MULTI-STATE PSO GSA FOR SOLVING DISCRETE COMBINATORIAL OPTIMIZATION PROBLEMS

ISMAIL BIN IBRAHIM

Thesis submitted in fulfilment of the requirements
for the award of the degree of
Doctor of Philosophy of Engineering

Faculty of Electrical and Electronic Engineering
UNIVERSITI MALAYSIA PAHANG

AUGUST 2016

ABSTRACT

There are various meta-heuristics exist in literature nowadays. However, not all meta-heuristics were originally developed to operate in discrete search space. Two examples of meta-heuristics are Particle swarm optimization (PSO) and gravitational search algorithm (GSA), which are based on the social behavior of bird flocks and the Newton's law of gravity and the law of motion, respectively. In order to solve discrete combinatorial optimization problems (COPs) using meta-heuristics, modification or enhancement is needed. In the context of the modification, a variety of discretization approaches have been proposed. Inspired by the design of a sequential circuit of digital system, a new discretization approach that leads to the establishment of a complete model of multi-state has been proposed. Based on the multi-state model, a multi-state search space is successfully built using the following two features; a current state and a radius. The multi-state model is then implemented in PSO and GSA. As a consequence, multi-state particle swarm optimization (MSPSO) and multi-state gravitational search algorithm (MSGSA) are developed. In the MSPSO and the MSGSA, the radius is represented by new velocity value. The extended version of the multi-state model is then formulated by introducing an embedded rule that ensures the updated solutions to be formed by unrepeated states. As a consequence, multi-state particle swarm optimization with an embedded rule (MSPSOER) and multi-state gravitational search algorithm with an embedded rule (MSGSAER) are developed. These four algorithms can be used to solve discrete combinatorial optimization problems (COPs). To evaluate the performances of the proposed algorithms, several experiments using eighteen sets of selected benchmarks instances of Travelling salesman problem (TSP) and a case study of assembly sequence planning (ASP) problem are conducted. The experimental results showed the newly introduced multi-state PSO GSA are promising compared to binary particle swarm optimization (BPSO) and binary gravitational search algorithm (BGSA) for the TSP and consistently outperformed simulated annealing (SA), genetic algorithm (GA), and BPSO for the ASP in finding optimal solutions.

ABSTRAK

Pada masa kini, terdapat pelbagai meta-heuristik yang wujud dalam kajian. Namun begitu, pada asalnya bukan semua meta-heuristik telah dibangunkan untuk beroperasi dalam ruang carian diskret. Dua contoh meta-heuristik ialah pengoptimuman kerumunan zarah (PSO) dan algoritma carian graviti (GSA) yang mana masing-masing berasaskan tingkah laku sosial kawanan burung dan hukum graviti Newton dan hukum gerakan. Dalam usaha untuk menyelesaikan masalah-masalah pengoptimumam kombinatorik diskret menggunakan meta-heuristik, proses pengubahsuaian atau penambahan diperlukan. Dalam konteks pengubahsuai, pelbagai pendekatan pendiskretan telah dicadangkan. Dengan berinspirasikan reka bentuk litar berjujukan dalam sistem digital, satu pendekatan baharu pendiskretan yang membawa kepada penubuhan satu model lengkap berbilang keadaan telah dicadangkan. Berdasarkan model berbilang keadaan itu, ruang carian berbilang keadaan dibina dengan jayanya menggunakan kedua-dua ciri-ciri berikut; satu keadaan semasa dan satu jejari. Kemudian, model berbilang keadaan ini diimplementasikan dalam PSO dan GSA. Lalu, pengoptimuman sekawan zarah berbilang keadaan (MSPSO) dan algoritma carian graviti berbilang keadaan (MSGSA) dihasilkan. Kemudian, satu versi model tambahan berbilang keadaan diformulasikan dengan memperkenalkan satu peraturan terbenam yang memastikan solusi-solusi yang telah dikemaskini itu dibina oleh keadaan-keadaan yang tidak berulang. Lalu, MSPSOER dan MSGSAER dihasilkan. Keempat-empat algoritma ini boleh digunakan untuk menyelesaikan masalah-masalah pengoptimumam diskret. Untuk menilai prestasi algoritma-algoritma yang dicadangkan, beberapa eksperimen yang menggunakan lapan belas set contoh penanda aras yang terpilih bagi TSP dan satu kajian kes bagi ASP dijalankan. Keputusan eksperimen menunjukkan algoritma-algoritma berasaskan pelbagai keadaan yang baru diperkenalkan ini mempunyai potensi yang baik jika dibandingkan kepada BPSO dan BGSA untuk TSP dan secara konsistennya mengatasi SA, GA, dan BPSO untuk ASP dalam menemui solusi-solusi yang optimal.

TABLE OF CONTENTS

Page

CHAPTER 3 METHODOLOGY

CHAPTER 5 CONCLUSION AND RECOMMENDATIONS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

c_1	Cognitive coefficient
c_2	Social coefficient
r	Random number
ω	Inertia weight
G	Gravitational constant
β	Constant β
ε	Small constant
A_a	Assembly time
ψ	Set of components that have been assembled
W+	The sums of the positive ranks
W-	The sums of the negative ranks
χ^2_{α}.	Critical value
χ^2_F	Friedman value
ωInitial	Initial inertia weight
ωFinal	Final inertia weight

LIST OF ABBREVIATIONS

ACO	Ant colony optimization
AIS	Artificial immune system
APV	Adjusted p-value
ASP	Assembly sequence planning
AUTOPASS	Automated parts assembly system
BA	Bat algorithm
BBA	Binary bat algorithm
BCO	Bee colony optimization
BEA	Bacterial evolutionary algorithm
BGSA	Binary gravitational search algorithm
BPSO	Binary particle swarm optimization
CAD	Computer-aided design
COP	Combinatorial optimization problem
CS	Cuckoo search
DE	Differential evolution
FA	Firefly algorithm
FAS	Feasible assembly sequence
FSO	Fish swarm optimization
GA	Genetic algorithm
GDP	Geometric design processor
GLS	Guided local search
GRASP	Greedy randomized adaptive search procedure
GSA	Gravitational search algorithm
HS	Harmony search

IS	Inner state
IWD	Intelligent water drop
JTF	Japanese tree frogs
MLS	Multi-start local search
MSGSA	Multi-state gravitational search algorithm
MSGSAER	Multi-state gravitational search algorithm with an embedded rule
MHS	Mosquito host-seeking
MSPSO	Multi-state particle swarm optimization
MSPSOER	Multi-state particle swarm optimization with an embedded rule
NP	Non-deterministic polynomial-time
OS	Outer state
PADL	Part and assembly description language
PCM	Pulse code modulation
PM	Precedence matrix
PSO	Particle swarm optimization
RFD	River formation dynamic
SA	Simulated annealing
S-PSO	Set-based particle swarm optimization
TS	Tabu search
TSP	Travelling salesman problem
VNS	Variable neighbourhood search

CHAPTER 1

INTRODUCTION

1.1 Background

Combinatorial or discrete optimization is well-known in various areas such as computer science, computational intelligence, electronic commerce, assembly of products, and applied mathematics. All the problems that exist in this field, when abstracted mathematically have a commonality of discreteness (Du & Pardalos, 2005). Typically, the goal of a combinatorial optimization algorithm is to identify a solution that minimizes or maximizes a given objective function among a discrete set of feasible solutions satisfying certain conditions. For instance, one can find an answer for a question as "*what is the minimal total length for visiting each vertice exactly once for a particular instance of the Travelling salesman problem?*" This kind of problem has commonly a set of input parameters. The nature of this problem is there is a set of vertices and a set of weighted edges that offer information of two connected vertices along with its cost.

Generally, a combinatorial optimization problem can be represented using these three elements (Z, Ω, f). The definition of the elements are:

1. Z is the search space for a finite set of variables $X = \{x_1,\ldots, x_n\}$. Discrete domain variables $E_1,\ldots, E_n$ are commonly used to solve this type of problems. Concurrently, continuous domain variables are used in solving continuous optimization problem. There are problems that combine these two domain variables as well.
2. Ω is a group of constraints that is included in variables.

3. $f: Z \rightarrow IR^+$ is the objective function that appoints a real value to each solution of Z.

The goal of an optimization algorithm is to investigate a solution $s \in Z$ where $f(z) \leq f(z')$, $\forall z' \in Z$ for the case of minimization of the objective function, or $f(z) \geq f(z')$, $\forall z \in Z$ for the case of maximization of the objective function.

In the context of the combinatorial optimization problem, all possible feasible tasks are formulated as Z = $\{s = \{(x_1, q_1),\ldots,(x_n, q_n)\} | q_i \in E_i\}$. Since s satisfied all the constraints, the group that constitutes with the feasible tasks can be now defined as a potential solution. The most typical constraint is a value cannot be used more than once in representing each solution. For example, two or more variables may not have same value. These problems are usually NP-Hard (nondeterministic polynomial-time hard). In addition, numerous substantial combinatorial optimization problem showed that the problem are somewhat as the most complex problems in NP. Often, NP-Hard problem cannot be solved in polynomial time. The computational time increases exponentially with regard to the number of inputs. In contrast, any problem that can be solved in polynomial time is in P class (polynomial time).

Methods of solving combinatorial optimization problem can be categorized into these two categories; complete or approximate (Blum & Roli, 2003). Using complete algorithms, the finding of the best solution for each instance can be guaranteed in the certain limited time (Papadimitriou & Steiglitz, 1998 and Watkins, 1990). As mentioned earlier, there is no polynomial time algorithm exists for combinatorial optimization problems that are NP-hard (Garey & Johnson, 1979). The case can be critically bad because complete methods naturally consume a lot of time. As such, this characteristic causes the methods to be incapable when dealing with a great number of real-life applications. As a consequence, many researchers shift their direction to other methods associated with approximate algorithms to handle NP-hard combinatorial optimization problems. These algorithms arguably can reduce the cost of times. However, the drawback is the best solutions cannot be assured to be found (Blum & Roli, 2003 and Talbi, 2009). These approximate algorithms can be categorized into three subclasses; constructive, iterative improvement heuristics and meta-heuristics (Blum & Roli, 2003). The term heuristic is derived from the Greek words "*heuriskein*" that means "discover" or "find". Heuristics able to provide sufficient solution even on big size of problem instances. This means regardless of the size of the optimization problems, the algorithms still able to offer good solution and agreeable costs. "*Meta*" is also one of the Greek words that is translated as "in upper level". Meta-heuristics can be described as a higher level of general methodology that offers guidance with regard to the heuristic search to solve numerous optimization problems. However, meta-heuristics need significant problem specific adaptation to attain good performance (Sorensen & Glover, 2013).

Constructive heuristics produces a solution faster than iterative improvement heuristics and meta-heuristics. However, the solution is worse compared with the iterative improvement heuristics and meta-heuristics in most cases (Talbi, 2009) and this subclass of heuristics applicable to solve just a particular problem. Iterative improvement heuristics is often very fast and able to provide solutions of inferior quality but this subclass of heuristics also applicable to solve just a particular problem. Meanwhile, meta-heuristics can solve almost all optimization problems (Blum & Roli, 2003 and Talbi, 2009). Some of those meta-heuristics are random Multi-start Local Search (MLS) (Papadimitriou & Steiglitz, 1998 and Reiter & Rice, 1966), Genetic Algorithms (GA) (Holland, 1962 and Wen et al., 2011), Simulated Annealing (SA) (Kalashnikov & Kostenko, 2008), Tabu Search (TS) (Porto & Ribeiro, 1994), Ant Colony Optimization (ACO) (Tumeo et al., 2008), Artificial Immune System (Yu, 2008), Particle Swarm Optimization (Kennedy & Eberhart, 1995), Variable Neighbourhood Search (VNS) (Mladenović & Hansen, 1997), and Gravitational Search Algorithm (Rashedi et al., 2009a).

1.2 Problem Statement

Discrete search space is commonly used in discrete combinatorial optimization problems. Sequencing and routing problems are two common problems that set their space in discrete. In these two problems, the certain arrangement of discrete components are needed to ultimately produce the best sequence or solution. To solve the discrete optimization problems, binary encoding optimization algorithms and continuous optimization algorithms were used. Binary encoding optimization algorithms were operated in binary search space. Several well-known binary encoding algorithms are Genetic Algorithm (Goldberg, 1989), Ant Colony Optimization (Dorigo et al., 1996), Mosquito Host-Seeking (Feng et al., 2013), Japanese Tree Frogs (Hernandez & Blum, 2012), River Formation Dynamic (Basalo et al., 2007), and Intelligent Water Drop (Hosseini, 2009). As discrete optimization problem can be expressed in binary notation, many binary optimization algorithms were proposed based on the existing continuous optimization algorithms using activation functions. Those binary optimization algorithms include, but are not restricted to, Binary Particle Swarm Optimization (BPSO) (Kennedy & Eberhart, 1997), Binary Gravitational Search Algorithm (BGSA) (Rashedi et al., 2009b), and Binary Bat Algorithm (BBA) (Mirjalili et al., 2013). Meanwhile, there were continuous optimization algorithms that used such as a trigonometric function (Pampara, Franken, & Engelbrecht, 2005, Pampara et al., 2006), smallest position value (Tasgetiren, Evkli, Liang, & Gencyilmaz; Ucar & Tasgetiren, 2006; Yousif et al., 2011; and Verma & Kumar, 2012), modified position equation (Tasgetiren, Suganthan, & Pan, 2007 and Pan et al., 2008), nearest integer (Burnwal & Deb, 2012), random key (Baykasoglu, Ozbakir, & Tapkan, 2007; Lin et al., 2010; Chen et al., 2011; Xiangyang et al., 2011; Balasz et al., 2012 and Fister et al., 2012), great value priority (Congying, Huanping, & Xinfeng, 2011), and set-based (Wei-Neng et al., 2010).

The motivation of this research is to extend the previous work to solve some combinatorial optimization problems (COPs) in discrete space using an alternative representation based on the transition between two states; current and next state. In synchronous sequential logic of digital circuit, particularly in triggering the flip-flop, the transition between two states can be found where the information of the current or present state is used to determine the next state. It is worth pointing out that in the flip-flop, one or more candidate states that can be called multi-state are available to be selected as the next state. This shows that the behaviour of states are quite similar compared with positions in PSO and GSA, where these two algorithms have the feature of current and next position. This alternative representation differs from the previous works which employed real numbers and binary digits to represent their positions. Any optimization algorithms associated with the new representation should be able to follow the general procedure of the respective continuous optimization algorithms to search solutions.

1.3 Objectives of Research

This research operates by the guidance of the following objectives:

1. To develop a new discretization approach associated with the multi-state model based on general principle of PSO and GSA called the multi-state PSO (MSPSO) and the multi-state GSA (MSGSA) that share same features; position and velocity.
2. To extend and improve the developed algorithms through design modification or rule called the multi-state PSO with embedded rule (MSPSOER) and the multi-state GSA with embedded rule (MSGSAER).
3. To investigate the application of the multi-state optimization algorithms to solve two discrete COPs namely travelling salesman problem (TSP) and assembly sequence planning problem (ASP).

1.4 Scope of the Study

The scope of this work includes the following:

1. This research focuses on the establishment of a new approach based on transition between two states that is used to allow continuous optimization algorithms to be operated in discrete space.
2. This research proposes new variants of PSO and GSA that employ the respective mechanism of state transition as next state generator of each solution. It is assumed to be more computationally efficient than Binary PSO (BPSO) as these algorithms avoids evolving a high-dimensional bit vector. PSO and GSA are chosen because these two algorithms are relatively the most popular optimization algorithms used to solve discrete COPs.
3. In this research, the developed algorithms applications are extended to travelling salesman problem (TSP) and assembly sequence planning problem (ASP), which are two of discrete COPs. Since just these two problems are considered in this research, the developed algorithms are designed to solve problems with single objective only, but not multi-objective.

4. The performance of the develop algorithms are heavily compared with Binary PSO (BPSO) and Binary GSA (BGSA). To design BPSO and BGSA, most researchers employ activation functions that allow PSO and GSA to be operated in binary space. This approach is selected from the rest because it is a common approach used to solve discrete COPs.

1.5 Significance of the Study

The contributions of this thesis with regard to the discretization approach and performance measures for the developed algorithms namely the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER are:

1. A new mechanism based on transition between two states; current and next state is introduced. Each next state that constitutes in each dimension of a solution is selected according to its velocity. A velocity is defined as a radius that produces a circle, that is, each state exists in the circle can be selected as the next state. The circle covers many states according to the different number of dimension of each solution. In this research, the term multi-state is used to show that one or more candidate states are available to be selected as the next state. With regard to this mechanism, four algorithms have been successfully developed, namely the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER to solve discrete COPs.
2. A comprehensive overview of performance measures that currently used to measure the performance of the MSPSO, the MSPSOER, the MSGSA and the MSGSAER, compared with BPSO and BGSA to solve TSP, and BPSO, GA, and SA to solve ASP.

1.6 Thesis Outlines

This thesis consists of five chapters. Chapter 1 embarks with the presentation of research background, problem statement, objectives of research, scope, significance of the study, and organization of chapters.

Chapter 2 starts with an explanation of the basic concepts of meta-heuristics, followed by a brief discussion of the type of optimization problems. Discrete COPs are then presented, including a discussion of the numerous published to the discretization approaches. This is followed by a description of the PSO, the GSA, the BPSO, and the BGSA. Two problem domains used in this research are then elaborated; TSP and ASP. Next, the research methodology that shows the overview of this research is presented.

Chapter 3 explains the representation mechanism designed to allow the continuous PSO and the continuous GSA to be operated in discrete space. With regard to the representation, four algorithms have been proposed. The algorithms called the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER are specifically developed to deal with discrete problems as in TSP and ASP. Next, the research methodology to show the overview of how this research is conducted is presented.

In Chapter 4, optimization results of the proposed algorithms (i.e. the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER) in solving TSP are compared with results produced by their binary variant (i.e. the BPSO and the BGSA). Following that, optimization results in solving ASP using the proposed algorithms are compared with the results presented in the literature. To analyse between the results produced by the algorithms, statistical analysis is then performed in order to see if there are significance differences between algorithms and tuning parameters.

Chapter 5 discusses and concludes the contribution of the research findings to the knowledge. Finally, this chapter discusses the recommendations of the future direction of the research.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter provides a brief overview of optimization algorithms. This is followed by a brief discussion of the type of optimization problems. Discrete COPs are then discussed in details. Next explanation of a swarm'based algorithm and a physics based algorithm, which are PSO and GSA is offered respectively, followed by a brief description of two significant selected problem domains which are TSP and ASP. As aforementioned, these two problems fall into the category of discrete COPs.

2.2 Optimization Algorithms

During past decades, many optimization algorithms have been recorded to be provenly effective for a broad range of discrete COPs arising in economic studies, applied mathematics, management studies, engineering, and science (Papadimitriou & Steiglitz, 1998; Du & Pardalos, 2005 and Talbi, 2009). Various discrete COPs are known to be NP-hard. To solve the discrete COPs, consumption of calculation time increases proportionally to bigger size of instance. (Papadimitriou & Steiglitz, 1998). In general, the discrete COPs may be solved by complete algorithms or approximate algorithms. Complete algorithms obtain optimal solutions and guarantee their optimality. Furthermore, the complete algorithms are non-polynomial time algorithms (unless P = NP), by what the complete algorithms suffer with the time complexity that increase exponentially. A traditional type of complete algorithms namely dynamic programming

(Bellman, 1952), branch & cut (Mitchell, 2002), branch & bound (Land & Doig, 1960) and branch & price algorithm (Barnhart et al., 1998) modelled for operation research, a heuristic determination of minimum cost paths (Hart, Nilsson, & Raphael, 1968) and constraint programming (Rossi, Beek, & Walsh, 2006) modelled to be used in algorithms that able to find acceptable solutions and assure their optimality for each size of discrete COPs instance in a limited run-time. Meanwhile, approximate algorithms generate reasonably good solutions in a reasonable time for use in practical, but there is no guarantee of finding a global optimal solution. A lot of approximate algorithms employ random feature to produce feasible solutions. Approximate algorithms also manage to obtain many candidate solution iteratively without hardly searching all candidate solutions. In other words, this approach allows the approximate algorithms to search many good solutions for bigger size of problem instances in a reasonable amount of time.

Approximate algorithms generally can be grouped into three; constructive, iterative improvement heuristics and meta-heuristics (Balaprakash, Birattari, Stützle, & Dorigo, 2010). The constructive heuristics present a great extent of appropriateness in discrete COPs. They are somewhat quicker than iterative improvement heuristics and meta-heuristics. The strategy is the constructive heuristics generate solutions initially from scratch by adding to an initially empty partial solution components, until a solution is complete. The primary shortcoming is extremely depicted in the solution capacity they produced. It seems that the capacity of the produced solution is below par compared to the iterative improvement heuristics and meta-heuristics in most cases (Talbi, 2009). Concerning this, it is advisable to use the constructive heuristics to produce initial solutions for other algorithms for solving discrete COPs, but not as a standalone algorithm. Previous studies recorded various constructing heuristics that include the Nearest Neighbour heuristic (Rosenkrantz, Stearns, & Lewis, 2009), Construct-Strike heuristic (Christofides, 1973), Path-Scanning heuristic (Golden, Dearmon, & Baker, 1983; Evans & Minieka, 1992), Augment-Merge heuristic (Golden & Wong, 1981; Golden et al., 1983), Ulusoy's Route-First Cluster-Second Method heuristic (Ulusoy, 1985), Parallel-Insert heuristic (Chapleau, Ferland, Lapalme, & Rousseau, 1984), Modified-Construct-Strike heuristic (Pearn, 1989), Modified Path Scanning heuristic (Wohlk, 2005), Double Outer Scan heuristic (Wohlk, 2005), Node Duplication heuristic (Wohlk, 2005), Ellipse rule heuristic (Santos, Coutinho-Rodrigues, & Current, 2009), and

Improved Tour Splitting heuristic (Prins, Labadi, & Reghioui, 2009) for solving discrete COPs.

Contrary to constructive heuristics, iterative improvement heuristics that also referred to hill climbing procedures (in the case of maximization problems) embark from an existing initial solution and make an effort to enhance their performance using a specific operation until a better neighbourhood solution is found, for example, a local search. The framework of neighbourhood is so vital in the efficiency of any iterative improvement heuristic. The selection of the neighbourhood framework follows the condition on the framework of a specific COPs. In other words, each COP possesses its own neighbourhood framework. The neighbourhood solution *x'* in each COP is generally procured by involving an interchange operation of some elements in a current solution. A function N: $X \rightarrow 2^x$ for each solution $x \in X$ is set, where X is search space of solution that is also a set of neighbourhood solution $N(x) \subset X$. As discussed earlier, the fundamental of iterative heuristics put faith in the searching operation of all available neighbourhood solutions with regard to the current solution *x*. The searching operation ends when the best neighbourhood solution *x'* is successfully obtained. The characteristic of the best neighbourhood solution which normally called local optima is the solution that is better than the current solution *x*. This happens because the iterative improvement heuristics is only assured to be better than its neighbourhood. Examples of such iterative improvement heuristics are Neighbourhoods Pruning Technique (Bentley, 1992; Johnson & McGeoch, 1997 and Congram et al., 2002), and Variable Neighbourhood Descent (Hansen & Mladenovic, 2001 and Hansen & Mladenovic, 2002).

Meta-heuristics are typical algorithmic structures that are inspired by nature, biology, statistical mechanics, physics, and neuroscience, to name but a few. For example; PSO emulates the natural behaviour of the flocking of birds while searching for the food. To describe the meta-heuristics in a complete sentence, a definition taken from (Osman & Kelly, 1996) can be digested:

> *"A meta-heuristic is an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solution"*

Conceptually, a meta-heuristic refers to the high-level guidance strategy that drives other heuristics to find solutions beyond those that are typically produced in a search for local optimal solutions. The dynamic balance between the exploration and exploitation mechanism of the searching ability in predefined search space is a main character of meta-heuristics. The word of exploration can be defined as the capacity of an algorithm to search the solutions in the predefined search space, while the term of exploitation can be defined as the capacity of an algorithm to exploit the assembled search observation (Blum& Roli, 2003). Meta-heuristics mainly include SA (Kalashnikov & Kostenko, 2008), GA (Goldberg, 1989), PSO (Kennedy & Eberhart, 1995), Bee Colony Algorithm (BCO) (Karaboga, 2005), and TS (Porto & Ribeiro, 1994).

Generally, meta-heuristics do not possess any particular standard to halt the searching operation of optimization algorithms. On the other hand, the constructive heuristics just ends the searching operation when a full solution is generated. Meanwhile, the iterative improvement heuristics ends the searching operation when a local optimum is successfully found.

2.3 Type of Optimization Problems

An optimization problem can be mathematically described in various manners, relying on the fundamental application. Such function, f: $A \rightarrow Y$ is normally described throughout a domain A which is popularly recognized as the search space, and with range, Y, that informs all associating relations throughout Y based on the optimization given.

The field of optimization can be somewhat divided with regard to the form of the objective function, the nature of the search space, and the nature of the problem. An easy class is provided and summarized in Figure 2.1. Various well-being and significant fields, with reference to the form of the objective function are:

1. Linear optimization (linear programming): Conditions in which the objective function and constraints are linear are investigated.

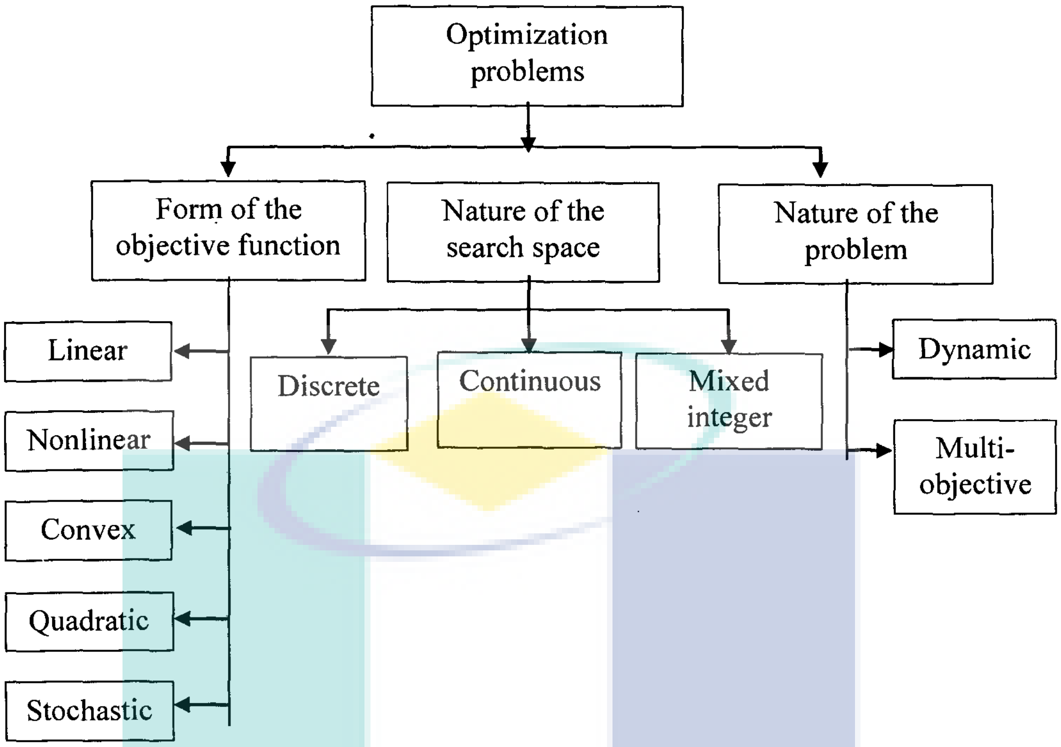


Figure 2.1. Optimization problems.

