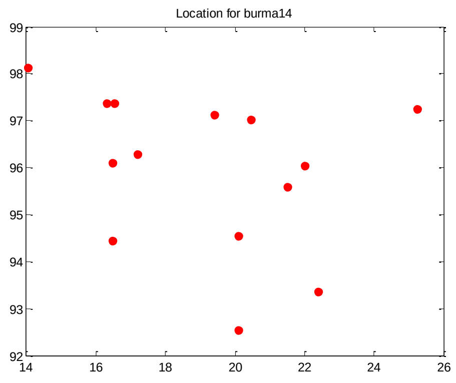


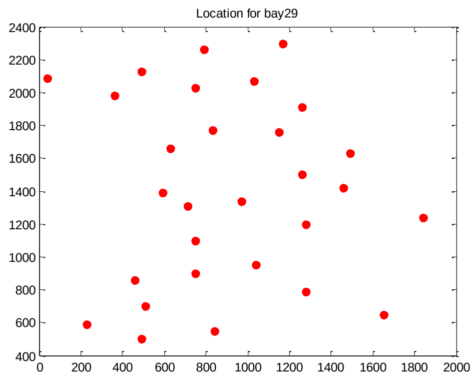
Figure 4.2: Optimal solution for five-city problem with PROX algorithm

4.3.3 Benchmark problem from TSPLIB

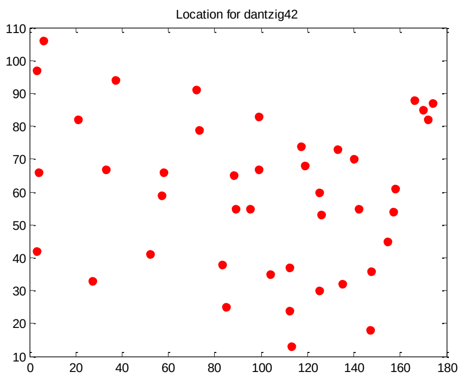
The TSP algorithms are now tested at several problems benchmarked from TSPLIB [26] in which the optimal solution is known. There are four problems called burma14, bay29, dantzig42 and eil51. The burma14, bay29, dantzig42 and eil51 have 14, 29, 42 and 51 instances, respectively. The datasets of the instances are given in **APPENDIX A-1**. All problems are tested at three different algorithms developed in chapter 3, which are PROX, RBOX and TSOX. Due to time constraint, the parameter values used for all experiments are obtained through trial and error. The locations of the cities are shown in Figure 4.3. A complete coding for the computational experiment using PROX, RBOX and TSOX algorithms are presented in **APPENDIX A-2**.



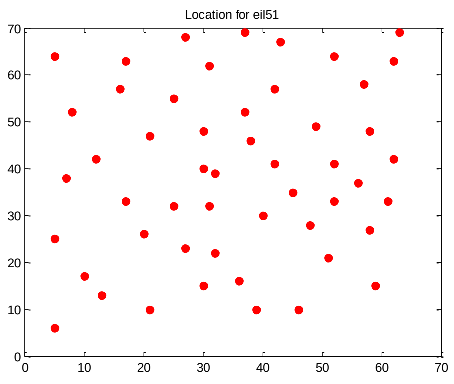
Location for burma14



Location for bay29



Location for dantzig42



Location for eil51

Figure 4.3: City location for burma14, bay29, dantzig42 and eil51

Results with PROX algorithm

The GA parameters setting used in each problem using PROX algorithm are shown in Table 4.3. Note that different parameters setting is required for different problems. The larger the problem size, the larger population size and maximum number of generations used to ensure better search process. Figure 4.4 through Figure 4.7 presents the optimal tour and the performance graph (showing best and average distance found in each generation) for all problems tested.

Table 4.3: GA parameters setting for PROX algorithm

Problem	*pop_size*	P_c	P_m	*ngener*
burma14	140	0.9	0.01	50
bay29	600	0.9	0.01	150
dantzig42	800	0.95	0.01	250
eil51	1000	0.88	0.01	300

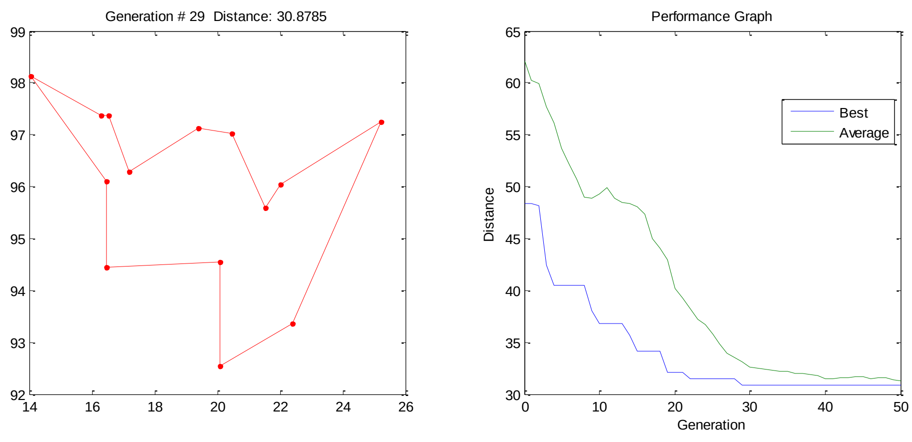


Figure 4.4: Optimal tour & performance graph for burma14 with PROX algorithm

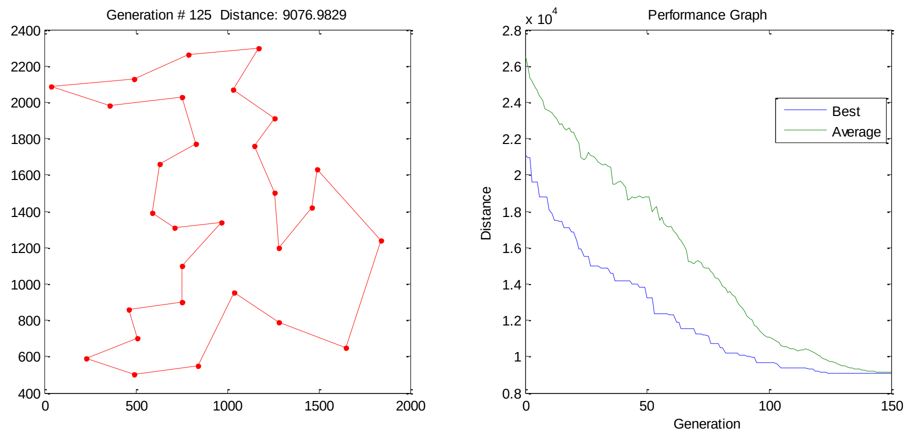


Figure 4.5: Optimal tour & performance graph for bay29 with PROX algorithm

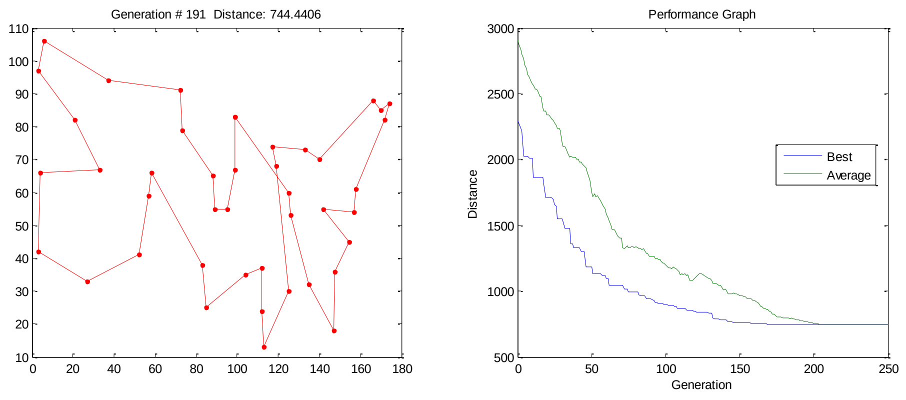


Figure 4.6: Optimal tour & performance graph for dantzig42 with PROX algorithm

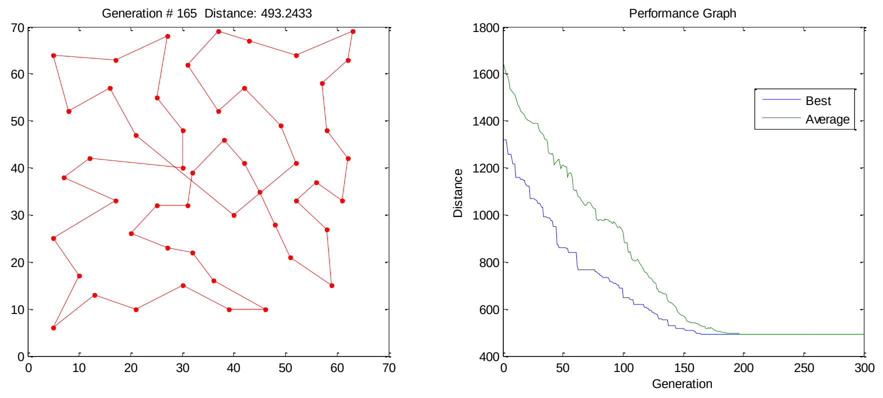


Figure 4.7: Optimal tour & performance graph for eil51 with PROX algorithm

Results with RBOX algorithm

In this experiment, the RBOX algorithm is tested for burma14, bay29, dantzig42 and eil51 using similar crossover and mutation operators as in PROX experiment. In contrast with PROX, RBOX use rank-based Roulette Wheel selection for parent selection. Table 4.4 shows the parameters used in the experiments. Figure 4.8 through Figure 4.11 shows the optimal tour and the performance graph of all TSP instances run with RBOX algorithm.

Table 4.4: GA parameters setting for RBOX algorithm

Problem	*pop_size*	P_c	P_m	*ngener*
burma 14	140	0.9	0.01	100
bay 29	600	0.89	0.01	500
dantzig42	800	0.79	0.1	1500
eil51	1000	0.92	0.05	1500

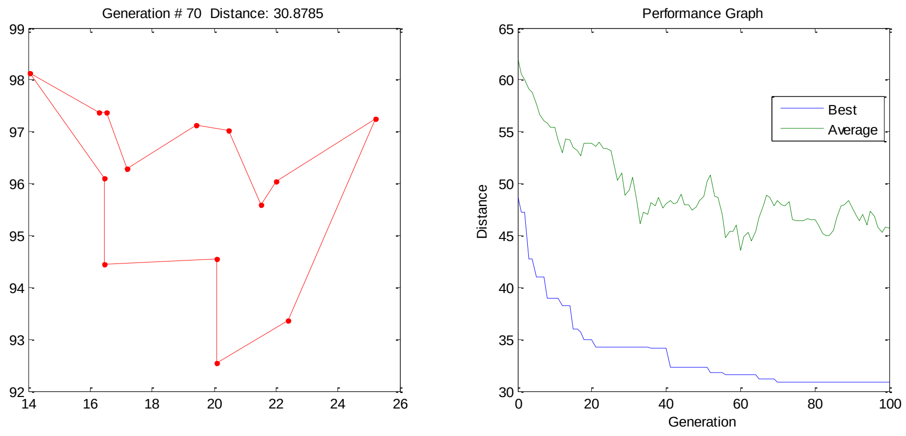


Figure 4.8: Optimal tour & performance graph for burma14 with RBOX algorithm

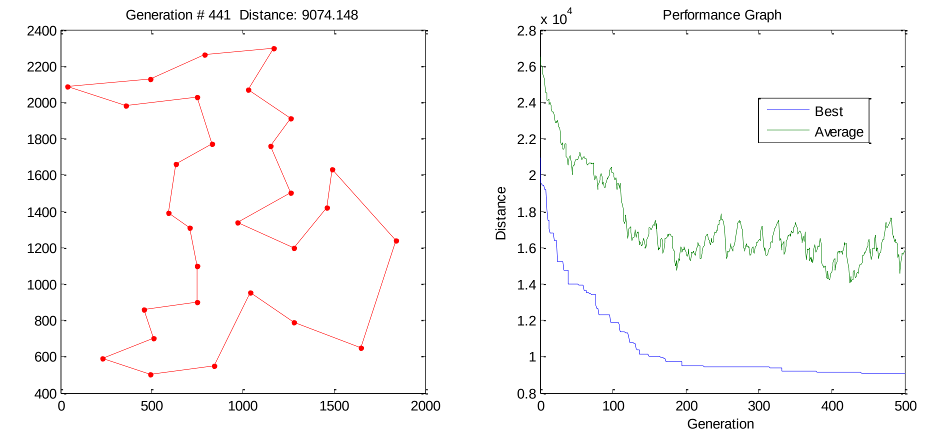


Figure 4.9: Optimal tour & performance graph for bay29 with RBOX algorithm

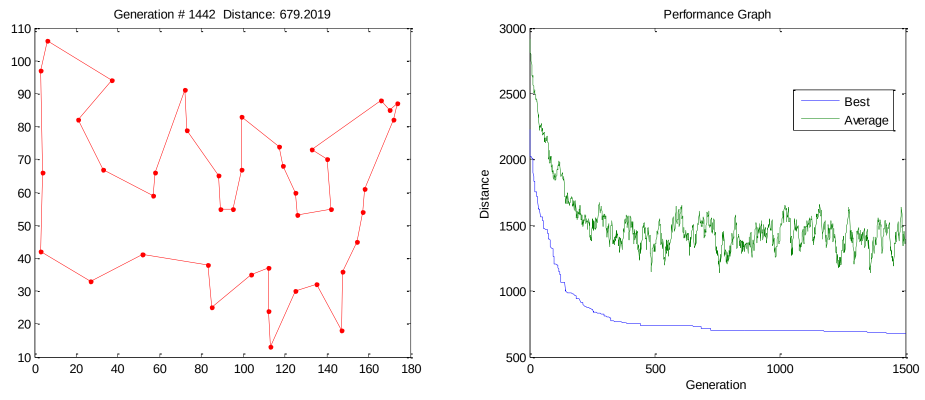


Figure 4.10: Optimal tour & performance graph for dantzig42 with RBOX algorithm

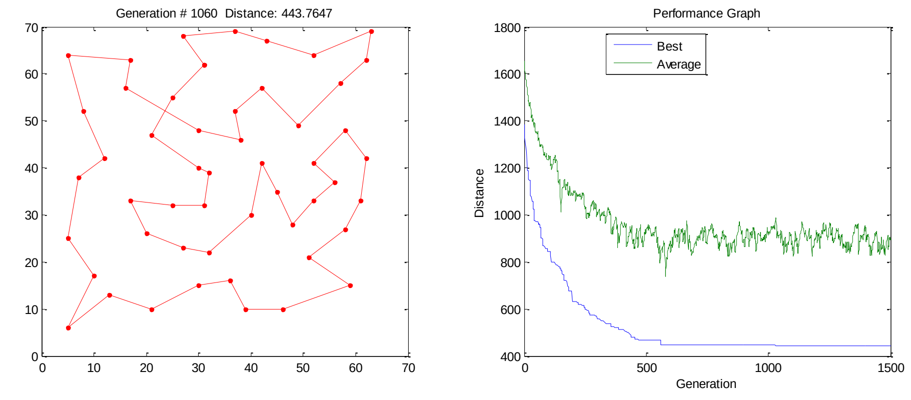


Figure 4.11: Optimal tour & performance graph for eil51 with RBOX algorithm

Results with TSOX algorithm

The TSOX procedure also utilizes the same crossover and mutation technique to generate offspring. This time tournament method is used in the selection stage and it is again tested at burma14, bay29, dantzig42 and eil51. The setting of parameters for each experiment is chosen based on trial and error as in Table 4.5. The optimal tour, the average and the minimum distance found in each generation for all TSP instances are depicted in Figure 4.12 through Figure 4.15.

Table 4.5: GA parameters setting for TSOX algorithm

Problem	*pop_size*	P_c	P_m	*ngener*
burma 14	140	0.9	0.01	50
bay 29	600	0.9	0.01	100
bantzig42	800	0.98	0.01	200
eil51	1000	0.88	0.01	300

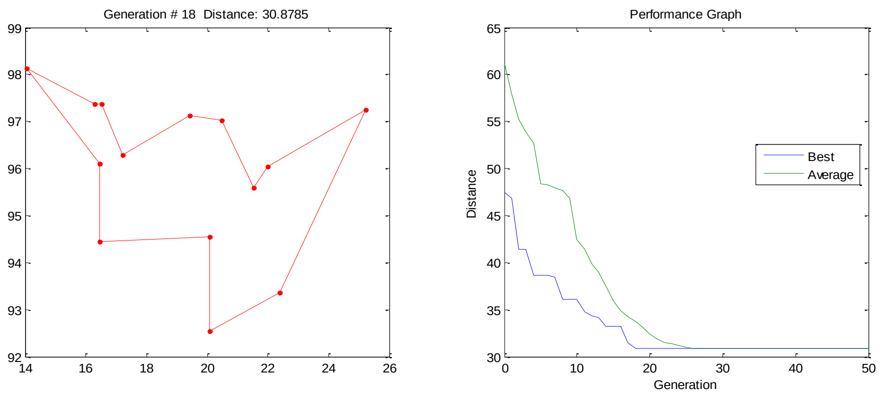


Figure 4.12: Optimal tour & performance graph for burma14 with TSOX algorithm

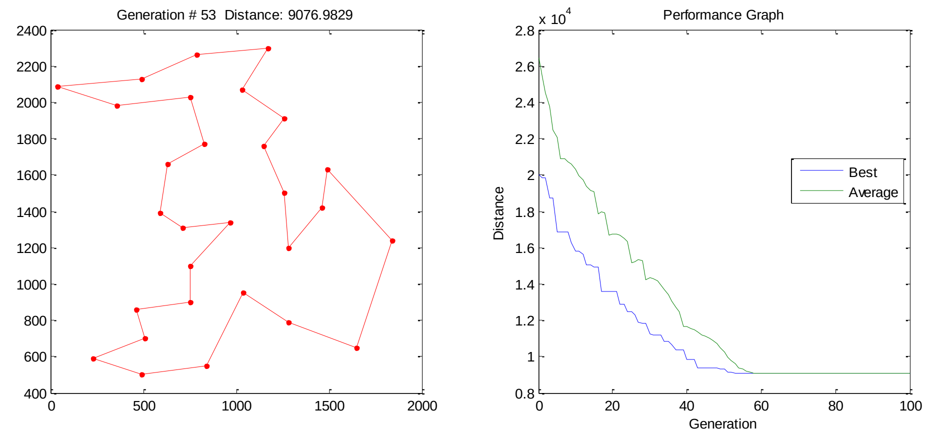


Figure 4.13: Optimal tour & performance graph for bay29 with TSOX algorithm

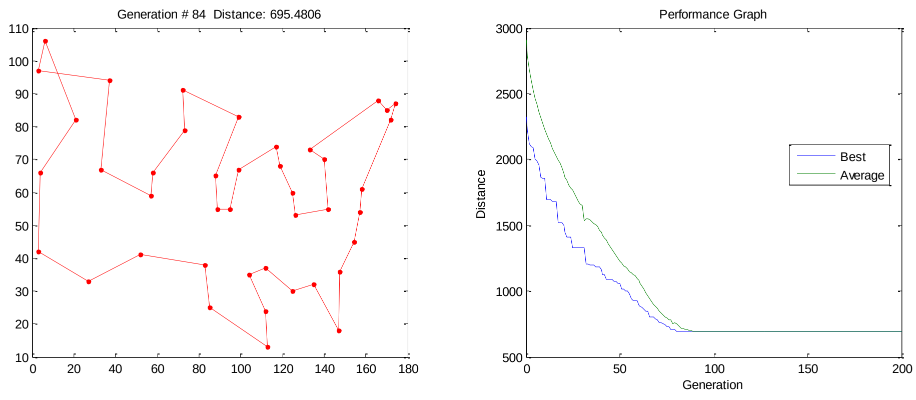


Figure 4.14: Optimal tour & performance graph for dantzig42 with TSOX algorithm

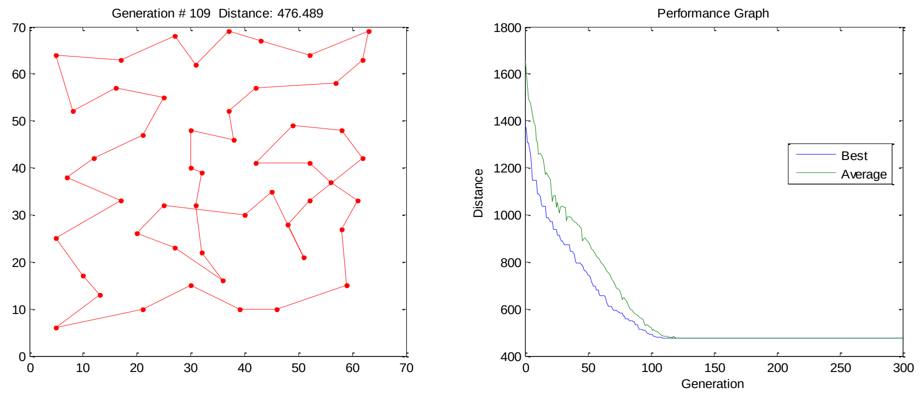


Figure 4.15: Optimal tour & performance graph for eil51 with TSOX algorithm

Table 4.6 compares the best solution found for four TSP instances run with three different procedures. The iteration time (in sec) for the TSPLIB problems using different algorithms are summarized in Table 4.7.

Table 4.6: Quality of solution (Best solution) comparisons for TSPLIB problem

Problem	Known optimal	PROX	RBOX	TSOX
burma 14	31	31	31	31
bay 29	9074	9077	9074	9077
dantzig42	679	744	679	695
eil51	425	493	444	476

Table 4.7: Iteration time for TSPLIB problem using three different algorithms

Problem	PROX	RBOX	TSOX
burma14	0.208291	0.34826	0.231
bay29	4.345619	14.31729	1.76294
dantzig42	13.72937	83.80755	5.810677
eil51	25.09628	133.6516	12.8272

4.4 Computational experiments for TSPPC

In this section, PROX algorithm which utilises proportional roulette wheel in the selection stage will be used for the whole TSPPC experiments. Although RBOX gives the highest quality solution, it seems to be computationally expensive. A modified PROX algorithm which integrates topological sort technique in the procedure is used to obtain feasible solutions for TSPPC. The topological sort technique is benchmarked from Moon’s work in order to repair infeasible chromosomes in the initial population as well as after reproduction process. The proposed GA procedure to solve TSPPC employs a proportional Roulette Wheel selection, linear order crossover and inversion mutation technique as in PROX algorithm. The results of the experiments using the proposed TSPPC algorithm will be compared with the results of the Moon’s algorithm. The moon’s algorithm uses a combination of Roulette Wheel selection, moon crossover and exchange mutation. Beside crossover and mutation, there is also different technique used in terms of selecting tasks in the representation stage as described in chapter 3. The main objective of the experiments is to obtain the feasible sequence of the tasks such that it does not violate the precedence relationships and the distance/cost/time of the complete order is minimised. In addition to that, the performance of the algorithm is also monitored through performance graph whereby efficient algorithm will produce optimal or near optimal solution with less number of generations and less iteration time.

4.4.1 TSPPC test problem

To confirm that the proposed algorithm is effective in solving TSPPC, three TSPPC problems with known optimal solution as well as randomly generated problem are tested. The two approaches i.e. proposed algorithm and Moon's algorithm are compared with respect to the quality of the best solution and the number of generations to reach the best solution. The coding of the proposed algorithm and Moon's algorithm are supplied in **APPENDIX A-3** and **APPENDIX A-4**, respectively.

a) Test problem 1: 6 tasks & 6 precedence constraints

The first test problem consists of 6 tasks and 6 precedence constraints taken from Moon [6]. Figure 4.16 and Table 4.8 illustrates the precedence diagram of the problem and the transition time between the tasks, respectively. The GA parameters used in the experiment are presented in Table 4.9. Figure 4.17 displays the performance graph, indicating best and average transition time found in each generation for both algorithms.