Source Parsopoulos and Vrahatis 2010

2. Nonlinear optimization (nonlinear programming): Conditions in which at least one nonlinear function is incorporated in the optimization problem are investigated.
3. Linear optimization (linear programming): Conditions in which the objective function and constraints are linear are investigated.
4. Nonlinear optimization (nonlinear programming): Conditions in which at least one nonlinear function is incorporated in the optimization problem are investigated.
5. Quadratic optimization (quadratic programming): It minimizes quadratic objective functions and linear constraints.
6. Convex optimization: Problems associated with convex objective functions and convex feasible sets are investigated.
7. Stochastic optimization: It concerns to minimization in the existence of randomness, in which presented in two ways; variables and parameters of

problems are probabilistically selected, and noise produced from the evaluation of function, with regard to the distributions statistically.

There is a large number of wide-ranging works on the previously specified optimization problems. Several of the most extensively-utilized resources are (Luenberger, 1989; Torn & Žilinskas, 1989; Horst & Pardalos, 1995; Polak, 1997; Nocedal& Wright, 1999; Zhigljavsky &Zhilinskas, 2002 and Horst & Tuy, 2003).

Normally, optimization problems are designed with an objective function that stays unchanged over time. In real-world, various engineering problems are designed one or a set of static or time-varying objective functions that require to be concurrently optimized. Effect of the scenario is the emergence of the coming significant optimization fields (Chiang, 1992; Deb, 2001; Branke, 2002 and Ehrgott & Gandibleux, 2002):

1. Dynamic optimization: This type of optimization field concerns on the minimization of time-varying objective functions. The first objective is to find the position of the global optimizer subsequently the global optimizer moves in the search space. The second objective is to contribute robust solutions. If an improvement in a small change occurred in the objective function, solutions are produced in inexpensive costs of times.
2. Multi-objective optimization (multiple criteria optimization): Two or more objective functions of problems are required to be minimized or maximized at the same time. When dealing with this type of problems, the optimality of solutions is redefined as global minima of variant objective functions are hardly ever accomplished at the similar minimizers.

A diversity of optimizations can be also constructed based on the nature of problem variables and the search space as well (Floudas, 1995 and Boros & Hammer, 2003):

1. Continuous optimization: All variables used in a particular objective function expect in the form of real values.

2. Discrete optimization: In these problems, an assumption should be made where all variables used in a particular objective function are in discrete. Integer variables that are also discrete are belong to integer optimization.
3. Mixed integer optimization: Two types of variables, namely integer and real variables occur in the representation of objective function.

There is an overabundance of methods that designed to easily solve the problems in majority of the aforesaid categories. Nonetheless, most of these methods are constructed using solid mathematical presumptions that do not fit in many applications. For instance, there are many efficient deterministic algorithms are designed for handling nonlinear optimization problems; but at the same time the algorithms need to hold some features namely convexity or differentiability of the objective function. Unfortunately, these features are not met in various significant applications. In addition, there is objective function of some problems which is not even analytically established, with its values being obtained through complex computer software or assessments yielding from observation equipment.

2.4 Discrete Combinatorial Optimization Problems

Various optimization problems are designated in a continuous space. To cope with these problems, meta-heuristics are represented in continuous encoding. Examples of the continuous meta-heuristics are SA (Kirkpatrick et al., 1983), Tabu Search (Glover, 1986), Guided Local Search (GLS) (Voudouris & Tsang, 1999 and Voudouris et al., 2010), Greedy Randomized Adaptive Search Procedure (GRASP) (Feo & Resende, 1995), Variable Neighbourhood Search (Mladenović & Hansen, 1997), Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995), Gravitational Search Algorithm (GSA) (Rashedi, Nezamabadi-pour, & Saryazdi, 2009a), Differential Evolution (DE) (Fleetwood, 1999), Bee Colony Optimization (BCO) (Karaboga, 2005), Cuckoo Search (CS) (Yang & Deb, 2009), Bat Algorithm (BA) (Yang, 2013), Harmony Search (HS) (Yang, 2009), Bacterial Evolutionary Algorithm (BEA) (Nawa & Furuhashi, 1999), Fish Swarm Optimization (FSO) (Li, Shao, & Qian, 2002), and Firefly Algorithm (FA) (Yang, 2008).

Although the continuous meta-heuristics are continuous by nature, some of these algorithms are adapted to deal with discrete domain problems. In the continuous domain, the elements have the property of varying smoothly, while the elements of a discrete domain (i.e. integers or binary digits) tolerate only distinct and separated values. The discrete domain is distinguished by handling countable sets, either finite or infinite. Two examples of discrete domain problems are binary and combinatorial applications. These two examples involve problems that need the ordering or arranging of discrete elements, such as scheduling and routing problems.

There are some of algorithms that are created specifically to handle discrete problems using the binary codification, namely Genetic Algorithm (GA) (Goldberg, 1989), Ant Colony Optimization (ACO) (Dorigo, Maniezzo, & Colorni, 1996), Mosquito Host-Seeking Algorithm (Feng, Lau, & Yu, 2013), the Calling Behaviour of Japanese Tree Frogs Algorithm (Hernández & Blum, 2012), the River Formation Dynamics Algorithm (Basalo, Laguna, & Rubio Díez, 2007), and the Intelligent Water Drops Algorithm (Hosseini, 2009). Swarm algorithms that use the binary codification are adapted to discrete domain since its beginning (i.e., initial swarm). Other than the binary-encoded algorithms, a variety of the discretization approaches are used to adjust the continuous formulated algorithms to discrete optimization problems (Krause, Cordeiro, Parpinelli, & Lopes, 2013). The methods that consist of the binary-encoded algorithms and the continuous algorithms that make use of various discretization approaches to solve discrete COPs are portrayed in Figure 2.2. These discretization approaches namely activation function, smallest position value, modified position equation, random-key, great value priority, nearest integer, set-based, and trigonometric function are then shown in Figure 2.3.

2.41 Discretization Approaches

Discretization approaches offer a significant aid to transform continuous space to discrete space. There are a few discretization approaches, and the primary are presented in this sub-section, namely activation function, smallest position value, modified position equation, nearest integer, great value priority, angle modulated, random-key, and set-based.

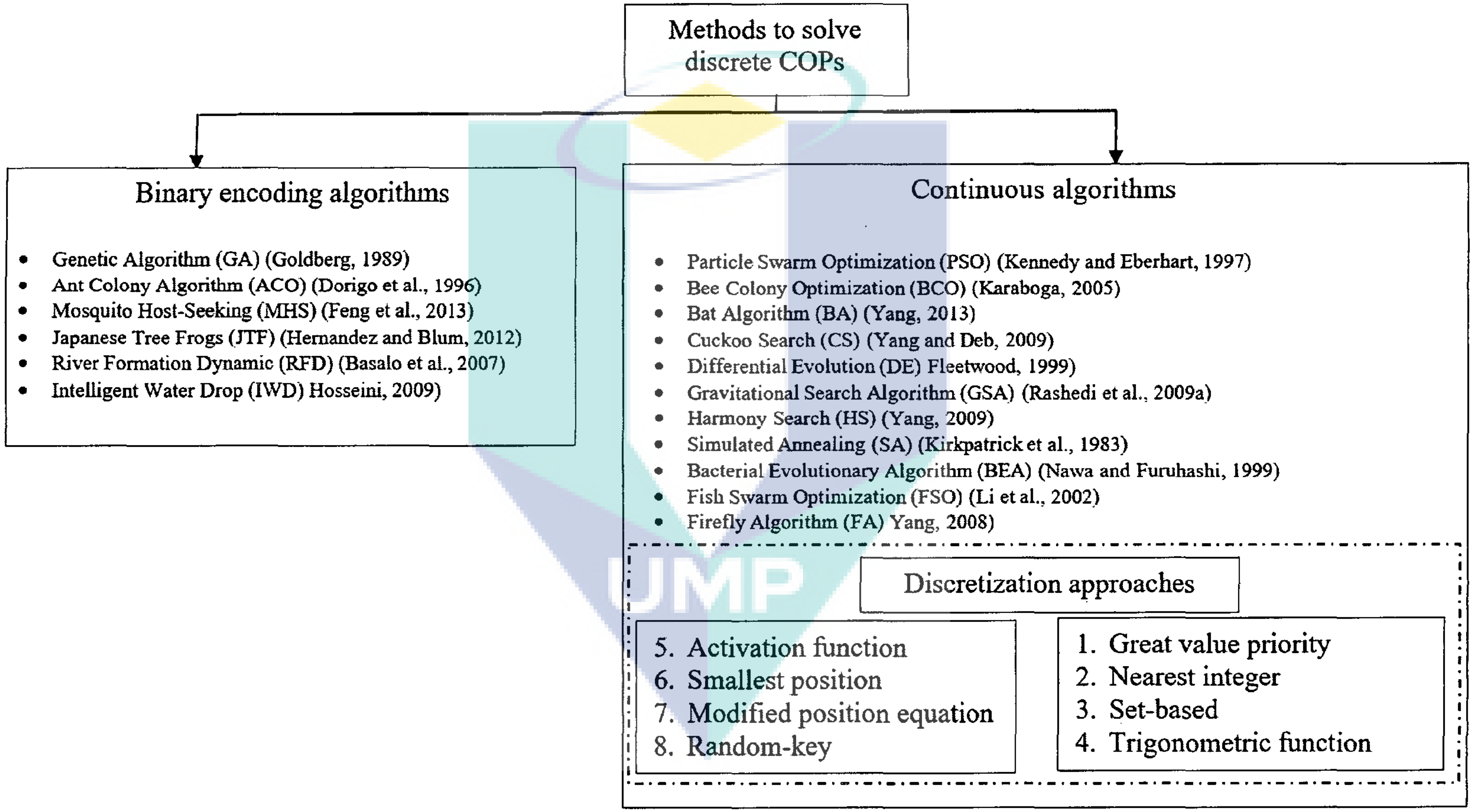

Methods to solve discrete COPs
Binary encoding algorithms
Genetic Algorithm (GA) (Goldberg, 1989)
Ant Colony Algorithm (ACO) (Dorigo et al., 1996)
Mosquito Host-Seeking (MHS) (Feng et al., 2013)
Japanese Tree Frogs (JTF) (Hernandez and Blum, 2012)
River Formation Dynamic (RFD) (Basalo et al., 2007)
Intelligent Water Drop (IWD) Hosseini, 2009)
Continuous algorithms
Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1997)
Bee Colony Optimization (BCO) (Karaboga, 2005)
Bat Algorithm (BA) (Yang, 2013)
Cuckoo Search (CS) (Yang and Deb, 2009)
Differential Evolution (DE) Fleetwood, 1999)
Gravitational Search Algorithm (GSA) (Rashedi et al., 2009a)
Harmony Search (HS) (Yang, 2009)
Simulated Annealing (SA) (Kirkpatrick et al., 1983)
Bacterial Evolutionary Algorithm (BEA) (Nawa and Furuhashi, 1999)
Fish Swarm Optimization (FSO) (Li et al., 2002)
Firefly Algorithm (FA) Yang, 2008)
Discretization approaches
1. Great value priority
2. Nearest integer
3. Set-based
4. Trigonometric function
5. Activation function
6. Smallest position
7. Modified position equation
8. Random-key

Figure 2.2. Methods to solve discrete COPs.

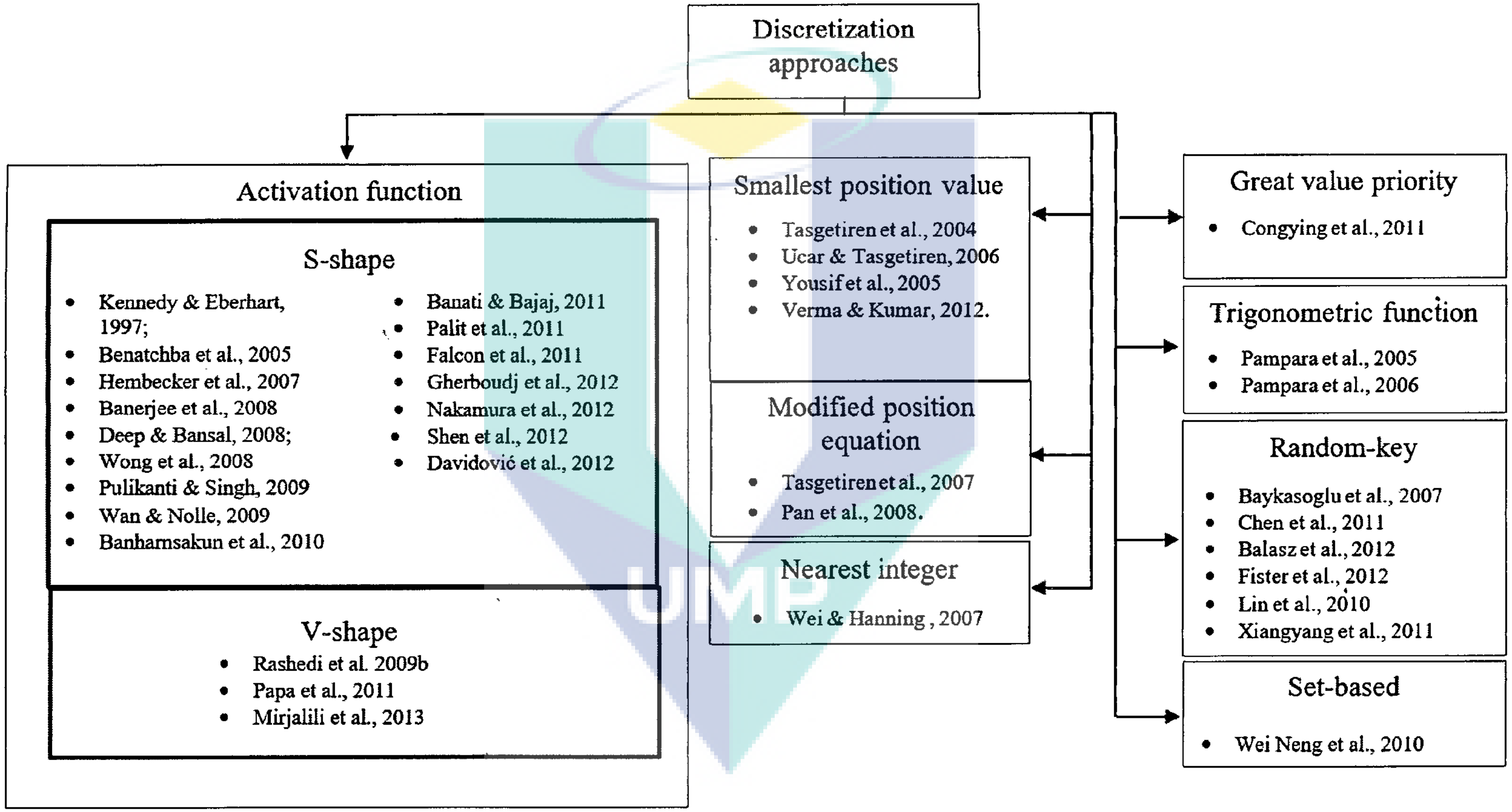


Figure 2.3. Previous studies for different discretization approaches.

1. Activation function: The activation function is needed to change a continuous space value into a binary one. This discretization approach is very common (Banati & Bajaj, 2011 and Palit et al., 2011) and the transformation is assigned to each dimension of the solution vector. As a consequence, each element is formed by binary digit. For instance, activation function that is used in the PSO algorithm (Kennedy and Eberhart, 1997) define the probability of changing position vector's element from “0”and “1” and vice versa. The activation function forces particles to move in a binary space. Two common functions used are S-shape function and V-shape function. There are several optimization algorithms that uses the S-shape function, such as (Kenneddy & Eberhart, 1997; Benatchba et al., 2005; Hembecker et al., 2007; Banerjee et al., 2008; Deep & Bansal, 2008; Wong et al., 2008; Pulikanti & Singh, 2009; Wan& Nolle, 2009; Banharnsakun et al., 2010; Banati & Bajaj, 2011; Palit et al., 2011; Falcon et al., 2011; Nakamura et al., 2012; Shen et al., 2012; Davidović et al., 2012 and Gherboudj, Layeb, & Chikhi, 2012). On the other hand, some optimization algorithms use the V-shape function, namely (Rashedi, Nezamabadi-pour, & Sayyazdi, 2009b; Papa et al., 2011; Mirjalili et al., 2013).
2. Smallest position value: The smallest position value approach plots the positions of the solution vector by putting the index of the smallest valued component as the first element on a permutated solution, the second smallest as the second, and so on. As a result, an integer vector solution is successfully created by indexing the position of all the particles (Tasgetiren, Evkli, Liang, & Gencyilmaz, 2004; Ucar & Tasgetiren, 2006; Yousif et al., 2011 and Verma & Kumar, 2012).
3. Modified position equation: Up-to-date, the modified position equation approach are solely used in the PSO algorithm (Tasgetiren, Suganthan, & Pan, 2007 and Pan et al., 2008). This method generates a uniform random number in the range [0, 1] and, if the value is smaller than inertia weight, the mutation operator is then applied to produce a perturbed permutation of the particle; otherwise current permutation is preserved.
4. Random-key: The random-key encoding approach is used to transform a position to an integer or combinatorial space from a continuous space. To interpret the position, the points are inspected in ascending order for each dimension. Those algorithms that are employed this method are (Baykasoglu, Ozbakir, & Tapkan,

2007; Lin et al., 2010; Chen et al., 2011; Xiangyang et al., 2011; Balasz et al., 2012 and Fister et al., 2012).

5. Great value priority: The great value priority scheme is used to transform a continuous space intó a binary one (Congying, Huanping, & Xinfeng, 2011). The position of the solution vector with the biggest element is firstly chosen. This position is put on the first position of a new vector called as permutation vector. The position of the second biggest element of the solution vector is then chosen and set in the next position of permutation vector. This process is repeated consecutively for all dimensions of the solution vector and, once the permutation vector is completed, the first and the second dimension of the permutation vector is then compared. If the value of the first dimension of the permutation vector is bigger than the value of the second dimension, the solution vector is set to 1, otherwise the solution vector is set to 0. This process is repeated to the second and the third dimension, the third and the fourth dimension, and so on.

6. Nearest integer: This approach converts a real value to the nearest integer either by rounding or trimming up or down (Burnwal & Deb, 2012).

7. Set based: A set-based PSO (S-PSO) approach is invented to spread the capability of PSO to handle various optimization problems in space featuring discrete (Wei-Neng et al., 2010) without using real-value and binary notation. With regard to the concept of sets and the possibility theory, S-PSO possesses two unique attributes. A set-based representation scheme is firstly modelled to distinguish the discrete search space as a universal set of components. Next, each possible solution in S-PSO correlates to a crisp subset out of the universal set. In the S-PSO, each component of a velocity is appointed with a possibility. The operators and procedures that defined on crisp sets and sets with possibilities are employed to modify associated arithmetic operators used in the formulation of velocity and position as recorded in the canonical PSO. The attributes facilitate S-PSO to comply with the framework of the canonical PSO to find solutions in a predefined discrete space.

8. Trigonometric function: The trigonometric function is firstly introduced by Pampara, Franken, and Engelbrecht (2005) called angle modulated PSO. In this scheme, the PSO algorithm is used to find the optimal coefficients of a trigonometric bit generating function using four coefficients, namely a, b, c, and d which use to control the form of the function. Coefficient a controls the

horizontal shift of the whole function. Meanwhile, coefficient b affects the frequency of the sin wave and controls the amplitude of the cos wave as well. Coefficient c and d affects the frequency of the cos wave and controls the vertical shift of the whole function respectively. To generate binary solutions, the generating function goes through sampling process at particular intervals in which the length of the required binary solution is the value of last interval. One more algorithm that employs this scheme is angle modulated differential evolution (DE) (Pampara, Engelbrecht, & Franken, 2006).

2.5 Particle Swarm Optimization (PSO)

2.5.1 PSO Background

Particle Swarm Optimization (PSO) is a population-based stochastic algorithms used to search solutions that are optimal (or near optimal) for optimization problems. Such algorithm that easily used (a few lines of code is just necesssary to write the code of this algorithm) can be awarded as a remarkable, virtuous and fast algorithm for different optimization problems.

Particle swarm optimization (PSO) algorithm is an optimization algorithm proposed by Kennedy and Eberhart (1995). It imitates swarms behavior as shown by a flock of bird and a swarm of fishes to search an optimal solution subjected to an objective function. This original PSO is predominantly utilized to discover solutions for continuous optimization problems.

2.5.2 Original PSO Algorithm

In the original PSO algorithm, an optimal or good enough solution is found by simulating social behaviour of bird flocking. The PSO algorithm consists of a group of individuals named particles which encode the possible solutions to the optimization problem using their positions. The group can attain the solution effectively by using the common information of the group and the information owned by the particle itself, which each particle share its current position to neighbouring particles. Using this information, each particle compares its current position with the best position found by its neighbours so far.

Optimization encompasses both minimization and maximization problems. Any maximization problem can be converted into a minimization problem by taking the negative of the objective function, and *vice versa*. Figure 2.4 portrays the pseudocode of the PSO algorithm. Consider the following optimization problem: there are *I*-particles flying around in a *D*-dimensional search space, where their position, $\boldsymbol{s}_i(d)$ ($i = 1,2,\ldots,I$; $d = 1,2,\ldots,D$), represent the possible solutions and *d* represents the dimension number. First, all particles are randomly positioned in the search space, and then designated with velocity $\boldsymbol{v}_i(k,d) = 0$, where *k* represents the iteration number. Next, the objective fitness $\vec{\boldsymbol{F}}_i(k)$, for each particle is evaluated by calculating the objective functions with respect to $\boldsymbol{S}_i(k)$. Each particle's best position is $\boldsymbol{pbest}_i(k)$ then initialized to its current position. The global best among the all $\boldsymbol{pbest}_i(k)$ is called $\boldsymbol{gbest}(k)$, is chosen as the swarm's best position. For minimization problem, $\boldsymbol{gbest}(k)$ is given as in Eq. (2.1). Meanwhile, for maximization problem, $\boldsymbol{gbest}((k)$ is given as in Eq. (2.2). In these two equations, $\boldsymbol{S}$ is the swarm of particles. Subsequently, the algorithm run until the stopping criteria is met, either a predefined acceptable amount of error is reached or the maximum number of iteration is used up.

$$gbest = \left\{ pbest_i \in S \,|\, f\left(pbest_i\right) = \min f\left(\forall pbest_i \in S\right) \right\} \tag{2.1}$$

```
procedure PSO
    Initialize particles with random positions and velocities.
    Set particles' pbest to their current positions.
    Calculate particles' fitness and set gbest.
    for T generations do
        Update particles' velocities.
        Update particles' positions.
        Recalculate particles' fitness.
        Update particles' pbest and gbest.
    end for
end procedure PSO
```

Figure 2.4. PSO algorithm.

$$\boldsymbol{gbest} = \{\boldsymbol{pbest}_i \in \boldsymbol{S} | f(\boldsymbol{pbest}_i) = \max f(\forall \boldsymbol{pbest}_i \in \boldsymbol{S})\} \tag{2.2}$$

Each iteration updates each particle's velocity and position using Eq. (2.3) and Eq. (2.4), respectively, where c_1 and c_2 are the cognitive and social coefficients, respectively. r_1 and r_2 are random number uniformly distributed between 0 and 1, and ω is called inertia weight, which used to control the impact of the previous history of velocities on the current velocity of each particle. After updating the velocity and position, $\vec{\boldsymbol{F}}_i(k)$ for each particle is calculated again. $\boldsymbol{pbest}_i(k)$ is then updated by a more optimal, obtained either from the new position of the i^{th} particle or $\boldsymbol{pbest}_i(k)$. The $\boldsymbol{gbest}((k)$ is also updated by the most optimal $\boldsymbol{pbest}_i(k)$ of all the particles, as denoted in Eq. (2.1) and Eq. (2.2). Finally, the best solution of the problem represented by $\boldsymbol{gbest}((k)$ is yielded when the stopping condition is met.

$$\begin{aligned} v_i(k+1,d) = \omega v_i(k,d) + c_1 r_1 (\boldsymbol{pbest}_i)(k,d) - \boldsymbol{s}_i(k,d) \\ + c_2 r_2 (\boldsymbol{gbest}(k,d) - \boldsymbol{s}_i(k,d) \end{aligned} \tag{2.3}$$

$$\boldsymbol{s}_i(k+1,d) = \boldsymbol{s}_i(k,d) + \boldsymbol{v}_i(k+1,d) \tag{2.4}$$

2.5.3 Binary Particle Swarm Optimization (BPSO) Algorithm

The original version of PSO algorithm is designed for search spaces of real valued vectors. A version of PSO algorithm for binary encoding is presented (Kennedy & Eberhart, 1997) to solve many optimization problems that are set in binary discrete space. The BPSO algorithm preserves the fundamental concept of PSO algorithm except that each particle in a swarm consists of binary string representing a particle's position vector.

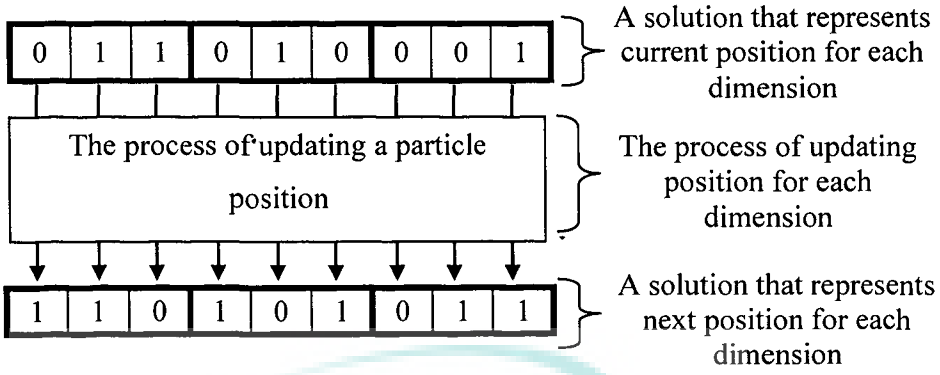

0 1 1 0 1 0 0 0 1
A solution that represents current position for each dimension
The process of'updating a particle position
The process of updating position for each dimension
1 1 0 1 0 1 0 1 1
A solution that represents next position for each dimension

Figure 2.5. Process of updating position in the BPSO algorithm.

This algorithm uses the concept of velocity as a probability that a bit flips to one or zero. The velocity equation remains unchanged, except Eq. (2.4) is redefined to:

$$s(k+1,d) = \begin{cases} 1 \text{ if } r_3 \leq H(v_i(k+1,d)), \text{ where } H(v_i(k+1,d)) = \frac{1}{1+e^{-v_i(k+1,d)}} \\ 0 \qquad\qquad\qquad \text{otherwise} \end{cases} \tag{2.5}$$

where r_3 is a random number uniformly distributed between 0 and 1, and $H(v_i(k+1,d)$ is a sigmoid function for transforming the velocity to be a probability value constrained to the interval [0.0, 1.0] that indicates the probability of the corresponding element of solution assuming the value 1. As a consequence, the particle's position now consists of the solution component, either 1 or 0, as shown in Figure 2.5.

2.6 Gravitational Search Algorithm (GSA)

2.6.1 GSA Background

In year 2009, a stochastic population-based metaheuristic, called Gravitational Search Algorithm (GSA) has been introduced (Rashedi, Nezamabadi-pour, & Saryazdi, 2009a) based on Newton's law of universal gravitation. This law indicates that all objects attract each other using a gravitational force. The first characteristic of the gravitational force between two objects is it directly proportional to the product of their masses. Heavier objects will attract each other with a bigger gravitational force.The second characteristic of the gravitational force is it inversely proportional to the distance between the two objects. The original version of GSA was originally modelled to deal with problems in space featuring real valued vectors.

2.6.2 Original GSA

Figure 2.6 portrays the pseudocode of the GSA. The GSA starts with a set of agents, which are randomly positioned in the search space. Let a system with I agents (masses), the position of agents that represent possible solutions to the problem can be defined as $\boldsymbol{s}_i(d)$ ($i = 1,2,\ldots,I$; $d = 1,2,\ldots,D$), where $\boldsymbol{s}_i(d)$ is the position of i^{th} agent in the d^{th} dimension. All agents are then assigned with random velocity $\boldsymbol{v}_i(k,d)$, where k represents the iteration number. Next, the objective fitness $\vec{\boldsymbol{F}}_i(k)$ for each agent is evaluated by calculating the objective function with respect to $\boldsymbol{S}_i(k,d)$, where $\vec{\boldsymbol{F}}_i(k)$ represent the fitness value of the agent i at k. The gravitational constant $G(k)$ is then updated based on Eq. (2.6). The gravitational constant is a decreasing function of time where it is set to G_0 at the beginning and is exponentially decreased towards zero at the last iteration to control the search accuracy.

```
procedure GSA
    Initialize agents with random positions and velocities.
    Set agents to their current positions.
    Calculate agents' fitness and set best agent and worst agent
    for K generations do
        Update the G, best agent and worst agent.
        Evaluate mass.
        Evaluate force of mass.
        Evaluate acceleration of mass.
        Update agents' velocities.
        Update agents' positions.
        Recalculate agents' fitness.
        Update the G, best agent and worst agent.
    end for
end procedure GSA
```

Figure 2.6. GSA.

$$G(\mathrm{k}) = G_0 e^{-\beta\frac{k}{K}} \tag{2.6}$$

where K is the maximum number of iteration, G_0 and β are constant value.

Next, *best*(k) and worst(k) are calculated. For minimization problem, the definition of *best* (k) and worst (k) are given as in Eq. (2.7) and Eq. (2.8).

$$best(k) = \min_{j\in\{1,\ldots,I\}} \vec{\boldsymbol{F}}_i(k) \tag{2.7}$$

$$worst(k) = \max_{\in\{1,\ldots,I\}} \vec{\boldsymbol{F}}_i(k) \tag{2.8}$$

For a maximization problem, the definition of *best*(k) and worst(k) are changed to:

$$best(k) = \max_{j\in\{1,\ldots,I\}} \vec{\boldsymbol{F}}_i(k) \tag{2.9}$$

$$worst(k) = \min_{j\in\{1,\ldots,I\}} \vec{\boldsymbol{F}}_i(k) \tag{2.10}$$

The gravitational and inertial mass are then updated using following equations:

$$m_i(k) = \frac{\vec{\boldsymbol{F}}_i(k) - worst(k)}{best(k) - worst(k)} \tag{2.11}$$

$$M_i(k) = \frac{m_i(k)}{\sum_{j=1}^{I} m_j(k)} \tag{2.12}$$

$M_{ii}(k)$ is the inertial mass of i^{th} agent. The acceleration α related to mass i in k and the d^{th} dimension is calculated as follows:

$$\alpha_i(k) = \frac{F_i(k,d)}{M_{ii}(k)} \tag{2.13}$$

The total force $\boldsymbol{F}_i(k,d)$ and the force acting on mass i from mass j $\boldsymbol{F}_{ij}(k,d)$ are calculated as follows:

$$\boldsymbol{F}_i(k,d) = \textstyle\sum_{j=1,\, j\neq 1}^{I} rand_j\, \boldsymbol{F}_{ij}(k,d) \tag{2.14}$$

$$\boldsymbol{F}_{ij}(k,d) = G(k)\frac{M_{aj}(k)}{R_{ij}(k)+\varepsilon}((\boldsymbol{s}_j(k,d - \boldsymbol{s}_i(k,d)) \tag{2.15}$$

where M_{aj} is the active gravitational mass related to agent j, ε is a small constant, R_{ij} is the Euclidean distance between agent i and j, and $rand_j$ is a random number uniformly distributed [0, 1]. Subsequently, the algorithm run until the stopping criteria is met, either a predefined acceptable amount of error is reached or the maximum number of iteration is used up.

$$v_i(k+1,d) = rand_i \times v_i(k,d) + \alpha_i(k) \tag{2.16}$$

$$s_i(k+1,d) = s_i(k,d) + v_i(k+1,d) \tag{2.17}$$

Each iteration updates each agent's velocity and position using Eq. (2.16) and Eq. (2.17), respectively, where $rand_i$ is random number uniformly distributed [0, 1]. After updating the velocity and position, $\vec{F}_i(k)$ for each agent is calculated again. Consider a minimization problem, the *best* of the population is then updated by a more minimal, obtained either from the new position of the i^{th} agent or the current *best*. The *worst* of the population is also updated by a more maximal, obtained either from the new position of the i^{th} particle or the current *worst*. Consider a maximization problem, the *best* of the population is then updated by a more maximal, obtained either from the new position of the i^{th} agent or the current *best*. The *worst* of the population is also updated by a more minimal, obtained either from the new position of the i^{th} particle or the current *worst*. Finally, the best solution of the problem represented by the *best* is yielded when the stopping condition is met.

2.6.3 Binary Gravitational Search Algorithm (BGSA)

The original version of GSA was designed for search spaces of real valued vectors. A version of GSA for binary encoding was presented (Rashedi et al., 2009b) to solve many optimization problems that are set in binary discrete space. The BGSA preserves the fundamental concept of the GSA except that each agent of a swarm consists of binary string representing an agent's position vector. The BGSA updates each agent's mass velocity using Eq. (2.16) and then updates each agent's position to be either 1 or 0 with a given probability based on Eq. (2.18), as for small $|v_i(d)|$, the probability of changing of an agent's position must be near zero and for a large $|v_i(d)|$, the probability of changing of an agent's position movement must be high. In addition, R_{ij} in the BGSA is the Hamming distance between two agents i and j. All agents are then moved according to Eq. (2.19) after $H(v_i(k+1,d))$ has been calculated using Eq. (2.18).

$$H(v_i(k+1,d)) = |\tanh|(v_i(k,d)) \qquad (2.18)$$

$$\begin{aligned} s_i(k+1,d) &= \text{complement } s_i(k,d) \text{ if } rand < H(v_i(k+1,d)) \\ &\text{else } s_i(k+1,d) = s_i(k,d) \end{aligned} \qquad (2.19)$$

where *rand* is a random number uniformly distributed between 0 and 1, and $H(v_i(k+1,d))$ is a logistic function for transforming the velocity to be a probability value constrained to the interval [0.0, 1.0], indicating the probability of the corresponding element of solution

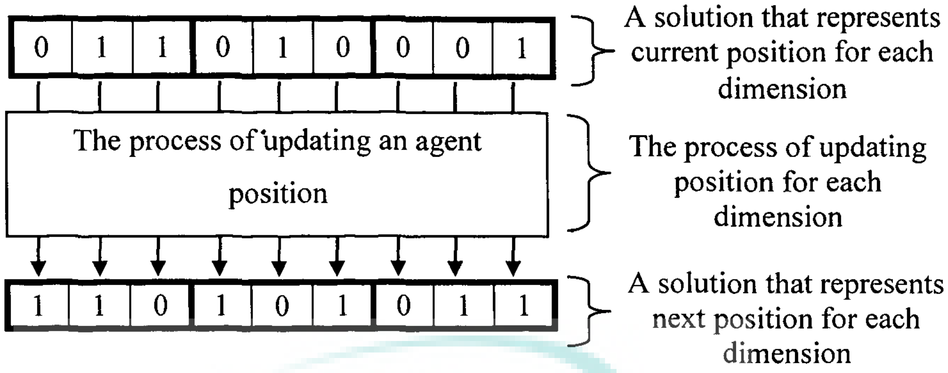

0 1 1 0 1 0 0 0 1
A solution that represents current position for each dimension
The process of updating an agent position
The process of updating position for each dimension
1 1 0 1 0 1 0 1 1
A solution that represents next position for each dimension

Figure 2.7. Process of updating position in the BGSA.

assuming the value 1. As a consequence, the agent's position $s_i(k+1,d)$ now consists of the solution component, either 1 or 0, as shown in Figure 2.7.

2.7 Domain Problems

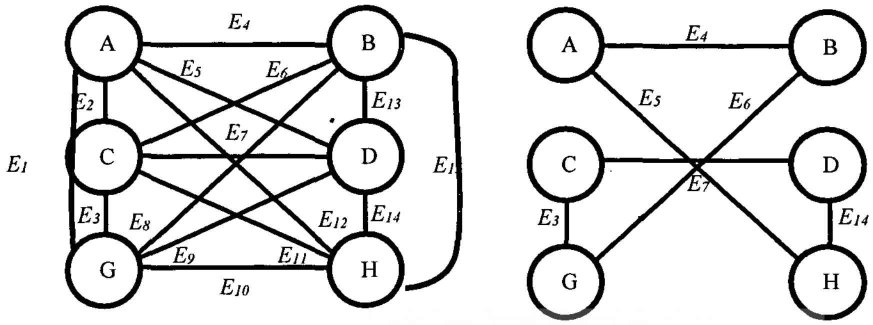
This section describes two problem domains that used in this study; TSP and ASP. Concerning this, the following sub-section discusses the fundamental information of each problem domain.

2.7.1 TSP

TSP is a common type of routing problem, which is one of the most popular problem in the context of COPs. The TSP can be defined using three features; an agent, a goal, and a constraint. A salesman (agent) starts his travel from a city to the other city and then to other consecutive cities to find the shortest path (goal) with a cost associated between each pair of city. The salesman can go through each city just once (constraint) (Lawler, Lenstra, Rinnooy, & Shmoys, 1985). Because the basic of the TSP is easy to learn, many researchers dedicated their works to design algorithms to solve this problem until now. These algorithms can be categorized into two categories of algorithms; complete and approximation algorithms. Many discoveries were recorded for solving the TSP which is proportional with many years spent. The maturity of the TSP is very helpful in designing new algorithms to solve other COPs in the fields of economic studies, managament studies, applied mathematic, engineering, and science.

The history of the TSP began in year 1800s with a creation of a puzzle game called Hamilton's Location Game by William Hamilton and Thomas Kinkman. The goal in solving the puzzle is to identify the shortest Hamiltonian cycle. The circle was built using a number of pair wise points that was finite associated with predefined distances. Hamiltonian cycle is actually a route in an undirected graph. Each point can just be visited once and the ending point should be similar with the starting point. The Hamiltonian cycle problem was then spreaded into the form of an optimization problem. Starting from this point, many researchers especially mathematicians extremely focused studying this problem. In year 1930, Karl Menger and his fellows introduced the Messenger Problem, that latter on called by the name of "Travelling Salesman Problem (TSP)" by Hassler Whitney from Princeton University in year 1934. The TSP was proven as a NP-hard problem by Richard M. Karp in year 1972, referring to the complexity of the TSP which it is difficult to obtain an optimal route. This discovery indicated that there were no exact algorithms that can solve the TSP benchmark instances in polynomial time.

The TSP can be also represented by a completed weighted graph G= (V, E), where V is a set of vertices representing citis, while E is a set of edges or arcs that are weighted by values. Each edge (I, J) $\in$ A is designated with a distance value or length between two cities *i* and *j* where *i* and *j* $\in$ V. The symmetrical feature makes the distance from *i* to *j* is equal to the distance from *j* to *i*. Using information of each pair of cities, the minimal length of Hamiltonian cycle in the graph can be now found. An example of the TSP directed graph is provided in Figure 2.8. The figure shows two plots. The plot at the left hand side presents the graph representation of a TSP problem. In the graph, a group of cities namely *A*, *B*, *C*, *D*, *G*, and *H*, are connected by undirected weighted edges namely E_1, E_2, E_3..., E_{14}. Meanwhile, the plot at the right hand side portrays one of many possible cities route of the problem presented in the left hand side plot. The possible route represents a path of the Hamiltonian cycle that involves edges E_3, E_4, E_5, E_6, E_6 and E_{14}.



a) Complete graph of TSP b) TSP cities tour

Figure 2.8. Example of the TSP graph.

Some of the TSP benchmark instances are represented in Euclidean plane. Such benchmark instances include Eil51 and Berlin52. Let considering each two cities connected in Eil51 and Berlin52 benchmark instances, the first city and second city are defined by two points (u_1, q_1) and (u_2, q_2), respectively in the Euclidean plane. Euclidean distance (*euc_dist*) between the two connected cities is formulated as in Eq. (2.20).

$$euc\_dist_{e_i,e_j} = \sqrt{(u_1 - u_2)^2 + (q_1 - q_2)^2} \qquad (2.20)$$

Meanwhile, some of the TSP benchmark instances should be calculated using geographical distance (*geog_dist*). Such benchmark instances includes Burma14, Ulysses16, Ulysses22, and Bays29. Consider two connected cities of geographical TSP benchmark instances. The first and city second city are firstly defined by two points (u_1, q_1) and (u_2, q_2). The coordinate of the two cities are then converted to latitude and longitude format, so that the new representation of the two cities are (lat_1, $long_1$) and (lat_2, $long_2$). These conversions can be done using Eq. (2.21), Eq. (2.22), Eq. (2.23), and Eq. (2.24).

$$lat_1 = \pi \frac{(floor(u_1)+5(x_1-floor)(u_1))/3}{180} \qquad (2.21)$$

$$lat_2 = \pi \frac{\left(floor(u_2)+5(u_2-floor)(u_2)\right)/3}{180} \tag{2.22}$$

$$long_1 = \pi \frac{\left(floor(q_1)+5(q_1-floor)(q_1)\right)/3}{180} \tag{2.23}$$

$$long_2 = \pi \frac{\left(floor(q_2)+5(q_2-floor)(q_2)\right)/3}{180} \tag{2.24}$$

The geographical distance (*geog_dist*) between the two connected cities is then denoted as in Eq. (2.25), where *floor* changes a value to be the largest integer smaller than the value, the value of radian is 6378.3888, acos is the inverse of the cosine function, the value of π is 3.141592, g_1 is cos ($long_1 - long_2$), g_2 is cos ($lat_1 - lat_2$) and g_3 is cos ($lat_1 + lat_2$).

$$geog\_dist_{e_i,e_j} \tag{2.25}$$

$$= floor\left(\left(radian \times acos\left(0.5 \times \left((1+g_1) \times g_2 - (1 - g_3) \times g_3\right)\right) + 1\right)\right)$$

2.7.2 ASP

The costs of assembly processes are determined by assembly plans. ASP, which is an important part of assembly process planning, plays an essential role in the manufacturing industry. Given a product-assembly model, the ASP determines the sequence of component installation to shorten assembly time or save assembly costs. The ASP is regarded as a large-scale, highly constrained combinatorial optimization problem because it is nearly impossible to generate and evaluate all assembly sequences to obtain the optimal sequence, either with human interaction or through computer programs.

Historically, the typical combinatorial explosion problem requires experienced assembly technicians to determine assembly plans. This manual assembly planning approach thus requires significant time investments and does not allow quantitative analysis of assembly costs before production begins. Thus, many studies in the last two decades have focused on geometric reasoning capabilities and full automatism to locate more efficient algorithms for automated ASP. The approaches used for assembly sequence planning can be categorized into four groups, which are:

1. **Graph-based representation.** In this representation, the data source originates from the user or a CAD system. Using this approach, many details of the assembly analysis can be determined. Mello and Sanderson (1990) and Zhang, (1989) proposed graph-based representation methods based on AND/OR and directed graph. Lee and Shin (1990), Moore et al. (2001) and Zha (2000) proposed graph-based representation methods based on PETRI nets.
2. **Lingual representation**. This representation uses a special language to represent subassemblies, their parts and the relations between them. A few approaches include but are not restricted to PADL, AUTOPASS and GDP (Mello & Sanderson, 1990).
3. **An ordered list representation**. This type of representation can be categorized as an ordered list of task representations, binary vectors, partitions of the set of parts and connections. Garrod (1989) represented each assembly sequence in the form of a set of list.
4. **Meta-heuristics based representation**. Meta-heuristic approaches include but are not restricted to the rule-based method (Chakrabarty & Wolter, 1997), heuristic search (Hong & Cho, 1995), neural networks (Chen, Tai, Deng, & Hsieh, 2008 and Hsin-Hao et al., 2000), genetic algorithms (Bonneville, Perrard, & Henrioud, 1995; Choi et al., 2008; Lit et al., 2010; Lu et al., 2006; Marian et al., 2003; Tseng et al., 2009 and Zhou et al., 2010), SA (Milner, Graves, & Whitney, 1994 and Motavalli & Islam, 1997), ACO (Wang, Liu, Mao, & Fei, 2004), memetic algorithms (Gao, Qian, Li, & Wang, 2009), PSO (Guo, Li, Mileham, & Owen, 2009; Mukred et al., 2012 and Tseng et al., 2011) and hybrid methods (Hui, Yuan, & Kai-fu, 2008 and Li et al., 2013).

The implementation of meta-heuristics in solving discrete optimization problems, particularly in the ASP problem, lead to significant reductions in computation times, which in turn sacrifices the guarantee of finding exact optimal solutions (Blum and Roli, 2003; Talbi, 2009). However, these approaches typically obtain acceptable performance at acceptable costs in a large number of possible assembly sequences; thus, these approaches have a capacity to find good solutions to large-sized problems.

The primary objective of the ASP is to generate a feasible assembly sequence in which it will take less time to assemble, thereby reducing assembly costs. The most important factor in reducing assembly time and costs include setup time, which includes transfer time, the number of tool changes, and proper fixture selection.

In this study, assumptions for the ASP include;

1. The setup time and actual assembly time for each part and component are given.
2. The transfer time between workstations is included in the setup time.
3. The downtime of machines and workstations is negligible.

The total assembly time calculated from a machine to assembly many components to be a homogenous final product is the combination of the setup time and the actual assembly time. It is assumed that regardless of the assembly sequence, the actual assembly time is constant, and a proper tool and setup for each component to be assembled is required. These two items depend on the geometry of the component itself and the components assembled up to that point. The setup time for a component can be predicted using Eq. (2.26) where (*a*) is the component to be assembled; P_{a0} is the setup time with product (*a*) being the first component; P_{ab} is the contribution to the setup time due to the presence of part (*b*) when entering part (*a*); and q_{ab} =1 if component (*b*) has already been assembled and q_{ab} =0 otherwise for $a = 1, 2, \ldots, c$. where *c* is the number of components in the assembly sequence planning.

$$Time_{\text{Setup}}(a) = p_{a0} + \sum_{b=1}^{c} p_{ab} q_{ab} \qquad (2.26)$$

The total assembly time is the summation of the setup time and the actual assembly time. Hence, the objective function for minimizing the assembly time is described by Eq. (2.27) where A_a is the assembly time for component *a*. The calculation of time is in time units.

$$\text{Min } Time_{\text{Assembly}} = \sum_{b=1}^{c} \left(Time_{\text{Setup}}(a) + A_a\right) \tag{2.27}$$

To obtain feasible assembly sequences, all of the assembly sequences produced must comply with all precedence constraints. One of the constraints of the assembly design is the precedence relationships between the components. In this study, the assembly of a product with four components is represented in a directed graph, as shown in Figure 2.9. In general, the determined input data required by the assembly process is readily extracted either from CAD or a disassembly analysis.

A precedence matrix (PM) is used to show the relationships between the components in the assembly using precedence constraints. These relationships include the nature of the connection (i.e., free or assembled components) and the relative assembly precedence between two components. For this purpose, Table 2.1 can be built according to the precedence constraints between components a and b. If component a must be assembled after component b, PM(P_a,P_b)=1; otherwise PM(P_a,P_b)=∅, where (P_a,P_b) is a pair of components with geometric information in which P_a must be assembled without interfering with P_b.

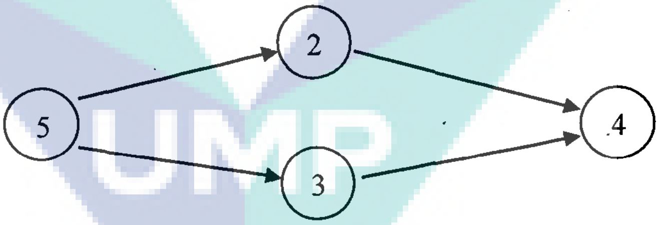


Figure 2.9. Example of a precedence diagram.

Table 2.1

Precedence Matrix (PM) based on Figure 2.9.

Component a	Component *b*			
	5	**2**	**3**	**4**
	0	0	0	0
	1	0	0	0
	1	0	0	0
	1	1	1	0

To decide which pair is feasible, precedence constraints for a product should be described using a PM. Assuming that ψ is the set of components that have been assembled before component a, and the union of the PM is a feasible assembly sequence $FAS(\mathrm{P}_a,P_b)$ with constraints, then:

$$FAS(P_a,P_b)=\cup\mathrm{PM}(P_a,P_b), P_b\in\psi \tag{2.28}$$

Eq. (2.28) explains that component a can be assembled if $FAS(P_a,P_b)$ is an empty set because all components that must be assembled before component a have been assembled. As an example, component 4 in Figure 2.8 can only be assembled if component 5, 2, and 3 have already been assembled. An assembly sequence 5-2-3-4 is thus feasible. It is shown that component 5 must be the first component to be assembled because P_5 is an empty set for all P_b. Next, either component 2 or 3 can be selected as the second component to be assembled because if $FAS(P_2,P_b)=(P_b=5)$. If component 2 is chosen, the component that must be assembled before component 2 can only be component 5. Component 3 can then be selected as the third component to be assembled because if $FAS(P_3,P_b)=(P_b=5)$, the component that must be assembled before component 3 has already been assembled (i.e., component 5). Finally, component 4 can be selected as the fourth component to be assembled because if $FAS(P_4,P_b)=(P_b=3)$, the components that must be assembled before component 4 have already been assembled (i.e., components 5, 2, and 3).

2.8 Summary

This chapter provides the previous and related studies for this research. Firstly, it covers a brief overview of optimization algorithms and the type of optimization problems. Methods to solve discrete COPs and discretization approaches of continuous optimization algorithms are then discussed in details. Next, an explanation and description of the fundamental concepts and operations for the original PSO, the original GSA and the binary version of PSO and GSA called the BPSO and the BGSA are offered. With regard to the literature review given in this chapter, it demonstrates that many previous studies have been proposed to solve discrete COPs that is naturally discrete either using the binary codification (i.e. Goldberg, 1989), integer codification (Tasgetiren, Evkli, Liang, & Gencyilmaz, 2004), or real-to-binary (Kennedy & Eberhart, 1997) or real-to-integer transformation (Wei & Hanning, 2007). There could be other approaches that search all possible solutions for solving discrete COPs in other type of search space rather than real-valued and binary search spaces, as provided by (Wei-Neng et al., 2010), where S-PSO makes use of set-based search space for searching all possible solutions.

CHAPTER 3

DESIGN OF MULTI-STATE MODEL AND ITS APPLICATION ON PSO AND GSA

3.1 Introduction

This chapter focuses on the design of a new discretization approach that should be able to solve discrete COPs as the other aforementioned discretization approaches. Inspired by the design of a sequential circuit of digital system, a new discretization approach, based on transition between two states, generates next state for each current state of each solution, leading to a complete model of multi-state. The multi-state model is then implemented in PSO and GSA. As a consequence, the MSPSO and the MSGSA are developed. The extended version of the multi-state model is then formulated by introducing an embedded rule that forces the updated solutions to be formed by unrepeated states. As a consequence, the MSPSOER and the MSGSAER are developed. The chapter starts by discussing the origin of multi-state model and then employed the fundamental description of multi-state model into the PSO and GSA later on. A few modifications required to operate this multi-state model is presented, which includes the operation of updating velocity, the operation of updating position and the operation of converting solution with repeated states to be non-repeated states.

This chapter is organized as follows: Section 3.2 provides the origin of the multi-state model. Section 3.3 discusses the multi-state model. Section 3.4 and Section 3.5 presents the implementation of multi-state model on PSO and GSA respectively. Section 3.6 offers the explanation of multi-state model with an additional rule called embedded rule. Section 3.7 and Section 3.8 provides the implementation of multi-state model with the embedded rule on PSO and GSA respectively. Section 3.9 and Section 3.10 offers the proposed approach based on the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER for solving TSP and the ASP. Section 3.11 presents the research methodology to show the overview of how this research is conducted. Finally, Section 3.12 presents the summary of this chapter.

3.2 Origin of Multi-State Concept

Digital systems represent information using only two possible values; one or zero. This binary system is sufficient to use in electronic circuit, where values are characterized either by the absence or presence of an electrical current flow. For instance, pulse code modulation (PCM) which is a method used to convert an analog signal into a digital signal is represented in binary numbers using high and low voltages value. There are two primary classes of logic circuits for digital system; combinational circuits and sequential circuits.

A combinatorial circuit performs an operation that can computes a Boolean function of its inputs. Two characteristics of this type of circuit is memoryless by definition and its outputs depend only on current input values. On the other hand, a sequential circuit (also called Finite State Machine) is an extension of combinatorial circuit with additional memory elements, called delays that is used to store the present state of the circuit. Its outputs depend on both circuit state and current inputs (Doyen et al., 2010). Figure 3.1 provides block diagram of sequential circuit.

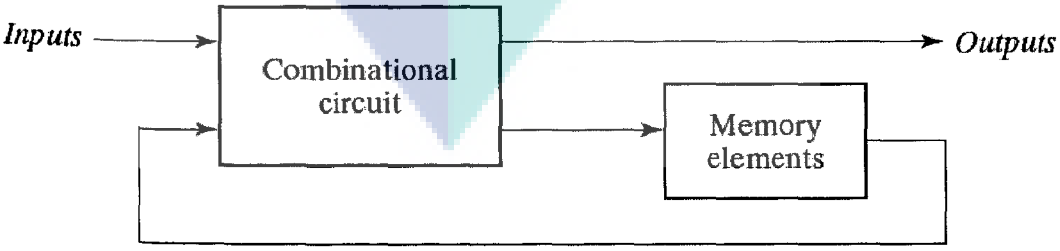


Figure 3.1. Block diagram of sequential circuit.

Source Mano and Ciletti 2012

A synchronous sequential circuit uses signals that change the storage elements at only discrete instants of time. Synchronization is carried out by a timing device named a clock generator, which offers a clock signal that have the shape of a periodic train of clock pulses. Clocked sequential circuits are sequential circuits that employ clock pulses to control storage elements. Figure 3.2 presents block diagram of clocked sequential circuit.

Flip-flops are the storage elements (memory) employed in clocked sequential circuits. A flip-flop is a binary storage device that able to store information with one bit, either 0 or 1. A sequential circuit may employ more than one flip-flop to store many bits. The concept of multi-state is drawn from one of flip-flops called JK flip-flop. Figure 3.3 and 3.4 offer the graphic symbol of JK flip-flop and its state diagram. With regard to state diagram presented in Figure 3.4, the term “0”and “1” can be replaced with state A and B.

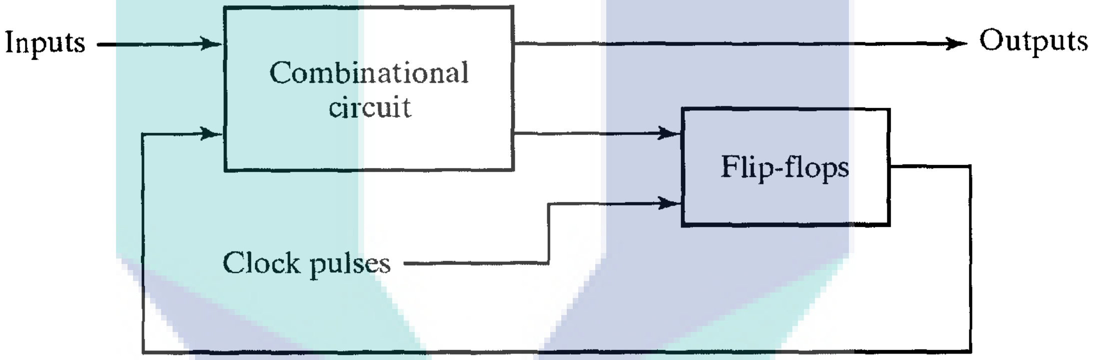


Figure 3.2. Block diagram of clocked sequential circuit.