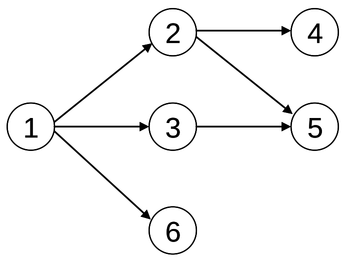


Figure 4.16: Precedence diagram for test problem 1

Table 4.8: Transition time matrix for test problem 1

	1	**2**	**3**	**4**	**5**	**6**
1	-	7	5	6	10	9
2	7	-	14	6	10	8
3	5	14	-	16	16	10
4	6	6	16	-	10	6
5	10	10	16	10	-	12
6	9	8	10	6	12	-

Table 4.9: GA parameters setting for test problem 1

Parameter	*pop_size*	P_c	P_m	*ngener*
Proposed	12	0.6	0.1	20
Moon	12	0.6	0.1	20

The results of the experiment show that both algorithms are able to produce the optimal solution at the first generation. However, the computation time for the proposed algorithm is slightly better than the Moon's algorithm. The best tour found by both algorithms is [1-3-6-2-4-5] which is feasible and similar as reported on Moon's work. Table 4.10 summarizes the results obtained from the experiment.

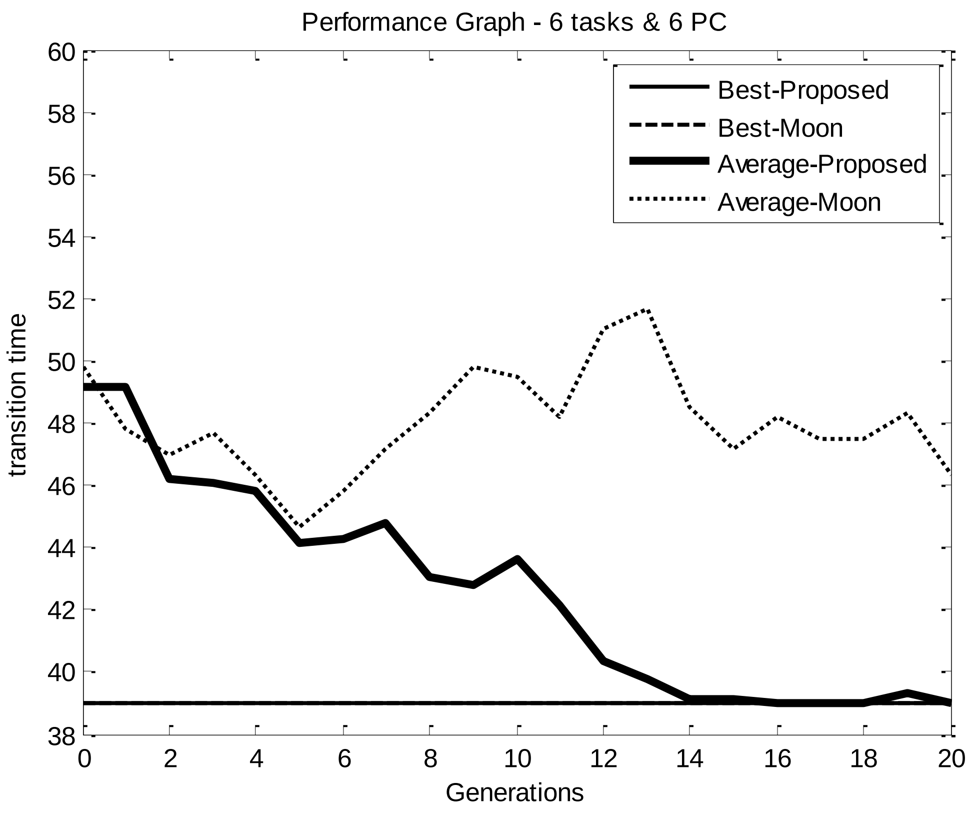


Figure 4.17: Performance graph for test problem 1

Table 4.10: Summary of results for test problem 1

	Gen#	Best (sec)	Convergence time (sec)	Completion time (sec)
Proposed	1	39	0.1786	0.2749
Moon	1	39	0.2226	0.3286

b) Test problem 2: 8 tasks & 9 precedence constraints

The second experiment uses the asymmetric TSP data from Escudero [74] to obtain the minimum cost in the FMS scheduling. The set of problems which consists of 8 tasks and 9 precedence constraints are shown in Figure 4.18, while the cost matrix is given in Table 4.11. Table 4.12 displays the GA parameters used in both experiments. For both experiments, similar settings of parameters are used except for the crossover rate. The probability of crossover used in the Moon’s experiment is set lower in order to get the optimal solution and similar task sequence as reported in the paper.

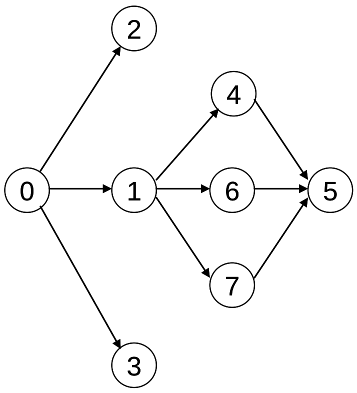


Figure 4.18: Precedence diagram for test problem 2

Table 4.11: Cost matrix for Test problem 2

Cost	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	-	0	0	0	-	-	-	-
1	-	-	1	2	0.75	0	3	1
2	0	4	-	5	3.25	4	6	0
3	0	7	8	-	5.5	7	9	8
4	-	2.75	2.5	2.25	-	2.75	5.25	2.5
5	0	0	1	2	0.75	-	3	1
6	-	10	11	12	10.75	10	-	11
7	-	4	0	5	3.25	4	6	-

Table 4.12: GA parameters setting for test problem 2

	pop_size	P_c	P_m	*ngener*
Proposed	16	0.9	0.1	20
Moon	16	0.6	0.1	20

Both algorithms are able to achieve optimal cost of $21.25 with a feasible task sequence of [0-1-4-2-7-6-5-3]. Although they present similar quality of solution, however the proposed algorithm appears to be more effective as it uses less number of generations and less computation time compared to a Moon's algorithm. The performance graph for the Moon and the proposed algorithm are illustrated in Figure 4.19. Table 4.13 summarizes the results of the experiments.

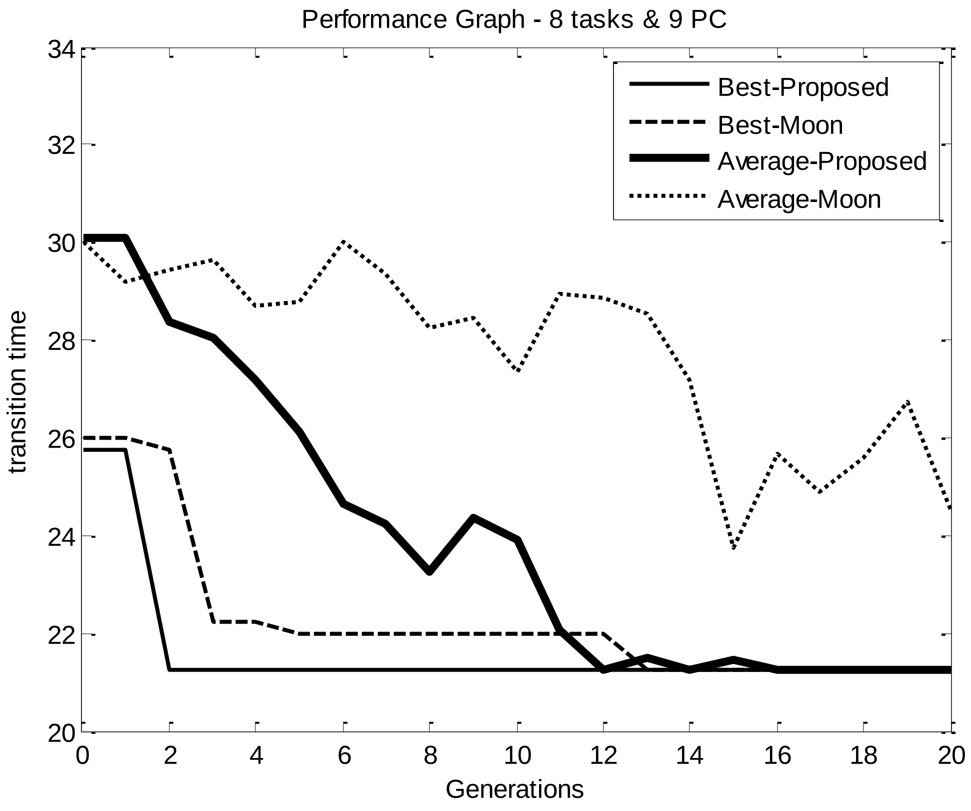


Figure 4.19: Performance graph for test problem 2

Table 4.13: Summary of results for test problem 2

	Gen#	Best Cost ($)	Convergence time (sec)	Completion time (sec)
Proposed	2	21.25	0.211	0.3063
Moon	13	21.25	0.3319	0.3729

c) Test problem 3: 20 tasks & 31 precedence constraints

The third experiment is again benchmarked from Moon's work which involves TSPPC problem with 20 tasks and 31 precedence constraints. The precedence diagram concerning the tasks and the precedence relationship is depicted in Figure 4.20. The transition time between the tasks are given in Table 4.14.

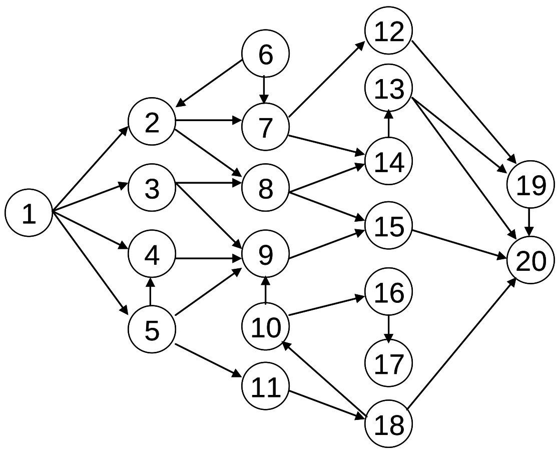


Figure 4.20: Precedence diagram for test problem 3

Table 4.14: Transition time matrix for test problem 3

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	-	2	10	4	10	2	11	9	1	3	7	12	5	8	8	8	13	7	2	12
2	2	-	12	12	4	6	2	8	6	7	14	10	11	9	2	6	13	14	1	3
3	10	12	-	5	8	2	5	9	7	8	13	6	9	2	6	6	14	2	4	5
4	4	12	5	-	6	6	11	12	5	11	4	5	11	1	3	10	17	10	14	14
5	10	4	8	6	-	7	2	13	3	10	6	7	14	8	7	7	5	1	8	13
6	2	6	2	6	7	-	2	6	14	6	9	3	7	13	13	3	13	10	13	3
7	11	2	5	11	2	2	-	11	14	6	4	10	7	6	12	9	10	8	5	4
8	9	8	9	12	13	6	11	-	14	14	3	11	1	1	3	10	6	5	14	14
9	1	6	7	5	3	14	14	14	-	3	12	12	2	12	4	2	14	13	11	7
10	3	7	8	11	10	6	6	14	3	-	8	4	8	5	4	4	3	6	12	11
11	7	14	13	4	6	9	4	3	12	8	-	5	9	4	9	9	6	12	14	11
12	12	10	6	5	7	3	10	11	12	4	5	-	7	6	10	13	7	1	6	8
13	5	11	9	11	14	7	7	1	2	8	9	7	-	1	1	4	1	12	4	6
14	8	9	2	1	8	13	6	1	12	5	4	6	1	-	4	9	12	4	9	13
15	8	2	6	3	7	13	12	3	4	4	9	10	1	4	-	1	1	1	1	5
16	8	6	6	10	7	3	9	10	2	4	9	13	4	9	1	-	8	5	2	14
17	13	13	14	17	5	13	10	6	14	3	6	7	1	12	1	8	-	14	7	10
18	7	14	2	10	1	10	8	5	13	6	12	1	12	4	1	5	14	-	5	6
19	2	1	4	14	8	13	5	14	11	12	14	6	4	9	1	2	7	5	-	14
20	12	3	5	14	13	3	4	14	7	11	11	8	6	13	5	14	10	6	14	-

The problem is tested using proposed and Moon’s algorithm. The values of GA parameters as given in Table 4.15 are based on trial and error. In a usual practice, crossover rate is set relatively high while the mutation rate is set exceptionally low.

Table 4.15: GA parameter settings for test problem 3

	pop_size	*ngener*	P_c	P_m
Proposed	150	100	0.9	0.01
Moon	150	100	0.75	0.01

The performance graph in Figure 4.21 demonstrates the best and the average transition time found by the algorithm in each generation. In both experiments, the best transition time reduced towards an optimal solution as the generation increased and finally converged at a certain generation.

The known optimal solution for this problem is 61 sec and the proposed algorithm is also able to produce the same result at generation 39 and this is better than Moon’s algorithm which is converged at generation 76. The number of generations produced using the Moon’s algorithm in this thesis is different from the results reported in [6]. This is mainly because of the different parameters used and different length of coding/step involved in the program developed by Moon *et al*.

The optimal tour obtained from both algorithms is [6-1-2-7-5-11-4-3-18-12-10-9-16-17-8-14-13-19-15-20] which is feasible and similar as reported in the paper. The results of the experiments are summarized in Table 4.16.

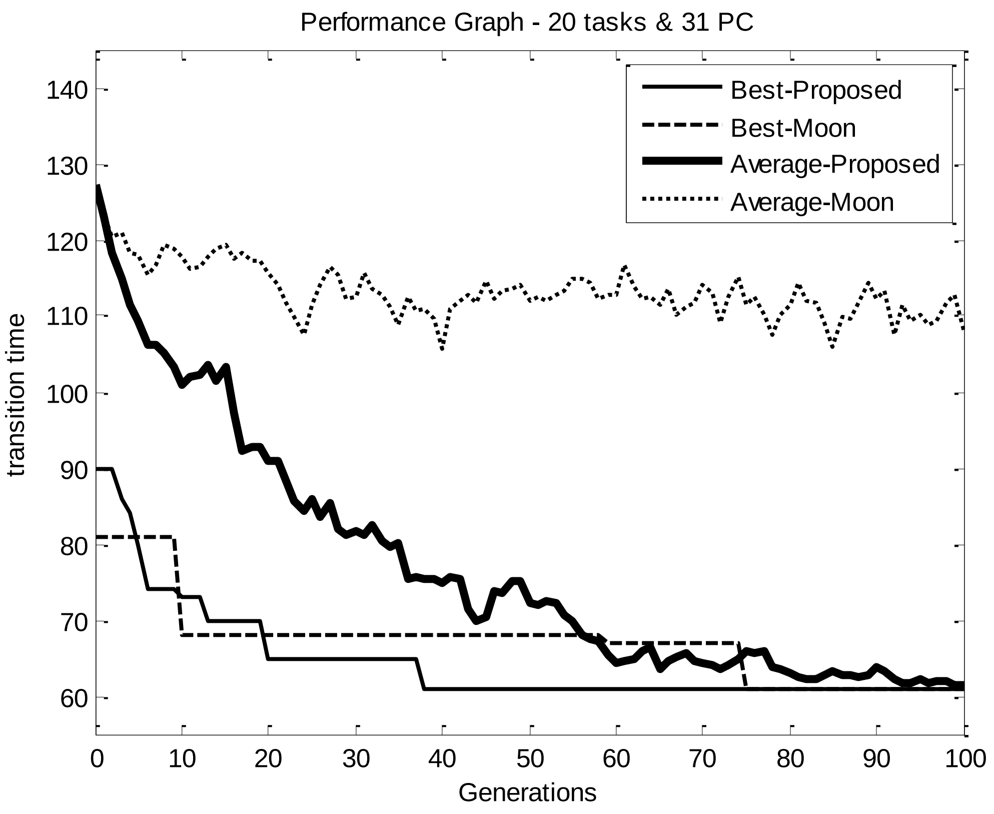


Figure 4.21: Performance graph for test problem 3

Table 4.16: Summary of results for test problem 3

	Gen#	Best (sec)	Convergence time (sec)	Completion time (sec)
Proposed	39	61	6.856	17.5
Moon	76	61	15.75	20.83

d) Test problem 4: 51 tasks & 71 precedence constraints

The test problem 4 consists of 51 tasks and 71 precedence constraints in which the location of the task and the transition time are randomly generated within [1, 15]. The precedence diagram of the problem is shown in Figure 4.22 while the transition time matrix is given in **APPENDIX A-5**. In order to assist the experimentation, full factorial DOE with 3 parameters each with 2 levels (2^3) is implemented. Therefore, the total number of computational runs will be 8 runs for one replication. Due to the large population size used in the experiment, each simulation experiment runs only one time in order to reduce computational time and resources.

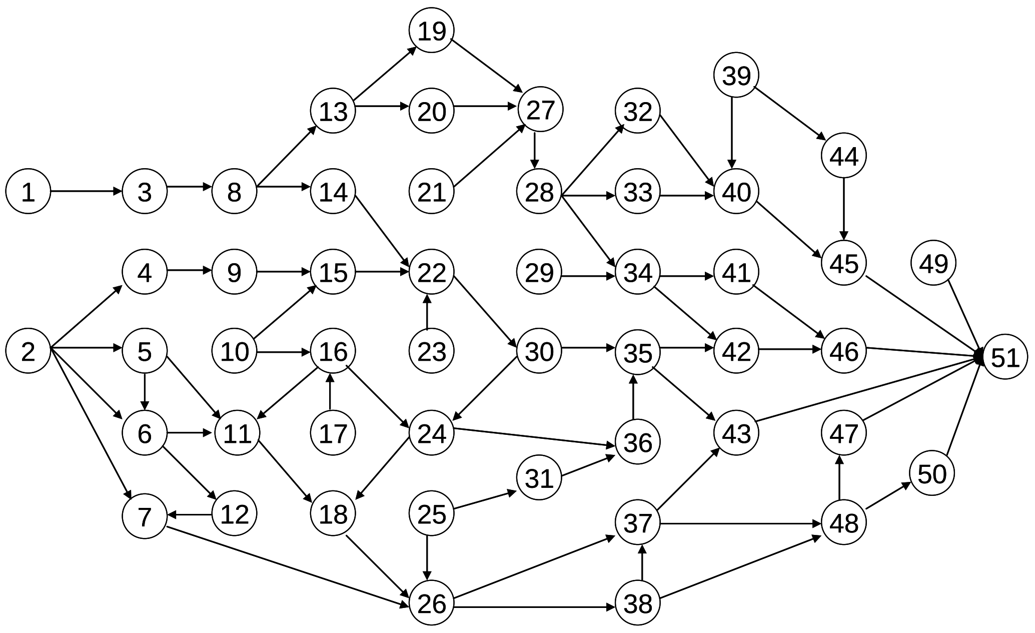


Figure 4.22: Precedence diagram for test problem 4

The experimental design and the range of the parameter's value considered for the proposed algorithm are shown in Table 4.17. The maximum number of generation is set to 200 for all the experiments. The DOE table along with the results obtained in each experiment is given in Table 4.18.

Table 4.17: GA parameters setting for proposed algorithm

Level	-1	+1
A (*pop_size*)	500	1000
B (P_c)	0.6	0.9
C (P_m)	0.001	0.2

Table 4.18: Results of experiment with proposed algorithm

Experiment	A	B	C	Gen#	best
1	-1	-1	-1	96	209
2	+1	-1	-1	192	204
3	-1	+1	-1	112	224
4	+1	+1	-1	157	194
5	-1	-1	+1	122	209
6	+1	-1	+1	74	218
7	-1	+1	+1	109	219
8	**+1**	**+1**	**+1**	**193**	**184**

From this experiment, it can be ascertained that the algorithm with a combination of large population size and high crossover rate as well as the high mutation rate had produced better outcomes. The final experiment (experiment number 8) offers the best sequence of tour with minimum transition time. The sequence of this tour is 2-4-39-17-9-1-3-44-10-5-8-13-14-15-20-16-19-21-23-22-25-31-27-28-33-32-30-49-6-11-40-45-24-18-36-12-7-26-38-37-48-50-47-29-34-41-35-42-46-43-51. This sequence confirmed that the chromosomes at the specified generation represent valid points in the search space, i.e. not violating the precedence constraints. The minimum total transition time is 184 sec and converged at generation 193. Thus, it can be expected that the quality of solution improves with the larger size of population and with a relatively high crossover and high mutation rate. The computation time to obtain the best solution is around 1358 sec (~ 22 minutes), which is still in an acceptable amount of time to spend.

Similar experiments were also carried out using the Moon's algorithm in order to investigate and compare the quality of solution and the performance of the algorithm with the proposed algorithm. The same parameters setting is applied as in the proposed algorithm (see Table 4.19).

Table 4.19: GA parameter settings for Moon's algorithm

Level	-1	+1
A (*pop_size*)	500	1000
B (P_c)	0.6	0.9
C (P_m)	0.001	0.2

Table 4.20: Results of experiment with Moon's algorithm

Experiment	A	B	C	Gen#	best
1	-1	-1	-1	150	268
2	**+1**	**-1**	**-1**	**163**	**256**
3	-1	+1	-1	83	277
4	+1	+1	-1	72	265
5	-1	-1	+1	101	272
6	+1	-1	+1	50	282
7	-1	+1	+1	51	279
8	+1	+1	+1	95	270

The results in Table 4.20 clearly show that the solution approaching minimum value with the utilization of high population size, low crossover rate and low mutation rate. The best solution found from the experiment is 256 and converged at generation 163. In order to check the feasibility of the solution from Moon's algorithm, the tasks sequences generated is recorded which is 1-10-2-4-9-15-49-5-6-12-23-3-8-14-22-30-13-19-20-17-21-27-28-33-32-29-34-41-25-31-16-11-24-18-36-35-42-46-39-40-44-45-7-26-38-37-48-50-43-47-51.

Figure 4.23 shows the performance graphs for 200 generations of 1000 chromosomes. The transition times are plotted against the number of generations for the two experiments, i.e. experiment number 8 for the proposed algorithm and experiment number 2 for the Moon's algorithm. It is observed that the 'Best' curves drop rapidly at the beginning of the run, but then as the population converges on the nearly optimal solution; it drops more slowly, and finally flattens at the end. The results of the computational experiments for both algorithms are summarized in Table 4.21. The iteration time for the proposed algorithm is larger than a Moon's algorithm. This is because a large number of generations are being utilized to converge on the best solution.

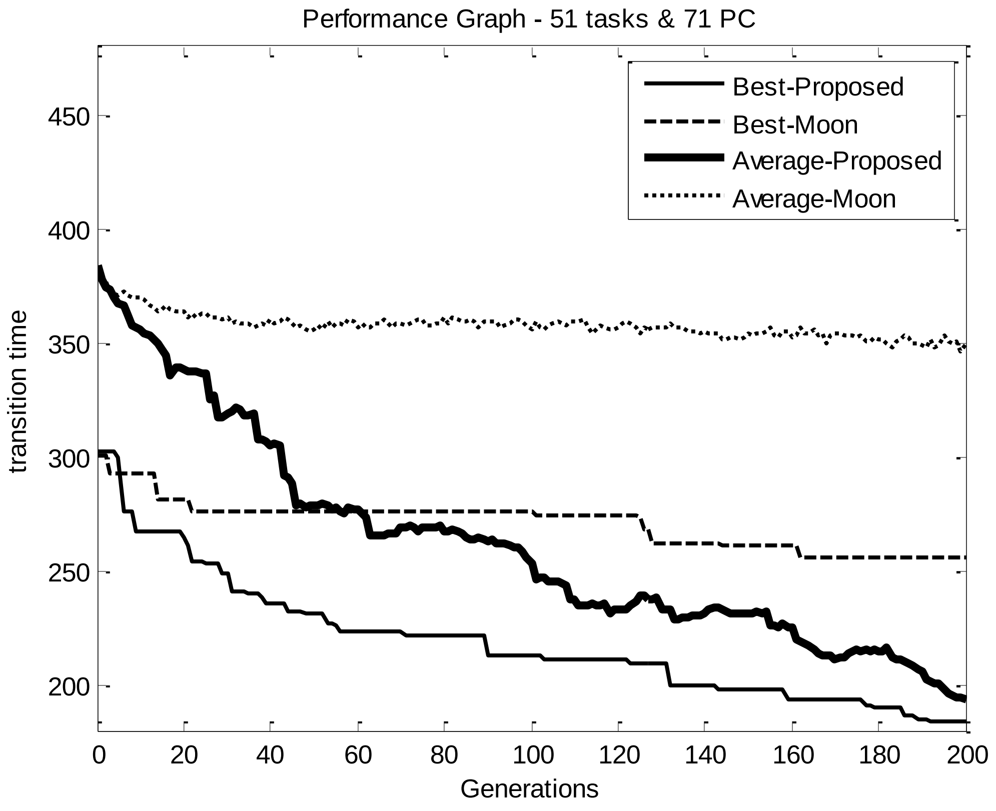


Figure 4.23: Performance graph for test problem 4

Table 4.21: Summary of results for test problem 4

	Gen#	Best	Convergence time (sec)	Completion time (sec)
Proposed	193	184	1358	1415
Moon	163	256	1109	1357

e) Test problem 5: Large-scale TSPPC

To examine the robustness and the stability of the proposed approach, the experiment is carried out on three large-scale TSPPC instances. They are 100 tasks with 141 precedence constraints, 200 tasks with 241 precedence constraints, and 500 tasks with 587 precedence constraints.

The precedence diagram for 100 tasks and 141 precedence constraints is depicted in Figure 4.24. In order to speed up the computation time, smaller population size and maximum number of generation are used for both experiments (Moon's and proposed algorithm). Note that the population size, maximum number of generation, the probability of crossover and probability of mutation are set similar for both algorithms. The parameter settings are as follows, *pop_size*=100, P_c=0.9, P_m=0.01 and *ngener*=400.

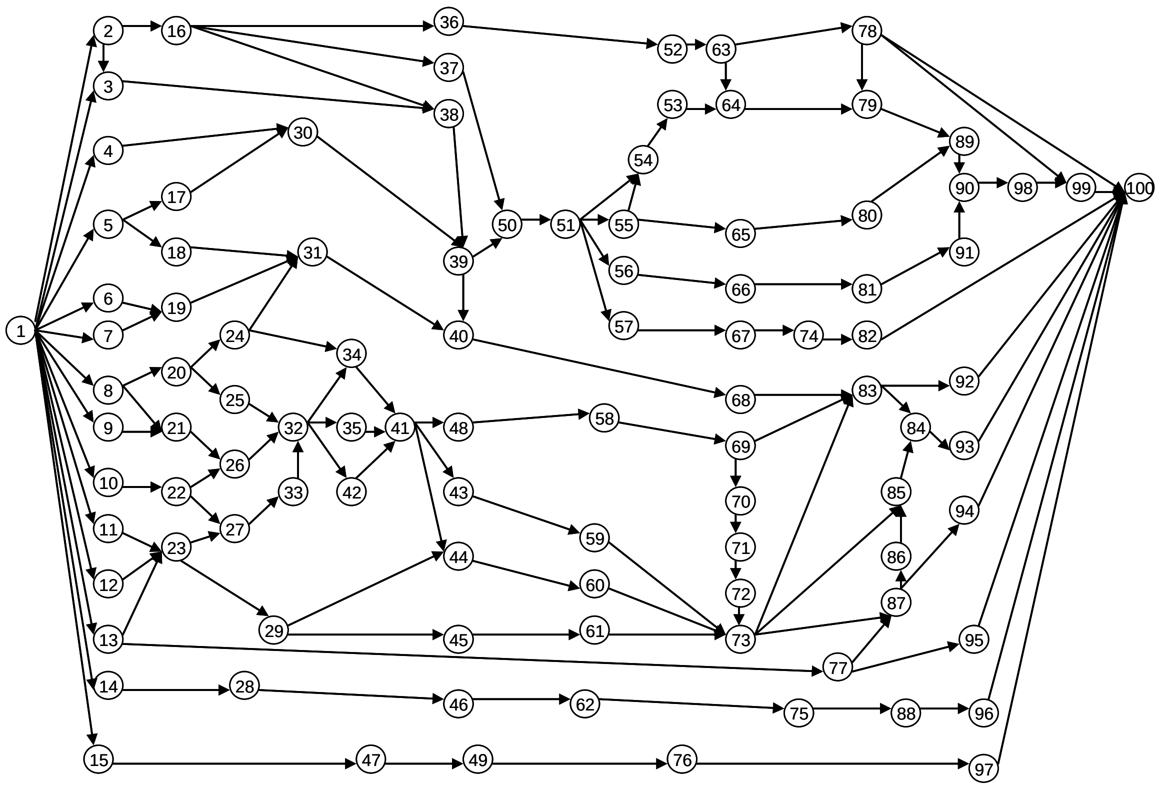


Figure 4.24: Precedence diagram for 100 tasks & 141 precedence constraints

The performance graph in Figure 4.25 demonstrates the best and the average transition time found by the algorithms in each generation. For the proposed algorithm, the best and average transition time steadily reduced over generations. The results of the experiments are summarized in Table 4.22.

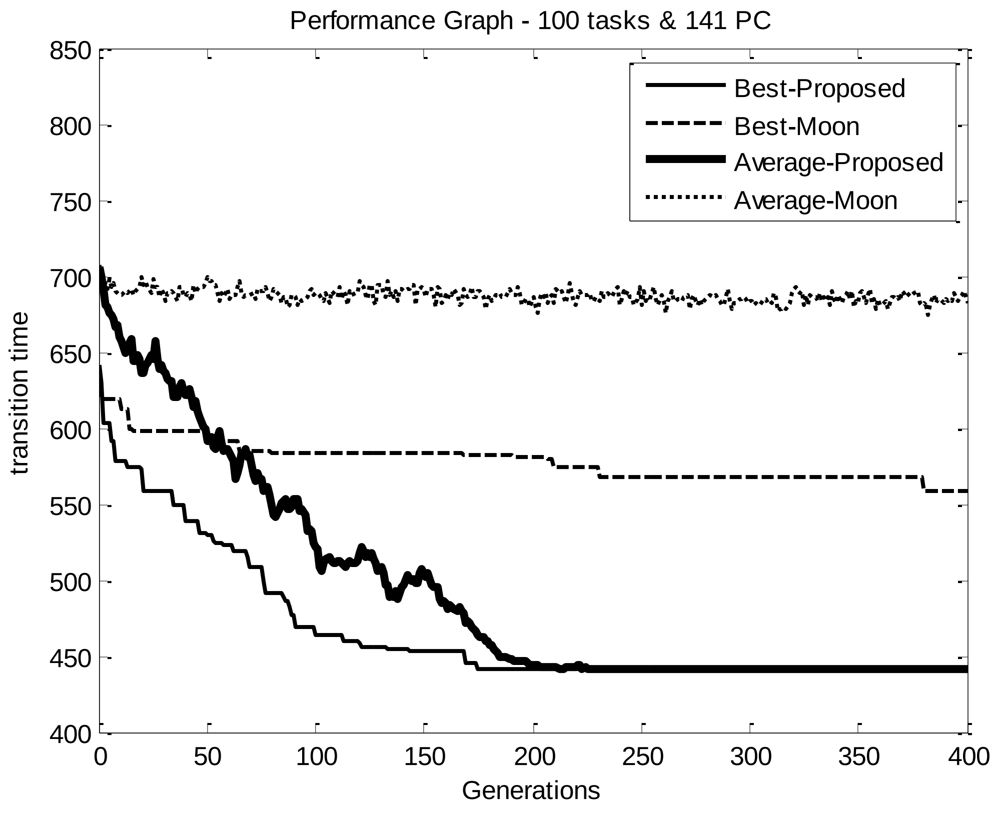


Figure 4.25: Performance graph for 100 tasks & 141 precedence constraints

Table 4.22: Summary of results for 100 tasks & 141 precedence constraints

	Gen#	Best	Convergence time (sec)	Completion time (sec)
Proposed	176	441	470.6	1058
Moon	381	559	950.6	1240

Figure 4.26 and Figure 4.27 show the performance graph for 200 tasks with 241 precedence constraints and 500 tasks with 587 constraints, respectively. The trends for both diagram are look similar in which the proposed algorithm always outperform Moon's algorithm both in finding the best (minimum) transition time and average transition time. The precedence diagram for 200 and 500 tasks problems are given in **APPENDIX A-6**.

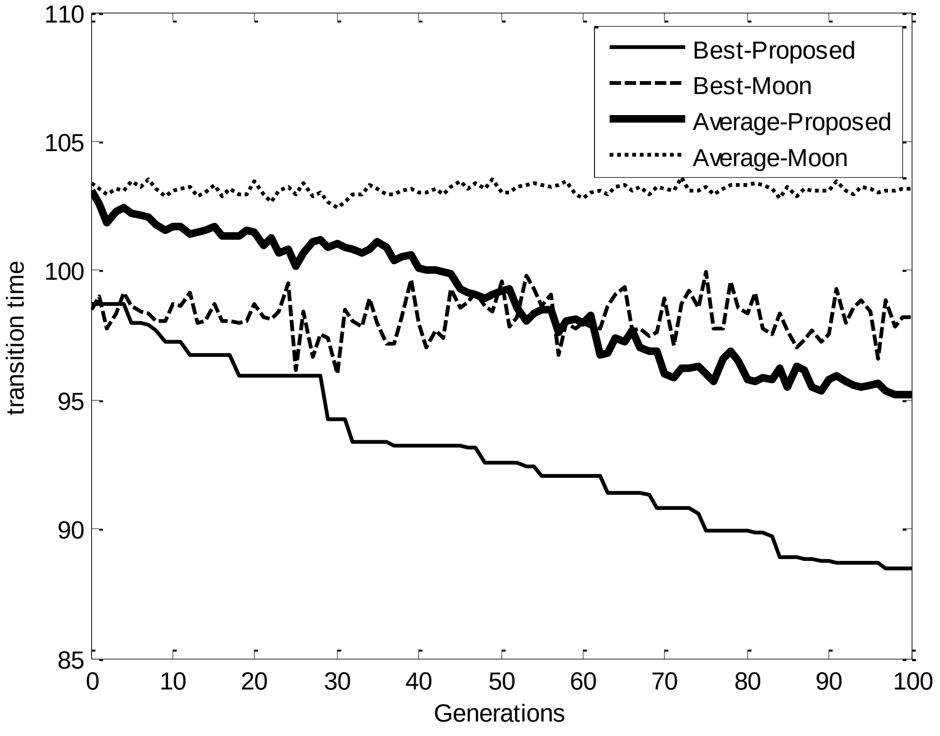


Figure 4.26: 200 tasks & 241 precedence constraints

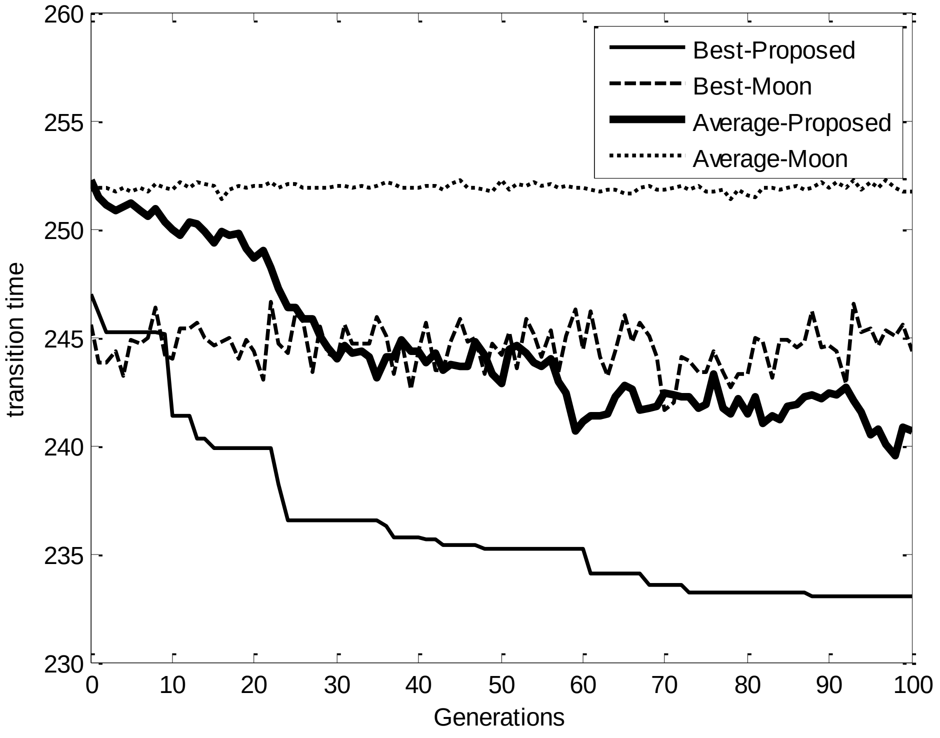


Figure 4.27: 500 tasks & 587 precedence constraints

4.4.2 TSPPC application examples

In this section three examples of process sequencing problem in assembly line are demonstrated and solved using the proposed TSPPC algorithm. The three examples are benchmarked from few published papers and have been simplified to reduce the complexity of the problem. The results of the process sequence and the optimal distance/cost/time obtain from the experiment are compared with the results reported in the published research papers which have used different optimisation approaches. The performance in terms of number of generations to attain optimal (best) solution is also compared with the Moon's algorithm.

a) Application example 1: Assembly sequence planning

The study of designing an intelligent robot assembly system has been gaining tremendous of attention especially in automotive manufacturing. One of the fundamental research issues concerning intelligent assembly is the assembly planning. The assembly planning problem is to generate the sequences for the assembly machines that transform the assembly operation to the desired product. Generally, the assembly process is to attach two components together with a formed sub-object. But due to the geometric constraint of the product structure, it happens that some components have to be done before other components can be performed. This problem is similar to the job shop, or scheduling problem with precedence constraints between tasks [45]. Based on the preceding knowledge obtained in the planning state and the cost or resources arrangement between tasks and robots, the assembly scheduler is to find the best ordering of the assembly task.

The objective of this example is to find the optimal ordering of gear box assembly which will minimize the total assembly cost. An exploded view of nine parts gearbox assembly and the precedence knowledge is given in Figure 4.28 and Figure 4.29, respectively while the assembly cost between two liaisons is given in Table 4.23. The GA parameters for this problem are set as follows; *pop_size*=10, P_c=0.9, P_m=0.01 and *ngener*=20.

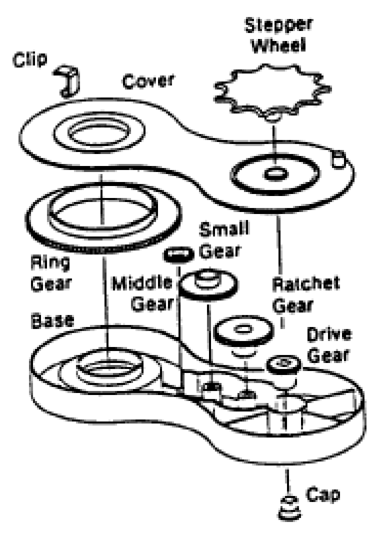
No	Task Name	
1	Cap & drive gear	 Clip Cover Stepper Wheel Ring Gear Small Gear Middle Gear Base Ratchet Gear Drive Gear Cap
2	Stepper wheel & cover	
3	Cover & base	
4	Clip & base	
5	Ring gear & base	
6	Small gear & base	
7	Middle gear & base	
8	Ratchet gear & base	
9	Drive gear & base	

Figure 4.28: Assembly parts for gearbox [45]

$$\begin{tikzcd}
6 \arrow[dr] & & 5 \arrow[dr] & & 1 \\
& 2 \arrow[ur] \arrow[r] \arrow[dr] & 8 \arrow[r] & 3 \arrow[ur] \arrow[dr] & \\
7 \arrow[ur] & & 9 \arrow[ur] & & 4
\end{tikzcd}$$

Figure 4.29: Precedence diagram for gearbox product

Table 4.23: Cost matrix for gearbox assembly

Cost	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	-	0.788	0.363	0.265	0.521	0.788	0.100	0.458	0.682
2	0.788	-	0.458	0.228	0.168	0.265	0.682	0.228	0.100
3	0.363	0.458	-	0.324	0.168	0.160	0.788	0.363	0.245
4	0.265	0.228	0.324	-	0.394	0.363	0.458	0.510	0.228
5	0.521	0.168	0.168	0.394	-	0.521	0.394	0.168	0.324
6	0.788	0.265	0.160	0.363	0.521	-	0.100	0.458	0.682
7	0.100	0.682	0.788	0.458	0.394	0.100	-	0.228	0.100
8	0.458	0.228	0.363	0.510	0.168	0.458	0.228	-	0.265
9	0.682	0.100	0.245	0.228	0.324	0.362	0.100	0.265	-

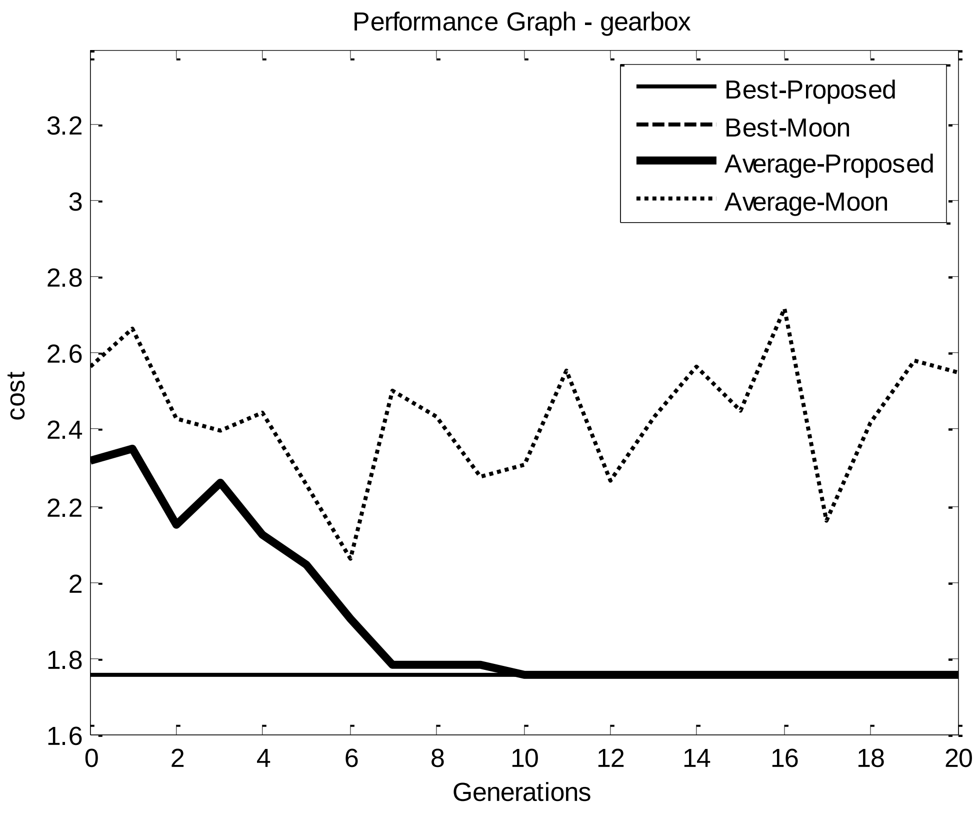


Figure 4.30: Performance graph for Application Example 1

The performance graph in Figure 4.30 shows that the GA has converged at generation 7 with minimum assembly cost of $1.755. The assembly sequence generated by the GA is [7 6 2 9 8 5 3 4 1]. These results are similar to the results reported in [45]. The summary of results obtained from the proposed and Moon's algorithm are given in Table 4.24.

Table 4. 24: Summary of results for test problem 4

	Gen#	Best	Convergence time (sec)	Completion time (sec)
Proposed	1	1.755	0.1939	0.3087
Moon	1	1.755	0.276	0.3874

b) Application example 2: Disassembly sequence planning

Generally, once electronics products (e.g. refrigerators, washing machines, coffee machines, computers, printers, copiers, telephones, TV sets) reach their end-of-life (EOL), they are sent to one of the EOL processes (i.e. remanufacturing, reuse and recycling) [42]. In order to perform remanufacturing, reusing and recycling, components and materials from EOL products must be obtained through the disassembly process. Disassembly is basically the process of systematic removal of the desired components or materials from the original assembly so that the components or materials are obtained in the desired form. Therefore, the disassembly sequence provides the order in which the components are disassembled. To ensure the maximum efficiency of the disassembly process, the disassembly sequence must be generated in such a way that the precedence relationship among the components is maintained and the disassembly complexity and time are minimised.

The assembly sequence planning models and the disassembly sequence planning models are performed independently as two tasks without interaction. An assembly operation with a low cost may not correspond to a disassembly operation with the same level of low cost. For example, in order to achieve a low cost, two components may be assembled with a welding operation because of a shorter time and lower cost. They may be assembled with a screw with higher cost. However, in the disassembly operations, disassembling two welded components can cost more, but disassembling two screw-fixed components usually costs less. As a result, an assembly sequence with low costs may result in a disassembly sequence with high costs for the same product [44].

Since assembly and disassembly operations are not necessarily reversible, there may be two different graphs representing the precedence relations for the same product, one corresponding to assembly and the other to disassembly operations. Usually, the minimum operation costs of disassembly sequence are used as an objective function. The cost items might include the disassembly operation cost to complete the disassembly operations, disassembly instability cost for maintaining the stability of the disassembled components, the accessibility cost to complete the disassembly operations, disassembly tool setup cost, and the cost for moving and handling needs for disassembly operations [44].

In this example, the disassembly process of hand phone product is benchmarked from [44]. The GA parameters used in the study are as follows; *pop_size*=50, P_c=0.9, P_m=0.1 and *ngener*=50. Figure 4.31 and Figure 4.32 illustrate the assembly components of the product and the precedence knowledge of the product, respectively. The cost to disassemble from one component to another is not supplied in the paper and therefore it is generated randomly within [1, 10] as depicted in Table 4.25.