Source Mano and Ciletti 2012

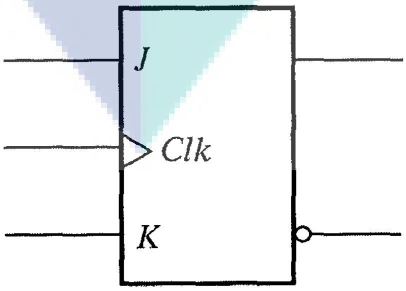


Figure 3.3. Graphic symbol of JK flip-flop.

Source Mano and Ciletti 2012

The changes and state transitions (1, 2, 3, and 4) that show present and next state can be seen in Figure 3.5.

State transition 1 and 3 indicate that next state for state A and B remain as their previous state. Meanwhile, for state transition 2 and 4, next state for state A and B change from state A to state B, and state B to state A, respectively. This condition applies to a problem with just two possible outputs (state A and stat B). In order to relate relationship between more than two states or called multi-state, state diagram in Figure 3.5 can be converted to Figure 3.6.

Figure 3.6 shares same attribute with Figure 3.5. The difference between these two figures is the number of possible states involved in relating one state to another. It is worth mentioning that state transition 1, 3, 5, and 8 produce next state that similar with their present state, while other state transitions produce next state which is different compared to their present state.

A search space is the set of all possible solutions for any given optimization problem. Each solution is generally represented either in real-value (Kennedy & Eberhart, 1995) or binary (Goldberg, 1989). In the context of discrete COPs, the application of continuous optimization algorithms require the transformation of a

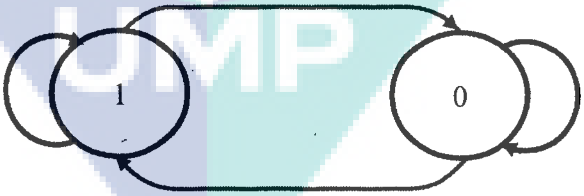


Figure 3.4. State diagram of JK flip-flop.

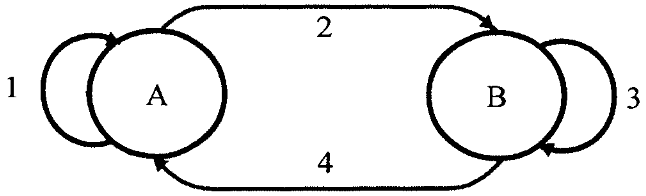


Figure 3.5. State diagram of two states.

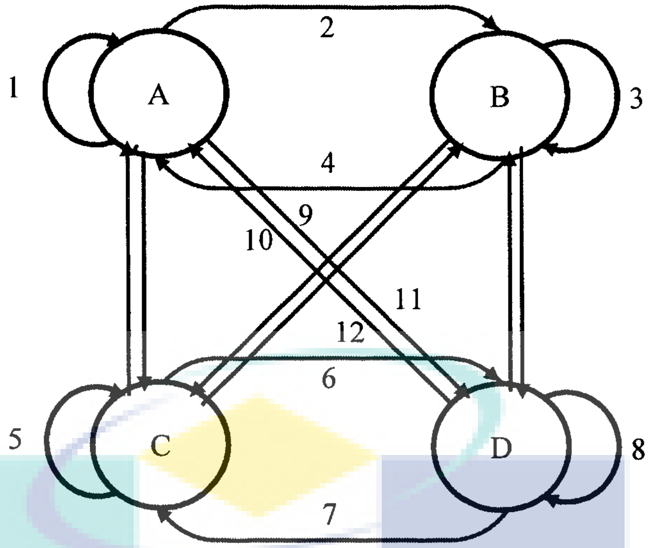


Figure 3.6. State diagram of multi-state.

continuous search space (real-valued search space) to a discrete one by discretization of the continuous decision variables. This mean real-valued optimization algorithms cannot be directly used to solve discrete COPs. Meanwhile, the binary search space consists of discrete points within in the coding range. By setting the number of bits and the coding range, the search space can be restricted. However, binary-based optimization algorithms require many bits, compared to continuous optimization algorithms that employ real-to-integer transformation to represent each solution. For instance, binary-based algorithms require 56 bits to represent each solution of Burma 14 benchmark instance of TSP, while continuous optimization algorithms that employ real-to-integer transformation only requires 14 bits. Other than these two aforementioned search spaces, a search space called set-based has been developed by (Wei-Neng et al., 2010) to search all possible solutions for solving discrete COPs.

To mapping all possible solutions produced by employing the mechanism of state transition as shown Figure 3.6, multi-state search space is introduced, as provided in Figure 3.7. In this research, the term multi-state is used to show that one or more candidate states are available to be selected as the next state.The multi-state search space provides information of current state, candidate states, and possible next states in each dimension. Such way is applied to other dimensions as well. For instance, by considering State A in Figure 3.6 as current state for a particular dimension, Figure 3.7 presents all candidate

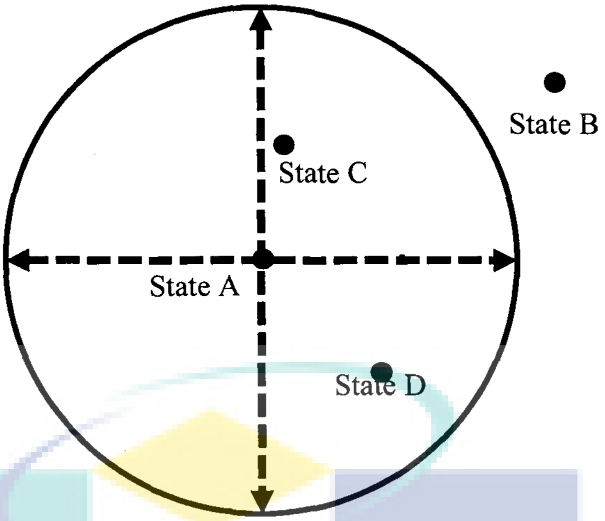


Figure 3.7. Example of a multi-state search space.

next states (State A, State B, State C, and State D), the area that covers possible next states (search space), and the possible next states (State A, State C, and State D. In Figure 3.7, the black small circles represent states and the circle represents an area that covers State A, State C, and State D, respectively. This representation is elaborated in Section 3.3 for more details.

3.3 Design of Multi-State Model

In this section, the proposed multi-state model is discussed. This model is built using the mechanism of state transition between two states. This optimization algorithms that employ this model called multi-state based algorithms. In the multi-state model, each solution's vector or dimension in the multi-state-based algorithm is represented as a collective of states; neither continuous nor discrete value.

The multi-state model can be elaborated using information of cities' location of Burma14 benchmark instance of TSP as depicted in Figure 3.8. This benchmark instance of TSP is chosen because it has just 14 cities; so it is easier to be illustrated. All 14 cities or vertices that extracted from Burma14 benchmark instance are represented as a collective of states.

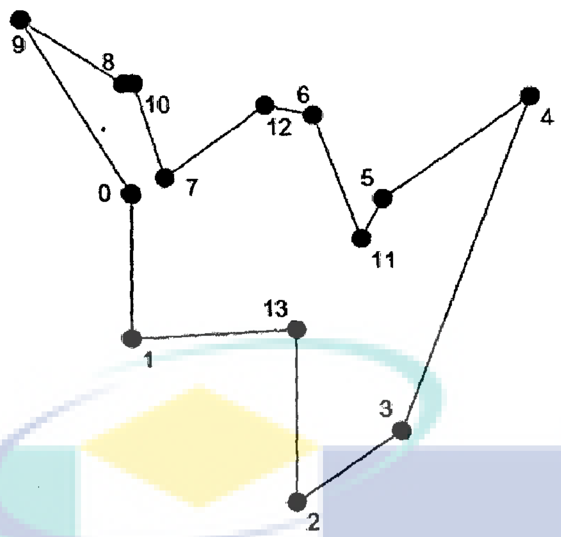


Figure 3.8. Burma14 benchmark instance of TSP.

Figure 3.9 presents the collective of states in the form of small black circle. With regard to this figure, the multi-state search space of each dimension is built using two features; a current state and a radius. Each current state of a 14-dimensional vector solution which is basically represented by each 14 small black circle, is plotted based on its location. Each current state of the 14-dimensional vector solution is then appointed as the centre of the multi-state search space. Subsequently, the length of the line from the centre of each current state called radius is determined. A circle can be eventually built when these two features; a current state and a radius are known. The process of producing the circle is applied on each 14-dimensional vector solution.

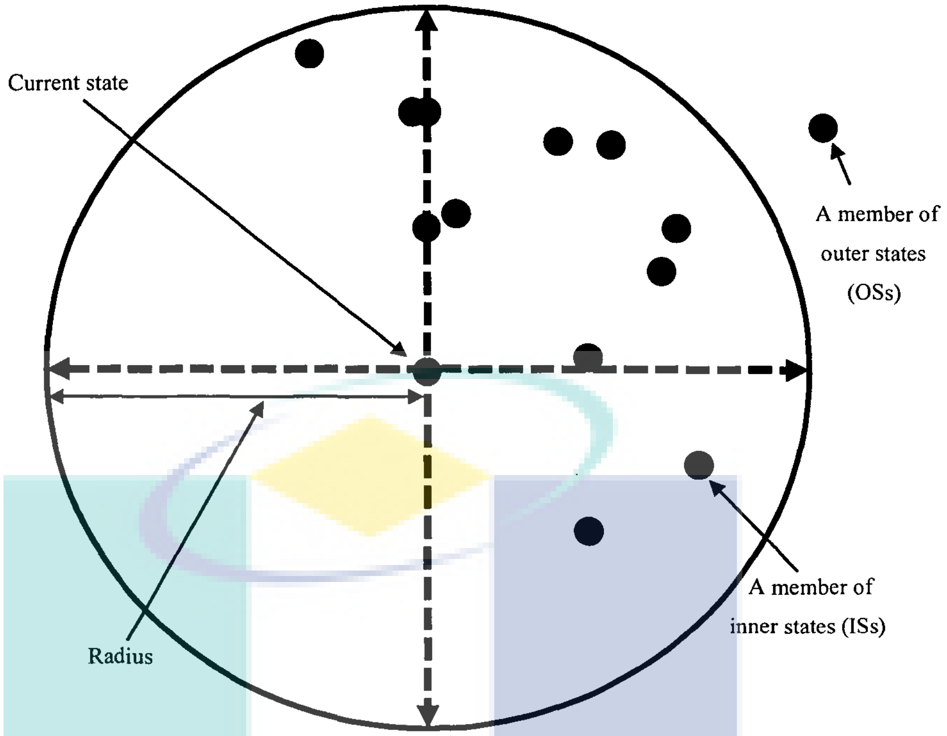


Figure 3.9. Multi-state search space of a dimension for Burma14 benchmark instance of TSP.

It seems that the small black circles can be found inside and outside of each circle. Each small black circle is basically a state. Each small black circle inside of the circle is called inner state (IS), while each black circle outside of the circle is outer state (OS). The IS can be defined as a state in which the cost value (i.e. distance, time or etc.) from a current state to a particular state is same or smaller than the length of line (radius). Otherwise; the state is OS. A small black circle is then selected randomly among the IS. The selected small black circle is now called the next state.

Figure 3.10 offers an explanation on the process of updating current state of each dimension for a solution to next state in the multi-state search space. For instance, a current 5- dimensional vector solution $\{x_5, x_3, x_2, x_1, x_4\}$ is used. In updating current state to next state for each dimension, radius from the current state for each dimension is firstly determined. As a consequence, all ISs are now identified. A state is then randomly selected among all ISs for each dimension. The selected state is the current state for each dimension. Let's say the selected states for dimension 1, 2, 3, 4, and 5 are $x_2, x_4, x_3, x_2,$ and x_1, respectively. Thus, the updated solution (next solution) is x_2, x_4, x_3, x_2, x_1.

One characteristic of multi-state model is, there might be one or more repeated states in an updated solution. Figure 3.10 shows that there is a repeated state in the updated solution which is x_2 in 1st and 4th dimension of the updated solution. It is in such case as in TSP necessarily to have no repeated states in each solution. Figure 3.11 portrays

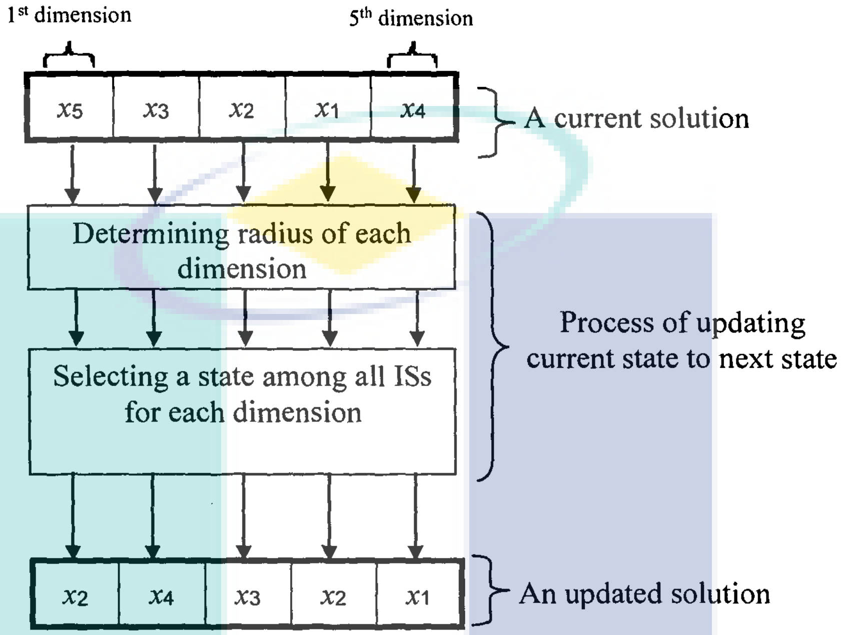

1st dimension
5th dimension
x5
x3
x2
x1
x4
A current solution
Determining radius of each dimension
Process of updating current state to next state
Selecting a state among all ISs for each dimension
x2
x4
x3
x2
x1
An updated solution

Figure 3.10. Process of updating current state of each dimension for a solution to next state in the multi-state search space.

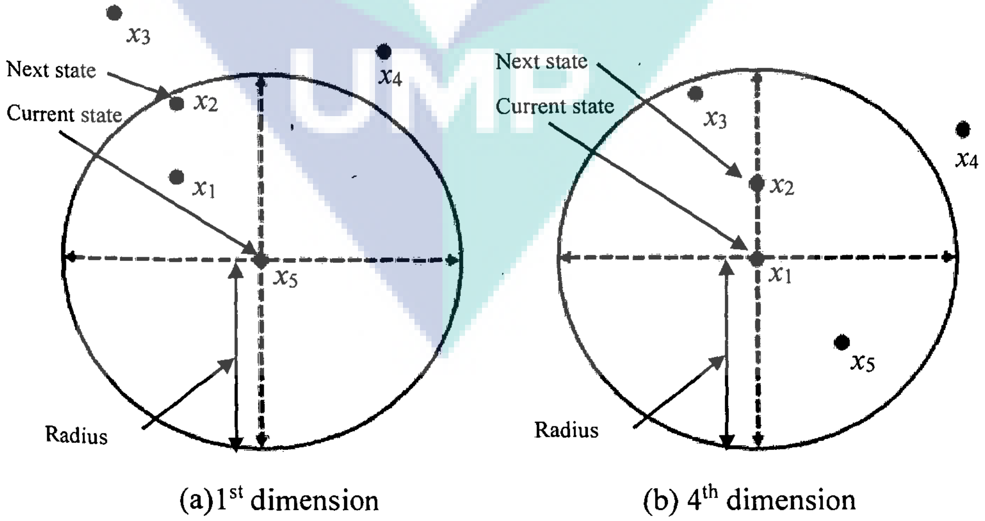


Figure 3.11. The identical state chosen, x_2 in (a) and (b).

the repetitive issue and its cause for 1st and 4th dimension of a particular solution. It is clearly seen that state x_2 has been randomly selected among IS to be the next state. Because states are discrete, it is worth to note that multi-state model can only be used to solve discrete COPs, but not continuous COPs.

To explain the exploration and exploration of the multi-state model, the characteristic of radius of the multi-state model for a dimension of a solution for a particular iteration are presented in Figure 3.12, Figure 3.13, and Figure 3.14. Similar characteristic are applied to each dimension of each solution. At the first iteration, the radius is set big as presented in Figure 3.12. Starting from the first iteration, the coverage of exploration of the multi-state search space is the biggest. As the iteration increases, the radius becomes smaller, as illustrated in Figure 3.13. As the iteration ends, the radius

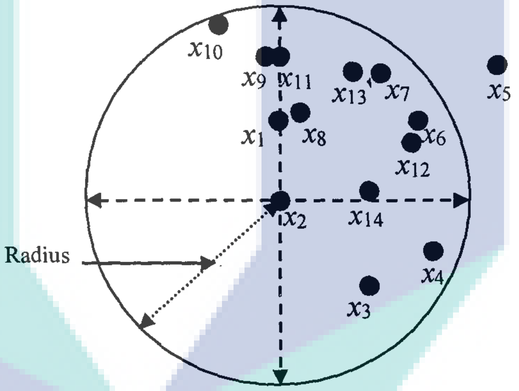


Figure 3.12. The characteristic of radius update of multi-state model at t^{th} iteration.

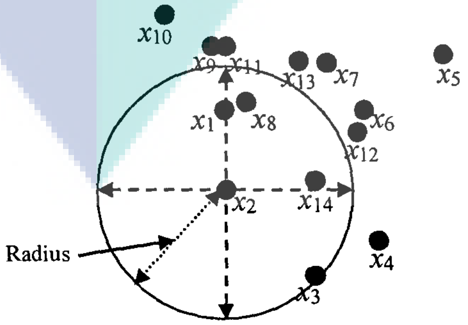


Figure 3.13. The characteristic of radius update of multi-state model at $t{+}{+}^{th}$ iteration.

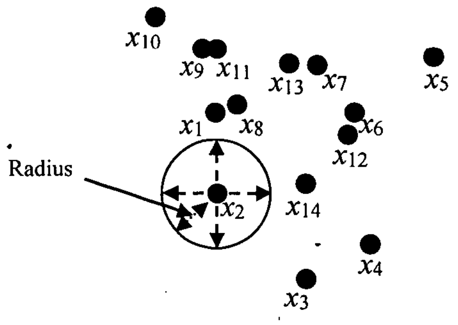


Figure 3.14. The characteristic of radius update of multi-state model at near the end of maximum iteration.

becomes smaller than the previous generated radius as presented in Figure 3.14.This means the coverage of exploration of the multi-state search space is reduced per iteration. At the end of iteration, the radius becomes smaller than previous iterations and the algorithms that use the multi-state model converge. The characteristic of radius in the multi-state model reduces the number of candidate states as iteration increases and hence, a good balance between exploration and exploitation can be achieved.

3.4 The Proposed Multi-State Particle Swarm Optimization (MSPSO)

In this section, the proposed MSPSO algorithm is explained. As aforementioned, the multi-state search space of each dimension is built using two features; a current state and a radius. In the MSPSO algorithm, the radius is represented by new velocity value. The proposed MSPSO practices similar general principle of the original PSO with a few modifications in term of exploitation of particle's velocity and a mechanism of state transition.

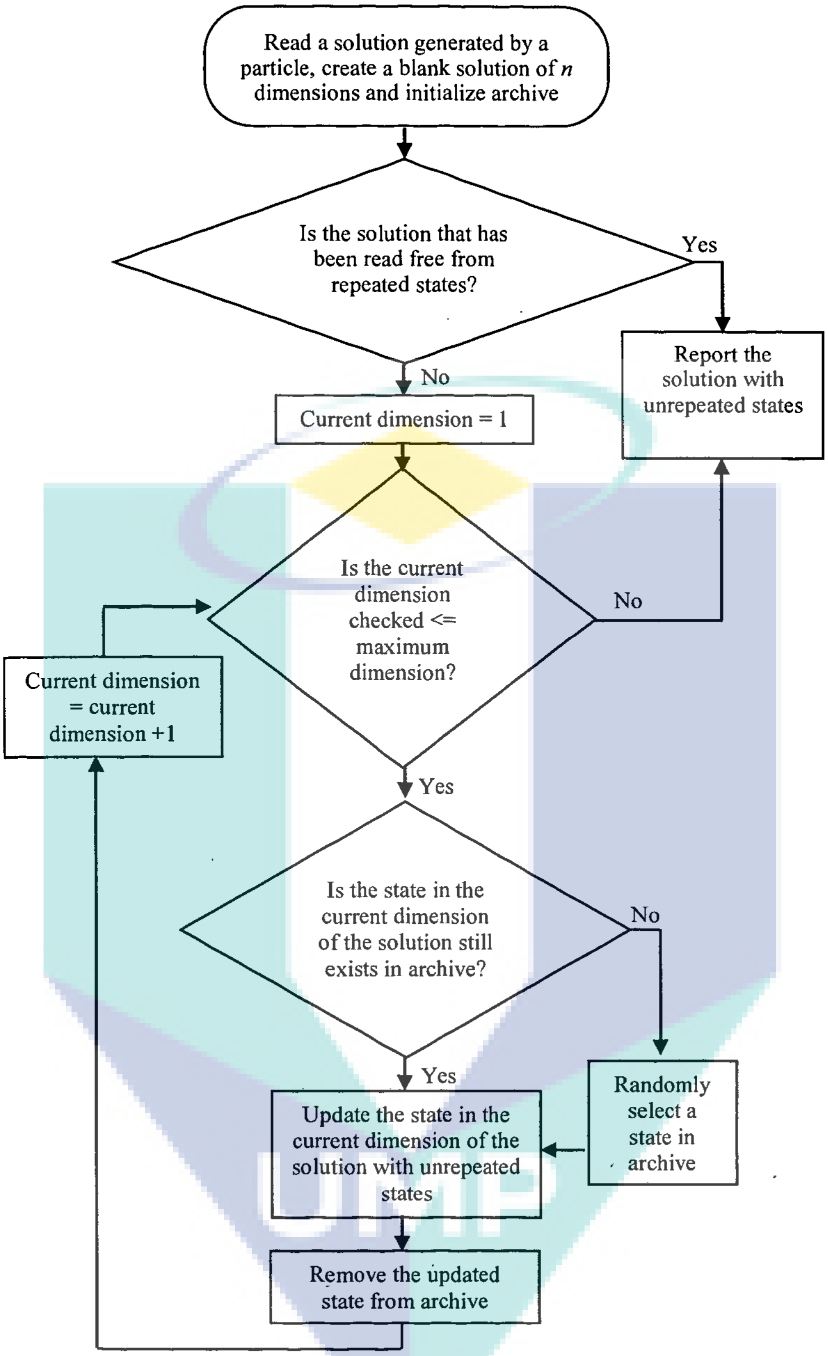
The main difference between the MSPSO and the PSO is the way on updating each particle's velocity and position based on state, while most of steps required in the MSPSO are identical with the PSO. In the implementation of the multi-state based representation, numerical calculation to find difference in the cognitive component of velocity calculation which is between $\boldsymbol{pbest}_i(k,d)$ and $\boldsymbol{s}_i(k,d)$ does not allow. This condition also occurs in the part of finding difference in the social component of velocity calculation which is between $\boldsymbol{gbest}(k,d)$ and $\boldsymbol{s}_i(k,d)$.

In the PSO and the BPSO algorithms, a particle has three movement components; the inertia, cognitive, and social component as shown in Eq. (2.2). The effect of the first, second, and third component are the particle bias to follow in its own way, to go back to its best previous position, and to go towards the global best particle, respectively. In implementing the multi-state representation, the velocity should be defined as the summation of previous velocity, cost of difference function with the effect of cognitive factor between each particle's best position and current particle's position, and cost of difference function with the effect of social factor between global best particle and current particle's position. By way of explanation, the velocity is in the form of cost function, i.e distance in TSP (Cirasella et al., 2001; Dorigo et al., 1996; Voudouris and Tsang, 1999). Defining the velocity operator in this manner causes the change of behaviour of velocity, which are the new form of velocity has no magnitude and probability calculation as the PSO algorithm and the BPSO algorithm, respectively. To make such an updating velocity formulation valid, a function called *cost* (.) is introduced to replace the direct subtraction operation incorporated in Eq. (2.2) into Eq. (3.1) as follows.

$$\begin{aligned} v_i(k+1,d) = \omega v_i(k,d) + c_1 r_1 cost(\mathbf{pbest}_i)(k,d), \mathbf{s}_i(k,d) \\ + c_2 r_2 cost(\mathbf{gbest}(k,d), \mathbf{s}_i(k,d) \end{aligned} \tag{3.1}$$

Then, a next position which is in the form of state $\mathbf{s}_i(k+1,d)$ is selected randomly among the IS using Eq. (3.2). Given a set of j IS members $\mathbf{I}_i(k,d) = \left(\mathbf{I}_{i_1}(k,d),\ldots,\mathbf{I}_{i_j}(k,d)\right)$.

$$s_i(k+1,d) = \text{random}\left(\mathbf{I}_{i_1}(k,d),\ldots,\mathbf{I}_{i_j}(k,d)\right) \tag{3.2}$$



Read a solution generated by a particle, create a blank solution of *n* dimensions and initialize archive
Is the solution that has been read free from repeated states?
Yes
Report the solution with unrepeated states
No
Current dimension = 1
Is the current dimension checked <= maximum dimension?
No
Current dimension = current dimension +1
Yes
Is the state in the current dimension of the solution still exists in archive?
No
Yes
Randomly select a state in archive
Update the state in the current dimension of the solution with unrepeated states
Remove the updated state from archive

Figure 3.15. The procedure of producing unrepeated states of each solution.

As mentioned in Figure 3.10, there might be one or more repeated states in an updated solution. In order to overcome the repetitive issue, an additional procedure is added after updating velocity and position, so the solution with unrepeated states is evolved from the solution with repeated states. To elaborate this procedure, this procedure is presented in Figure 3.15.

Initially, a solution of a particle is read. Also, a blank solution of n dimensions is created and an archive is initialized. The archive is then sorted in natural order. The solution that has been read is then checked, whether the solution has repeated states or not. If the solution has no repeated states, the process is stopped. Otherwise, a solution with unrepeated states must be generated from the solution with repeated states. To generate the solution with unrepeated states, the current dimension d of the solution with repeated states is then checked whether it has exceeded the maximum number of dimension n or not. If the maximum number of dimension is not exceeded, the state in the current dimension d of the solution with repeated states is read. Otherwise, the process is stopped. If the state in the current dimension d of the solution with repeated states is read, the state is checked whether the state still exists in the archive. If the state still exists in the archive, the state is put in the current dimension d of the solution with unrepeated states. Otherwise, a state is randomly chosen from the archive at first and the state is then put in the current dimension d of the solution with unrepeated states. Next, the state chosen is removed from the archive. Finally, a solution with unrepeated states is produced when all dimensions in the solution with repeated states has been checked. It is worth mentioning that this procedure is applied to the solution of each particle.

For instance, the process illustrated in Figure 3.15 can be elaborated by Table 3.1. In this table, each dimension of solution is checked, whether the state in the dimension has been used or not. For instance, at the 4th dimension, the current state is x_2. The next state can be either state x_1 or state x_5. State x_2, state x_4, and state x_3 cannot be selected to be the next state because these three states have been selected in previous dimensions.

Table 3.1

The Explanation of the Procedure Conducted in Figure 3.15

Dimension	**Current state of each dimension**	**All states that can be chosen as a potential next state of each dimension**	**Selected state (next state)**	**All unused states**
1	x_2	$\{x_1, x_2, x_3, x_4, x_5\}$	x_2	$\{x_1, x_3, x_4, x_5\}$
2	x_4	$\{x_1, x_3, x_4, x_5\}$	x_4	$\{x_1, x_3, x_5\}$
3	x_3	$\{x_1, x_3, x_5\}$	x_3	$\{x_1, x_5\}$
4	x_2	$\{x_1, x_5\}$	x_5	$\{x_1\}$
5	x_1	$\{x_1\}$	x_1	-

A solution with unrepeated states = $\{ x_2, x_4, x_3, x_5, x_1 \}$

```
procedure MSPSO
    Initialize particles solution to feasible and velocities set to zero.
    Set the particles' pbests to their current positions.
    Calculate the particles' fitness and set gbest.
    for T generations do
        Update the particles' velocities.
        Update the particles' positions.
        Evolve infeasible solutions to feasible solutions
        Recalculate the particles' fitness.
        Update the particles' pbest and gbest.
    end for
end procedure
```

Figure 3.16. The general principle of the MSPSO.

As consequence, this procedure will successfully evolve a solution with repeated states to be a solution with unrepeated states. The pseudo code of the general principle of MSPSO is then presented in Figure 3.16.

3.5 The Proposed Multi-State Gravitational Search Algorithm (MSGSA)

In this section, the proposed MSGSA is explained. As aforementioned, the multi-state search space of each dimension is built using two features; a current state and a radius. In the proposed MSGSA, the radius is represented by new velocity value. The proposed MSGSA practices similar general principle of the original GSA. There are some modifications have been executed on the proposed MSGSA in the state transition and the force formulation.

The MSGSA implements same procedure as in the MSPSO in updating each agent's position by exploiting agent's velocity. The formulation of updating each agent's position in the form of state and velocity still similar as in the GSA. In addition, most of steps required in the MSGSA are identical with the GSA. Then, a next position in the form of state $s_i(k+1,d)$ is selected randomly among the IS using Eq. (3.3). Given a set of j IS members $\boldsymbol{I}_i(k,d) = \left(\boldsymbol{I}_{i_1}(k,d),\ldots,\boldsymbol{I}_{i_j}(k,d)\right)$.

$$s_i(k+1,d) = \text{random}\left(\boldsymbol{I}_{i_1}(k,d),\ldots,\boldsymbol{I}_{i_j}(k,d)\right) \tag{3.3}$$

The force formulation of the GSA as revealed in Eq. (2.15) in Chapter II shows the subtraction operation between two agents' positions, $s_j(k,d)$ and $s_i(k,d)$, is executed to calculate the difference of vector value in a particular dimension and time, resulting a numerical value. In the MSGSA, each agent's position is represented as a state. Since a state is not associated to any value, the subtraction operation cannot be used to find the difference between two states. Eq. (3.4) is derived to accommodate the calculation of force, $\boldsymbol{F}_{ij}(k,d)$ in the MSGSA. A *cost* function, *cost*(.), is introduced and incorporated in the force formulation. The cost (i.e. distance in the TSP) between two positions is a positive number given by cost(($s_j(k,d)$, $s_i(k,d)$)). In this force formulation, $R_{ij}(k)$ is the difference of fitness between agent *i* and agent *j*.

$$\boldsymbol{F}_{ij}(k,d) = G(k)\frac{M_{aj}(k)}{R_{ij}(k)+\varepsilon}\mathrm{cost}((\boldsymbol{s}_j(k,d),\ \boldsymbol{s}_i(k,d)) \qquad (3.4)$$

Similarly as in the MSPSO, there might be one or more repeated states in an updated solution. To solve this problem, the solution with repeated states must be evolved to be the solution with unrepeated states. The details of this procedure can be seen in Section 3.3 in this chapter. The pseudo code of the general principle of the MSGSA is then presented in Figure 3.17.

```
procedure MSGSA
    Initialize agents with random positions and velocities.
    Set the agents to their current positions.
    Calculate the agents' fitness and set G, the best agent and the worst agent
    for T generations do
        Evaluate the mass.
        Evaluate the force of the mass.
        Evaluate the acceleration of the mass.
        Update the agents' velocities.
        Update the agents' positions.
        Convert feasible solutions from infeasible solutions.
        Recalculate the agents' fitness.
        Update the G, the best agent and the worst agent.
    end for
end procedure
```

Figure 3.17. The general principle of the MSGSA.

3.6 The Multi-State Model with an Embedded Rule

In this section, the concept of multi-state model with an embedded rule is discussed. The concept is fundamentally introduced to solve the repetitive issue occurred in the MSPSO and the MSGSA. The embedded rule are implemented in the MSPSO and the MSGSA by designing such procedure to directly produce a solution with unrepeated states from a solution with repeated states. The embedded rule is; each state can only occur once in each solution. The introduction of the embedded rule in the multi-state model efficiently removes the limitation of the MSPSO and the MSGSA in which all solutions generated with unrepeated states.

With regard to the characteristic of the multi-state model with the embedded rule, it seems possible that the implementation of embedded rule will produces two different cases. Figure 3.18 presents the first case that can occur in updating position which position of each dimension in the form of state. This means each solution consists a state in its each dimensional vector. In Figure 3.18, (a) state x_2 is the next state for the 1st dimension, (b) state x_1 is the next state for the 2nd dimension, (c) state x_3 is the next state for the 3rd dimension, (d) state x_5 is the next state for the 4th dimension and (e) state x_4 is the next state for 5th dimension. In the first case, the next state of each dimension of the solution is selected just from ISs. To obtain the next state, the process is explained as follows.

Figure 3.18 (a) shows five states, namely state x_1, state x_2, state x_3, state x_4, and state x_5. The current solution is $\{x_5, x_3, x_2, x_1, x_4\}$. The process starts by updating the current state of the 1st dimension of the current solution, which is state x_5 using the velocity value obtained from derivation of Eq. (3.1). Due to the existence of the ISs, a state is randomly chosen from the ISs and then updated in the 1st dimension of the updated solution. The state placed in the 1st dimension of the updated solution is state x_2. The state x_2 is then removed from the search space; state x_2 is excluded for the next selection.

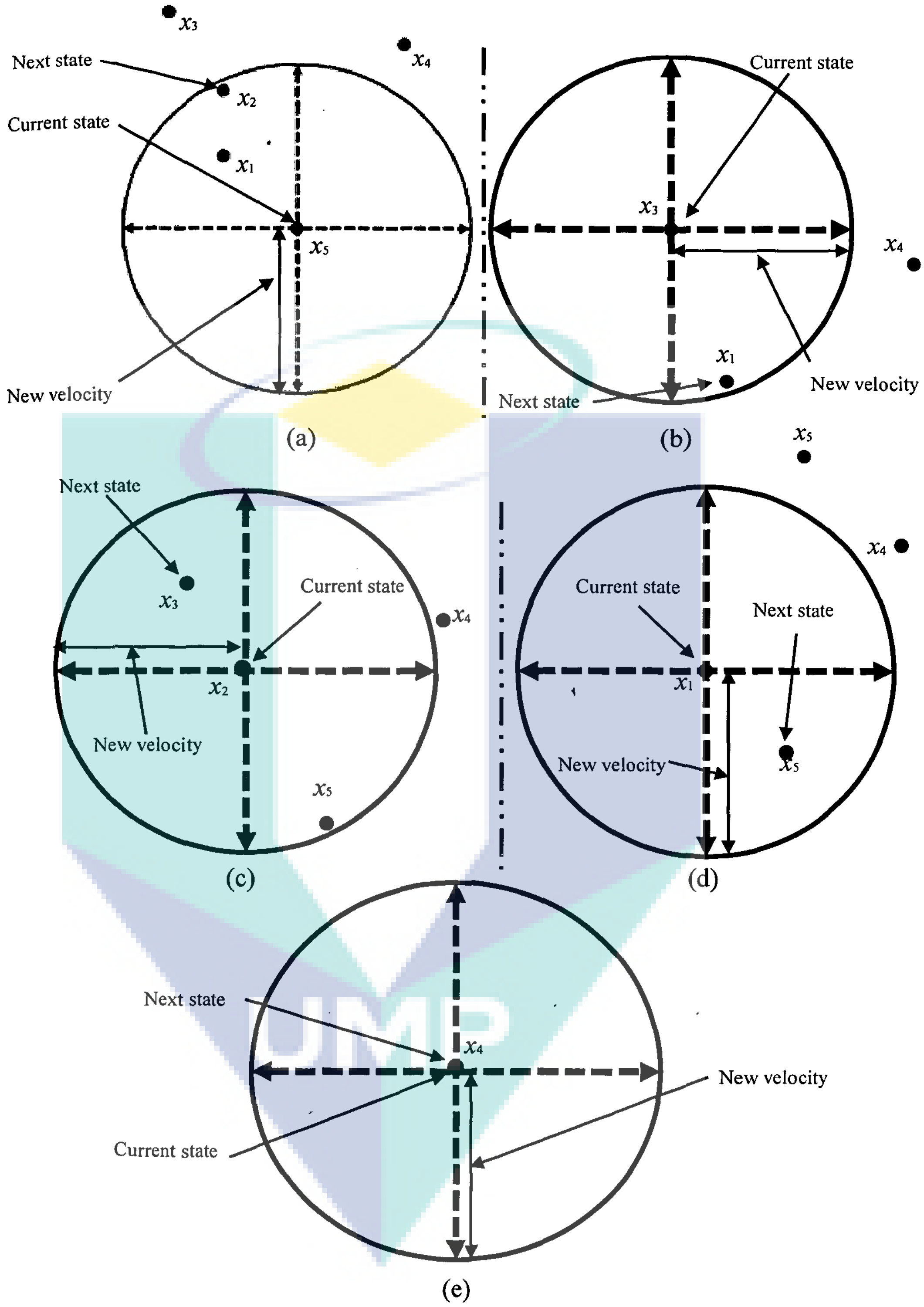


Figure 3.18. The first case occurred in updating state of each dimension in the MSPSOER and the MSGSAER.

Figure 3.18 (b) shows the current state of the 2nd dimension of the current solution, state x_3 using the velocity value obtained from derivation of Eq. (3.1). Due to the existence of the ISs, a state is randomly chosen from the ISs and then updated in the 2nd dimension of the updated solution. The state placed in the 2nd dimension of the updated solution is state x_1. The state x_1 is then removed from the search space; state x_1 is excluded for the next selection.

Figure 3.18 (c) shows the current state of the 3rd dimension of the current solution, state x_2 using the velocity value obtained from derivation of Eq. (3.1). Due to the existence of the ISs, a state is randomly chosen from the ISs and then updated in the 3rd dimension of the updated solution. The state placed in the 3rd dimension of the updated solution is state x_3. The state x_3 is then removed from the search space; state x_3 is excluded for the next selection.

Figure 3.18 (d) shows the current state of the 4th dimension of the current solution, state x_1 using the velocity value obtained from derivation of Eq. (3.1). Due to the existence of the ISs, a state is randomly chosen from the ISs and then updated in the 4th dimension of the updated solution. The state placed in the 4th dimension of the updated solution is state x_5. The state x_5 is then removed from the search space; state x_5 is excluded for the next selection.

Figure 3.18 (e) shows the current state of the 5th dimension of the current solution, state x_4 using the velocity value obtained from derivation of Eq. (3.1). Due to the existence of the ISs, a state is randomly chosen from the ISs and then updated in the 5th dimension of the updated solution. The state placed in the 5th dimension of the updated solution is state x_4. The state x_4 is then removed from the search space; state x_4 is excluded for the next selection. The procedure portrayed in Figure 3.18 eventually produces an updated solution $\{x_2, x_1, x_3, x_5, x_4\}$ which is a solution with unrepeated states.

Meanwhile, Figure 3.19 presents the second case can occur in updating state of each dimension with the implementation of the embedded rule. In this case, next state of each dimension of the solution is selected either from the ISs or the unselected states. In Figure 3.19, (a) state x_1 is the next state for the 1st dimension, (b) state x_3 is the next state for the 2nd dimension, (c) state x_5 is the next state for the 3rd dimension, (d) state x_2 is the

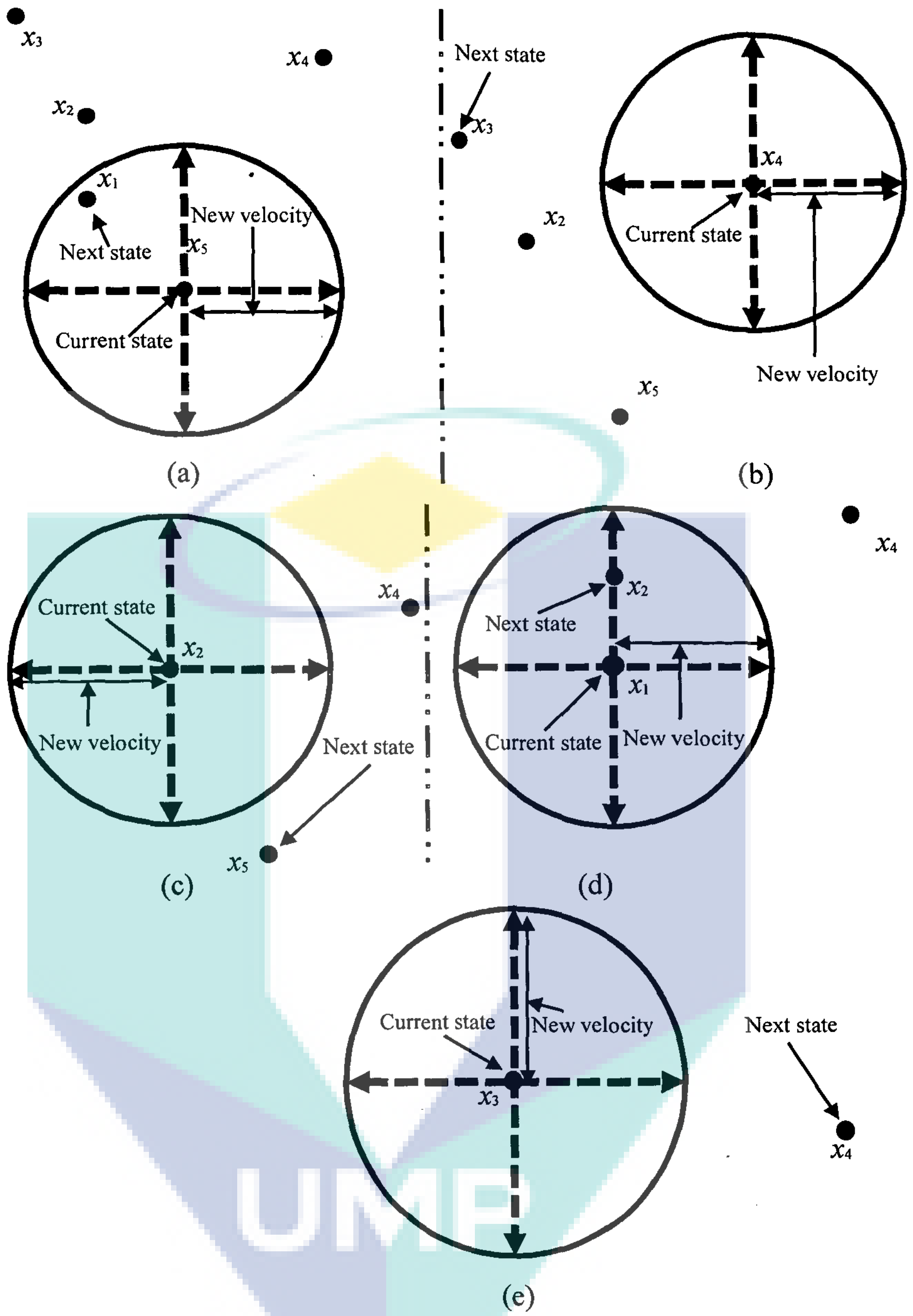


Figure 3.19. The second case occurred in updating state of each dimension in the MSPSOER and the MSGSAER.

next state for the 4th dimension and (e) state x_4 is the next state for 5th dimension. In the second case, the next state of each dimension of the solution is selected either from the ISs or the unselected states. To obtain the next state, the process is explained as follows.

Figure 3.19 (a) shows five states, namely state x_1, state x_2, state x_3, state x_4, and state x_5. The current solution is $\{x_5, x_4, x_2, x_1, x_3\}$. The process starts by updating the current state of the 1st dimension of the current solution, which is state x_5 using the velocity value obtained from derivation of Eq. (3.1). Due to the existence of the ISs, a state is randomly chosen from the ISs and then updated in the 1st dimension of the updated solution. The state placed in the 1st dimension of the updated solution is state x_1. The state x_1 is then removed from the search space; state x_1 is excluded for the next selection.

Figure 3.19 (b) shows the current state of the 2nd dimension of the current solution, state x_4 using the velocity value obtained from derivation of Eq. (3.1). Due to the non-existence of the ISs, a state is randomly chosen from the unselected states and then updated in the 2nd dimension of the updated solution. The state placed in the 2nd dimension of the updated solution is state x_3. The state x_3 is then removed from the search space; state x_3 is excluded for the next selection.

Figure 3.19 (c) shows the current state of the 3rd dimension of the current solution, state x_2 using the velocity value obtained from derivation of Eq. (3.1). Due to the non-existence of the ISs, a state is randomly chosen from the unselected states and then updated in the 3rd dimension of the updated solution. The state placed in the 3rd dimension of the updated solution is state x_5. The state x_5 is then removed from the search space; state x_5 is excluded for the next selection.

Figure 3.19 (d) shows the current state of the 4th dimension of the current solution, state x_1 using the velocity value obtained from derivation of Eq. (3.1). Due to the existence of the ISs, a state is randomly chosen from the ISs and then updated in the 4th dimension of the updated solution. The state placed in the 4th dimension of the updated solution is state x_2. The state x_2 is then removed from the search space; state x_2 is excluded for the next selection.

Figure 3.19 (e) shows the current state of the 5th dimension of the current solution, state x_3 using the velocity value obtained from derivation of Eq. (3.1). Due to the non-existence of the ISs, a state is randomly chosen from the unselected states and then updated in the 5th dimension of the updated solution. The state placed in the 5th dimension of the updated solution is state x_4. The state x_4 is then removed from the search space;

state x_4 is excluded for the next selection. The procedure offerred in Figure 3.19 eventually produces an updated solution $\{x_1, x_3, x_5, x_2, x_4\}$ which is a solution with unrepeated states. The implementation of the embedded rule in the MSPSOER and MSGSAER efficiently eliminate the step of evolving a solution with repeated states to a solution with unrepeated states because each solution yielded is originally without repeated states. This means the procedure of producing unrepeated states of each solution provided in Figure 3.15 is now can be omitted.

3.7 The Proposed Multi-State Particle Swarm Optimization with an Embedded Rule (MSPSOER)

In this section, the proposed MSPSOER is explained. The proposed MSPSOER fundamentally improves the proposed MSPSO, so the repeated states are not generated in a solution. Generally, the improved proposed algorithm practices similar principle of the proposed MSPSO except in the mechanism of state transition between two states.

3.7.1 Embedded Rule in the MSGSAER

In the MSPSO algorithm, once the new velocity is updated, the process of updating current state to next state of each dimension's solution is executed. As aforementioned in Section 3.2, the current state, the new velocity value, the ISs, the outer states (OSs) are the elements occur in the multi-state search space. Information of all the ISs and the OSs should be known before applying the embedded rule. Beside, all selected states (SSs) that are appointed as the next state in a solution must be taken into account. The origin of the SS could come either from one of the ISs or OSs. A set of h SSs can be derived as $\boldsymbol{T}_i(k,d) = \left(\boldsymbol{T}_{i_1}(k,d),\ldots,\boldsymbol{T}_{i_h}(k,d)\right)$. All the SS should be identified because this type of state is invalid to be selected as the next state.

For the ISs and OSs, let us consider a set of j IS $\boldsymbol{I}_i(k,d) = \left(\boldsymbol{I}_{i_1}(k,d),\ldots,\boldsymbol{I}_{i_j}(k,d)\right)$ and a set of l OS $\boldsymbol{O}_i(k,d) = \left(\boldsymbol{O}_{i_1}(k,d),\ldots,\boldsymbol{O}_{i_l}(k,d)\right)$. Based on the current state and the new velocity value of each dimension, a next state of each dimension can be selected using

Eq. (3.5) where $\emptyset$ is empty set. We observe that each solution with unrepeated states are successfully generated using the formulation in Eq. (3.5) in which the information of the ISs, OSs and SSs are taken into account. The algorithmic details of the process of updating each current state into a next state in the MSPSOER is presented in Figure 3.20.

$$s_i(k+1,d) = \begin{cases} \text{random}\Big(\big(\boldsymbol{I}_i(k,d) - (\boldsymbol{I}_i(k,d) \cap \boldsymbol{T}_i(k,d))\big)\Big) \\ \text{if}\Big(\big(\boldsymbol{I}_i(k,d) - (\boldsymbol{I}_i(k,d) \cap \boldsymbol{T}_i(k,d))\big)\Big) \neq \emptyset \\ \\ \text{random}\Big(\big(\boldsymbol{O}_i(k,d) - (\boldsymbol{O}_i(k,d) \cap \boldsymbol{T}_i(k,d))\big)\Big) \\ \text{if}\Big(\big(\boldsymbol{I}_i(k,d) - (\boldsymbol{I}_i(k,d) \cap \boldsymbol{T}_i(k,d))\big)\Big) = \emptyset \end{cases} \tag{3.5}$$

To sum up, the next state is randomly chosen from the remaining ISs after all the SSs are removed from the ISs. If remaining members of the ISs do not exist, the next state is then randomly chosen from the OSs. This process is basically applied to each dimension of each solution. This process ends when all dimensions of all solutions have been updated.

```
input: $I$ is the number of agents, $D$ is the maximum dimensions,
       $s_i(k,d)(i=1,2,\ldots,I;\ d=1,2,\ldots,D)$ is current position of each agent and
       $v_i(k+1,d)(i=1,2,\ldots,I;\ d=1,2,\ldots,D$ is current velocity of each agent.
agent = 1;
repeat
  Initialize the SSs.
  dimension = 1;
  repeat
     Generate a circle subjected to current state and new velocity value to
     determine the ISs and the OSs.
     Exclude the SSs from the members of the ISs.
     if any of the ISs still occurs as the subject of selection then
        Randomly choose one of the ISs to be next state.
     Randomly choose one of the OSs to be next state.
     Register the selected next state to be one of the SSs.
     dimension ++;
   until dimension == $D$;
   agent ++;
 until agent == I;
 output: Solutions with unrepeated states generated.
```

Figure 3.20. The process of updating current state to be next state in the MSPSOER.

```
procedure MSPSOER
    Initialize particles solution and velocities set to zero.
    Set the particles' pbests to their current positions.
    Calculate the particles' fitness and set gbest.
    for $T$ generations do.
        Update the particles' velocities.
        Update the particles' positions.
        Recalculate the particles' fitness.
        Update the particles' pbest and gbest.
    end for
end procedure
```

Figure 3.21. The general principle of the MSPSOER.

The proposed MSPSOER is depicted in Figure 3.21. The main difference between the proposed MSPSO and the proposed MSPSOER is that the proposed MSPSO has a probability having repeated states in a solution. In the case of the proposed MSPSOER, this issue is not occur as there is a procedure based on an embedded rule guarantees that any generation of solution free from unrepeated states.

3.8 The Proposed Multi-State Gravitational Search Algorithm with an Embedded Rule (MSGSAER)

In this section, the proposed MSGSAER is explained. The proposed MSGSAER fundamentally improves the proposed MSGSA in term of producing the unrepeated states in each solution. The improved proposed algorithm typically practices similar principle of the proposed MSGSA except in the mechanism of state transition between two states.

3.8.1 Embedded Rule in the MSGSAER

Based on the proposed MSPSOER, the proposed MSGSAER is designed in a similar way, in which this algorithm also employs the similar embedded rule. As already discussed in Section 3.7.1 the embedded rule eliminates the need of evolving solutions with repeated states into solutions with unrepeated states. When the velocity is updated and the new velocity value is obtained, the process of updating current state to next state of each dimension's solution is performed. To operate the embedded rule in the multi-state search space, some of elements should be identified, namely current state, new velocity value, the ISs, the OSs as described in Section 3.2.

For the ISs and the OSs, let's consider a set of j IS $\boldsymbol{I}_i(k,d) = \left(\boldsymbol{I}_{i_1}(k,d),\ldots,\boldsymbol{I}_{i_j}(k,d)\right)$ and a set of l OS $\boldsymbol{O}_i(k,d) = \left(\boldsymbol{O}_{i_1}(k,d),\ldots,\boldsymbol{O}_{i_l}(k,d)\right)$. A set of h SSs can be then derived as $\boldsymbol{T}_i(k,d) = \left(\boldsymbol{T}_{i_1}(k,d),\ldots,\boldsymbol{T}_{i_h}(k,d)\right)$. All the SSs should be identified because this type of state is invalid to be selected as the next state. Based on the current state and the new velocity value of each dimension, a next state of each dimension can be selected using Eq. (3.6) where $\emptyset$ is empty set. Eq. (3.6) operates in similar way as derived in Eq. (3.5). The observation of the solutions yielded can be made as Eq. (3.6) successfully generates unrepeated states in all solutions.

$$s_i(k+1,d) = \begin{cases} \text{random}\Big(\big(\boldsymbol{I}_i(k,d) - (\boldsymbol{I}_i(k,d) \cap \boldsymbol{T}_i(k,d))\big)\Big) \\ \text{if}\Big(\big(\boldsymbol{I}_i(k,d) - (\boldsymbol{I}_i(k,d) \cap \boldsymbol{T}_i(k,d))\big)\Big) \neq \emptyset \\ \\ \text{random}\Big(\big(\boldsymbol{O}_i(k,d) - (\boldsymbol{O}_i(k,d) \cap \boldsymbol{T}_i(k,d))\big)\Big) \\ \text{if}\Big(\big(\boldsymbol{I}_i(k,d) - (\boldsymbol{I}_i(k,d) \cap \boldsymbol{T}_i(k,d))\big)\Big) = \emptyset \end{cases} \qquad (3.6)$$

The algorithmic details of the process of updating each current state into a next state in the MSGSAER is explained in Figure 3.22. Meanwhile, the proposed MSGSAER is offered in Figure 3.23. The proposed MSGSAER has no chance having any repeated states in the generated solutions, affected by the implementation of the embedded rule, as discussed in Section 3.7.1.

```
input: $I$ is the number of agents, $D$ is the maximum dimensions,
        $s_i(k,d)(i=1,2,\ldots,I; d=1,2,\ldots,D)$ is current position of each agent and
        $v_i(k+1,d)(i=1,2,\ldots,I; d=1,2,\ldots,D$ is current velocity of each agent.
agent = 1;
repeat
    Initialize the SSs.
    dimension = 1;
    repeat
        Generate a circle subjected to current state and new velocity value to
        determine the ISs and the OSs
        Exclude the SSs from the ISs.
        if any of the ISs still occurs as the subject of selection then
            Randomly choose one of the ISs to be next state.
        Randomly choose one of the OSs to be next state.
        Register the selected next state to be one of the SSs.
        dimension ++;
    until dimension == $D$;
    agent ++;
until agent == I;
output: Solutions with unrepeated states generated.
```

Figure 3.22. The process of updating current state to be next state in the MSGSAER.

```
procedure MSGSAER
    Initialize agents with random positions and velocities.
    Set the agents to their current positions.
    Calculate the agents' fitness and set $G$, the best agent and the worst agent
    for $T$ generations do
        Evaluate the mass.
        Evaluate the force of the mass.
        Evaluate the acceleration of the mass.
        Update the agents' velocities.
        Update the agents' positions.
        Recalculate the agents' fitness.
        Update the $G$, the best agent and the worst agent.
    end for
end procedure
```

Figure 3.23. The general principle of the MSGSAER.