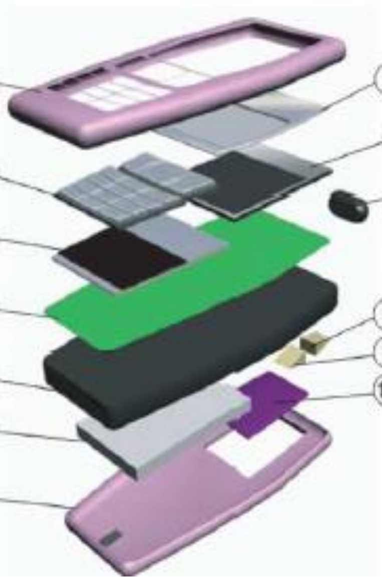
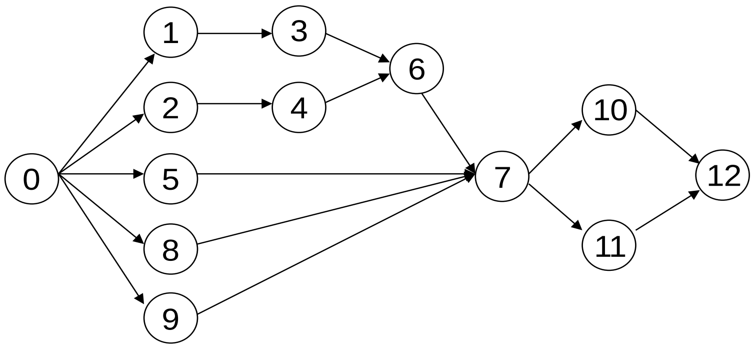
No	Component Name	
0	Upper case	 0 1 2 3 4 5 6 7 8 9 10 11 12
1	Screen cover	
2	Keyboard	
3	LCD panel	
4	Keyboard PCB	
5	Upper button	
6	Main PCB	
7	Structure frame	
8	Right button (1)	
9	Right button (2)	
10	Brand name label	
11	Battery container	
12	Bottom case	

Figure 4.31: Disassembly components for hand phone product [44]



$$\begin{tikzcd}
 & 1 \arrow[r] & 3 \arrow[dr] & & & \\
 & & & 6 \arrow[dd] & & \\
 & 2 \arrow[r] & 4 \arrow[ur] & & 10 \arrow[dr] & \\
0 \arrow[uuur] \arrow[ur] \arrow[r] \arrow[dr] \arrow[ddr] & 5 \arrow[rr] & & 7 \arrow[ur] \arrow[dr] & & 12 \\
 & 8 \arrow[urr] & & & 11 \arrow[ur] & \\
 & 9 \arrow[uurr] & & & &
\end{tikzcd}$$

Figure 4.32: Disassembly precedence diagram for hand phone product

Table 4.25: Disassembly cost matrix for hand phone product

Cost	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0	0	6.65	6.11	9.17	3.45	5.11	6.76	1.01	1.16	5.05	4.50	7.05	6.38
1	8.92	0	7.70	2.90	8.52	4.36	9.34	8.18	4.69	1.96	1.94	1.04	8.82
2	3.52	7.03	0	4.03	4.08	3.34	3.54	9.55	2.10	4.34	2.08	7.80	8.71
3	2.94	9.89	5.84	0	5.86	7.31	6.86	7.52	5.49	5.59	3.55	6.03	3.41
4	4.60	9.77	8.32	8.94	0	9.68	6.51	4.64	2.76	5.21	5.40	3.02	1.77
5	1.67	3.45	7.11	9.97	2.36	0	1.84	1.97	1.36	1.72	2.10	5.06	8.45
6	4.41	1.75	6.19	9.74	9.42	9.71	0	8.85	4.52	3.83	5.27	9.91	5.89
7	5.84	4.03	5.48	7.09	7.55	1.87	7.89	0	3.72	5.82	5.65	3.19	1.36
8	7.76	1.56	9.41	4.41	5.54	7.73	2.25	5.65	0	5.46	1.87	9.10	3.68
9	6.70	1.35	5.53	2.14	9.28	3.80	5.97	2.09	1.01	0	7.27	8.61	2.26
10	2.43	5.53	8.38	5.70	1.30	4.42	2.13	1.80	5.56	3.87	0	3.26	3.48
11	2.89	2.02	6.57	9.50	2.37	6.44	9.10	3.79	4.76	1.07	4.55	0	7.47
12	6.38	8.82	8.71	3.41	1.77	8.45	5.89	1.36	3.68	2.26	3.48	7.47	0

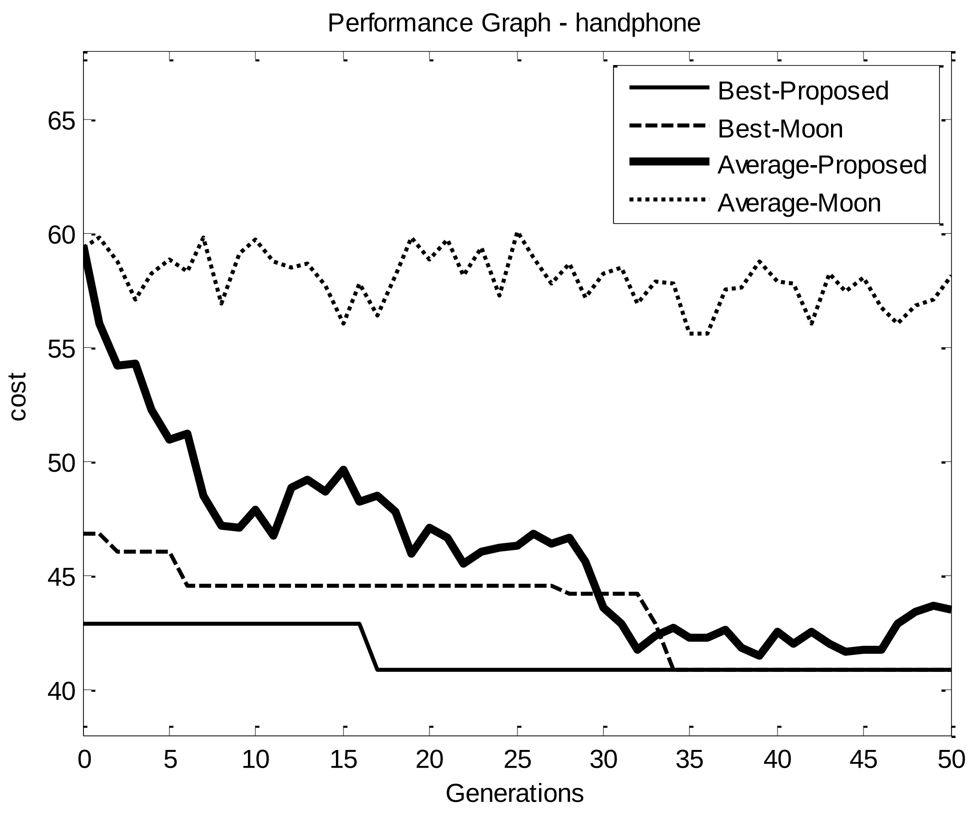


Figure 4.33: Performance graph for Application Example 2

The computational results show that the proposed GA is able to generate feasible sequence without violating precedence constraints. The generated sequence obtained by the GA is [0 8 1 3 2 5 4 6 9 7 11 10 12] with minimum cost of $40.81. The trend of the best and the average curve in Figure 4.33 looks normal and it is implied that the genetic operators used in the proposed algorithm are effective. The summary of the results obtained from the experiments are summarized in Table 4.26.

Table 4.26: Summary of results for test problem 4

	Gen#	Best	Convergence time (sec)	Completion time (sec)
Proposed	17	40.81	0.7472	1.704
Moon	34	40.81	1.718	2.329

c) Application example 3: Assembly line balancing

In the third example, a simple assembly line balancing (ALB) problem that uses the sequencing concept is presented. The purpose of performing task sequencing in this assembly line is to find the optimal sequence that comes out with minimum idle time. There are costs due to the idle times which represent unused capacities of workers and machines. An assembly line is a sequence of workstations, connected together by a material handling system, which is used to assemble components into a final product [14]. The assembly process consists of a sequence of tasks or work elements. The tasks in an assembly process are typically ordered. The assembly line balancing problem is to assign a set of tasks to workstation with some measure of performance to be optimized under the following restrictions: (i) each task is assigned to one and only one workstation, (ii) the precedence relationship among the tasks cannot be violated, and (iii) the sum of the task times of any workstation should not exceed the cycle time [120]. The purpose of balancing the line is to prevent the occurrence of bottlenecks in the final production line which will result in a stoppage and unnecessary accumulation of inventory [13].

Since the task times allotted to workstations may be unequal, parts are produced at different speeds on the line. Accordingly, stations may either be starved or a queue may build up in front of a station. To regulate the flow of parts, assembly lines are often paced. In a paced line, each workstation is given a fixed amount of time called cycle time. The cycle time of an assembly line is predetermined by a desired production rate. Such production rate is set so that the desired amount of the end product is produced within a certain time period. Material handling systems are designed so that after every certain cycle time, the system indexes, advancing the part to the next station. If a workstation finishes in less than cycle time given, it is idle for the remaining period. The difference between the time required by any station to complete its operations and the cycle time is called the idle time of the station. It is conventional to take the sum of all station idle times, i.e. total idle time, as a measure of the efficiency of the design of a line.

The ALB problem is usually presented by the precedence graph. Consider a precedence graph in Figure 4.32 which specifies the order or sequence in which the task must be performed. The number in each circle refers the task number, and the number above the circle refers the duration of the operation (task). The arrow represents directions of flow of operation. The variable of interest for the ALB consists of a number of tasks, processing time, precedence relationships, and the cycle time (*CT*). Several goals can be achieved such as to minimize the number of workstations (*m*), minimize the idle time (T_{id}), and maximize the line efficiency, *E*. Formulations of idle time and line efficiency are given in (4-1) and (4-2) respectively, where T_i is the processing/task time of the *i*th workstation.

$$T_{id} = \sum_{i=1}^{m} (CT - T_i) \tag{4-1}$$

$$E = \sum_{i=1}^{m} T_i / (mCT) \tag{4-2}$$

For ALB problem, the objective function is different from TSP, and it is not classified as TSP. However the proposed GA procedure can be used to generate the feasible sequence of the task and the objective function is developed to obtain the minimum idle time in the station.

The example presents in this thesis use a benchmark data sets from Scholl [121] called Gunther problem which consists of 35 tasks and 45 precedence constraints as depicted in Figure 4.34. The total task time and the predetermined cycle time of the Gunther problem are 483 min and 60 min, respectively. The GA parameters setting used in the experiment are as follows; *pop_size* = 20, P_c = 0.9, P_m = 0.1, and *ngener* = 50. The coding of the objective function is supplied in **APPENDIX A-7**. The results in terms of the number of stations, total idle time, and the line efficiency are presented in Table 4.27. These results are compared with the results obtained by S. Suwannarongsri and D. Puangdownreong [122] which used Tabu search to simulate the same problem.

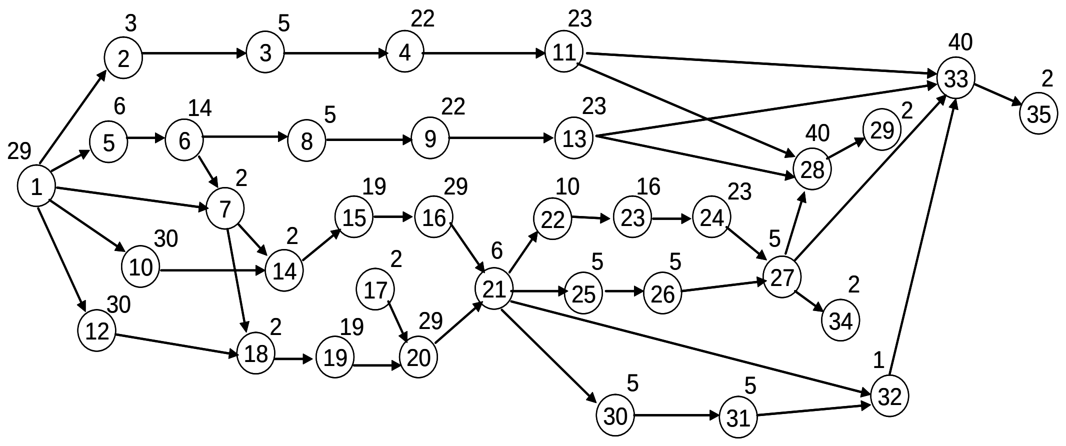


Figure 4.34: Precedence diagram for Gunther problem [122]

Table 4.27: Optimal solution for Gunther problem

Optimal task sequence: 1,5,6,2,3,7,4,11,8,12,18,19,17,10,14,15,16,9,13,20,21, 22,30,31,23,24,25,26,27,28,34,29,32,33,35			
Station	**Assigned Task**	**Processing Time (min)**	**Idle Time (min)**
1	1,5,6,2,3,7	59	1
2	4,11,8	50	10
3	12,18,19,17	53	7
4	10,14,15	51	9
5	16,9	51	9
6	13,20,21	58	2
7	22,30,31,23,24	59	1
8	25,26,27,28,34,29,32	60	0
9	33,35	42	12
		Total Idle Time, T_{id}	**57**

Based on equation (4-1), the total idle time, T_{id} = (60-59) + (60-50) + (60-53) + (60-51) + (60-51) + (60-58) + (60-59) + (60-60) + (60-42) = 57 minutes

The optimal number of workstations (m) obtained from GA are 9 stations, and the total processing time (T_i) is 483 minutes. Based from these data, the line efficiency, E can be obtained using equation (4-2).

The line efficiency, E= (483/9*60) =89.44%

Table 4.28: Summary of results for gunther problem

	Suwannarongsri & Puangdownreong	Proposed method
Total idle time	117 min	57 min
No. of station	10	9
Line efficiency	80.5%	89.44%

Table 4.28 shows the results comparison between the proposed GA used in this example and the results obtained from [122]. The utilization of 9 workstations gives a great reduction in idle time as well as improving the line efficiency. The idle time reduces 51.28% and line efficiency improves about 10% of the results reported from [122]. Therefore the proposed GA procedure use in this study outperforms the method used from the previous work. The proposed GA well address the number of tasks assigned for each workstation giving a minimum idle time in the workstation as well as minimizes the number of stations for a given cycle time. The result of such solution would be increased production efficiency. The performance of the proposed algorithm to come out with the minimum idle time is also compared with the Moon's algorithm. The performance graph for both algorithms is depicted in Figure 4.35.

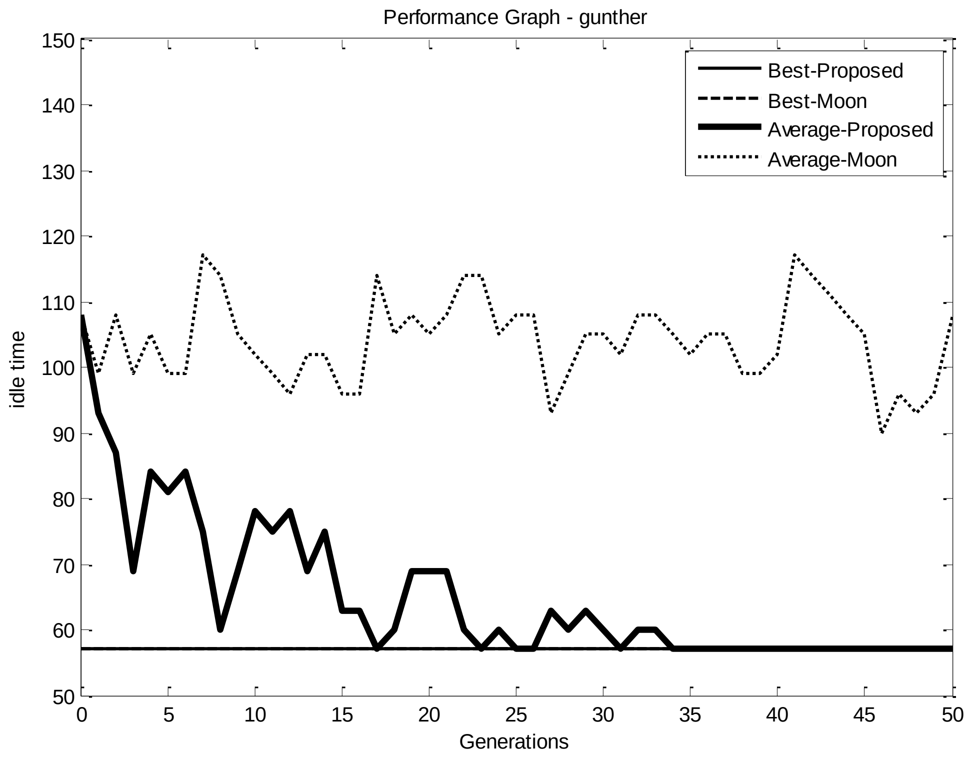


Figure 4.35: Performance graph for Application Example 3

Chapter 5

Discussions

In this chapter the results of the experiments as illustrated in the previous chapter are discussed. The discussion is divided into two sections. The first section discusses the results of TSP experiments while the second section discusses the results of TSPPC experiments.

5.1 TSP Experiments

The purpose of TSP experiments is to evaluate and to test the effectiveness of the GA procedure that uses different combination of genetic operators described in the methodology. This procedure is further modified to solve TSP with precedence constraints (TSPPC). All procedures developed for TSP in chapter 3 maintained the crossover and mutation techniques. The only difference in the procedure is the selection strategy used for parent selection.

To indicate the quality of the returned solution, the *relative error* is calculated based on equation (2-14). The percentage of *relative error* of the best solution obtained with respect to the known optimal solution is presented in Table 5.1. The results from this table demonstrates that GA with rank-based Roulette Wheel selection (i.e. RBOX algorithm) is superior than that of a tournament and proportional Roulette Wheel where the results of RBOX algorithm do not give any error (0%) from the optimal solution for the three instances: burma14, bay29, and dantzig42, and less than 5% for eil51. TSOX algorithm apparently gives better results than PROX algorithm for larger size instances.

Table 5.1: Percentage of relative error for all TSP instances

Problem	PROX (%)	RBOX (%)	TSOX (%)
burma14	0	0	0
bay29	0.003	0	0.003
dantzig42	9.57	0	2.36
eil51	16	4.47	12

The performance graph in Figure 5.1 through Figure 5.4 shows the minimum distance found by the algorithm in each generation. As can be seen from the graphs, the distance reduced towards an optimal solution as the generation increased and finally converged at a certain generation. For instance in dantzig42, it shows that the algorithm with TSOX and PROX algorithm converged at generation 82 and 195 respectively, where there is no more improvement made after this generation. On the other hand RBOX algorithm is able to reach an optimal solution without premature convergence. Although with slower convergence, RBOX performs highly competitive in terms of solution quality, achieving minimum travelling distance.

The graph in Figure 5.5 compares the iteration time between three different procedures. Obviously, RBOX algorithm consumes the largest iteration time, hence high computation time due to a large number of generations involved to complete the evolution process. The iteration time for TSOX algorithm is somewhat better than PROX algorithm in producing comparable results of minimum travelling distance. This indicates that in general tournament selection is superior to proportional Roulette Wheel selection in achieving a good quality solution with less computation time.

It has been stated in the literature that individuals are selected for reproduction on the basis of their fitness, i.e., the fittest individuals have a higher likelihood of reproducing [71]. Selection determines which individuals will reproduce. The results of the experiments confirmed that the selection method can have an important impact on the effectiveness of a GA.