3.9 Applying the MSPSO, the MSGSA, the MSPSOER, and the MSGSAER to the TSP

This section presents the proposed approaches based on the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER applied to the TSP. Figure 3.24 shows the outline of the proposed approaches based on the MSPSO which includes initialization (parameters and the initialization of solutions), the evaluation of particles' fitness, the *pbest* of the population update, the *gbest* of the population update, the particles' velocities and positions update, and the evolvement of the infeasible particles solution to be feasible.

Similar to the original PSO, the MSPSO starts with an initialization operation that is necessary to configure the parameters and an initial solutions of the algorithm (see Chapter II). During the initialization, the initial solutions for the population are randomly generated. Prior to this, the generated initial solutions for the population are subjected to a validity check to ensure the initial solutions are valid and feasible.

```
procedure MSPSO for solving the TSP
    Initialize particles' solution and initialize velocities to zero.
    Build a distance matrix.
    Set the particles' pbests to their current positions.
    Calculate the particles' fitness and set gbest.
    for T generations do
        Update the particles' velocities.
        Update the particles' positions.
        Evolve the solution of particles with repeated states to the solution
        with repeated states
        Recalculate the particles' fitness.
        Update the particles' pbest and gbest.
    end for
end procedure
```

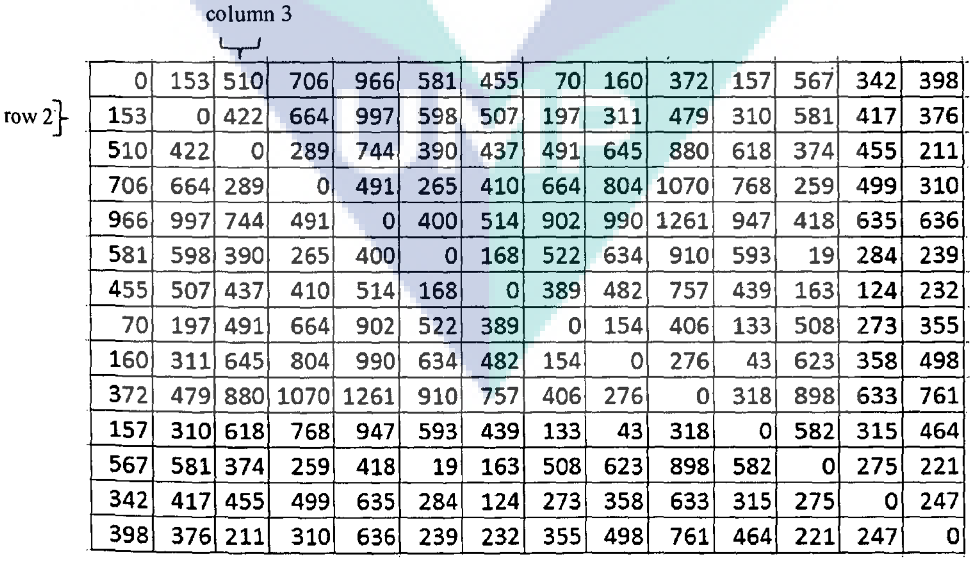
Figure 3.24. Outline of the proposed approach based on the MSPSO for solving the TSP.

In the TSP, the validity check is operated to ensure no city is repeated in each solution. The vector representation corresponding to the cities of a particle or an agent is represented in Figure 3.25; in this case, the sequence of the solution is 2-3-5-4-1. The length of a string depends on the total number of cities used in the calculating the tour distance. A distance matrix is then built in order to be used to calculate the fitness of each particle represented by a TSP sequence. For example, the distance matrix for Burma14 benchmark instance is depicted in Figure 3.26.

Figure 3.26 shows a distance matrix with size 14 x 14. The number of rows and columns are identical to the size of Burma14 benchmark instance of the TSP. The calculation of the fitness of each particle starts with calculating the distance between the first two cities in an initial solution (i.e. the first city is 2 and the second city is 3). For instance, the distance between city 2 (row 2) and city 3 (column 3) is 422 KM. This calculation is executed for other pairs of two cities and then all the distances are sum to obtain an ultimate distance in which all cities are visited such that no city is visited more than once and the salesman returns to the initial visited city at the end of the tour, that is

2	3	5	4	1

Figure 3.25. Example of a TSP sequence represented by a particle or an agent.


column 3
row 2

0	153	510	706	966	581	455	70	160	372	157	567	342	398
153	0	422	664	997	598	507	197	311	479	310	581	417	376
510	422	0	289	744	390	437	491	645	880	618	374	455	211
706	664	289	0	491	265	410	664	804	1070	768	259	499	310
966	997	744	491	0	400	514	902	990	1261	947	418	635	636
581	598	390	265	400	0	168	522	634	910	593	19	284	239
455	507	437	410	514	168	0	389	482	757	439	163	124	232
70	197	491	664	902	522	389	0	154	406	133	508	273	355
160	311	645	804	990	634	482	154	0	276	43	623	358	498
372	479	880	1070	1261	910	757	406	276	0	318	898	633	761
157	310	618	768	947	593	439	133	43	318	0	582	315	464
567	581	374	259	418	19	163	508	623	898	582	0	275	221
342	417	455	499	635	284	124	273	358	633	315	275	0	247
398	376	211	310	636	239	232	355	498	761	464	221	247	0

Figure 3.26. Distance matrix of Burma14 benchmark instance of the TSP.

the fitness of the particle. This calculation is applied to each particle. With regard to the fitness of the particles calculated, the *pbest* of each particle is then set to its position. The best of the *pbest* that is the *gbest* is then set.

Each iteration updates each particle's velocity and position. After updating the velocity and position, the solution of each particle with repeated states is evolved to the solution with unrepeated states. This operation is previously illustrated in Figure 3.15. The fitness of each particle is then calculated again. The *pbest* of each particle is then updated by a more optimal, obtained either from the new position of the particle of the *pbest*. The *gbest* is then updated by the most optimal *pbest* of all the particles. The optimum solution of the TSP represented by the *gbest* is yielded when the stopping condition is finally met. After the stopping condition is met, the performance of the proposed approach based on the MSPSO can be investigated.

Figure 3.27 provides the outline of the proposed approaches based on the MSPSOER that is designed mostly identical to the MSPSO except the evolvement of the solution with repeated states to the solution with unrepeated states does not require to be conducted. This is because by using the MSPSOER, all elements or states in any solution occurs just once.

```
procedure MSPSOER for solving the TSP
    Initialize particles solution and initialize velocities set to zero.
    Build a distance matrix.
    Set the particles' pbests to their current positions.
    Calculate the particles' fitness and set gbest.
    for T generations do
        Update the particles' velocities.
        Update the particles' positions.
        Recalculate the particles' fitness.
        Update the particles' pbest and gbest.
    end for
end procedure
```

Figure 3.27. Outline of the proposed approach based on the MSPSOER for solving the TSP.

Figure 3.28 presents the general architecture of the proposed approach based on the MSGSA, which includes initialization (parameters and the initialization of solutions), the evaluation of the agents' fitness, the *G*, the best agent and the worst agent update, the evaluation of the mass, the force of the mass, and the acceleration of the mass, the agents' velocities and positions update, and the conversion of the solution of agents with repeated states to the solution with unrepeated states.

Similar to the original GSA, the MSGSA starts with an initialization operation that is necessary for configuring the parameters and an initial solutions of the algorithm (see Chapter II). During the initialization, the initial solutions for the population are randomly generated. Prior to this, the generated initial solutions for the population are subjected to a validity check to ensure the initial solutions are valid and feasible. In the TSP, the validity check is operated to ensure no city is repeated in each solution. The

```
procedure MSGSA
    Initialize agents with random positions and velocities.
    Build a distance matrix.
    Set the agents to their current positions.
    Calculate the agents' fitness and set G, the best agent and the worst agent
    for T generations do
        Evaluate the mass.
        Evaluate the force of the mass.
        Evaluate the acceleration of the mass.
        Update the agents' velocities.
        Update the agents' positions.
        Convert the solution agents with repeated states to the solution
        with repeated states
        Recalculate the agents' fitness.
        Update the G, the best agent and the worst agent.
    end for
end procedure
```

Figure 3.28. Outline of the proposed approach based on the MSGSA for solving the TSP.

2	3	5	4	1

Figure 3.29. Example of a TSP sequence represented by an agent.

vector representation corresponding to the cities of an agent is represented in Figure 3.29; in this case, the sequence of the solution is 2-3-5-4-1. The length of a string depends on the total number of cities used in the calculating the tour distance.

The proposed approach based on the MSGSAER is provided in Figure 3.30. Similarly, as presented in the MSPSOER, the MSGSAER needs a distance matrix to calculate the fitness of each agent represented by a TSP sequence (please refer Figure 3.25). With regard to the fitness of the agents calculated, the *best* agent and the *worst* agent is then set. The gravitational constant *G* is also initialized. The mass, the force of the mass, and the acceleration for each agent is then calculated. Next, the velocity and position for each agent is updated. The updated solution of each agent is then evolved to the solution with unrepeated states. Prior to this, the operation is portrayed in Figure 3.15. The optimum solution is then selected from the feasible solutions by evaluating the fitness of each agent. The best of the population is the solution that has the sequence that is more optimal up until the stopping condition is met. After the stopping condition is met, the performance of the proposed approach based on the MSGSAER can be investigated. Similarly as in the MSPSOER, the MSGSAER does not require the step that converts the feasible solutions from the infeasible solutions.

```
procedure MSGSAER
    Initialize agents with random positions and velocities.
    Build a distance matrix.
    Set agents to their current positions.
    Calculate agents' fitness and set best agent and worst agent
    for $T$ generations do
        Update the G, best agent and worst agent.
        Evaluate mass.
        Evaluate force of mass.
        Evaluate acceleration of mass.
        Update agents' velocities.
        Update agents' positions.
        Recalculate agents' fitness.
        Update the G, best agent and worst agent.
    end for
end procedure
```

Figure 3.30. Outline of the proposed approach based on the MSGSAER for solving the TSP.

3.10 Applying the MSPSO, the MSGSA, the MSPSOER, and the MSGSAER to the ASP

This section presents the proposed approaches based on the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER applied to the ASP. Figure 3.31 shows the outline of the proposed approach based on the MSPSO which includes initialization sequence. The vector representation corresponding to the assembly components of a particle is represented in Figure 3.32; in this case, the sequence of the solution is 2-3-1-5-4. The length of a string depends on the total number of components used in the assembly process.

```
procedure MSPSO for solving the ASP
    Initialize particles' solution to feasible and initialize velocities to zero.
    Load PM, coefficient table, and actual assembly time.
    Set the particles' pbests to their current positions.
    Calculate the particles' fitness and set gbest.
    for T generations do
        Update the particles' velocities.
        Update the particles' positions.
        Evolve the updated assembly sequence of each particle to feasible
        assembly sequence.
        Recalculate the particles' fitness.
        Update the particles' pbest and gbest.
    end for
end procedure
```

Figure 3.31. Outline of the proposed approach based on the MSPSO for solving the ASP.

2	3	1	5	4

Figure 3.32. Example of an assembly sequence represented by a particle.

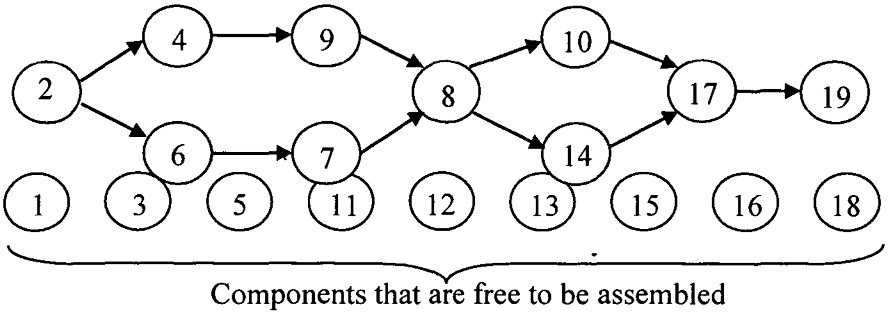


Figure 3.33. Assembly precedence diagram for the case study.

Source Choi, Lee, & Cho 2008

Figure 3.33 shows the assembly precedence diagram of a hypothetical product with nineteen components for the case study (Choi, Lee, & Cho, 2008). In this diagram, the components that are free to be assembled are the components that can be placed regardless of any part of a sequence.The precedence diagram can be translated into a PM as displayed in Table 3.2.

Occasionally, some respective assembly components cannot be integrated into a feasible assembly sequence. The determination of the assembly components that do not correspond to a feasible assembly sequence is achieved by satisfying all PM constraints between the components in the assembly, which are determined earlier, either from CAD or a disassembly analysis (Gao, Qian, Li, & Wang, 2009).

Table 3.2

Precedence Matrix (PM) for the Case Study.

Component to be assembled	Component *i*																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1																			
2																			
3																			
4		1																	
5																			
6		1																	
7		1				1													
8		1		1		1	1		1										
9		1		1															
10		1		1		1	1	1	1										
11																			
12																			
13																			
14		1		1		1	1	1	1										
15																			
16																			
17		1		1		1	1	1	1	1				1					
18																			
19		1		1		1	1	1	1	1				1			1		

Source: Choi et al. (1997)

Table 3.3

Coefficient of Various Components in the Assembly

Component to be	**Component *i***																		
assembled	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
1	10	1	2	3	4	5	6	7	8	9	3.2	4.3	7	6.1	1.2	3.4	0	0	7.4
2	1.5	10	2	2	2	2	2	2	2	2	0	3.1	6	4.3	2.7	4.8	0	3	0.5
3	1	2.3	10	0	4	5	0	4	2.3	4.3	9.8	2.4	5	1.2	3.4	4.5	5.6	3.4	3.1
4	0	2	3.4	10	4.5	0	4	0	8	0	3.4	5.6	5	0	0	3.4	0	0	9.8
5	1.2	1	2	3	10	7.9	8.9	0	1.2	2	2.3	0	3	0	3.6	0	2.8	· 9.8	0
6	9.8	4.5	0	1.2	3.6	10	3.4	4	0	2.3	4.6	5.6	0	4	3	2	0	0.4	3.2
7	0.5	1.4	2.3	0.5	1.9	1	10	13.4	1.2	4	2.3	0	3	5.7	8.3	2	0.1	0	0.5
8	0	0	0	0	0	1.8	9.8	10	2.3	3	8.9	2.3	0	0	2.3	0.5	9.8	0	2.3
9	1	3	4.5	2.3	4.6	9.8	7.5	6.8	10	6	2.3	3.4	5	12.3	3.4	5.61	1	0	0
10	2.3	4.5	2.3	0	2.3	0	2.1	0	4.5	10	1.1	2.2	2	0	0	2.1	1.2	5.4	9.2
11	1	1	2	3	4	5	6	7	8	9	10	4.5	3	6.1	1.2	3.4	0.3	0	1.3
12	1.5	0	2	2	2	2	2	1	2	2	11.2	10	6	4.3	2.7	4.8	0	3	0.5
13	1	2.3	0	0	4	5	0	4	2.3	4.3	9.8	2.4	10	1.2	2.4	4.5	1.6	2.4	3.1
14	0	2	3.4	0	4.5	0	4	0	8	0	3.4	5.6	5	10	2.1	1.4	1	0	2.8
15	1.2	1	2	3	0	7.9	8.9	0	1.2	2	1.3	4	3	1.4	10	1.3	9.8	9.8	2
16	9.8	4.5	0	1.2	3.6	0	3.4	4	0	2.3	4.6	3.6	0	4	3	10	1.5	0	3.2
17	1	3	4	5	0	5	4	3.4	1.2	4	1.3	0	2	3.7	4.3	2.3	10	3.8	10
18	0.6	0.5	3.4	1.2	3	2	9.8	2	2.3	3	5.9	2.3	0	1.0	2.3	0.5	9.8	10	2.3
19	1	3	4.5	2.3	4.6	9.8	7.5	6.8	0	6	3.3	3	2	3.3	4.4	2.6	0.3	2.5	10

Source: Choi et al. (1997)

The coefficient table of the assembly is outlined in Table 3.3. A coefficient table represented in a matrix is then used to calculate the fitness of each particle represented by a feasible assembly sequence. With regard to the fitness of each particle calculated, the *pbest* of each particle is then set to its position. The best of the *pbest* that is the *gbest* is then set. In this study, the actual assembly time is set to 5 time unit.

To confirm the production of a feasible assembly sequence of each particle, the updated assembly sequence of each particle produced by the updating position process is evolved to a feasible assembly sequence. To assemble the components of the product in a valid manner, only the feasible assembly sequences should be used. The feasibility of the sequences can be determined by referring to the PM. Based on the PM, the algorithm that can evolve the updated assembly sequence of each particle to feasible assembly sequence is developed, as shown in the previous Figure 3.34.

Archiving has become an important part of the algorithm because it stores components that were not used during this feasibility evolution period of the updated assembly sequence of each particle. The maximum archive size is a fixed value according to how many components should be assembled. This algorithm removes each component in the archive with the selected feasible component based on the PM.

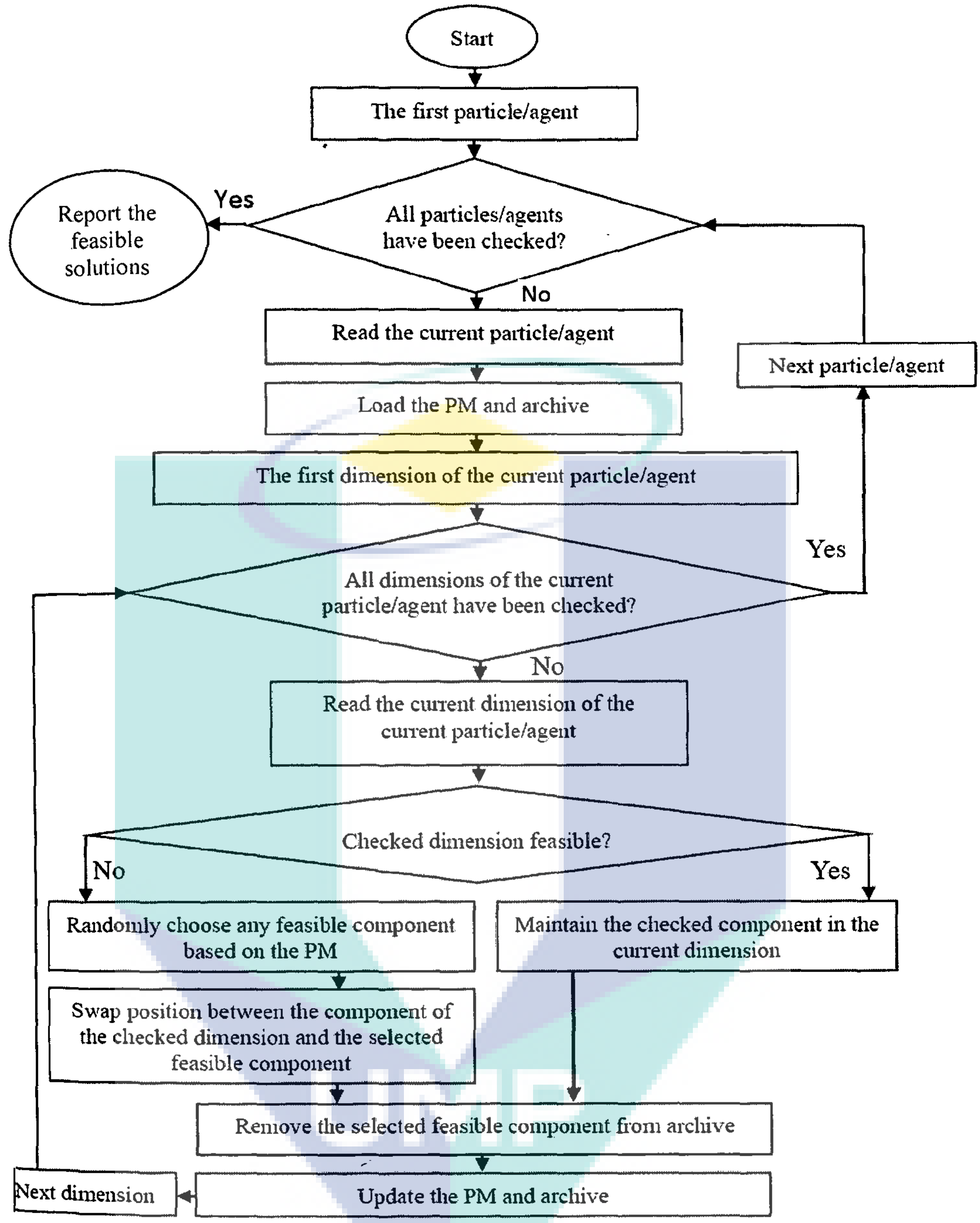

Start
The first particle/agent
All particles/agents have been checked?
Yes
Report the feasible solutions
No
Read the current particle/agent
Load the PM and archive
The first dimension of the current particle/agent
All dimensions of the current particle/agent have been checked?
Yes
Next particle/agent
No
Read the current dimension of the current particle/agent
Checked dimension feasible?
No
Yes
Randomly choose any feasible component based on the PM
Maintain the checked component in the current dimension
Swap position between the component of the checked dimension and the selected feasible component
Remove the selected feasible component from archive
Update the PM and archive
Next dimension

Figure 3.34. Process of evolving the updated assembly sequence of each particle or agent to be feasible.

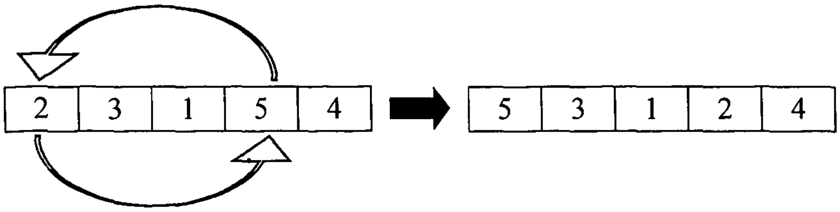


Figure 3.35. Swap position between two components in a sequence.

In this algorithm, the position of each sequence is swapped randomly between an infeasible component and a feasible component. For instance, the updated assembly sequence and the swap position between two components are shown in Figure 3.35. The swapping process ends when each component occurs in an assembly sequence in which the sequence is now feasible.

The evaluation of fitness of each particle is performed to improve the objective value. Hence, for each iteration, each particle represented by a feasible assembly sequence is evaluated. The *pbest* of each particle is then updated by a higher objective value, obtained either from the new position of the particle of the *pbest*. The *gbest* is then updated by the *pbest* with the highest objective value. The optimum solution of the ASP represented by the *gbest* is yielded when the stopping condition is finally met. After the stopping condition is met, the performance of the proposed approach based on the MSPSO can be investigated.

To evaluate the fitness for each agent, the total assembly time should be found. The total assembly time is the combination of the setup time and the actual assembly time. It is assumed that regardless of the assembly sequence, the actual assembly time is constant, and a proper tool and setup for each component to be assembled is required. These two items depend on the geometry of the component itself and the components assembled up to that point. The setup time for a component can be predicted using Eq. (2.26) as described in Section 2.7.2. The total assembly time is the summation of the setup time and the actual assembly time. Because the objective in this work to minimize assembly costs and time, the fitness function for minimizing the assembly time should be calculated by Eq. (2.27) as provided in Section 2.7.2.

Figure 3.36 provides the outline of the proposed approaches based on the MSPSOER that is designed mostly identical to the MSPSO except the evolvement of the infeasible solutions to feasible solutions does not require to be conducted. This is because by using the MSPSOER, all solutions obtained are valid and feasible. Meanwhile, Figure 3.37 presents the general architecture of the proposed approach based on the MSGSA, which includes initialization (parameters and initial population), the evaluation of fitness

```
procedure MSPSOER for solving the ASP
    Initialize particles' solution to feasible and initialize velocities to zero.
    Load PM, coefficient table, and actual assembly time.
    Set the particles' pbests to their current positions.
    Calculate the particles' fitness and set gbest.
    for T generations do
        Update the particles' velocities.
        Update the particles' positions.
        Recalculate the particles' fitness.
        Update the particles' pbest and gbest.
    end for
end procedure
```

Figure 3.36. Outline of the proposed approach based on the MSPSOER for solving the ASP.

```
procedure MSGSA for solving the ASP
    Initialize particles' solution to feasible and initialize velocities to zero.
    Load PM, coefficient table, and actual assembly time.
    Set the agents to their current positions.
    Calculate the agents' fitness and set G, the best agent and the worst agent.
    for T generations do
        Evaluate the mass.
        Evaluate the force of the mass.
        Evaluate the acceleration of the mass.
        Update the agents' velocities.
        Update the agents' positions.
        Convert the updated assembly sequence of each agent to feasible
        assembly sequence.
        Recalculate the agents' fitness.
        Update the G, the best and the worst agent.
    end for
end procedure
```

Figure 3.37. Outline of the proposed approach based on the MSGSA for solving the ASP.

of each agent, the G, the best agent and the worst agent update, the mass, the force of the mass, the acceleration of the mass, the update of the agents' velocities and positions, and the evolvement of the updated assembly sequence of each agent to feasible assembly sequence.

Similar to the original GSA, the MSGSA starts with an initialization operation that is necessary for configuring the parameters and the initial population (see Chapter II). During the initialization, the initial population is randomly generated. Prior to this, the generated initial population are subjected to a validity check using the PM described in Section 2.7.2 to ensure the initial population is valid and feasible. Each agent is represented by a sequence. The vector representation corresponding to the assembly components of an agent is represented in Figure 3.38; in this case, the sequence of the solution is 2-3-1-5-4. The length of a string depends on the total number of components used in the assembly process.

Because the similar assembly of a hypothetical product with nineteen components is considered, (Choi, Lee, & Cho, 2008), the PM and the assembly coefficient table, as presented in Table 3.2 and Table 3.3 are referred. The PM table is referred in order to ensure feasibility of the sequences, so that the components of the product can be assembled in a valid manner. Based on the PM, the algorithm that can evolve the updated assembly sequence of each agent to feasible assembly sequence is developed, as shown in the previous Figure 3.34.

The assembly coefficient table as presented in previous Table 3.3 is used to calculate the fitness of each agent represented by a feasible assembly sequence. With regard to the fitness of each agent calculated, the *best* agent and the *worst* agent is then set. In this study, the actual assembly time is set to 5 time unit. The gravitational constant G is also initialized. The mass, the force of the mass, and the acceleration for each agent

2	3	1	5	4

Figure 3.38. Example of an assembly sequence represented by an agent.

is then calculated. Next, the velocity and position for each agent is updated. The updated assembly sequence of each agent produced by the updating position process is then evolved to a feasible assembly sequence.

Archiving has become an important part of the algorithm because it stores components that were not used during this feasibility evolution period of the updated assembly sequence of each agent. The maximum archive size is a fixed value according to how many components should be assembled. This algorithm removes each component in the archive with the selected feasible component based on the PM. In this algorithm, the position of each sequence is swapped randomly between an infeasible component and a feasible component. The swapping process ends when each component occurs in an assembly sequence in which the sequence is now feasible.

The evaluation of fitness of each agent is performed to improve the objective value. Hence, for each iteration, each agent represented by a feasible assembly sequence is evaluated. The agent with the highest objective value for current iteration is then compared with the agent with the highest objective value for previous iterations. If the agent with the highest objective value for the current iteration is better than the agent with the highest objective value for previous iterations, the agent with the highest objective value for the current iteration is updated to be the agent with the highest objective value for all iterations. The optimum solution of the ASP represented by the *best* agent is yielded when the stopping condition is finally met. After the stopping condition is met, the performance of the proposed approach based on the MSGSA can be investigated. To evaluate the fitness for each agent, the total assembly time should be found. The total assembly time is the combination of the setup time and the actual assembly time. It is assumed that regardless of the assembly sequence, the actual assembly time is constant, and a proper tool and setup for each component to be assembled is required. These two items depend on the geometry of the component itself and the components assembled up to that point. The setup time for a component can be predicted using Eq. (2.26) as described in Section 2.7.2. The total assembly time is the summation of the setup time and the actual assembly time. Because the objective in this work to minimize assembly costs and time, the fitness function for minimizing the assembly time should be calculated using Eq. (2.27) as provided in Section 2.7.2.

```
procedure MSGSAER for solving the ASP
    Initialize particles' solution to feasible and initialize velocities to zero.
    Load PM, coefficient table, and actual assembly time.
    Set the agents to their current positions.
    Calculate the agents' fitness and set $G$, the best agent and the worst agent.
    for $T$ generations do
        Evaluate the mass.
        Evaluate the force of the mass.
        Evaluate the acceleration of the mass.
        Update the agents' velocities.
        Update the agents' positions.
        Recalculate the agents' fitness.
        Update the $G$, the best agent and the worst agent.
    end for
end procedure
```

Figure 3.39. Outline of the proposed approach based on the MSGSAER for solving the ASP.

Figure 3.39 provides the outline of the proposed approaches based on the MSGSAER that is designed mostly identical to the MSGSA except the evolvement of the infeasible solutions to feasible solutions does not require to be conducted. This is because by using the MSGSAER, all solutions obtained are valid and feasible.

3.11 The Evaluation

This thesis records evaluation by comparing between the algorithms with regard to the quality of solutions. The comparison is carried out in order to observe the average or minimum cost of solution. A statistical (descriptive test) is then executed to calculate the minimum, maximum, mean, and standard deviation of the algorithms. Firstly, a statistical test (Wilcoxon's Signed Rank Test) substitute for the t-test is executed for paired-sample data in doing inferences concerning the value of the median of the population of differences. Secondly, a statistical test (Friedman) is then performed for more than two sample data in order to test if there is a significant difference between the tested algorithms or the parameters used that yield the quality of solutions. A post-hoc procedure (Holm) is eventually conducted to find which pairs of algorithms of parameters that have the significant difference.

3.11.1 Accuracy Test: the Descriptive Test

One of critical aspect of this research is to acquire a significant interpretation of the experimental results, so the performance for a certain algorithm can be ultimately clarified. With regard to this matter, descriptive test is selected to describe the main feature of the results quantitatively. Inferential or inductive statistics employ the sample of data to get information about the population on the ground of the probability theory (Upton & Cook, 2002). Meanwhile, the descriptive test emphasizes in observing the big picture of the sample of data. The significant variable used in descriptive test is the mean or average; to determine the pivotal propensity of data (Mann, 2007). The median, mean, and mode are the measurement of the pivotal propensity, while the minimum, the maximum variable, and the standard deviation are the measurement of variability.

This study uses the mean values to determine the average performance of the algorithm, which represent the quality of solution. The performance of the algorithms based on the quality of solutions are then compared and analysed.

3.11.2 Significant Test: the Statistical Wilcoxon's Signed-Rank Test, Friedman Test and Holm Procedure

In this research, using descriptor test solely to value the results is insufficient to justify the assumption or hypothesis stated. So, significant testing is used to impose statistical significance of differences between the performances of the algorithms or parameters setting. The commonly used paired test, i.e., the parametric *t*-test and its nonparametric alternative Wilcoxon's Signed Rank tests, are sufficient when comparing paired samples. However, these two tests are not sufficient when employing multiple comparisons due to the so-called multiplicity effect (Salzberg, 1997). Thus, one of the rank-based nonparametric called Friedman test (Friedman, 1937) is recommended to be used followed by proper post-hoc procedures for identifying pairs of algorithms or parameters which differ significantly.

The most popular statistical tests used to determine significant differences between two algorithms are the *t*-test and the Wilcoxon's Signed-Rank test (Wilcoxon, 1945). These two tests are parametric that require the necessary conditions for a safe usage of parametric tests should be carried out; independence, normality, heteroscedasticity (Sheskin, 2011 and Zar, 2009). Hence, the nonparametric Wilcoxon matched pairs test, in which less powerful that *t*-test should be conducted. However, when the best one of a set of several algorithms should be found out, the pairwise comparisons are not suitable because the losing control over the so-called familywise error rate subjected to an accumulated error yielded from the combination of pairwise comparisons. Thus, sufficient tests and post-hoc procedures should be employed to multiple comparisons to compare between a control algorithm and other algorithms (1 x N comparisons) or to perform all possible pairwise comparisons (N x N comparisons).

In our investigation, nonparametric Wilcoxon's Signed-Rank test (Wilcoxon, 1945) and Friedman test (Friedman, 1937) are performed. In performing Wilcoxon's Signed-Rank test, normally, the population would have a median, symmetrical, and be continuous. The differences between the variates are listed and ranked; the largest is assigned as the highest rank. If the case of ties occur, each should be appointed to a shared rank.

The test statistic for the Wilcoxon Signed Rank Test is W, defined as the smallest of $W+$ and $W-$ which are the sums of the positive and negative ranks, respectively. The level of significance and is set and sample size is set. The critical value for two-sided test is found in statistical table and the decision rule is as follows: Reject the null hypothesis if $W \leq 1$. If the null hypothesis is rejected, a justification can be made that the two samples are significantly different. The test statistic can be obtained using Eq. (3.7).

$$W = \min(W+, W-) \tag{3.7}$$

where min is a minimum function which is used to find the smaller value between $W+$ and $W-$.

On the other hand, Friedman test ranks the algorithms or parameters from the best performing one to the poorest one. In other words, for the ranking of the algorithms, the best performing algorithm is states as the rank of 1, the second best received the second rank, etc.; in the case of ties average ranks are assigned. Let r_i^j be the rank of the j^{th} of k algorithms for the i^{th} of n data sets. The Friedman test compares the average or mean ranks of algorithms, using $R_j = 1/n\Sigma_i r_i^j$. The null hypothesis state that all the algorithms perform evenly and therefore their ranks R_j should be equal. Based on this hypothesis the Friedman statistic is given as in Eq. (3.8) where χ^2_F distributed with k-1 degrees of freedom.

$$\chi_F^2 = \frac{12n}{k(k+1)}\left[R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{3.8}$$

In order to determine whether to accept or reject the null hypothesis, it is essential to have a critical value χ^2_α. The critical value χ^2_α depends on significance level α. Table 3.4 that represents the table for the chi-square distribution can only be used after knowing

Table 3.4
Chi-Square Table

D.f.	Chi-square							
	$\chi^2_{.005}$	$\chi^2_{.025}$	$\chi^2_{.05}$	$\chi^2_{.90}$	$\chi^2_{.95}$	$\chi^2_{.975}$	$\chi^2_{.99}$	$\chi^2_{.995}$
	0.0004	0.0010	0.0039	2.706	3.8410	5.0240	6.6350	7.8790
	0.0100	0.0506	0.1030	4.6050	5.9910	7.3780	9.2100	10.5970
	0.0717	0.2160	0.0352	6.2510	7.8150	9.3480	11.3450	12.8380
	0.2070	0.4840	0.7110	7.7790	9.4880	11.1430	13.2770	14.8600
	0.4120	0.8310	1.1450	9.2360	11.0700	12.8320	15.0860	16.7500
	0.6760	1.2370	1.6350	10.6450	12.5920	14.4490	16.8120	18.5480
	0.9890	1.6900	2.1670	12.0170	14.0670	16.0130	18.4750	20.2780
	1.3440	2.1800	2.7330	13.3620	15.5070	17.5350	20.0900	21.9550
	1.7350	2.7000	3.3250	14.6840	16.9190	19.0230	21.6660	23.5890
	2.1560	3.2470	3.9400	15.9870	18.3070	20.4830	23.2090	25.1880
	2.6030	3.8160	4.5750	17.2750	19.6750	21.9200	24.7250	26.7570
	3.0740	4.4040	5.2260	18.5490	21.0260	23.3360	26.2170	28.3000
	3.5650	5.0090	5.8920	19.8120	22.3620	24.7360	27.6880	29.8190
	4.0750	5.6290	6.5710	21.0640	23.6850	26.1190	29.1410	31.3190
	4.6010	6.2620	7.2610	22.3070	24.9960	27.4880	30.5780	32.8010
	5.1420	6.9080	7.9620	23.5420	26.2960	28.8450	32.0000	34.2670

Note: D.f. = degree of freedom
Source: Rothman (2012)

the significance level α. The correct critical value χ^2_α should be selected from Table 3.4 by knowing which row and column to be used. The column is determined by the significance level α. The row is determined by $k - 1$ degree of freedom. The intersection of the column and row produces the critical value χ^2_α. For example, if the degree of freedom is 5, the $k - 1$ degree of freedom should be 4 and the critical value $\chi^2_{.95}$ is 9.488. For any selected significance level α, the null hypothesis can be rejected if the computed value χ^2_F is greater than critical value χ^2_α for chi-square distribution having $k - 1$ degrees of freedom.

The Friedman test can only detect the presence of differences over the whole multiple comparison. If the null hypothesis of similarity of rankings is rejected by the Friedman test, post-hoc procedures can be then conducted to find the particular pairs of algorithms which has a significant difference. In this research, Holm procedure is employed (Holm, 1979), which checks sequentially the ordering of hypotheses based on p-values from the lowest to the highest. All hypotheses for which P-value is less than the significance level α divided by the number of algorithms minus the number of a successive step are rejected. All hypotheses that have greater P-values are supported. A brief of Holm procedure and the formula for computation of the adjusted P-value (APV) is given in Table 3.5. In this study, a multiple comparison (N x N comparison) in which all possible pairwise comparisons need to be computed is carried out.

Table 3.5
Post-Hoc Procedure (Holm) for N x N Comparison.

Procedure	**Description**	***APV$_i$***
Holm	Step-down method, it rejects H$_1$ to H$_{i-1}$ if i is the smallest integer such that $p_i > \alpha/(k(k - 1)/2 - i + 1)$	$\min\{v;1\}$, where $v = \max\{(k(k - 1)/2 - j + 1)p_j : 1 \leq j \leq i\}$

Source: Holm (1979)

The notation used in Table 3.5 is outlined as follows:

- indexes i and j are used in comparison or hypothesis in the family of hypotheses. Index i is subjected to the hypothesis where APV is being determined and index j refers to another hypothesis.

- p_j is the *P*-value that calculated for the j^{th} hypothesis. *P*-value can be found using the table of two tails of Z as shown in Table 3.6.
- *k* is the number of predictors that being compared.

Table 3.6

Two Tails of Z.

Tenths	**Hundredths**									
	0.00	**0.01**	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**
0.0	0.9203	0.9124	0.9045	0.8966	0.8887	0.8808	0.8729	0.8650	0.8572	0.8493
0.1	0.8415	0.8337	0.8259	0.8181	0.8103	0.8026	0.7949	0.7872	0.7795	0.7718
0.2	0.7642	0.7566	0.7490	0.7414	0.7339	0.7263	0.7189	0.7114	0.7040	0.6965
0.3	0.6892	0.6818	0.6745	0.6672	0.6599	0.6527	0.6455	0.6384	0.6312	0.6241
0.4	0.6171	0.6101	0.6031	0.5961	0.5892	0.5823	0.5755	0.5687	0.5619	0.5552
0.5	0.5485	0.5419	0.5353	0.5287	0.5222	0.5157	0.5093	0.5029	0.4965	0.4902
0.6	0.4839	0.4777	0.4715	0.4654	0.4593	0.4533	0.4473	0.4413	0.4354	0.4295
0.7	0.4237	0.4179	0.4122	0.4065	0.4009	0.3953	0.3898	0.3843	0.3789	0.3735
0.8	0.3681	0.3628	0.3576	0.3524	0.3472	0.3421	0.3371	0.3321	0.3271	0.3222
0.9	0.3173	0.3125	0.3077	0.3030	0.2983	0.2937	0.2891	0.2846	0.2801	0.2757
1.0	0.2713	0.2670	0.2627	0.2585	0.2543	0.2501	0.2461	0.2420	0.2380	0.2341
1.1	0.2301	0.2263	0.2225	0.2187	0.2150	0.2113	0.2077	0.2041	0.2006	0.1971
1.2	0.1936	0.1902	0.1868	0.1835	0.1803	0.1770	0.1738	0.1707	0.1676	0.1645
1.3	0.1615	0.1585	0.1556	0.1527	0.1499	0.1471	0.1443	0.1416	0.1389	0.1362
1.4	0.1336	0.1310	0.1285	0.1260	0.1236	0.1211	0.1188	0.1164	0.1141	0.1118
1.5	0.1096	0.1074	0.1052	0.1031	0.1010	0.0989	0.0969	0.0949	0.0930	0.0910
1.6	0.0891	0.0873	0.0854	0.0836	0.0819	0.0801	0.0784	0.0767	0.0751	0.0735
1.7	0.0719	0.0703	0.0688	0.0673	0.0658	0.0643	0.0629	0.0615	0.0601	0.0588
1.8	0.0574	0.0561	0.0549	0.0536	0.0524	0.0512	0.0500	0.0488	0.0477	0.0466
1.9	0.0455	0.0444	0.0434	0.0424	0.0414	0.0404	0.0394	0.0385	0.0375	0.0366
2.0	0.0357	0.0349	0.0340	0.0332	0.0324	0.0316	0.0308	0.0300	0.0293	0.0285
2.1	0.0278	0.0271	0.0264	0.0258	0.0251	0.0245	0.0238	0.0232	0.0226	0.0220
2.2	0.0215	0.0209	0.0203	0.0198	0.0193	0.0188	0.0183	0.0178	0.0173	0.0169
2.3	0.0164	0.0160	0.0155	0.0151	0.0147	0.0143	0.0139	0.0135	0.0131	0.0128
2.4	0.0124	0.0121	0.0117	0.0114	0.0111	0.0108	0.0105	0.0102	0.0099	0.0096
2.5	0.0093	0.0091	0.0088	0.0085	0.0083	0.0081	0.0078	0.0076	0.0074	0.0072
2.6	0.0069	0.0067	0.0065	0.0063	0.0061	0.0060	0.0058	0.0056	0.0054	0.0053
2.7	0.0051	0.0050	0.0048	0.0047	0.0045	0.0044	0.0042	0.0041	0.0040	0.0039
2.8	0.0037	0.0036	0.0035	0.0034	0.0033	0.0032	0.0031	0.0030	0.0029	0.0028
2.9	0.0027	0.0026	0.0025	0.0025	0.0024	0.0023	0.0022	0.0021	0.0021	0.0020
3.0	0.0019	0.0019	0.0018	0.0018	0.0017	0.0016	0.0016	0.0015	0.0015	0.0014
3.1	0.0014	0.0013	0.0013	0.0012	0.0012	0.0012	0.0011	0.0011	0.0010	0.0010
3.2	0.0010	0.0009	0.0009	0.0009	0.0008	0.0008	0.0008	0.0008	0.0007	0.0007
3.3	0.0007	0.0007	0.0006	0.0006	0.0006	0.0006	0.0005	0.0005	0.0005	0.0005
3.4	0.0005	0.0005	0.0004	0.0004	0.0004	0.0004	0.0004	0.0004	0.0003	0.0003
3.5	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0002	0.0002	0.0002
3.6	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002
3.7	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
3.8	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001

Source: Gerstman (2006)

3.12 Research Methodology

The research methodology employed in this study is presented in Figure 3.40. The ultimate goal of presenting this figure is to offer a brief summary of the research flow in a scientific manner. The research methodology has to be robust to optimize the time consumption in collecting and analysing data. The research process is a step by step process of developing a scientific investigation. These steps give the basic foundations of a systematic approach in a research. The steps conducted in this research are listed as follows: formulation of the research problem, identification of existing literature, identification of objectives and scope, design and propose a model, implementation of the model into the considered algorithms, enhancement of the proposed model,

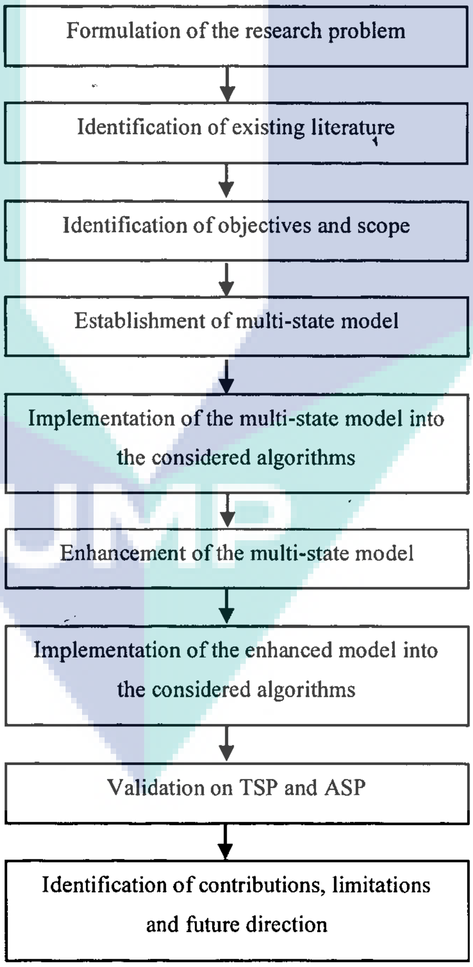

Formulation of the research problem
Identification of existing literature
Identification of objectives and scope
Establishment of multi-state model
Implementation of the multi-state model into the considered algorithms
Enhancement of the multi-state model
Implementation of the enhanced model into the considered algorithms
Validation on TSP and ASP
Identification of contributions, limitations and future direction

Figure 3.40. Research methodology.

implementation of the enhanced model into the considered algorithms, validation of the continuous optimization algorithms that adopted the multi-state model and the enhanced multi-state mode on the selected case studies, and identification of contributions, limitations and future direction.

3.12.1 Formulation of the Research Problem

This is the most vital part of research because the problem decides not only the nature and quality of the investigation, but also the design, methods and procedures of the study. There are 3 steps that should be carried out to identify the research problem; the selection of a topic area, the selection of a general problem in that area, and the reduction of the general problem to some precise and well defined problem. In this research, optimization problem is selected as the topic area. In the context of optimization problem, one of general problems called discrete optimization problems is selected. Discrete COPs is then considered to be the precise and well defined problem among discrete optimization problems.

Other than binary-coded optimization algorithms, there are plenty of continuous optimization algorithms that use different discretization approaches to solve discrete COPs. These approaches operate either in real-valued, binary or set-based search space, but none in multi-state search space. Owing to this, the multi-state model is identified as a potential model to handle discrete COPs and at the same time, it has an advantage over binary-based optimization algorithms in representing solutions with less computational complexity.

3.12.2 Identification of Existing Literature

Basically, background information is required to understand the research problem clearly. For this reason, the process of reviewing what others have done in a related area should be executed. In this research, the literature review is conducted by reading the conference papers and reviewed journal, book chapters and thesis. It begins by overviewing continuous optimization algorithms such as PSO and GSA that make use of different discretization approaches in solving discrete COPS in order to grasp the big picture of achievement of this area. PSO and GSA are reviewed because these two algorithms share a few similar features in their operation and also there has been intense interest in the use of these two algorithms to solve many optimization problems. After that, several discrete COPs are identified as the selected case studies. During this stage, the research activity is separated into two primary tasks, as follows.

1. **Literature survey.** A substantial literature is devoted to identify the existing continuous optimization algorithms that make use of the developed discretization approaches to solve discrete COPs. It starts with specification of keyword such as "discrete optimization algorithms", "binary optimization algorithms", "the improved optimization algorithms", and "the enhanced optimization algorithms". Based on the selected keywords, an extensive search of papers is performed. The electronic databases used for the search include Google Scholar and Scopus. The papers are then filtered to select only relevant papers based on information contained in the abstract, title and keywords. The selected papers are eventually fully studied and evaluated to obtain a clear picture in the research area.
2. **Identify the research gaps.** The major contribution of this study that is to fill the research gap in identifying the new discretization approach adoption for discrete COPs has been done.

3.12.3 Identification of Objectives and Scope

The research objectives are determined based on the research problem. The objectives are defined with respect to research gaps noted from literature review, so the list of objectives are in alignment with current research trends and handle particular limitations of previous works. The research objectives serves as guidelines of the task list of things that need to be done. Meanwhile, the research scope refers to the boundaries which the study is operated.

3.12.4 Establishment of Multi-State Model

In this research, multi-state model is developed to provide a new kind of discretization approaches. The proposed model must be able to solve discrete COPS. The design of the multi-state is illustrated using Burma 14 benchmark instance of TSP.

3.12.5 Implementation of the Multi-State Model into the Considered Algorithms

In this stage, the multi-state model is embedded in the considered continuous optimization algorithms. Concerning this, a few modifications have been done on the formulation without violating the core of updating mechanism in the original algorithms.

3.12.6 Enhancement of the Multi-State Model

In this research, the enhancement of multi-state model with an embedded rule is developed to eliminate limitation of multi-state model in which the state of its solution can be repeated. The enhanced model must be able to solve discrete COPS. The design of the enhanced multi-state is illustrated using Burma 14 benchmark instance of TSP.

3.12.7 Implementation of the Enhanced Model into the Considered Algorithms

In this stage, the enhanced multi-state model with an embedded rule is adopted in the considered continuous optimization algorithms. Concerning this, a few modifications have been done on the formulation without violating the core of updating mechanism in the original algorithms.

3.12.8 Validation on TSP and ASP

Validation of the continuous optimization algorithms that adopted the multi-state model (MSPSO and MSGSA) and the enhanced multi-state mode (MSPSOER and MSGSAER) are conducted to evaluate the performance of these four algorithms using two selected test problems which are TSP and ASP. In this stage, two primary research tasks are planned.

1. **Validate on TSP.** From literature, TSP has been identified as the first test problem. This problem covers small, medium, and large size of cities in order to analyse the performance of the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER on different sizes of cities. For comparison purposes, the MSPSO and the MSPSOER are compared with the BPSO. Meanwhile, the MSGSA and MSGSAER are compared with the BGSA.
2. **Validate on ASP.** From literature, ASP has been identified as the second test problem. ASP is selected because it shares similar characteristic with TSP but with an additional constraint which is the solutions should be generated according to its precedence matrix. For comparison purposes, the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER are compared with the BPSO, GA, and SA.

3.12.9 Identification of Contributions, Limitations and Future Direction

As the final stage, the research contributions to knowledge are identified by reviewing the proposed research model and its outcomes in line with the research objectives. On the other hand, the research gaps are also figured out to ensure how the contributions have filled the gaps. Following that, the limitations of this research are identified with regard to the respect future direction for this research in order to overcome the limitations and at the same time enhance the research in this area.

3.13 Summary

This chapter demonstrates the origin of multi-state model that is inspired from a sequential circuit in digital system. There are two primary features in the multi-state model; a current state and a radius. By implementing this model on PSO and GSA, two discrete optimization algorithms have been proposed, which are the MSPSO and the MSGSA. These two algorithms have modified the way to update their velocity and position. As aforementioned, the MSPSO and the MSGSA have a chance to produce solutions with repeated states. To solve the repetitive issue, the multi-state model is then extended by implementing an embedded rule which ensure that "each state can be chosen only once" in a solution. By implementing the extended multi-state model, the MSPSOER and the MSGSAER have been developed. In these two developed algorithms, solutions with unrepeated states have been successfully generated. For validation purposes, the proposed approaches based on the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER for solving the TSP and the ASP are then provided. Next, the chapter offers a description of the statistical tests to analyse either there are significant differences in term of performance between the algorithms in comparison. The chapter eventually elaborates the general process of this research step by step in research methodology.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

In this section, computation results of the proposed algorithms (i.e. the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER) and their binary-based algorithm (i.e. the BPSO and the BGSA) are presented. To evaluate the performance of these algorithms, several experiments were conducted on two categories of COPs; the TSP and the ASP.

In investigating the performance of the algorithms for the TSP, eighteen sets of TSP benchmark instances taken from TSPLIB are used (Reinelt, 1991). The experiments are divided into two stages; the first stage is the experiment that uses small size of TSP benchmark instances and the second stage uses medium and big size of TSP benchmark instance. The primary objective of the second stage of the experiment is to determine the performance of the proposed algorithms with bigger size of TSP benchmark instances.

On the other hand, an ASP consisting nineteen components is chosen as the case study (Motavalli & Islam, 1997; Choi et al., 2008 and Mukred et al., 2012). Each solution is in the form of a feasible assembly sequence. The optimum sequence is then selected from the feasible assembly sequences by evaluating the fitness of each solution. The best of the population is the sequence that is more optimal up until the stopping condition is met. After the stopping condition is met, the performance of the proposed approach using the proposed approach based on the MSPSO, the proposed approach based on the MSPSOER, the proposed approach based on the MSGSA, and the proposed approach based on the MSGSAER can be investigated. To conclude the finding, the performance of the proposed approach using the MSPSO and the MSGSA are compared against some related approaches embedded with meta heuristics, such as SA (Motavalli & Islam, 1997), the GA (Choi, Lee, & Cho., 2008), and the BPSO (Mukred et al., 2012). Table 4.1 offers an explanation for the eighteen sets of TSP benchmark instances that considered in this study by name, size, number of cities, and optimal route length.

The quality of results is measured based on the objective values of the best solutions found by each algorithm. For instance, if the number of independent run is 50, the quality of results is determined based on the fitness values of 50 solutions. The average (mean), minimum (min) and maximum (max) of fitness values of 50 solutions, and the standard deviation (SD) are recorded. The mean value is heavily used in the analysis of results that obtained after conducting experiments using the TSP benchmark instances to identify the consistency of each proposed algorithms.

Table 4.1
Characteristics of TSP Benchmark Instance.

TSP instance	Size	Number of cities	Optimal route length
Burma14	Small	14	3323
Ulysses16	Small	16	6859
Ulysses22	Small	22	7013
Bays29	Small	29	2020
Eil51	Small	51	426
Berlin52	Small	52	7542
Eil76	Medium	76	538
Rd100	Medium	100	7910
Eil101	Medium	101	629
Bier127	Medium	127	118282
Ch130	Medium	130	6110
Ch150	Medium	150	6528
Rd400	Big	400	15281
Rat783	Big	783	8806
Pr1002	Big	1002	259045
Rl1304	Big	1304	252948
Rl1323	Big	1323	270199
Rl1889	Big	1889	316536

Source: Reinelt (1991)

Meanwhile, the min value is heavily used in the analysis of results that obtained by conducting experiments using the ASP case study to identify the minimum assembly time and its associated best parameter settings.