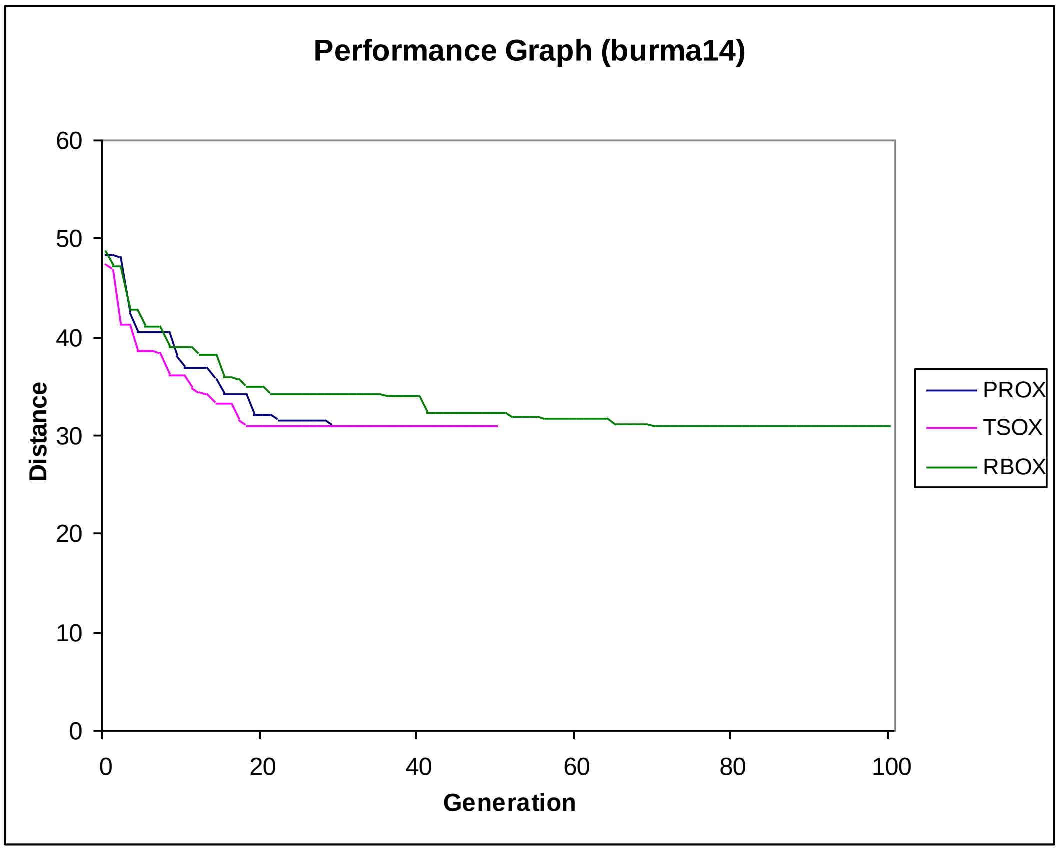


Figure 5.1: Performance comparisons between PROX, RBOX and TSOX for burma14

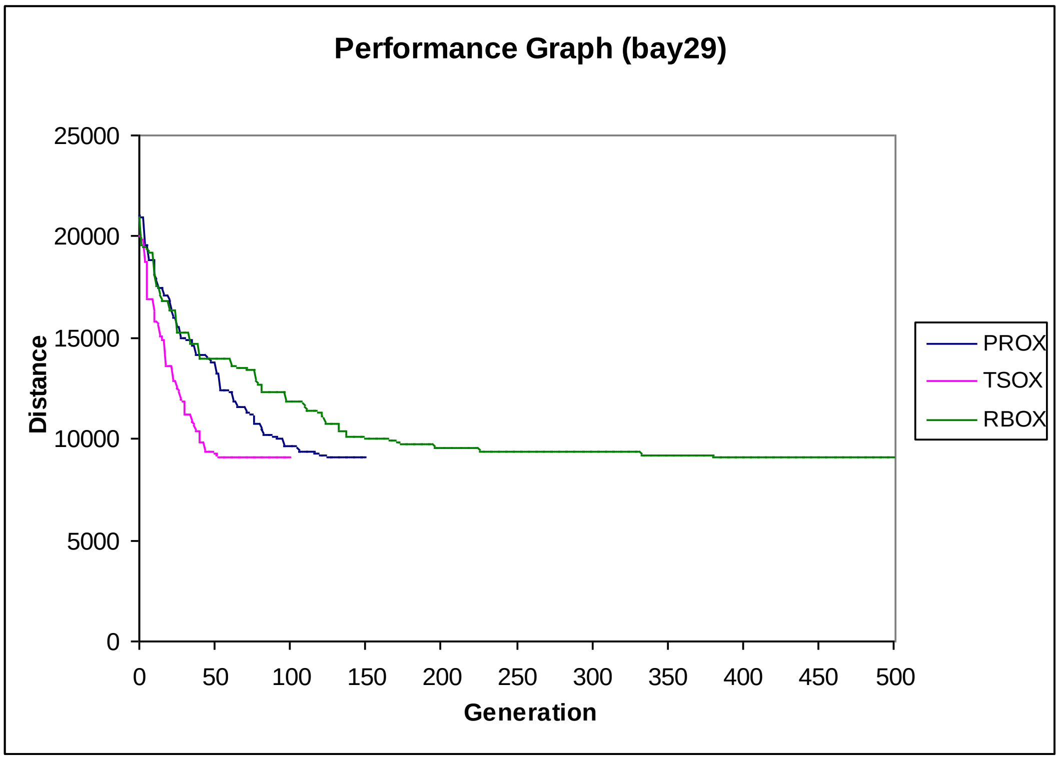


Figure 5.2: Performance comparison between PROX, RBOX and TSOX for bay29

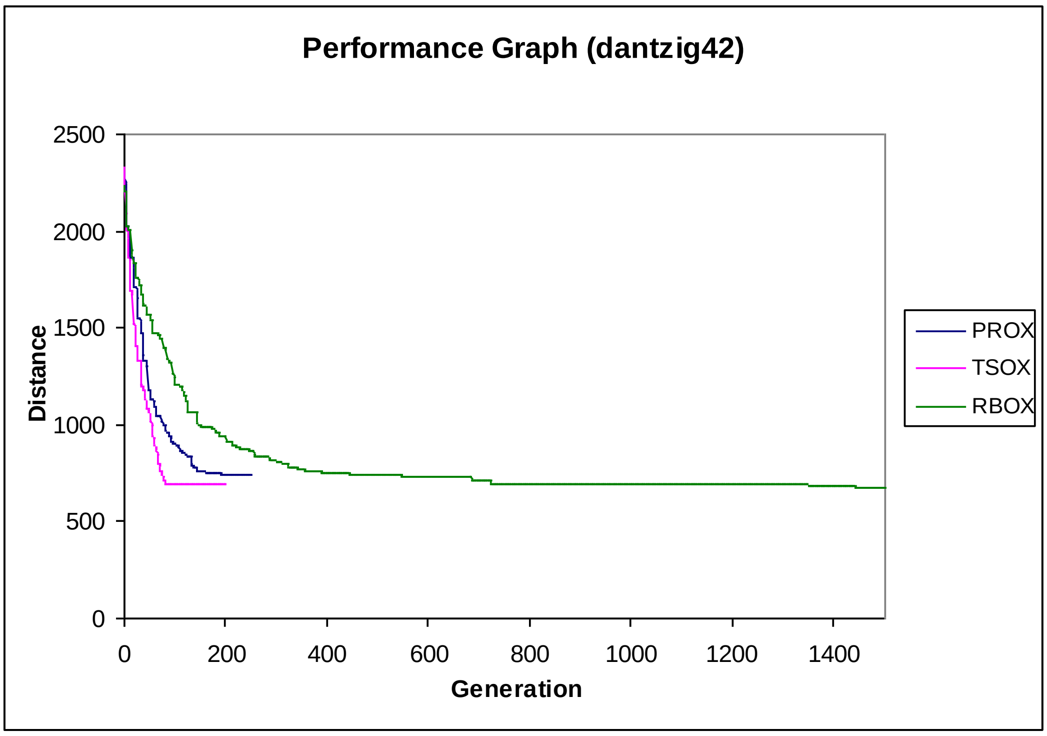


Figure 5.3: Performance comparisons between PROX, RBOX and TSOX for dantzig42

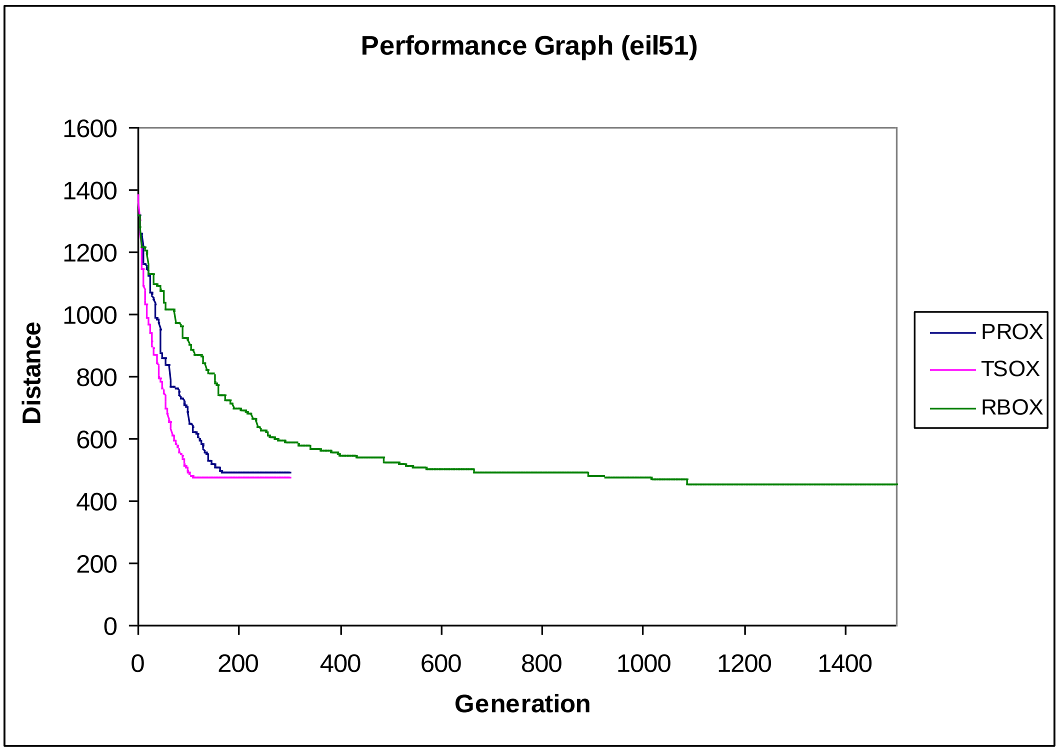


Figure 5.4: Performance comparisons between PROX, RBOX and TSOX for eil51

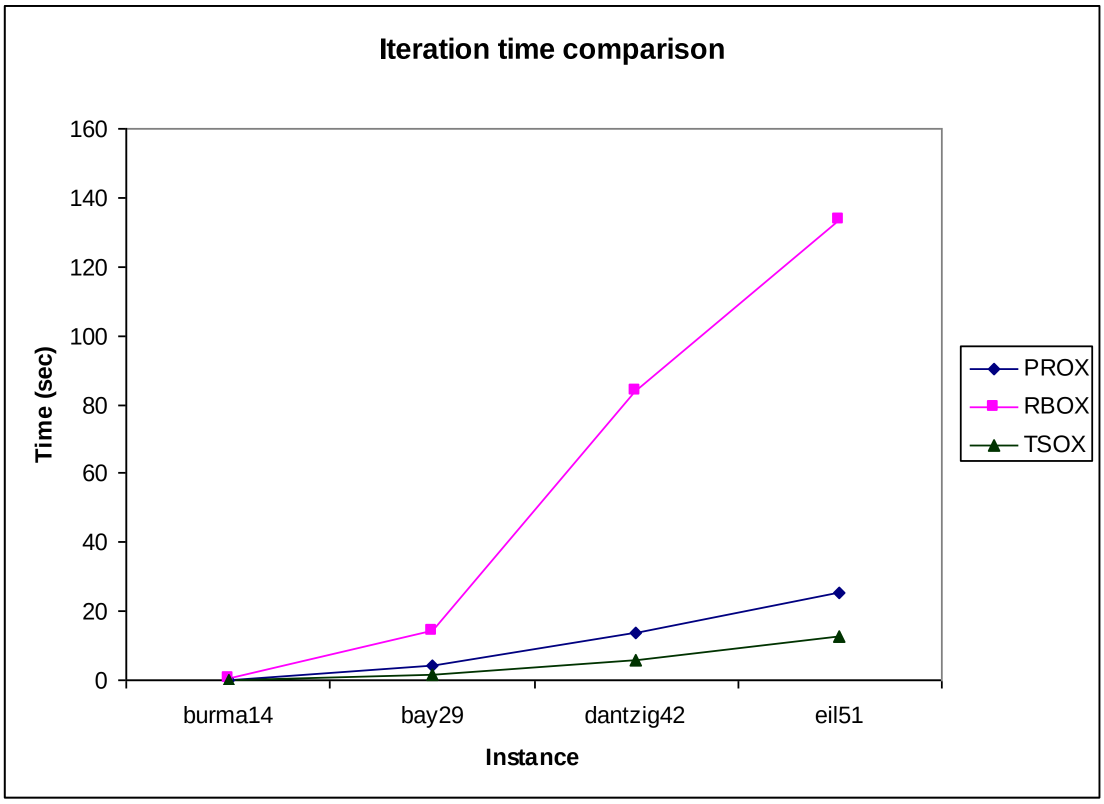


Figure 5.5: Iteration time comparisons between PROX, RBOX and TSOX algorithm

5.2 TSPPC Experiments

The computational experiment for TSPPC was divided into two parts. In the first part of the experiment, three test problems with known optimal solution (test problem 1, test problem 2 and test problem 3) and one randomly generated test problem (test problem 4) were run at two different algorithms, i.e. proposed algorithm and Moon's algorithm. The test problem with larger size instances (test problem 5) were tested in order to observe the behaviour of both algorithms. In the second part of the experiment, three simple application examples benchmarked from the published papers were tested to further demonstrate the effectiveness of the proposed algorithm. The proposed and the Moon's algorithm were compared and the results were monitored in terms of optimal solution (denoted by 'Best'), the number of generations to come out with the optimal solution (denoted by 'Gen#'), the convergence time and the feasible sequence of the optimal solution.

Table 5.2: Summary of results for all TSPPC test problems

Test Problem	Proposed			Moon		
	Gen#	Best	Convergence time (sec)	Gen#	Best	Convergence time (sec)
6 tasks & 6 PC	1	39	0.1786	1	39	0.2226
8 tasks & 9 PC	2	21.25	0.211	13	21.25	0.3319
20 tasks & 31 PC	39	61	6.856	76	61	15.75
51 tasks & 71 PC	193	184	1358	163	256	1109
100 tasks & 141 PC	174	441	1058	381	559	1240
Application ex. 1	1	1.755	1.1939	1	1.755	0.276
Application ex. 2	17	40.81	0.7472	34	40.81	1.718
Application ex. 3	1	57	0.501	1	57	1.950

The overall results of the experiment are summarized in Table 5.2. For test problem with known optimal solution, the results have shown that both algorithms were capable to achieve similar quality of solution as reported in the published papers. However the proposed algorithm has better results in terms of number of generations and computation time to converge on the optimal solution. For larger size problem (i.e. 51 tasks & 71 PC, and 100 tasks & 141 PC), the proposed algorithm had produced a better quality of the solution compared to a Moon's algorithm. The proposed algorithm was capable to generate optimal solutions with less number of generations and less convergence time for all test problems. For instance, the convergence time improved 130% by using the proposed algorithm for test problem 3 (i.e. 20 tasks & 31 PC). This indicates that the proposed algorithm performs highly competitive in terms of solution quality and has better efficiency compared to a Moon's algorithm.

One of the factors that contribute to better efficiency is because the proposed algorithm used sequence of task as a chromosome. When the sequence of task as chromosome is straight away used, the changes in chromosome will directly affect to the similar element in the sequence of task. Therefore, the changes of sequence will directly follow the GA towards a better solution.

The iteration time for the Moon's algorithm to complete the evolution process (i.e. completion time) is much higher than the proposed algorithm. This result is associated with the use of a priority factor in the Moon’s algorithm. It requires longer iteration time to complete one generation because it needs to compare the highest priority for all available tasks before selecting them to be placed in sequence. It is believed that the ‘earliest position’ based task assignment technique that's used in the proposed algorithm can substantially improve the efficiency of the algorithm, particularly for large problem instances. The ‘earliest position’ based task assignment technique eliminates the procedure of comparing the priority with each of the tasks in the chromosome.

For test problem 4 (51 tasks & 71 PC), the proposed algorithm had produced very encouraging result in terms of the quality of solution. Although slow convergence which requires longer computation time compared to a Moon’s algorithm, the quality of the solution is still promising. For larger problems, it may not be able to achieve the optimal solution, however near optimal solution with feasible tour is guaranteed. There is always a trade-off between computation time and the quality of solution. If quality of the solution is the main concern and computation time is still negotiable, then rank-based selection strategy may be used to replace the roulette wheel selection.

It is also observed that the ‘Average’ curve for Moon seems to be stagnant which was apparently being seen in all test problems. This indicates that the crossover technique used in the procedure was not capable to introduce the new fittest offspring and therefore the search space contains almost identical individuals which have the same characteristics as their parents. Moon’s algorithm showed a very slow progress and finally fails to converge on the best/optimal solution.

It can be said that the genetic operators used in the proposed approach are having the characteristics to exploit and explore. For all the experiments done, the progress of the curve in the performance graphs indicates that the combinations of linear order crossover and inversion mutation are able to preserve good chromosomes and add new chromosomes in the population. This is because in linear order crossover, the absolute positions of some elements of the first parent are retained and the relative positions of some elements of the second parent are also kept. This in turns will transfer the good characteristics of the parents to their offspring. The use of inversion mutation on the other hand provides sufficient variance in fitness across the population to drive further evolution. Based on analysis and survey, it is suggested that a GA for TSPPC should utilise the linear order crossover as this is simple to implement which is also the most appropriate and commonly crossover technique applied for TSP.

The application examples presented in the last part of chapter 4 verified that the proposed algorithm was a stable and robust approach to solve TSPPC problem. The results of the experiments demonstrate that the genetic algorithm approach combined with a topological sort procedure can be applied for solving assembly and disassembly sequence as well as Assembly line balancing problems. The role of topological sort was to generate feasible task sequence while the GA was to further improve the quality of the solution. The computational results revealed that the proposed approach is superior to the Moon's approach in both quality of the solution and computation time. It is believed that a GA with these claimed results will become a robust tool for TSPPC and potential applications in manufacturing industry.

5.2.1 Individual Fitness diversity

In order to measure the diversity of the individual fitness in the population, the standard deviation is calculated and plotted. Figure 5.6 and Figure 5.7 shows the standard deviation versus the number of generations for test problem 3 (20 tasks & 31 PC) and test problem 4 (51 tasks & 71 PC), respectively. The trends of the curves for both test problems are almost similar. The standard deviation for Moon's algorithm does not decrease and looks slightly increase towards the generations. This indicates that the population diversity is maintained, however the algorithm does not introduce new fitter individuals in the population. On the other hand the standard deviation for the proposed algorithm steadily decreases towards the generations. This implies that the population diversity is reduced and the population contains a large number of fitter individuals.

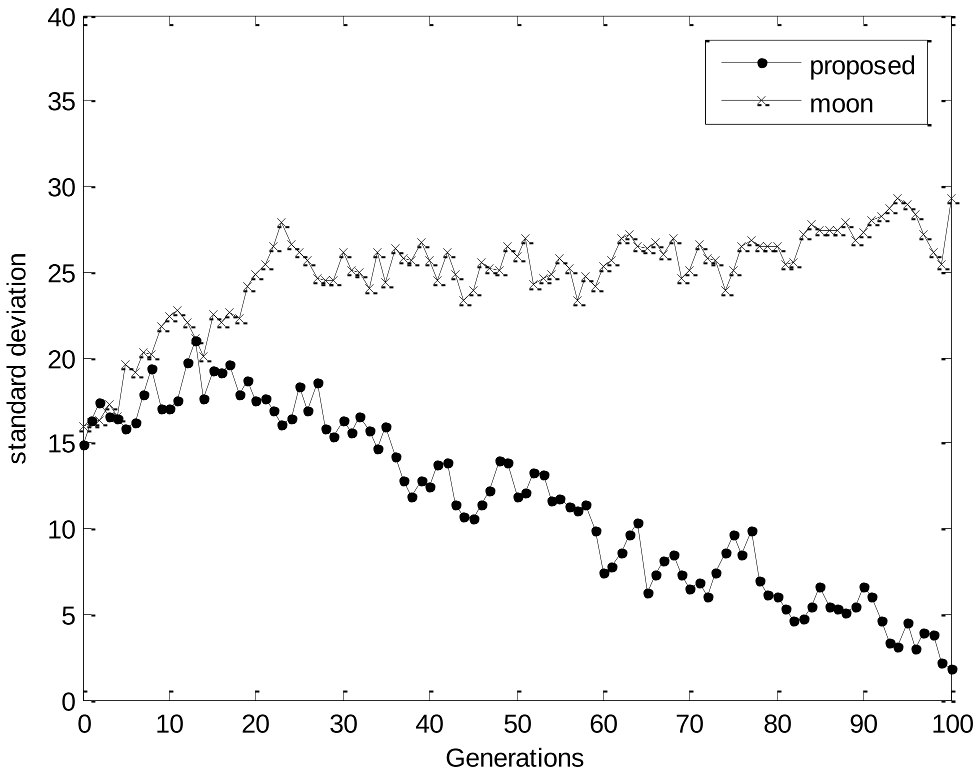


Figure 5.6: Standard deviation vs. generations graph for test problem 3

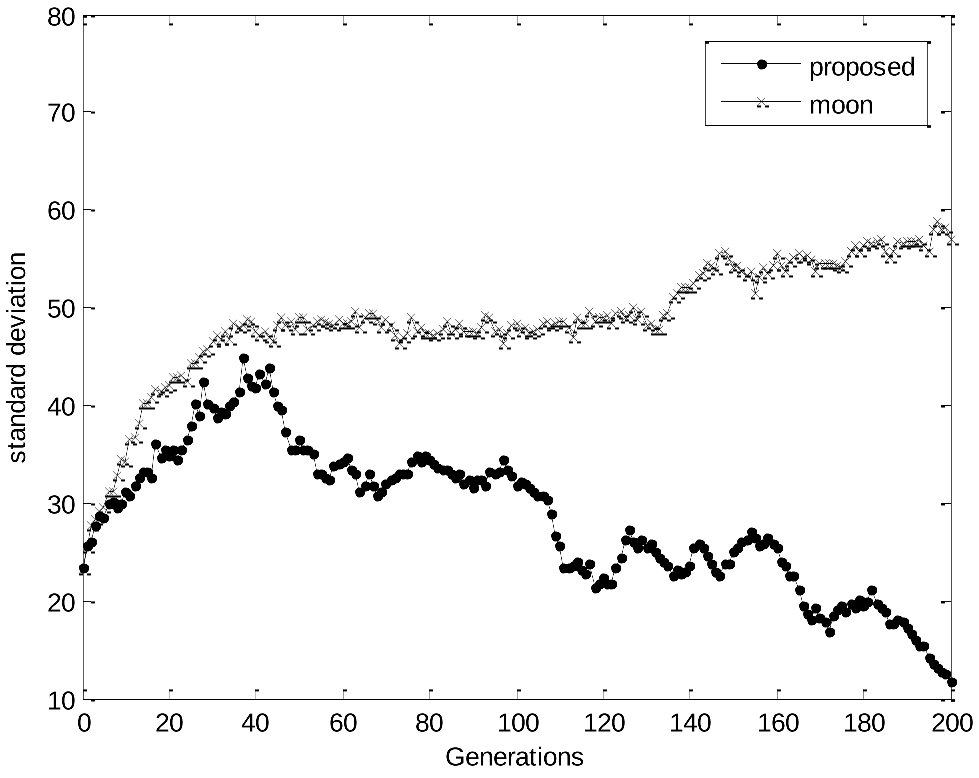


Figure 5.7: Standard deviation vs. generations graph for test problem 4

5.2.2 CPU Time

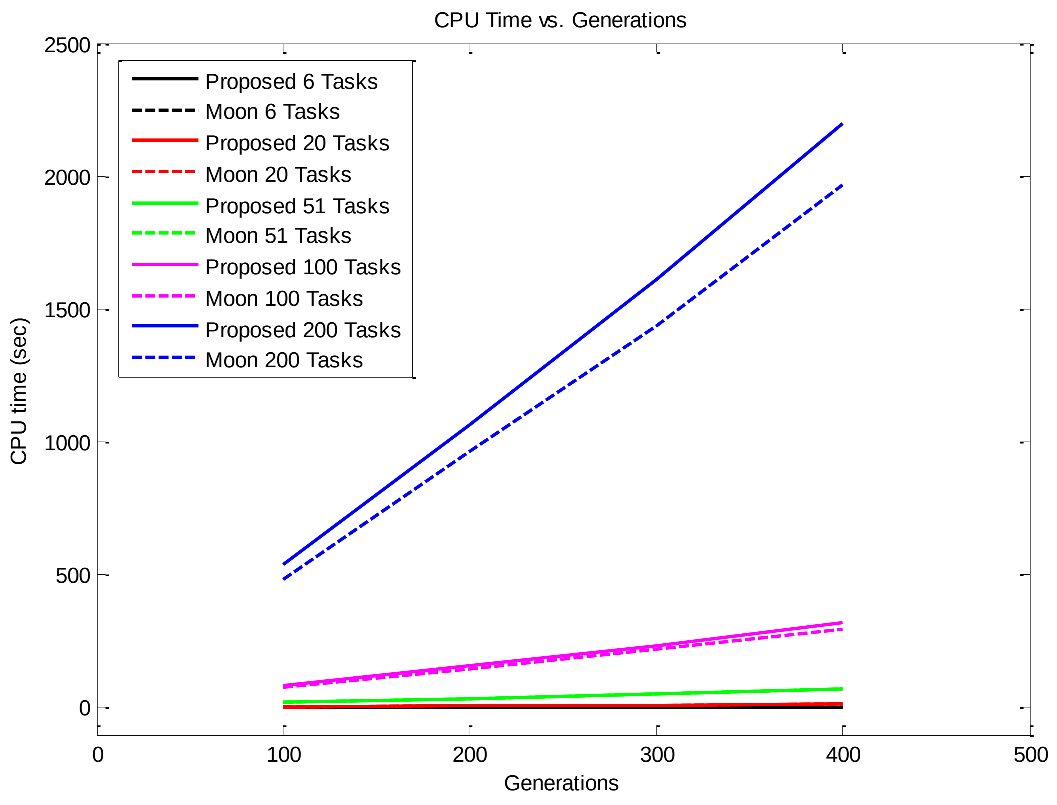


Figure 5.8: CPU time vs. number of generations

The graph in Figure 5.8 shows the CPU time versus the number of generations for 5 tasks, 20 tasks, 51 tasks, 100 tasks and 200 tasks problem size. From the graph, as the number of generation increases, the CPU time increases linearly. The trend is steadily increasing from 100 generations to 400 generations. Although the CPU time for Moon algorithm is slightly better compared to the proposed algorithm especially for large size problems, however this does not indicate that Moon algorithm produces better result at this particular number of generations. For all problems tested, the proposed algorithm achieved better quality of the solution (i.e. small transition time) compared to a Moon's algorithm with almost similar CPU time. Moon's algorithm converges early and end up with trapping at a local optimum rather than global optimum, whereas the proposed algorithm continuously search for the best solution and does not trap at a local optimum.

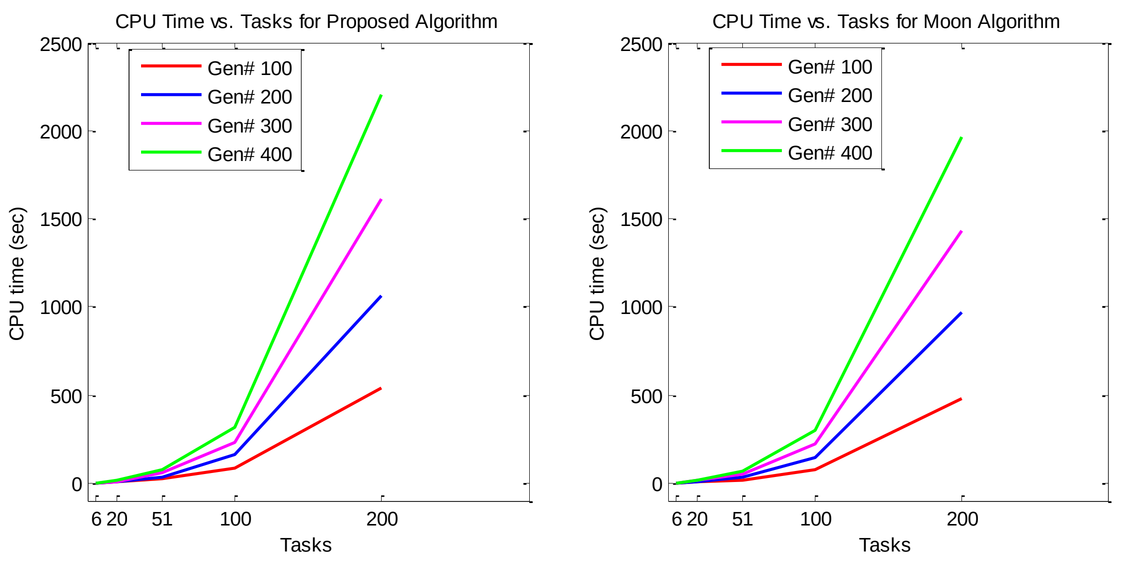


Figure 5.9: CPU time vs. number of tasks

In Figure 5.9, the CPU time is plotted against various problem sizes for different number of generations. The trend for proposed and Moon's algorithm are about similar for all test problems. From the graphs, it can be seen that CPU time increases exponentially with the number of tasks. As the size of the problem increases, the CPU time increases tremendously. For instance, in the proposed algorithm, the CPU time at generation 100 increases from 0.369 seconds for 5 tasks problem to 536.729 seconds for 200 tasks problem. Likewise for Moon's algorithm, the CPU time at generation 100 increases from 0.547 seconds for 5 tasks problem to 482.958 seconds for 200 tasks problem. In this experiment, the population size is set similarly for both algorithms.

Chapter 6

Conclusions & Future Work

This chapter concludes the research with respect to the research objectives and the main contributions of the research. Then, some potential avenues for further research are presented.

6.1 Conclusions

The aim of the study was to develop genetic algorithm in order to solve TSP subject to precedence constraints. An efficient algorithm must be able to generate feasible and optimal solution with less number of generations and fast iteration time. In relation to the research objectives set at the beginning of the thesis, the study had successfully

1. Covered the operation and components of GA and demonstrated the mechanism of each GA components
2. Developed an efficient GA procedure for TSP and TSPPC in order to generate feasible and optimal solution
3. Compared and verified the quality of solution and the performance of the proposed algorithm with a previously developed algorithm (i.e. Moon's algorithm)
4. Applied the proposed algorithm to various TSPPC problems benchmarked from related published papers

Although Moon *et al.* claimed that their algorithm to solve TSPPC is efficient, however in comparison to the approach developed in this thesis; it requires a larger number of generations and longer computation time to obtain the optimal solution. This is because the Moon's algorithm requires more steps to compare priorities before selecting the next task. Therefore, it takes longer time and more generations to come out with an optimal solution. The 'earliest position' based selection of task was implemented in the proposed algorithm which helped to reduce the iteration time, hence improve the GA performance. In addition, the utilisations of simple linear order crossover technique combined with inversion mutation were able to introduce new fitter individuals in the search space hence avoiding premature convergence.

In conclusions, the proposed GA approach combined with a topological sort procedure developed in this thesis is effective in solving TSP subject to precedence constraints with the objective to minimise the cost/time/distance. The role of topological sort is to ensure only feasible solutions exist in the search space while the GA operators is to further improve the quality of the solution. The proposed GA approach has proved suitable for solving the TSPPC and it is obvious that it could be easily applied to all other types of process sequencing problem which can be modelled as TSPPC.

6.2 Summary of contributions

The contributions to this field cover the development of a GA procedure to solve TSPPC, the analysis and improvement of recently proposed algorithm, and the exploration of new applications of the specific sequencing problem. From the review and experimental work, the main contributions of the research can be summarized as follows;

1. Developed TSPPC procedure – developed a clear genetic algorithm procedure for TSPPC in which route repair based topological sort is inserted in the procedure in order to generate only feasible chromosomes.
2. Developed fitness evaluation procedure for TSPPC – the objective function to evaluate the fitness of each chromosome has been developed.
3. Improved GA performance – the proposed algorithm has used ‘earliest position’ selection of tasks in order to reduce iteration time, hence improved GA performance.
4. Improved quality of the solution – the proposed algorithm has used simple linear order crossover and inversion mutation to introduce new fitter chromosomes from generations to generations in order to prevent premature convergence.

6.3 Suggestions for future work

This thesis leads to a number of opportunities for future research. The contributions and observations made in this work also pose a number of interesting open questions for the specific research issues attacked in this thesis for the research in the field of optimisation, in general. The following are possible areas for further investigation that could prove profitable to the genetic algorithm community and also to other research areas.

Apply different GA operators

The proposed GA procedure in this thesis appears to find good solutions for the TSPPC, however it depends very much on the way the problem is encoded and which parent selection, crossover and mutation methods are used. It seems that the biggest problem with the GA devised for the TSPPC is that it is difficult to maintain structure from the parent chromosomes. Perhaps a better crossover or mutation routine that retains structure from the parent chromosomes would give a better solution that has already found for some TSPPC. Therefore, some further works could be conducted to investigate the efficiency and effectiveness of the various crossover and mutation operators introduced in the earlier chapter.

Perform GA parameter study

The quality of the solution and the performance of the proposed GA also depend on various control parameters such as the population size, the crossover rate, the mutation rate and the generation gap. The GA parameter study could be performed to investigate the main effect and interaction among the parameters towards the quality of the solution and the performance of the GA.

Real world case study

In this thesis, a demonstration of application examples is benchmarked and simplified from the published papers. In the future, real case study with real data collected from the industry could be conducted especially that relate with assembly and disassembly sequence planning and scheduling problem.

Develop GA for MTSP with precedence constraints

In this thesis, a single salesman with precedence constraints is considered in the tour. In many real problem situations, more than a single salesman with multiple constraints present in the tour/operation. An example of the problem which considers more than one salesman is vehicle routing scheduling with precedence, time and capacity constraints. Another example is sequencing parallel machine with sequence dependent setup times and costs. Therefore, an efficient algorithm which can handle multiple salesmen with precedence constraints should be developed.

Multi-objective optimisation

It is also possible to consider multi-objective optimistion for TSPPC problem. This is an interesting subject to explore and a more realistic problem to be tackled as most of the real problems in manufacturing involve with more than one objective to be achieved such as maximising the profit while minimising the cost or maximising customer satisfaction (e.g. on time delivery) while minimising on-hand inventories. Such objectives are often conflicting in nature in that realizing improvements in one often requires accepting inferior solutions for others, i.e. a trade-off must be addressed. An example area of study is sequencing mixed-model assembly problem in a just in time (JIT) environment where the objectives of dependent setup times and production rate are considered simultaneously. Since the objective may conflict with each other, a sequence that can optimise both objectives at the same time may not exist. Therefore Pareto front must be constructed and can be used to assist the decision making. Furthermore, this type of problem is NP-hard, and obtaining multi-objective genetic algorithm is a practical option.

References

[1] T. P. Bagchi, *et al.*, "A review of TSP based approaches for flowshop scheduling," *European Journal of Operational Research* vol. 169, pp. 816-854, 2006.

[2] E. L. Lawler, *et al.*, *The Traveling Salesman Problem*: John Wiley & Sons 1986.

[3] S. Chatterjee, *et al.*, "Genetic Algorithms and Traveling Salesman Problems," *European Journal of Operational Research*, vol. 93, pp. 490-510, 1996.

[4] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*, Second Edition ed.: John Wiley & Sons, Inc., 2004.

[5] C. M. Reidys and P. F. Stadler, "Combinatorial Landscapes," *SIAM review, Society for Industrial and Applied Mathematics*, vol. 44, No. 1, pp. 3-54, 2002.

[6] C. Moon, *et al.*, "An efficient genetic algorithm for the traveling salesman problem with precedence constraints," *European Journal of Operational Research*, vol. 140, pp. 606-617, 2002.

[7] K. Fagerholt and M. Christiansen, "A traveling salesman problem with allocation, time window and precedence constraints - an application to ship scheduling," *International Transactions in Operational Research*, vol. 7, pp. 231-244, 2000.

[8] J. Hurink and S. Knust, "A tabu search algorithm for scheduling a single robot in a job-shop environment," *Discrete Applied Mathematics* vol. 119, pp. 181-203, 2002.

[9] L. M. Gambardella and M. Dorigo, "An ant colony system hybridized with a new local search for the sequential ordering problem," *Informs Journal on Computing*, 2000.

[10] J. Renaud, *et al.*, "A heuristic for the pickup and delivery traveling salesman problem," *Computers & Operations Research* vol. 27, pp. 905-916, 2000.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the Theory of NP-Completeness*. New York: W. H. Freeman and company.

[12] T. G. Stutzle, "Local search algorithms for combinatorial problems-analysis, improvements, and new applications," PhD Thesis, Technical University Darmstadz, 1998.

[13] G. Farber and A. Coves, "Overview on sequencing in mixed model flowshop production line with static and dynamic context," Universitat Politecnica De Catalunya, 2005.

[14] B. Rekiek and A. Delchambre, *Assembly line design: The balancing of mixed-model hybrid assembly lines with genetic algorithms*: Springer Series in Advanced Manufacturing, Springer-Verlag London Limited, 2006.

[15] G. Mitchell, "Evolutionary Computation Applied to Combinatorial Optimisation Problems," PhD Thesis, School of Electronic Engineering, Dublin City University, 2007.

[16] K. Bryant, "Genetic Algorithms and the Traveling Salesman Problem," Dept of Maths, Harvey Mudd College, 2000.

[17] C. R. Reeves, "A Genetic Algorithm for Flowshop Sequencing," *Computers & Operations Research*, vol. 22, pp. 5-13, 1995.

[18] M. T. Keller, "Knot theory: history and applications with a connection to graph theory," Thesis, Dept of Maths, North Dakota State University, 2004.

[19] P. Ji and W. Ho, "The traveling salesman and quadratic assignment problems: integration, modeling and genetic algorithm," in *Proceedings of International symposium on OR and its applications*, 2005.

[20] L. J. Schmitt, "Performance characteristics of alternative genetic algorithmic representation: An empirical study," *European Journal of Operational Research* vol. 108, pp. 551-570, 1998.

[21] N. Cummings. 5/2/2010). *A Brief History of TSP*. Available: http://www.orsoc.org.uk/about/topic/news/article_news_tspjune.html

[22] G. Dantzig, *et al.*, "Solution of a large-scale traveling salesman problem," *Operations Research Letters*, vol. 2, pp. 393-410, 1954.

[23] D. Applegate, *et al.*, "Implementing the Dantzig Fulkerson-Johnson Algorithm for large traveling salesman problems," *Mathematical programming* vol. 97, pp. 91-153, 2003.

[24] A. Schrijver, "On the history of combinatorial optimization (till 1960)," 1998.

[25] W. Cook. 13/4/2010). *Milestone in the Solution of TSP Instances*. Available: http://www.tsp.gatech.edu/history/milestone.html

[26] G. Reneilt. 20/12/2009). *TSPLIB* Available: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[27] P. Larranaga and C. M. H. Kuijpers, "Genetic algorithms for the traveling salesman problem: A review of representation and operators," in *Artificial Intelligence Review*. vol. 13, ed: Kluwer Academic Publishers, 1999, pp. 129-170.

[28] C. W. Ping, "A visual genetic algorithm tool, Masters Dissertation," Faculty of Computer Science and Information Technology, University of Malaya, 2006.

[29] C. A. Tovey, "Tutorial on Computational Complexity," *Informs Journal on Computing*, vol. 32, pp. 30-61, 2002.

[30] M. Grotschel and O. Holland, "Solution of large-scale symmetric travelling salesman problems," *Mathematical Programming*, vol. 51, pp. 141-201, 1991.

[31] A. Allahverdi, *et al.*, "A review of scheduling research involving setup considerations," *The International Journal of Management Science*, vol. 27, pp. 219-239, 1999.

[32] A. E. Carter, "Design and application of Genetic Algorithms for the multiple traveling salesperson assignment problem," PhD Thesis, Virginia Polytech Institute, 2003.

[33] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *The International Journal of Management Science*, vol. 34, pp. 209-219, 2006.

[34] A. Guinet, "Scheduling sequencing-dependent jobs on identical parallel machines to minimize completion criteria," *International Journal of Production Research*, vol. 31, pp. 1579-1594, 1993.

[35] K. Kotecha and N. Gambhava, "Solving Precedence Constraint Traveling Salesman Problem Using Genetic Algorithm," in *Proceedings of National Conference on Software Agents and embedded System*, 2003.

[36] R. K. Wysocki, *Effective Project Management: traditional, agile, extreme, 5th ed.* . Indianapolis: Wiley Publishing, 2009.

[37] H. N. Psaraftis, "k-Interchange procedures for local search in a precedence-constrained routing problem," *European Journal of Operational Research*, vol. 13, pp. 391-402, 1983.

[38] L. F. Escudero, "An inexact algorithm for the sequential ordering problem," *European Jornal of Operational Research*, vol. 37, pp. 236-249, 1988.

[39] L. M. Gambardella and M. Dorigo, "HAS-SOP: Hybrid ant system for the sequential ordering problem," 1997.

[40] A. Mingozzi, *et al.*, "Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints," *Operations Research* vol. 45, pp. 365-77, 1997.

[41] S. C. Sarin, *et al.*, "A precedence-constrained asymmetric traveling salesman model for disassembly optimization," *IIE Transactions,* vol. 38, pp. 223-237, 2006.

[42] E. Kongar and S. M. Gupta, "Disassembly sequencing using genetic algorithm," *International Journal of Advanced Manufacturing Technology,* vol. 30, pp. 497-506, 2006.

[43] P. Imtanavanich and S. M. Gupta, "Generating disassembly sequences for multiple products using genetic algorithm," *Proceedings of the 2007 POMS-Dallas Meeting,* 2007.

[44] Y. J. Tseng, *et al.*, "Integrated assembly and disassembly sequence planning using a GA approach," *International Journal of Production Research,* vol. 48, pp. 5991-6013, 2009.

[45] C. L. P. Chen, "AND/OR Precedence Constraint Traveling Salesman Problem and its Application to Assembly Schedule Generation," Department of Computer Science and Engineering, Wright State University, Dayton, OH, 1990.

[46] K. I. Srikanth and B. Saxena, "Improved genetic algorithm for the permutation flowshop scheduling problem," *Computers & operations Research* vol. 31, pp. 593-606, 2004.

[47] Z. Othman, "An integration approach in production scheduling using GA," PhD Thesis, Universiti Sains Malaysia, 2002.

[48] C. Dimopoulos and A. Zalzala, "Recent developments in evolutionary computation for manufacturing optimization: Problems, solutions, and comparisons," *IEEE Transactions on evolutionary computation,* vol. 4, 2000.

[49] M. W. P. Savelsberg and M. Sol, "The general Pickup and delivery problem," School of Industrial and systems eng. GIT, Atlanta, USA.

[50] G. R. Bitran and S. M. Gilbert, "Sequencing Production on Parallel Machines with Two Magnitudes of Sequence Dependent Setup Costs," *MIT Sloan School Working Paper,* 1989.

[51] P. C. Gilmore and R. E. Gomory, "Sequencing a one-state variable machine: a solvable case of the traveling salesman problem," *Operations Research,* vol. 12, pp. 655-679, 1964.

[52] F. Yalaoui and C. Chu, "An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times," *IIE Transactions,* vol. 35:2, pp. 183-190, 2010.

[53] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. New Jersey: Prentice Hall, 1995.

[54] M. Junger, *et al.*, "The Traveling Salesman Problem," in *Handbooks in OR & MS*. vol. 7, ed, 1995.

[55] S. Minic and R. Krishnamurti, "The multiple TSP with time windows: vehicle bounds based on precedence graphs," *Operations Research Letters* vol. 34, pp. 111-120, 2006.

[56] C. Moon and Y. Seo, "Evolutionary algorithm for advanced process planning and scheduling in a multi-plant," *Computers & Industrial Engineering* vol. 48, pp. 311-325, 2005.

[57] S. Kirkpatrick, *et al.*, "Optimization by Simulated Annealing," *Science,* vol. 220, pp. 671-680, 1983.

[58] E. H. L. Aarts and J. K. Lenstra, *Local search in combinatorial optimization*: Princeton University Press, 2003.

[59] D. Savic, "Single objective vs. Multiobjective optimization for integrated decision support," Centre for Water Systems, Department of Engineering School of Engineering and Computer Science, University of Exeter, UK.

[60] A. Jaszkiewicz, "Genetic local search for multi-objective combinatorial optimization," *European Journal of Operational Research,* vol. 137, pp. 50-71, 2002.

[61] P. Ngatchou, *et al.*, "Pareto Multi Objective Optimization," University of Washington, Seattle, Washington, USA2005.

[62] H. Tamaki, *et al.*, "Multi-objective optimization by Genetic Algorithms: A review," in *Proc. 1996 IEEE ICEC,* 1996, pp. 517–522.

[63] M. R. Gholamian, *et al.*, "A hybrid system for multiobjective problems - a case study in NP-hard problems," *Knowledge-Based Systems,* vol. 20, pp. 426–436, 2007.

[64] C. Malandraki and R. B. Dial, "A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem," *European Journal of Operational Research* vol. 90, pp. 45-55, 1996.

[65] A. Kusiak and G. Finke, "Modeling and solving the flexible forging module scheduling problem," *Engineering Optimization,* vol. 12, pp. 1-12, 1987.

[66] M. Fischetti and P. Toth, "An additive bounding procedure for combinatorial optimization problems," *Operations Research,* vol. 37, pp. 319-328, 1989.

[67] K. Fagerholt and M. Christiansen, "A traveling salesman problem with allocation, time window and precedence constraints - an application to ship scheduling," *International Transactions in Operational Research* vol. 7, pp. 231-244, 2000.

[68] Y. N. Chong, "Heuristic algorithms for routing problems," PhD Thesis, Curtin Univ. of technology, 2001.

[69] C. R. Reeves, *Modern heuristic techniques for combinatorial problems*: John Wiley & Sons, Inc., 1993.

[70] I. H. Osman and J. P. Kelly, *Meta-heuristics: Theory and applications,* 2nd Edition ed.: Springer-Verlag, 1996.

[71] J. Holland, *Adaptation in Natural and Artificial Systems*: Ann Arbor: University of Michigan Press, 1975.

[72] F. Glover, "Tabu Search-Part 1," *ORSA Journal on Computing*
vol. 1, 1989.

[73] S. H. Zanakis and J. R. Evans, "Heuristic methods and applications: A categorized survey," *European Journal of Operational Research,* vol. 43, pp. 88-110, 1989.

[74] L. F. Escudero, "On improving a solution to the ATSP with fixed origin and precedence relationships," *Trabajos de investigacion operativa,* vol. 3, pp. 117-140, 1988.

[75] E. A. Silver, "An overview of heuristic solution methods," *The Journal of the Operational Research Society,* vol. 55, pp. 936-956, 2004.

[76] D. Dannenbring, "An Evaluation of Flow Shop Sequencing Heuristics," *Management Science,* vol. 23, pp. 1174–1182, 1977.

[77] A. A. Hopgood, "Intelligent systems for engineers and scientists," ed: CRC Press, 2000.

[78] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*: MIT Press, 1992.

[79] S.Wang and A. Zhao, "An Improved Hybrid Genetic Algorithm for Traveling Salesman Problem," College of Computer and Information Technology, Henan Normal University, Xinxiang, China2009.

[80] G. Vahdati1, *et al.*, "A New Approach to Solve Traveling Salesman Problem Using Genetic Algorithm Based on Heuristic Crossover and Mutation Operator," presented at the International Conference of Soft Computing and Pattern Recognition, 2009.

[81] L. Zhang, *et al.*, "Optimization and Improvement of Genetic Algorithms Solving Traveling Salesman Problem," College of Computer Science, Zhejiang University, Hangzhou, China2009.

[82] F. Farhadnia, "A New Method Based on Genetic Algorithms for Solving Traveling Salesman Problem," presented at the International Conference on Computational Intelligence, Modelling and Simulation, 2009.

[83] B. F. Al-Dulaimi and H. A. Ali, "Enhanced Traveling Salesman Problem Solving by Genetic Algorithm Technique (TSPGA)," in *Proceedings of World Academy of Science, Engineering and Technology*, 2008, pp. 2070-3740

[84] G. Zhao, *et al.*, "A genetic algorithm balancing exploration and exploitation for the travelling salesman problem," presented at the Fourth international conference on Natural Computation, 2008.

[85] W. Youping, *et al.*, "An advanced Genetic Algorithm for Traveling Salesman Problem," presented at the Third International Conference on Genetic and Evolutionary Computing, 2009.

[86] S. S. Ray, *et al.*, "New Operators of Genetic Algorithms for Traveling Salesman Problem," in *Proceedings of the 17th International Conference on Pattern Recognition*, 2004.

[87] J. Y. Potvin, "Genetic Algorithms for the Traveling Salesman Problem," *Annals of Operations Research,* vol. 63, 1996.

[88] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*. New York: Wiley, 1997.

[89] M. Cepin, *Assessment of Power System Reliability: Methods and Applications*: Springer-Verlag London Limited, 2011.

[90] K. Katayama and H. Sakamoto, "The efficiency of hybrid mutation genetic algorithm for the travelling salesman problem," *Mathematical and Computer Modelling,* vol. 31, pp. 197-203, 2000.

[91] H. Youssef, *et al.*, "Evolutionary algorithms, simulated annealing and tabu search: a comparative study," *Engineering Applications of Artificial Intelligence* vol. 14, pp. 167–181, 2001.

[92] A. E. Carter and C. T. Ragsdale, "A new approach to solving the multiple traveling salesperson problem using genetic algorithms," *European Journal of Operational Research* vol. 175, pp. 246-257, 2006.

[93] M. Netnevitsky, *Artifial intelligence: A guide to intelligent systems*. England: Pearson Education Limited, 2002.

[94] Z. Michalewicz, *Genetic Algorithms+Data Structures=Evolution Programs*, 2nd Edition ed.: Springer-Verlag, 1994.

[95] A. Marczyk. 10/11/2010). *Genetic Algorithms and Evolutionary Computation*. Available: http://www.talkorigins.org/faqs/genalg/genalg.html

[96] Y. J. Kim, *et al.*, "A heuristic-based genetic algorithm for workload smoothing in assembly lines," *Computers Ops Res. ,* vol. 25, pp. 99-111, 1998.

[97] D. E. Goldberg and K. Deb, *A comparative analysis of selection schemes used in genetic algorithms*. Los Altos: in: G.J.E. Rawlins (Ed.), Foundations of Genetic Algorithms, Morgan Kaufmann, 1991.

[98] A. P. J. Dreo, P. Siarry, E. Taillard, *Metaheuristics for Hard Optimization; Methods and Case Studies*: Springer Berlin Heidelberg 2006.

[99] Y. K. Kim, *et al.*, "Genetic algorithms for assembly line balancing with various objectives," *Computers ind. Engng,* vol. 30, pp. 397-409, 1996.

[100] L. Gouveia and J. M. Pires, "The asymmetric travelling salesman problem and a reformulation of the Miller-Tucker-Zemlin constraints," *European Journal of Operational Research,* vol. 112, pp. 134-146, 1999.

[101] T. Watanabe, *et al.*, "Line balancing using a genetic evolution model," *Control engineering Practice,* vol. 3, pp. 69-76, 1995.

[102] J. E. Baker, "Adaptive Selection Methods for Genetic Algorithms," presented at the Proceedings of the 1st International Conference on Genetic Algorithms, Hillsdale, NJ, USA, 1985.

[103] C. Grosan and A. Abraham, *Intelligent Systems: A modern Approach* vol. 17: Springer-Verlag Berlin Heidelberg, 2011.

[104] L. Y. Wan and W. Li, "An improved particle swarm optimization algorithm with rank-based selection," *Proceedings of the seventh International Conference on Machine Learning and Cybernetics,* 2008.

[105] E. Parkinson, "Using improvement location and improvement preference to create meta-heuristic," PhD Thesis, City University, London, 2004.

[106] D. B. Fogel, "An Evolutionary Approach to the Traveling Salesman Problem," *Biology Cybernetic,* vol. 60, pp. 139-144, 1988.

[107] I. M. Oliver, *et al.*, "A Study of Permutation Crossover Operators on the TSP " in *Grefenstette, J. J. (ed) Genetic Algorithms and Their Applications, Proceedings of the Second International Conference*, Hillsdale, New Jersey, 1987.

[108] I. Kaya and O. Engin, "A new approach to define sample size at attributes control chart in multistage processes: An application in engine piston manufacturing process," *Journal of Materials Processing Technology* vol. 183, pp. 38-48, 2007.

[109] W. Banzhaf, "The "Molecular" Travelling Salesman," *Biological Cybernetics* vol. 60, pp. 139-1444:7-14, 1990.

[110] M. Mitchell, *An Introduction to Genetic Algorithms*: MIT Press, 1996.

[111] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*: Addison-Wesley, 1989.

[112] C. A. Coello and G. T. Pulido, "Multiobjective Structural Optimization using a Micro-Genetic Algorithm," *Structural and Multidisciplinary Optimization,* vol. 30, pp. 388-403, 2005.

[113] D. E. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Norwell, MA, USA: Kluwer Academic, 2002.

[114] S. N. Sivanandam and S. N. Deepa, *Introduction to Genetic Algorithms*: Springer Berlin Heidelberg New York, 2008.

[115] K. A. DeJong and J. Sarma, "Generation Gaps Revisited," George Mason University, USA1993.

[116] D. Beasley, *et al.*, "An Overview of genetic algorithms: Part 1, Fundamentals," *University Computing,* vol. 2, pp. 58-69, 1993.

[117] P. Pongcharoen, *et al.*, "Determining optimum Genetic Algorithm parameters for scheduling the manufacturing and assembly of complex products," *Int. J. Production Economics* vol. 78, pp. 311–322, 2002.

[118] M. Mokhlesian, *et al.*, "Economic lot scheduling problem with consideration of money time value," *International Journal of Industrial Engineering Computations,* vol. 1, pp. 121-138, 2010.

[119] A. Homaifar, "A new approach on the travelling salesman problem by genetic algorithms," in *Proceedings of the 5th International Conference on Genetic Algorithms*, 1993, pp. 460-466.

[120] U. Ozcan and B. Toklu, "Multiple-criteria decision-making in two-sided assembly line balancing: A goal programmingand a fuzzy goal programming models," *Computers & Operations Research,* vol. 36, pp. 1955-1965, 2009.

[121] A. Scholl. 25/3/2011). *Data of assembly line balancing problems*. Available: www.assembly-line-balancing.de

[122] S. Suwannarongsri and D. Puangdownreong, "Optimal assembly line balancing using tabu search with partial random permutation technique," *International Journal of Management Science,* vol. 3, pp. 3-18, 2007.

APPENDIX A-1

TSP_LIB data for burma14, bay29, dantzig42 and eil51

burma14	
16.47	96.10
16.47	94.44
20.09	92.54
22.39	93.37
25.23	97.24
22.00	96.05
20.47	97.02
17.20	96.29
16.30	97.38
14.05	98.12
16.53	97.38
21.52	95.59
19.41	97.13
20.09	94.55

bay29	
1150	1760
630	1660
40	2090
750	1100
750	2030
1030	2070
1650	650
1490	1630
790	2260
710	1310
840	550
1170	2300
970	1340
510	700
750	900
1280	1200
230	590
460	860
1040	950
590	1390
830	1770
490	500
1840	1240
1260	1500
1280	790
490	2130
1460	1420
1260	1910
360	1980

dantzig42	
170	85
166	88
133	73
140	70
142	55
126	53
125	60
119	68
117	74
99	83
73	79
72	91
37	94
6	106
3	97
21	82
33	67
4	66
3	42
27	33
52	41
57	59
58	66
88	65
99	67
95	55
89	55
83	38
85	25
104	35
112	37
112	24
113	13
125	30
135	32
147	18
147.5	36
154.5	45
157	54
158	61
172	82
174	87

eil51			
37	52	30	48
49	49	43	67
52	64	58	48
20	26	58	27
40	30	37	69
21	47	38	46
17	63	46	10
31	62	61	33
52	33	62	63
51	21	63	69
42	41	32	22
31	32	45	35
5	25	59	15
12	42	5	6
36	16	10	17
52	41	21	10
27	23	5	64
17	33	30	15
13	13	39	10
57	58	32	39
62	42	25	32
42	57	25	55
16	57	48	28
8	52	56	37
7	38	30	40
27	68		

APPENDIX A-2

```
function [opt_rte,ave,best]=TSP_PROX(city_location,pop_size,Pc,Pm,ngener)

%Distance matrix calculation
N=size(city_location,1);
a=meshgrid(1:N);
city_distance=reshape(sqrt(sum((city_location(a,:)-city_location(a',:)).^2,2)),N,N);

%generate initial random chromosome
ngenes=N; %number of genes in a chromosome
chrom=zeros(pop_size,ngenes);
for k=1:pop_size
   chrom(k,:)=randperm(ngenes);
end

%fitness evaluation in the initial population
ObjV=zeros(1,pop_size);
for p=1:pop_size
   d=city_distance(chrom(p,ngenes),chrom(p,1)); % closed path-same starting and ending point
   for k=2:ngenes
      d=d+city_distance(chrom(p,k-1),chrom(p,k));
   end
   ObjV(p)=d;
end

best=min(ObjV); %minimum distance in the initial population
ave=mean(ObjV); %average distance in the initial population

for i=1:ngener
   %Parent selection-proportional roulette wheel
   proportional;

   %Crossover mechanism-Linear order
   points=round(rand(floor(numsel/2),1).*(ngenes-1))+1;
   points=[points round(rand(floor(numsel/2),1).*(ngenes-1))+1];
   points=sort((points*(rand(1)<Pc)),2);
   for j=1:length(points(:,1))
      swap_sect=newchrom(2*j-1:2*j,points(j,1)+1:points(j,2));
      remain_sect=newchrom(2*j-1:2*j,:);
      for k=1:ngenes
         for n=1:length(swap_sect(1,:))
            if newchrom(2*j-1,k)==swap_sect(2,n);
               remain_sect(1,k)=0;
            end
            if newchrom(2*j,k)==swap_sect(1,n);
               remain_sect(2,k)=0;
            end
         end
      end
      [a b c1]=find(remain_sect(1,:));
      [a b c2]=find(remain_sect(2,:));
      remain_sect=[c1; c2];
      newchrom(2*j-1:2*j,:)=[remain_sect(1:2,1:points(j,1)),...
         flipud(newchrom(2*j-1:2*j,points(j,1)+1:points(j,2))),...
         remain_sect(1:2,points(j,1)+1:length(remain_sect(1,:)))];
   end

   %Mutation mechanism-Inversion (flip left to right)
   for q=1:numsel
```

```
        if rand(1)<Pm
            points=sort((round(rand(floor(numsel/2),1).*(ngenes-1))+1)');
            newchrom(q,:)=[newchrom(q,1:points(1)),...
                fliplr(newchrom(q,points(1)+1:points(2))),...
                newchrom(q,points(2)+1:ngenes)];
        end
    end

    if pop_size-numsel %preserving a part of the parent chromosome population
        [answ,Index]=sort(fitness); %sort the fitness of parent chromosome & preserving the
        %best nind-numsel chromosomes
        chrom=[chrom(Index(numsel+1:pop_size),:);newchrom];
    else %replacing the entire parent chromosome population with a new one
        chrom=newchrom;
    end

    %Fitness Evaluation
    ObjV=zeros(1,pop_size);
    for p=1:pop_size
        d=city_distance(chrom(p,ngenes),chrom(p,1));
        for k=2:ngenes
            d=d+city_distance(chrom(p,k-1),chrom(p,k));
        end
        ObjV(p)=d;
    end
    best=[best min(ObjV)]; %minimum distance in every generation
    ave=[ave mean(ObjV)]; %average distance in every generation

    [min_dist index]=min(ObjV);
    opt_rte=chrom(index,:); %optimal route
    [c d]=min(best);
end

subplot(1,2,1);
rtes=opt_rte([1:ngenes 1]);
plot(city_location(rtes,1),city_location(rtes,2),'r.-');
title(['Generation # ',num2str(d),'  Distance: ',num2str(min_dist)])
subplot(1,2,2);
plot(0:ngener,best,0:ngener,ave);
legend('Best','Average',0);
xlabel('Generation')
ylabel('Distance')
title('Performance Graph');
```

sub-program: proportional

```
fitness=(1./ObjV)';
numsel=round(pop_size*0.9);
cumfit=cumsum(fitness);
chance=cumfit(pop_size).*rand(numsel,1);
Mf=cumfit(:,ones(1,numsel));
Mt=chance(:,ones(1,pop_size))';
[selind,idx]=find(Mt < Mf & [zeros(1,numsel); Mf(1:pop_size-1,:)] <= Mt);
newchrom=chrom(selind,:);
```

sub-program: rank-based

```
fitness=(1./ObjV)';
fitnessrank=sort(fitness);
for f=1:pop_size
    SP=1.1; %selection pressure
```

```
   fitnessrank(f,:)=2-SP+(2*(SP-1)*(f-1)/(pop_size-1));
end

cumfit=cumsum(fitnessrank);
numsel=round(pop_size*0.9);
chance=cumfit(pop_size).*rand(numsel,1);
Mf=cumfit(:,ones(1,numsel));
Mt=chance(:,ones(1,pop_size))';
[selind,idx]=find(Mt < Mf & [zeros(1,numsel); Mf(1:pop_size-1,:)] <= Mt);
newchrom=chrom(selind,:);
```

sub-program: tournament

```
Tsize=2; %tournament size is 2
N=pop_size/Tsize;
M=1;
s=0;
while s<Tsize
   ts=randperm(pop_size);
   j=0; ii=0;
   while j<pop_size/Tsize
      j=j+1;
      for k=1:Tsize
         dists(k)=ObjV(ts(k+ii));
         Index(k)=ts(k+ii);
       end
       [Y,idx]=min(dists);
       winners(j,:)=chrom(Index(idx),:);
       ii=ii+Tsize;
    end
    s=s+1;
    AA(M:N,1:num)=winners;
    N=N+pop_size/Tsize;
    M=M+pop_size/Tsize;
end
newchrom=AA;
```

APPENDIX A-3

```
%Proposed TSPPC algorithm for 20 tasks and 38 precedence constraints

clear all; clc;

tic

 trans_time =   [ 0   2   10   4   10   2   11   9   1   3   7   12   5   8   8   8   13   7   2   12
                  2   0   12   12   4   6   2   8   6   7   14   10   11   9   2   6   13   14   1   3
                  10  12   0   5   8   2   5   9   7   8   13   6   9   2   6   6   14   2   4   5
                  4   12   5   0   6   6   11   12   5   11   4   5   11   1   3   10   17   10   14   14
                  10   4   8   6   0   7   2   13   3   10   6   7   14   8   7   7   5   1   8   13
                  2   6   2   6   7   0   2   6   14   6   9   3   7   13   13   3   13   10   13   3
                  11   2   5   11   2   2   0   11   14   6   4   10   7   6   12   9   10   8   5   4
                  9   8   9   12   13   6   11   0   14   14   3   11   1   1   3   10   6   5   14   14
                  1   6   7   5   3   14   14   14   0   3   12   12   2   12   4   2   14   13   11   7
                  3   7   8   11   10   6   6   14   3   0   8   4   8   5   4   4   3   6   12   11
                  7   14   13   4   6   9   4   3   12   8   0   5   9   4   9   9   6   12   14   11
                  12   10   6   5   7   3   10   11   12   4   5   0   7   6   10   13   7   1   6   8
                  5   11   9   11   14   7   7   1   2   8   9   7   0   1   1   4   1   12   4   6
                  8   9   2   1   8   13   6   1   12   5   4   6   1   0   4   9   12   4   9   13
                  8   2   6   3   7   13   12   3   4   4   9   10   1   4   0   1   1   1   1   5
                  8   6   6   10   7   3   9   10   2   4   9   13   4   9   1   0   8   5   2   14
                  13   13   14   17   5   13   10   6   14   3   6   7   1   12   1   8   0   14   7   10
                  7   14   2   10   1   10   8   5   13   6   12   1   12   4   1   5   14   0   5   6
                  2   1   4   14   8   13   5   14   11   12   14   6   4   9   1   2   7   5   0   14
                  12   3   5   14   13   3   4   14   7   11   11   8   6   13   5   14   10   6   14   0];

 prec_data1 =   [ 1   0   0   0   0   0   0   0
                  2   0   1   6   0   0   0   0
                  3   0   1   0   0   0   0   0
                  4   0   1   5   0   0   0   0
                  5   0   1   0   0   0   0   0
                  6   0   0   0   0   0   0   0
                  7   0   2   6   0   0   0   0
                  8   0   2   3   0   0   0   0
                  9   0   3   4   5   10   0   0
                  10   0   18   0   0   0   0   0
                  11   0   5   0   0   0   0   0
                  12   0   7   0   0   0   0   0
                  13   0   14   0   0   0   0   0
                  14   0   7   8   0   0   0   0
                  15   0   8   9   0   0   0   0
                  16   0   10   0   0   0   0   0
                  17   0   16   0   0   0   0   0
                  18   0   11   0   0   0   0   0
                  19   0   12   13   0   0   0   0
                  20   0   13   15   17   19   0   0 ];

[pn,pm]=size(prec_data1);
rand('seed',1.4929e+009); %seed to set the same number every time run the program
num=length(trans_time(1,:));
```

```
%genetic parameters
pop_size=150;
ngenes=num;
Pc=0.6;
Pm=0.1;
ngener=100;

%initialize random chromosomes
chrom=zeros(pop_size,ngenes);
for k=1:pop_size
   chrom(k,:)=randperm(ngenes);
end
chrom; %chromosomes in the initial population

% repairing chromosomes in the population taking into account precedence constraint route_repair;

routtemp; %repaired chromosomes (feasible route)

%Fitness evaluation
ObjV=zeros(1,pop_size);
for p=1:pop_size
   d=0; %open path (different start and end point)
   for k=2:ngenes
      d=d+trans_time(routtemp(p,k-1),routtemp(p,k));
   end
   ObjV(p)=d;
end
best=min(ObjV);
ave=mean(ObjV);

%genetic operators
ccount=0;
for i=1:ngener
   %parent selection mechanism-proportional roulette wheel
   fitness=(1./ObjV)';
   numsel=round(pop_size*0.9);
   cumfit=cumsum(fitness);
   chance=cumfit(pop_size).*rand(numsel,1);
   Mf=cumfit(:,ones(1,numsel));
   Mt=chance(:,ones(1,pop_size))';
   [selind,idx]=find(Mt < Mf & [zeros(1,numsel); Mf(1:pop_size-1,:)] <= Mt);
   newchrom=chrom(selind,:);

   %Crossover mechanism-Linear order
   point1=round(rand(floor(numsel/2),1).*(ngenes-1))+1;
   point2=round(rand(floor(numsel/2),1).*(ngenes-1))+1;
   points=[point1 point2];
   points=sort((points*(rand(1)<Pc)),2);
   for j=1:length(points(:,1))
      swap_sect=newchrom(2*j-1:2*j,points(j,1)+1:points(j,2));
      remain_sect=newchrom(2*j-1:2*j,:);
      for k=1:ngenes
         for n=1:length(swap_sect(1,:))
            if newchrom(2*j-1,k)==swap_sect(2,n);
               remain_sect(1,k)=0;
            end
            if newchrom(2*j,k)==swap_sect(1,n);
               remain_sect(2,k)=0;
            end
         end
      end
```

```
        [a b c1]=find(remain_sect(1,:));
        [a b c2]=find(remain_sect(2,:));
        remain_sect=[c1; c2];
        newchrom(2*j-1:2*j,:)=[remain_sect(1:2,1:points(j,1)),...
            flipud(newchrom(2*j-1:2*j,points(j,1)+1:points(j,2))),...
            remain_sect(1:2,points(j,1)+1:length(remain_sect(1,:)))];
    end

    %Mutation mechanism-Inversion (flip left to right)
    for q=1:numsel
        if rand(1)<Pm
            points=sort((round(rand(floor(numsel/2),1).*(ngenes-1))+1)');
            newchrom(q,:)=[newchrom(q,1:points(1)),...
                fliplr(newchrom(q,points(1)+1:points(2))),...
                newchrom(q,points(2)+1:ngenes)];
        end
    end
    %creating a new population of chromosomes
    if pop_size-numsel %preserving a part of the parent chromosome population
        [answ,Index]=sort(fitness); %sort the fitness of parent chromosome & preserving the
        %best nind-numsel chromosomes
        chrom=[chrom(Index(numsel+1:pop_size),:);newchrom];
    else %replacing the entire parent chromosome population with a new one
        chrom=newchrom;
    end

    route_repair;

    ObjV=zeros(1,pop_size);
    for p=1:pop_size
        d=0;
        for k=2:ngenes
            d=d+trans_time(routtemp(p,k-1),routtemp(p,k));
        end
        ObjV(p)=d;
    end
    best=[best min(ObjV)];
    ave=[ave mean(ObjV)];
    ccount=ccount+1;
    time(ccount)=toc;
    aan(ccount)=min(ObjV);

    [min_time index]=min(ObjV);
    opt_rte=routtemp(index,:);
    [a b]=min(best);
end

toc

figure('name','Performance Graph1');
plot(0:ngener,best,0:ngener,ave);
legend('Best','Average',0);
xlabel('Generations');
ylabel('transition time (sec)')
title(['Generation # ',num2str(b),'  Transition time : ',num2str(min_time)])

figure('name','Performance Graph2');
plot(time,aan);
xlabel('iteration time (sec)');
ylabel('transition time (sec)');
title(['best route: ',num2str(opt_rte)])
```

sub-program: route_repair

```
routtemp=zeros(pop_size,ngenes);
for f=1:pop_size
   route=chrom(f,:);
   prec_data=zeros(pn,pm);
   prec_data(1:pn,1:pm)=prec_data_ori;
   available=[];

   %check available for 1st loop
   x=0;
   for i=1:num
      if prec_data(i,3)==0;
         x=x+1;
         available(x)=prec_data(i,1);
      end
   end

   %select the route
   bbreak=0;
   for numroute=1:length(route(1,:))
      for numav=1:length(available(1,:))
         if available(numav)==route(numroute)
            selected=route(numroute);
            bbreak=1;
            break
         end
      end
      if bbreak==1;
        break
      end
   end

   route(numroute)=[];
   available(numav)=[];

   %write to first route
   route2=zeros(1,num);
   route2(1,1)=selected;
   prec_data(route2(1,1),2)=1;

   %loop for searching next available
   for loop=2:num %n1
      xx=length(available);
      yd=length(route);
      for j=1:yd
         poss=0;
         if isempty(available) || min(abs(available-route(j)))~=0
            if prec_data(route(j),3)==0
               poss=1;
            else
               s=4;
               while s<=pn && prec_data(route(j),s)~=0
                  s=s+1;
               end
               s=s-3;

               ss=1;
               while ss<=s && prec_data(prec_data(route(j),ss+2),2)==1
                  ss=ss+1;
               end
```

```
                    ss=ss-1;

                    if ss==s
                        poss=1;
                    end
                end
            end

            if poss==1
                xx=xx+1;
                available(xx)=route(j);
            end
        end

        bbreak=0;
        selected=[];
        for numroute=1:length(route(1,:))
            for numav=1:length(available(1,:))
                if available(numav)==route(numroute)
                    selected=route(numroute);
                    bbreak=1;
                    break
                end
            end
            if bbreak==1;
                break
            end
        end

        route(numroute)=[];
        available(numav)=[];

        %write to route
        route2(1,loop)=selected;
        prec_data(selected,2)=1;
    end

    routtemp(f,:)=route2;
end
```

APPENDIX A-4

```
%Moon's algorithm for 20 tasks and 38 precedence constraints

clear all; clc;

tic

trans_time;
prec_data1;

[pn,pm]=size(prec_data1);
rand('seed',1.4929e+009); %seed to set the same number every time run the program
num=length(trans_time(1,:));

%genetic parameters
pop_size;
ngenes=num;
Pc;
Pm;
ngener;

%initialize the population (not considering prec. constraint)
chrom=zeros(pop_size,ngenes);
for k=1:pop_size
   chrom(k,:)=randperm(ngenes);
end

% repairing chromosomes in the population taking into account precedence constraint
moon_repair;

routtemp;

%Fitness evaluation
ObjV=zeros(1,pop_size);
for p=1:pop_size
   d=0; %open route (different start and end point)
   for k=2:ngenes
      d=d+trans_time(routtemp(p,k-1),routtemp(p,k));
   end
   ObjV(p)=d;
end
best=min(ObjV);
ave=mean(ObjV);

%genetic operators
ccount=0;
for i=1:ngener
   %Parent selection mechanism-proportional roulette wheel
   fitness=(1./ObjV)';
   numsel=round(pop_size*0.9);
   cumfit=cumsum(fitness);
   chance=cumfit(pop_size).*rand(numsel,1);
   Mf=cumfit(:,ones(1,numsel));
   Mt=chance(:,ones(1,pop_size))';
   [selind,idx]=find(Mt < Mf & [zeros(1,numsel); Mf(1:pop_size-1,:)] <= Mt);
   newchrom=chrom(selind,:);

   %Crossover mechanism-moon crossover
   point1=round(rand(floor(numsel/2),1).*(ngenes-1))+1;
```

```
point2=round(rand(floor(numsel/2),1).*(ngenes-1))+1;
points=[point1 point2];
points=sort((points*(rand(1)<Pc)),2);
for j=1:length(points(:,1))
   swap_sect=newchrom(2*j-1:2*j,points(j,1)+1:points(j,2));
   remain_sect=newchrom(2*j-1:2*j,:);
   Pa=remain_sect(1,:);
   Pb=remain_sect(2,:);
   if rand(1)<Pc
      moon_cross;
   end
   c1=Pa;
   c2=Pb;
   remain_sect=[c1;c2];
   newchrom;
   newchrom(2*j-1:2*j,:)=remain_sect(1:2,:);
   remain_sect=[];
end

%Mutation mechanism-exchange (swap position)
for q=1:numsel
   if rand(1)<Pm
      P=newchrom(i,:);
      J=20;
      sel=randint(1,2,[1 20]);
      PP=P;
      PP(sel(1))=P(sel(2));
      PP(sel(2))=P(sel(1));
      P=PP;
      newchrom(i,:)=P;
   end
end

%creating a new population of chromosomes
if pop_size-numsel %preserving a part of the parent chromosome population
   [answ,Index]=sort(fitness); %sort the fitness of parent chromosome & preserving the
   %best nind-numsel chromosomes
   chrom=[chrom(Index(numsel+1:pop_size),:);newchrom];
else %replacing the entire parent chromosome population with a new one
   chrom=newchrom;
end

moon_repair;

ObjV=zeros(1,pop_size);
for p=1:pop_size
   d=0;
   for k=2:ngenes
      d=d+trans_time(routtemp(p,k-1),routtemp(p,k));
   end
   ObjV(p)=d;
end
best=[best min(ObjV)];
ave=[ave mean(ObjV)];
ccount=ccount+1;
time(ccount)=toc;
aan(ccount)=min(ObjV);

[min_time index]=min(ObjV);
opt_rte=routtemp(index,:);
[a b]=min(best);
```

```
end

toc

figure('name','Performance Graph1');
plot(0:ngener,best,0:ngener,ave);
legend('Best','Average',0);
xlabel('Generations');
ylabel('transition time (sec)')
title(['Generation # ',num2str(b),'  Transition time : ',num2str(min_time)])

figure('name','Performance Graph2');
plot(time,aan);
xlabel('iteration time (sec)');
ylabel('transition time (sec)');
title(['best route: ',num2str(opt_rte)])
```

sub-program: moon_repair

```
routtemp=zeros(pop_size,ngenes);
for f=1:pop_size
   route=chrom(f,:);
   route_ori=route;
   prec_data=zeros(pn,pm);
   prec_data(1:pn,1:pm)=prec_data_ori;
   available=[];

   %check available for 1st loop
   x=0;
   for i=1:num
      if prec_data(i,3)==0;
         x=x+1;
         available(x)=prec_data(i,1);
      end
   end

   %select the route

   kk=route_ori(available);
   [ax,bx]=max(kk);
   selected=available(bx);
   ax=1;
   while ax<=length(route) && route(ax)~=available(bx)
      ax=ax+1;
   end
   if route(ax)==available(bx)
      route(ax)=[];
   end
   available(bx)=[];

   %write to first route
   route2=zeros(1,num);
   route2(1,1)=selected;
   prec_data(route2(1,1),2)=1;

   %loop for searching next available
   for loop=2:num %n1
      xx=length(available);
      yd=length(route);
      for j=1:yd
         poss=0;
```

```
            if isempty(available) || min(abs(available-route(j)))~=0
               if prec_data(route(j),3)==0
                  poss=1;
               else
                  s=4;
                  while s<=pn && prec_data(route(j),s)~=0
                     s=s+1;
                  end
                  s=s-3;

                  ss=1;
                  while ss<=s && prec_data(prec_data(route(j),ss+2),2)==1
                     ss=ss+1;
                  end
                  ss=ss-1;

                  if ss==s
                     poss=1;
                  end
               end
            end

            if poss==1
               xx=xx+1;
               available(xx)=route(j);
            end
         end

         selected=[];
         kk=[];
         kk=route_ori(available);
         [ax,bx]=max(kk);
         selected=available(bx);
         ax=1;
         while ax<=length(route) && route(ax)~=available(bx)
            ax=ax+1;
         end
         if route(ax)==available(bx)
            route(ax)=[];
         end
         available(bx)=[];

         %write to route
         route2(1,loop)=selected;
         prec_data(selected,2)=1;
      end
      routtemp(f,:)=route2;
   end
```

<u>sub-program: moon_cross</u>

```
Pa;
Pb;
J=20;
%crossover
a=randint(1,1,[1,20]);
if a==20
   aa=[a-1 a];
else
   aa=[a a+1];
end
```

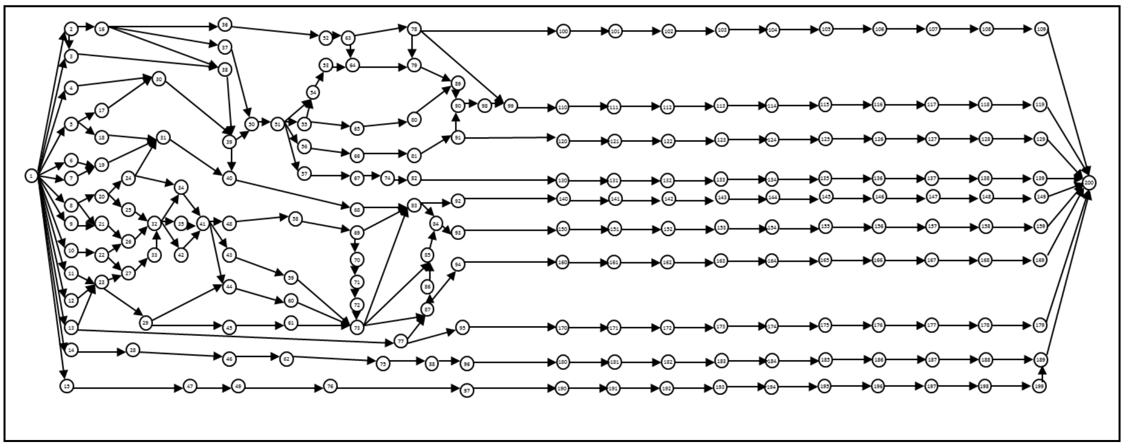
```
osp=[];
osp=Pa(aa);
%osp=[7 2]
ssub_Pb=osp;
%creating sub_Pb
sub=Pb;
for ip=1:length(Pb(1,:))
    for ii=1:length(osp(1,:))
        ibreak=0;
        if Pb(1,ip)==osp(1,ii)
            ibreak=1;
            sub(ip)=0;
        end
    end
end
in=0;

for iii=1:length(sub(1,:))
    if sub(1,iii)>0
        in=in+1;
        sub_Pb(in)=sub(1,iii);
    end
end
sub_Pb;

k=0;
i=0;
%loop for osp
i=length(osp(1,:))+1;
while length(osp(1,:))<J
    if i==1
        i=J+1;
        i=i-1;
        k=k+1;
        if Pa(i)~=Pb(k)
            ap=Pa(i);
            bp=Pb(k);
            for ki=1:length(osp(1,:))
                if osp(ki)==Pa(i)
                    ap=[];
                end
                if osp(ki)==Pb(k)
                    bp=[];
                end
            end
            osp=[osp ap bp];
        else
            ap=Pa(i);
            bp=Pb(k);
            for ki=1:length(osp(1,:))
                if osp(ki)==Pa(i)
                    ap=[];
                end
                if osp(ki)==Pb(k)
                    bp=[];
                end
            end
            osp=[osp ap];
        end
    elseif k==J
        i=i-1;
        k=k+1;
        if Pa(i)~=Pb(k)
            ap=Pa(i);
            bp=Pb(k);
            for ki=1:length(osp(1,:))
                if osp(ki)==Pa(i)
                    ap=[];
                end
                if osp(ki)==Pb(k)
                    bp=[];
                end
            end
            osp=[bp ap osp];
        else
            ap=Pa(i);
            bp=Pb(k);
            for ki=1:length(osp(1,:))
                if osp(ki)==Pa(i)
                    ap=[];
                end
                if osp(ki)==Pb(k)
                    bp=[];
                end
            end
            osp=[ap osp];
        end
    else
        i=i-1;
        k=k+1;
        if Pa(i)~=Pb(k)
            ap=Pa(i);
            bp=Pb(k);
            for ki=1:length(osp(1,:))
                if osp(ki)==Pa(i)
                    ap=[];
                end
                if osp(ki)==Pb(k)
                    bp=[];
                end
            end
            osp=[ap osp bp];
        else
            ap=Pa(i);
            bp=Pb(k);
            for ki=1:length(osp(1,:))
                if osp(ki)==Pa(i)
                    ap=[];
                end
                if osp(ki)==Pb(k)
                    bp=[];
                end
            end
            osp=[ap osp];
        end
    end
end
Pa=osp;

%representation of second offspring
osp2=[sub_Pb ssub_Pb];
Pb=osp2
```

APPENDIX A-5

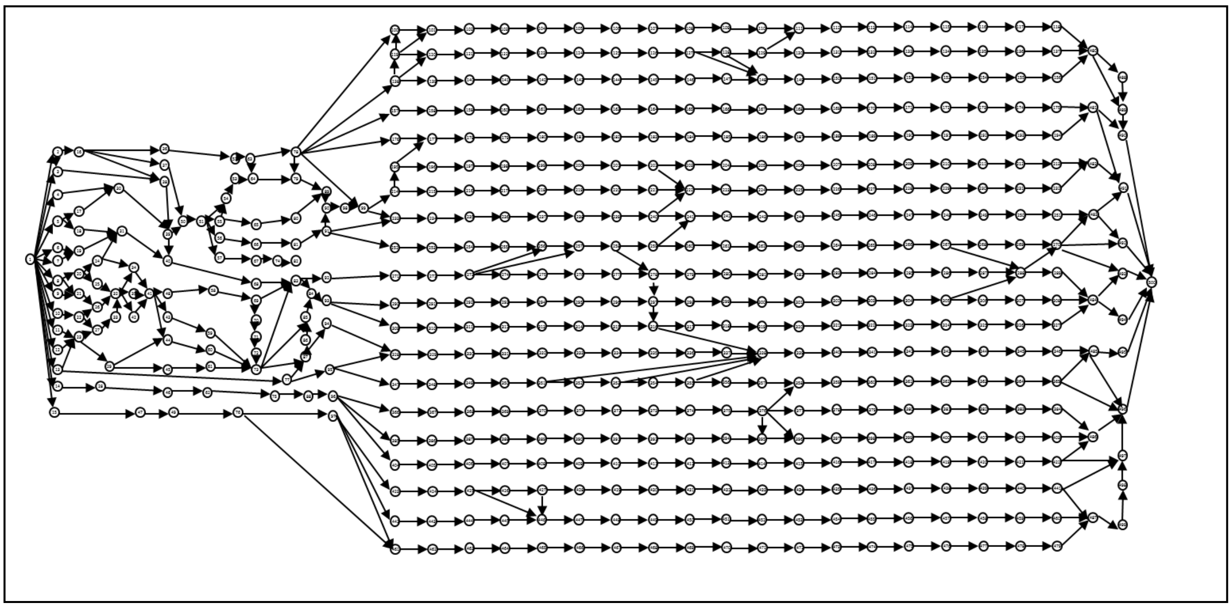
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
1	0	13	3	8	7	5	13	12	2	3	13	14	1	6	5	10	13	6	2	12	1	2	10	2	9	13	1	6	9	5	9	1	2	4	6	14	13	11	9	6	14	13	13	8	4	8	6	8	13	3	9
2	13	0	4	5	8	12	2	11	3	7	2	1	6	10	6	5	9	11	11	4	1	7	8	7	10	11	10	13	6	7	5	10	10	1	12	7	6	4	13	12	4	10	6	10	14	5	6	9	10	4	10
3	3	4	0	8	9	13	13	1	7	6	14	1	2	7	5	4	14	3	14	10	13	8	4	1	4	14	3	10	4	9	14	9	4	14	5	9	3	6	12	13	13	14	3	3	14	13	4	6	5	7	9
4	8	5	8	0	8	14	5	12	4	13	5	10	8	5	6	2	14	12	14	9	5	7	8	8	5	12	11	10	3	14	12	11	11	3	5	8	8	6	1	9	3	13	4	5	14	7	6	11	3	8	9
5	7	8	9	8	0	13	3	1	9	3	8	2	13	11	14	8	1	12	11	6	4	3	7	9	1	1	13	3	1	13	14	6	6	10	10	12	11	3	10	8	4	1	4	6	7	11	9	4	3	5	5
6	5	12	13	14	13	0	12	12	2	10	3	6	13	13	5	10	7	11	4	1	12	2	10	14	14	9	5	5	10	10	5	6	14	7	13	9	4	8	5	5	2	9	5	5	13	10	11	12	2	2	4
7	13	2	13	5	3	12	0	4	14	1	8	11	5	4	4	13	14	12	12	7	10	10	6	3	1	3	9	13	13	6	8	5	8	10	9	9	2	14	12	4	9	8	6	10	6	11	3	3	1	9	2
8	12	11	1	12	1	12	4	0	2	6	6	2	1	4	2	7	9	7	3	13	9	9	8	10	2	6	2	11	3	9	6	2	5	3	8	1	4	13	5	11	8	8	14	8	14	3	7	1	7	6	2
9	2	3	7	4	9	2	14	2	0	12	5	8	4	8	2	6	2	8	12	13	2	2	12	14	9	12	10	6	5	6	1	3	13	13	13	13	7	12	4	8	14	11	12	13	11	11	2	1	2	10	6
10	3	7	6	13	3	10	1	6	12	0	7	8	5	8	12	9	6	10	11	8	5	7	2	13	4	2	2	6	9	13	13	5	11	6	3	8	3	14	14	3	9	4	7	3	3	1	5	9	6	4	3
11	13	2	14	5	8	3	8	6	5	7	0	14	2	6	8	5	1	11	11	3	11	9	5	5	5	1	3	5	9	1	7	2	10	1	8	7	6	2	8	4	13	8	2	14	7	13	9	3	13	14	8
12	14	1	1	10	2	6	11	2	8	8	14	0	10	12	7	11	8	2	8	8	3	3	3	2	10	11	2	5	9	5	9	5	6	8	12	2	7	4	1	9	2	11	8	13	5	6	2	11	13	14	12
13	1	6	2	8	13	13	5	1	4	5	2	10	0	1	8	3	4	13	4	14	10	12	4	2	6	3	1	1	12	3	7	8	9	14	6	9	9	8	4	4	3	4	8	14	11	10	10	7	6	7	5
14	6	10	7	5	11	13	4	4	8	8	6	12	1	0	1	3	5	9	8	13	12	11	11	12	6	2	11	7	10	1	3	14	8	2	11	9	14	8	1	5	13	2	2	14	7	8	5	13	9	7	5
15	5	6	5	6	14	5	4	2	2	12	8	7	8	1	0	3	10	3	6	4	8	7	9	11	13	5	6	4	13	8	2	5	14	11	14	3	1	1	1	4	14	14	14	11	13	11	1	1	6	6	3
16	10	5	4	2	8	10	13	7	6	9	5	11	3	3	3	0	12	1	2	3	9	5	11	10	10	5	8	8	13	12	9	13	3	11	1	7	4	10	11	9	1	3	11	1	14	6	10	6	8	5	2
17	13	9	14	14	1	7	14	9	2	6	1	8	4	5	10	12	0	10	6	2	2	14	14	9	3	10	13	9	4	5	7	9	10	10	5	5	6	9	2	5	2	5	2	5	6	10	7	8	1	9	7
18	6	11	3	12	12	11	12	7	8	10	11	2	13	9	3	1	10	0	13	11	12	11	7	8	10	10	5	13	1	12	5	5	13	12	5	2	7	1	2	5	10	3	1	2	14	7	9	3	14	14	13
19	2	11	14	14	11	4	12	3	12	11	11	8	4	8	6	2	6	13	0	6	1	4	11	13	2	4	14	14	6	4	2	10	2	2	9	5	1	9	2	11	14	14	3	10	8	13	3	11	9	13	7
20	12	4	10	9	6	1	7	13	13	8	3	8	14	13	4	3	2	11	6	0	14	7	3	1	1	2	10	1	11	14	3	1	4	2	9	11	10	2	14	8	13	13	13	13	2	3	2	11	7	10	2
21	1	1	13	5	4	12	10	9	2	5	11	3	10	12	8	9	2	12	1	14	0	13	2	6	4	13	3	5	10	7	14	6	9	14	9	1	11	11	9	4	6	13	9	5	9	10	14	6	12	12	14
22	2	7	8	7	3	2	10	9	2	7	9	3	12	11	7	5	14	11	4	7	13	0	8	10	1	1	8	14	13	10	10	13	8	14	7	10	8	7	13	14	3	7	7	13	10	12	3	12	6	11	3
23	10	8	4	8	7	10	6	8	12	2	5	3	4	11	9	11	14	7	11	3	2	8	0	13	4	9	6	7	14	2	10	2	14	5	7	7	6	9	14	7	4	7	9	11	1	10	7	7	8	5	6
24	2	7	1	8	9	14	3	10	14	13	5	2	2	12	11	10	9	8	13	1	6	10	13	0	14	10	14	14	11	11	13	6	11	10	10	12	14	10	6	7	8	5	9	1	1	12	13	4	4	8	7
25	9	10	4	5	1	14	1	2	9	4	5	10	6	6	13	10	3	10	2	1	4	1	4	14	0	4	10	7	1	10	1	6	3	12	6	7	12	13	14	12	4	3	12	8	12	5	12	1	3	11	4
26	13	11	14	12	1	9	3	6	12	2	1	11	3	2	5	5	10	10	4	2	13	1	9	10	4	0	10	3	2	14	3	8	5	13	5	14	6	4	14	5	12	3	10	2	14	12	5	3	1	11	5
27	1	10	3	11	13	5	9	2	10	2	3	2	1	11	6	8	13	5	14	10	3	8	6	14	10	10	0	9	13	5	2	8	2	6	10	5	4	6	1	9	6	4	10	7	11	3	12	1	11	2	1
28	6	13	10	10	3	5	13	11	6	6	5	5	1	7	4	8	9	13	14	1	5	14	7	14	7	3	9	0	9	6	11	14	7	14	10	12	14	6	12	1	13	7	8	2	14	9	5	4	4	8	5
29	9	6	4	3	1	10	13	3	5	9	9	9	12	10	13	13	4	1	6	11	10	13	14	11	1	2	13	9	0	13	10	3	4	2	9	2	4	9	3	13	14	8	12	11	3	1	2	2	4	9	5
30	5	7	9	14	13	10	6	9	6	13	1	5	3	1	8	12	5	12	4	14	7	10	2	11	10	14	5	6	13	0	3	1	12	8	12	5	11	8	9	14	4	7	6	9	12	2	1	7	1	12	10
31	9	5	14	12	14	5	8	6	1	13	7	9	7	3	2	9	7	5	2	3	14	10	10	13	1	3	2	11	10	3	0	11	3	3	7	4	7	11	14	3	2	7	4	7	4	7	3	7	8	1	2
32	1	10	9	11	6	6	5	2	3	5	2	5	8	14	5	13	9	5	10	1	6	13	2	6	6	8	8	14	3	1	11	0	2	11	4	9	1	5	14	13	14	5	5	3	12	12	4	9	7	10	5
33	2	10	4	11	6	14	8	5	13	11	10	6	9	8	14	3	10	13	2	4	9	8	14	11	3	5	2	7	4	12	3	2	0	5	8	10	1	11	9	12	12	14	1	9	12	4	10	14	9	12	7
34	4	1	14	3	10	7	10	3	13	6	1	8	14	2	11	11	10	12	2	2	14	14	5	10	12	13	6	14	2	8	3	11	5	0	14	10	8	11	13	7	3	3	11	2	11	14	13	10	7	4	6
35	6	12	5	5	10	13	9	8	13	3	8	12	6	11	14	1	5	5	9	9	9	7	7	10	6	5	10	10	9	12	7	4	8	14	0	10	9	5	11	1	2	7	13	8	7	9	12	6	5	5	11
36	14	7	9	8	12	9	9	1	13	8	7	2	9	9	3	7	5	2	5	11	1	10	7	12	7	14	5	12	2	5	4	9	10	10	10	0	13	8	5	1	7	13	8	14	7	3	13	3	9	13	10
37	13	6	3	8	11	4	2	4	7	3	6	7	9	14	1	4	6	7	1	10	11	8	6	14	12	6	4	14	4	11	7	1	1	8	9	13	0	8	5	1	3	10	14	10	4	14	4	10	7	7	3
38	11	4	6	6	3	8	14	13	12	14	2	4	8	8	1	10	9	1	9	2	11	7	9	10	13	4	6	6	9	8	11	5	11	11	5	8	8	0	3	11	9	10	6	14	4	4	8	12	6	8	9
39	9	13	12	1	10	5	12	5	4	14	8	1	4	1	1	11	2	2	2	14	9	13	14	6	14	14	1	12	3	9	14	14	9	13	11	5	5	3	0	2	11	14	8	9	11	6	12	8	9	1	12
40	6	12	13	9	8	5	4	11	8	3	4	9	4	5	4	9	5	5	11	8	4	14	7	7	12	5	9	1	13	14	3	13	12	7	1	1	1	11	2	0	13	9	10	6	11	3	9	14	9	10	1
41	14	4	13	3	4	2	9	8	14	9	13	2	3	13	14	1	2	10	14	13	6	3	4	8	4	12	6	13	14	4	2	14	12	3	2	7	3	9	11	13	0	6	7	10	9	11	8	11	12	13	13
42	13	10	14	13	1	9	8	8	11	4	8	11	4	2	14	3	5	3	14	13	13	7	7	5	3	3	4	7	8	7	7	5	14	3	7	13	10	10	14	9	6	0	11	7	3	4	5	1	1	3	4
43	13	6	3	4	4	5	6	14	12	7	2	8	8	2	14	11	2	1	3	13	9	7	9	9	12	10	10	8	12	6	4	5	1	11	13	8	14	6	8	10	7	11	0	7	4	2	9	4	7	4	9
44	8	10	3	5	6	5	10	8	13	3	14	13	14	14	11	1	5	2	10	13	5	13	11	1	8	2	7	2	11	9	7	3	9	2	8	14	10	14	9	6	10	7	7	0	7	14	14	2	5	12	4
45	4	14	14	14	7	13	6	14	11	3	7	5	11	7	13	14	6	14	8	2	9	10	1	1	12	14	11	14	3	12	4	12	12	11	7	7	4	4	11	11	9	3	4	7	0	11	4	1	11	6	11
46	8	5	13	7	11	10	11	3	11	1	13	6	10	8	11	6	10	7	13	3	10	12	10	12	5	12	3	9	1	2	7	12	4	14	9	3	14	4	6	3	11	4	2	14	11	0	7	2	10	10	9
47	6	6	4	6	9	11	3	7	2	5	9	2	10	5	1	10	7	9	3	2	14	3	7	13	12	5	12	5	2	1	3	4	10	13	12	13	4	8	12	9	8	5	9	14	4	7	0	13	12	11	7
48	8	9	6	11	4	12	3	1	1	9	3	11	7	13	1	6	8	3	11	11	6	12	7	4	1	3	1	4	2	7	7	9	14	10	6	3	10	12	8	14	11	1	4	2	1	2	13	0	3	1	9
49	13	10	5	3	3	2	1	7	2	6	13	13	6	9	6	8	1	14	9	7	12	6	8	4	3	1	11	4	4	1	8	7	9	7	5	9	7	6	9	9	12	1	7	5	11	10	12	3	0	4	4
50	3	4	7	8	5	2	9	6	10	4	14	14	7	7	6	5	9	14	13	10	12	11	5	8	11	11	2	8	9	12	1	10	12	4	5	13	7	8	1	10	13	3	4	12	6	10	11	1	4	0	1
51	9	10	9	9	5	4	2	2	6	3	8	12	5	5	3	2	7	13	7	2	14	3	6	7	4	5	1	5	5	10	2	5	7	6	11	10	3	9	12	1	13	4	9	4	11	9	7	9	4	1	0

APPENDIX A-6

200 tasks & 241 precedence constraints



500 tasks & 587 precedence constraints



APPENDIX A-7

```
%Fitness evaluation for ALB problem

%idle is total idle time in the workstation
%wst is a workstation
%gtime is total task time in the workstation
%seq is routtemp (task/operation sequence)
%time is task time for each operation
%ct is a pre-determined cycle time in the workstation

[n,m]=size(seq);
ITs=zeros(n,1);
wst=zeros(2*n,m);
gtime=zeros(n,1);
for z=1:n,
    temptime=zeros(1,m);
    for j=1:m
        temptime(1,j)=time(1,seq(z,j));
    end
    [ITs(z,1),wst(2*z-1:2*z,:),gt]=idletime_ALB(seq(z,:),temptime,ct);
    [s,t]=size(gt);
    gtime(z,1:t)=gt;
end
ObjV=ITs';
best=min(ObjV);
ave=mean(ObjV);
```

<u>sub-program: idletime_ALB</u>

```
function [idle,wst,gtime]=idletime_ALB(seq,time,ct)
[n,m]=size(seq);
temp=time(1,1);
wst=zeros(2,m);
wst(1,:)=seq;
cg=0;
cwst=1;
cgrst=0;
gtime=zeros(1,n);
for r=2:m
    temp=temp+time(1,r);
    if temp>ct
        cg=cg+1;
        gtime(1,cg)=temp-time(1,r);
        cgrst=cgrst+1;
        wst(2,cwst:r-1)=cgrst*ones(1,r-cwst);
        cwst=r;
        temp=time(1,r);
    end
    if r==m
       cg=cg+1;
       gtime(1,cg)=temp;
       cgrst=cgrst+1;
       wst(2,cwst:r)=cgrst*ones(1,r-cwst+1);
    end
end
idle=sum(ct-gtime);
```