4.2 The Performance of the MSPSO, the MSPSOER, and the BPSO for solving the TSP

This section offers the comparisons of the performance between the MSPSO, the MSPSOER, and the BPSO for problems with small and bigger size of the TSP benchmark instance. The small and bigger size of the TSP benchmark instances are considered with the number of city ranged from 14 to 52 and 76 to 1889, respectively. Due to sensitivity of algorithmic parameters, c_1, and c_2, for the MSPSO, the MSPSOER and the BPSO were chosen according to previous reported values (Kennedy & Eberhart, 1995 and Blum & Roli, 2003) that suggested a fixed value of 2.0. With the conviction that a large ω facilitates a global search while a small ω facilitates a local search, ωInitial and ωFinal are usually set to 0.9 to 0.4, respectively (Shi & Eberhart, 1998). This means that the ω can be interpreted as the fluidity of the medium in which a particle moves, indicating that setting it to 0.9 makes particles move in a low viscosity medium and performs extensive exploration. Gradually reducing it to 0.4 makes the particle moves in a high viscosity medium and performs more exploitation. The number of iteration T for the small and bigger size of the TSP benchmark instances are set to 10000 and 1000, respectively. T for the bigger size of the TSP benchmark instances is set smaller in order to speed up the computation in obtaining results. The total number of runs is 50 for the both two size categories.

In order to evaluate the performance of the algorithms in consideration, the indicators used are the quality of results and speed of convergence, and the superiority of results on individual runs. With regard to the speed of convergence, each particle of each algorithm records the objective value of the best solution found in every iteration. Convergence patterns for each algorithm are then constructed using the best solution of

Table 4.2

Parameter Settings of the MSPSO, the MSPSOER, and the BPSO on the Small Size of the TSP Benchmark Instances.

Parameter		Algorithm	
	MSPSO	MSPSOER	BPSO
Number of run	50	50	50
Number of iteration	10000	10000	10000
Number of particle	30	30	30
c_1 and c_2	2	2	2
r_1 and r_2	[0,1]	[0,1]	[0,1]
ω Initial	0.9	0.9	0.9
ω Final	0.4	0.4	0.4

Table 4.3

Parameter Settings of the MSPSO, the MSPSOER, and the BPSO on the Bigger Size of the TSP Benchmark Instances.

Parameter		Algorithm	
	MSPSO	MSPSOER	BPSO
Number of run	50	50	50
Number of iteration	1000	1000	1000
Number of particle	30	30	30
c_1 and c_2	2	2	2
r_1 and r_2	[0,1]	[0,1]	[0,1]
ω Initial	0.9	0.9	0.9
ω Final	0.4	0.4	0.4

each algorithm. Table 4.2 and Table 4.3 list the parameters and their respective value for the small and bigger size of the TSP benchmark instances of the MSPSO, the MSPSOER, and the BPSO, respectively.

4.2.1 The Performance of the MSPSO, the MSPSOER, and the BPSO for Small Size of the TSP Benchmark Instances

A summary of the performance of the MSPSO, the MSPSOER, and the BPSO for small size of the TSP benchmark instances is given in Table 4.4. The values reported are best, worst, average, and standard deviation of solutions over 50 independent runs. The MSPSOER yields smaller values for the minimum for Burma14, Ulysses16, and Berlin52 benchmark instances compared to the MSPSO and the BPSO, thus verifying that the MSPSOER produces higher quality solutions. Similarly, the MSPSOER yields smaller values for the minimum compared to the MSPSO and the BPSO in finding the quality of the results of Ulysses22, Bays29, and Eil51 benchmark instances; the MSPSOER produces higher quality solutions than the MSPSO and the BPSO.

Table 4.4
Performance of the MSPSO, the MSPSOER, and the BPSO on the Small Size of the TSP Benchmark Instances.

TSP instance	Algorithm	Min	Mean	Max	SD
Burma14	MSPSO	3411.00	3753.26	3955.00	144.00
	MSPSOER	3475.00	**3712.66**	4004.00	121.41
	BPSO	3527.00	3739.88	3829.00	101.69
Ulysses16	MSPSO	7499.00	7913.66	8204.00	200.64
	MSPSOER	7011.00	**7765.40**	8087.00	210.28
	BPSO	7572.00	7828.26	8203.00	163.38
Ulysses22	MSPSO	9603.00	9907.80	10297.00	234.09
	MSPSOER	9457.00	**9677.00**	9834.00	154.81
	BPSO	9565.00	10084.40	10499.00	367.60
Bays29	MSPSO	3669.00	3950.02	4126.00	99.78
	MSPSOER	3646.00	**3912.42**	4107.00	110.99
	BPSO	3747.00	3952.52	4114.00	96.25
Eil51	MSPSO	1184.00	1226.64	1266.00	20.85
	MSPSOER	1143.00	**1219.22**	1258.00	25.82
	BPSO	1218.00	1241.80	1264.00	16.38
Berlin52	MSPSO	21261.00	22021.40	22606.00	352.81
	MSPSOER	19980.00	**21686.10**	22360.00	497.20
	BPSO	21124.00	21853.20	22393.00	364.29

Note: Bold signifies the best mean result for each instance

On the other hand, the best result found from these 50 independent runs of each algorithm for each TSP benchmark instance algorithm are selected and then portrayed in several convergence patterns. Gbest fitness or global best, which is the best fitness value among swarm particles is recórded in the convergence patterns of each iteration. Figure 4.1 and Figure 4.2 present that the MSPSO converges slower for Burma14, Ulysses16, and Berlin52 benchmark instances.

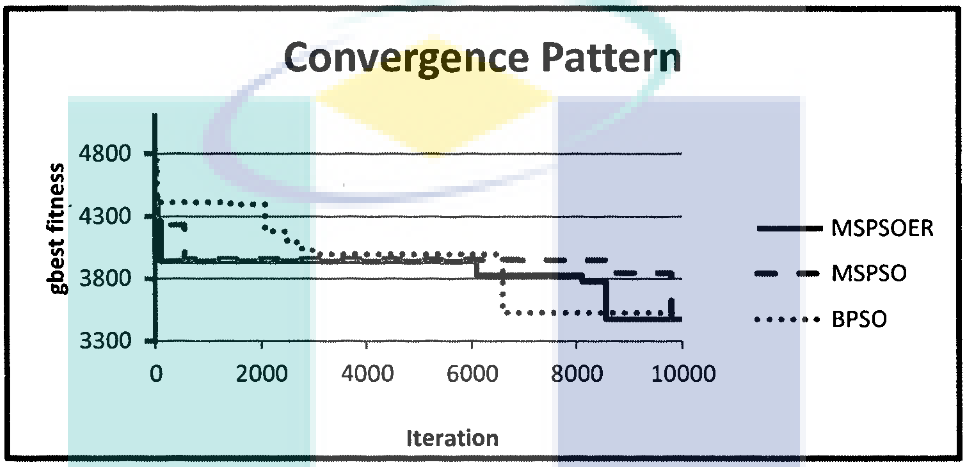


Figure 4.1. Comparison of the convergence pattern of the MSPSO, the MSPSOER, and the BPSO for Burma14 benchmark instance.

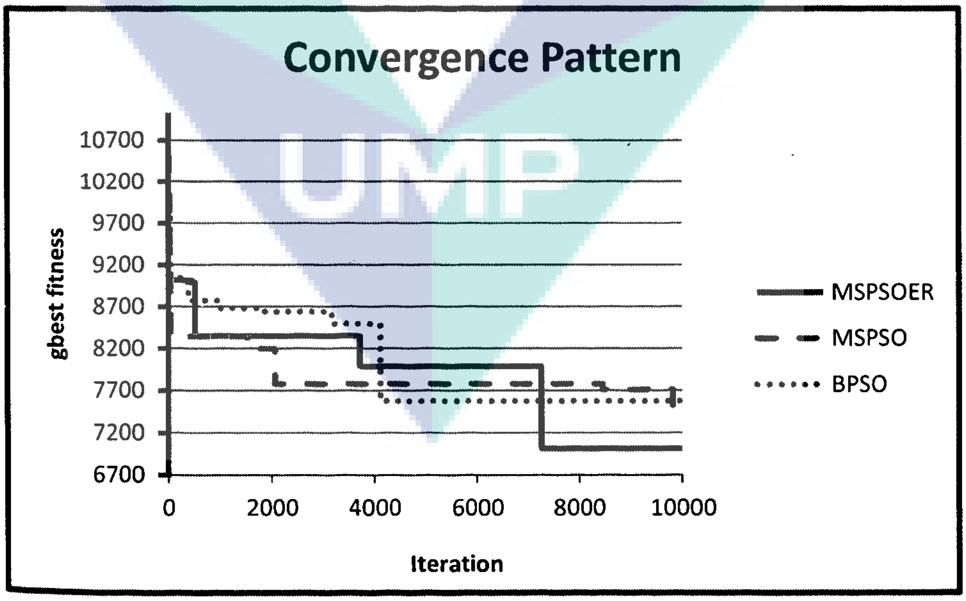


Figure 4.2. Comparison of the convergence pattern of the MSPSO, the MSPSOER, and the BPSO for Ulysses16 benchmark instance.

Figure 4.3 shows the BPSO converges slower for Ulysses22 benchmark instance. Meanwhile, Figure 4.4 illustrates that the MSPSOER converges slower for Bays29 and Eil51 benchmark instances.

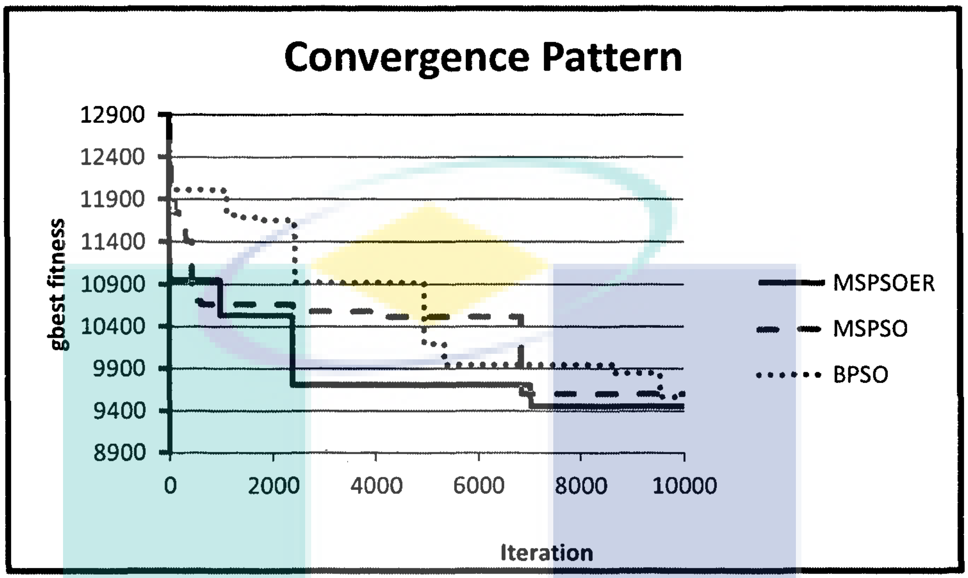


Figure 4.3. Comparison of the convergence pattern of the MSPSO, the MSPSOER, and the BPSO for Ulysses22 benchmark instance.

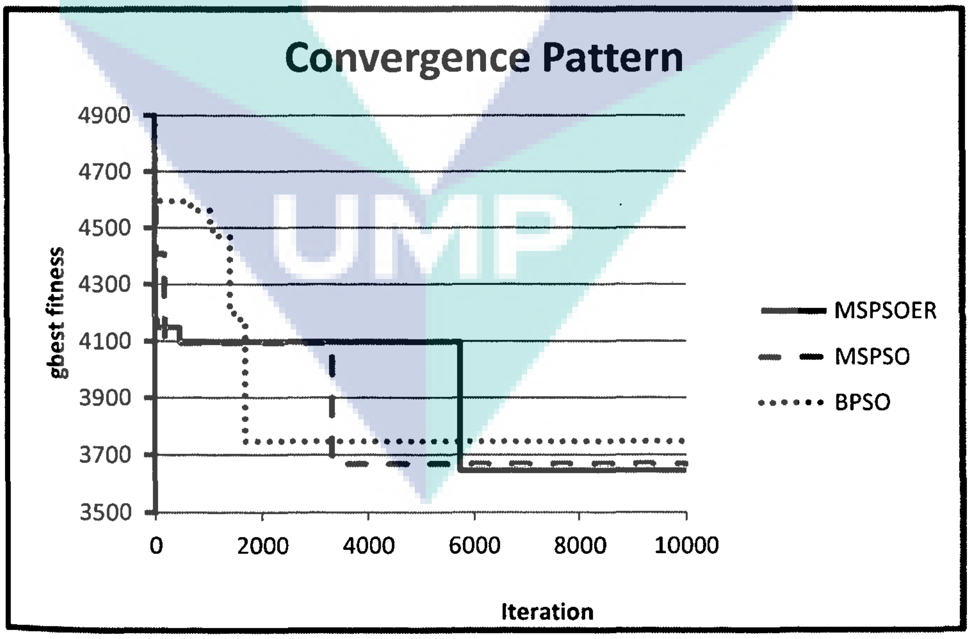


Figure 4.4. Comparison of the convergence pattern of the MSPSO, the MSPSOER, and the BPSO for Bays29 benchmark instance.

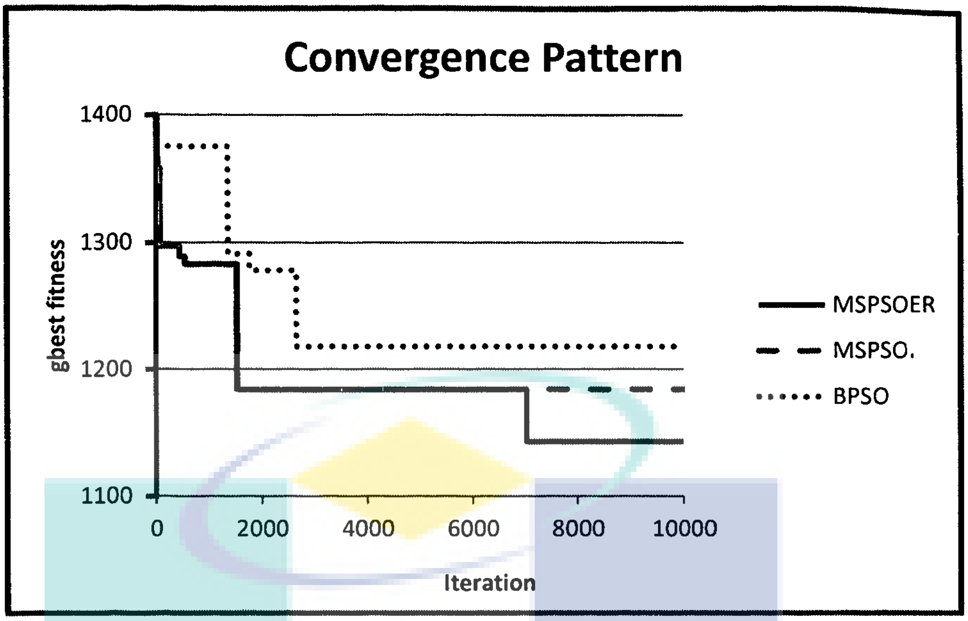


Figure 4.5. Comparison of the convergence pattern of the MSPSO, the MSPSOER, and the BPSO for Eil51 benchmark instance.

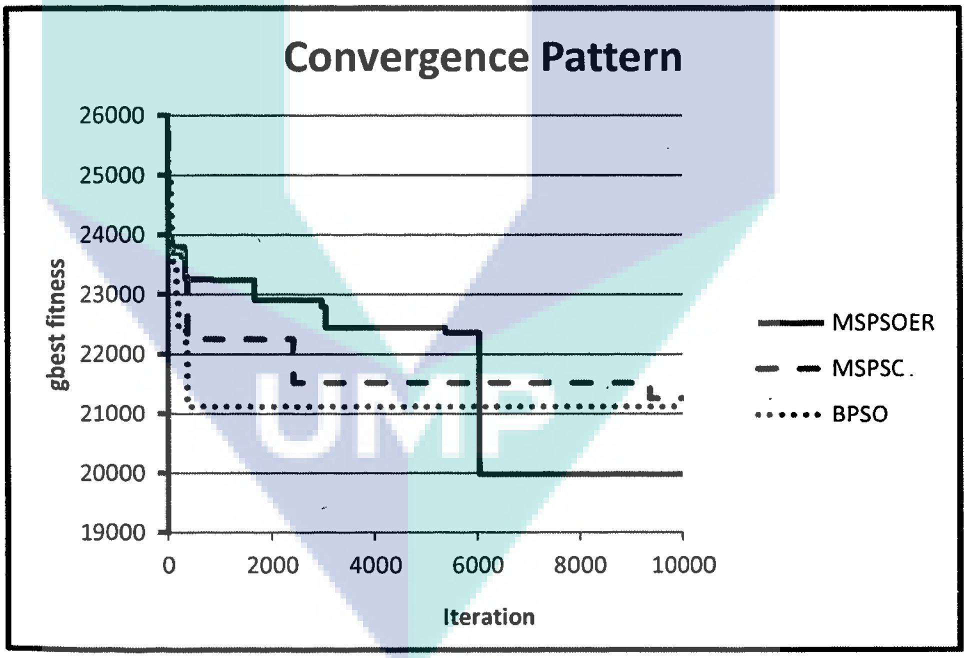


Figure 4.6. Comparison of the convergence pattern of the MSPSO, the MSPSOER, and the BPSO for Berlin52 benchmark instance.

Figure 4.5 shows that the MSPSOER converges slower for Bays29 and Eil51 benchmark instances. Figure 4.6 presents that the MSPSO converges slower for Burma14, Ulysses16, and Berlin52 benchmark instances.

Two comparisons between the individual runs for the MSPSO and the BPSO, and the MSPSOER and the BPSO are shown in Table 4.5 and 4.6. In these two tables, the number of times of the MSPSO and the MSPSOER perform better, similar, or worse than the BPSO are recorded. With regard to the individual runs conducted, it is observed that the MSPSO and the MSPSOER performs better than the BPSO in obtaining the best solution in each benchmark instance.

The difference between the solutions obtained by these four algorithms is also analysed using boxplots, as shown in Figure 4.7. In this figure, each rectangle represents one of the six benchmark instances (ranging from (a) to (f)). Inside each rectangle, boxplots representing the distribution of the best solution value for the MSPSO, the MSPSOER, and the BPSO are drawn. In each boxplot, the minimum and maximum values are the lowest and highest lines, the upper and lower ends of the box are the upper

Table 4.5
Number of Times of the MSPSO Performs Better, Similar, or Worse compared to the BPSO.

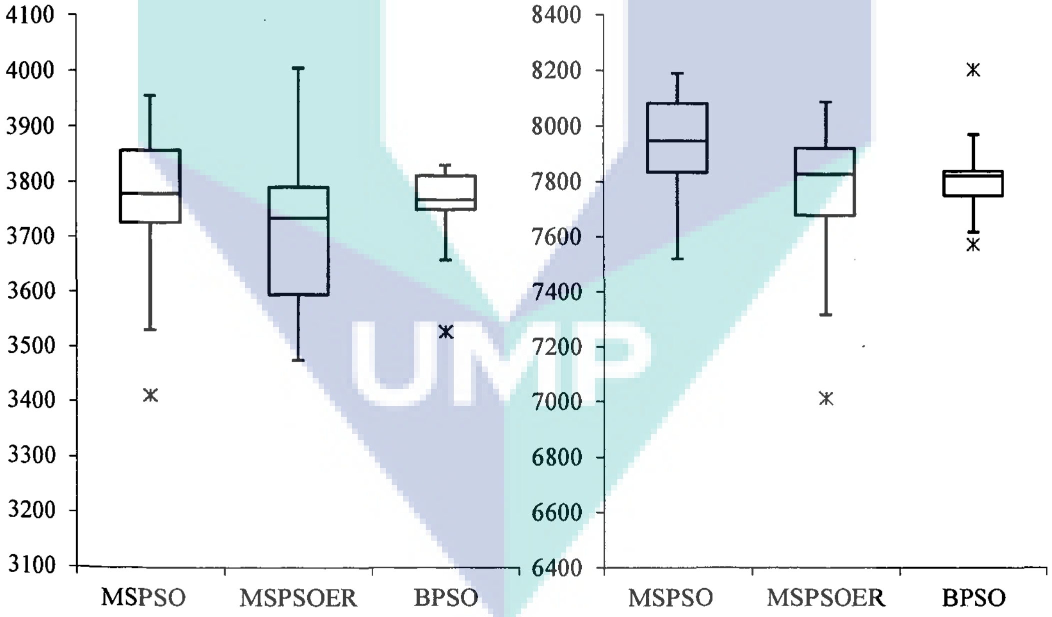
TSP instance		MSPSO vs. BPSO	
	Better	Similar	Worse
Burma14	28	0	22
Ulysses16	27	0	23
Ulysses22	35	0	15
Bays29	28	0	22
Eil51	35	0	15
Berlin52	27	0	23

Table 4.6
Number of Times of the MSPSOER Performs Better, Similar, or Worse compared to the BPSO.

TSP instance		MSPSO vs. BPSO	
	Better	Similar	Worse
Burma14	30	0	20
Ulysses16	29	0	21
Ulysses22	40	0	10
Bays29	30	0	20
Eil51	38	0	12
Berlin52	28	0	22

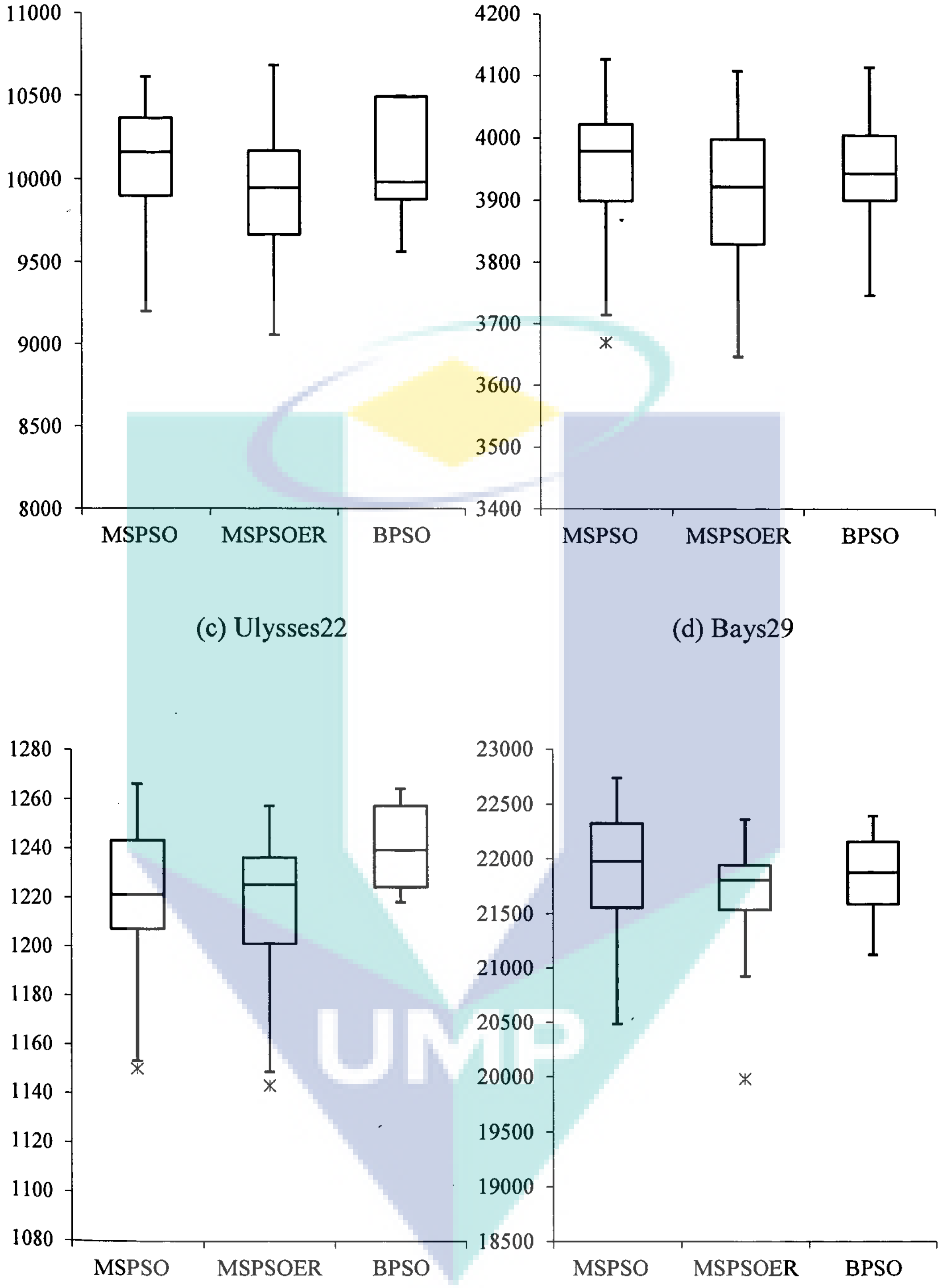
and lower quartiles, a line within the box shows the median, and the isolated points are the outliers of the distribution.

Each boxplot offers the information about the quality and the performance of the MSPSO, the MSPSOER, and the BPSO. The size of the box represents how varies the results. A smaller box indicates a constant performance of the parameters. In some occasions, the results obtained by the three algorithms contain outliers. The outliers should not be casted from measurement because they are feasible solutions. The pure outliers are actually produced with an actual measurement from the experiments and without clerical errors. The unusual marks are generated by the stochastic behaviour of the algorithms. Because the TSP is a minimization problem, a boxplot that its lower line closes to the minimum value is covetable since the boxplot demonstrates better quality of the solutions. It can be observed from Figure 4.7 that the MSPSOER give good performance on the six sets of TSP benchmark instances; the MSPSOER is promising.



(a) Burma14

(b) Ulysses16



(c) Ulysses22

(d) Bays29

(e) Eil51

(f) Berlin52

Figure 4.7. Boxplots of the MSPSO, the MSPSOER and the BPSO for (a) Burma14 (b) Ulysses16 (c) Ulysses22 (d) Bays29 (e) Eil51 and (f) Berlin52 benchmark instances.

It seems that nonparametric test should be used to compare the algorithms because the solutions are not normally distributed. Wilcoxon's Signed-Rank test is performed to the MSPSO and the BPSO in order to investigate either the quality of results between these two algorithms are identical or not; the null hypothesis is that the quality of results of these two algorithms are identical. The mean value should be used in executing the Wilcoxon's Signed-Rank test. As a consequence, rank information is provided as shown in Table 4.7.

The level of significance is set at 5% ($p < 0.05$) and sample size is 6. The critical value for two-sided test is 1 and the decision rule is as follows: Reject the null hypothesis if the test statistic, $W \leq 1$. With regard to the Wilcoxon's Signed Rank Test, the test statistic is $W = 6$, where $W^{+} = [3 + 2 + 1] = 6$ and $W^{-} = [6 + 4 + 5] = 15$. Therefore, the null hypothesis cannot be rejected because 6 > 1. The result is statistically no significant at $p < 0.05$, where the performance of the MSPSO is similar to the BPSO.

To investigate the performance of the MSPSO, the MSPSOER, and the BPSO, other nonparametric test is performed; the Friedman test with significance level 0.05 is performed using the experimental results from Table 4.4. This test is advisable for comparison that is more than two sets of algorithms (Dieterich, 2012).

Table 4.7
Wilcoxon's Signed-Rank Test Table.

TSP instance	**MSPSO**	**BPSO**	**Diff**	**Rank**
Burma14	3753.26	3739.88	13.38	3
Ulysses16	7913.66	7828.26	85.40	2
Ulysses22	9907.80	10084.40	-176.60	6
Bays29	3950.02	3952.52	-2.50	4
Eil51	1226.64	1241.80	-15.16	5
Berlin52	22021.40	21853.20	168.2	1

The test begins with sorting the three algorithms based on their average performance, as presented in Table 4.8. The average rank is then utilized to measure the Friedman statistic value. The Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (9.3468 > 5.9910). Thus, significant differences exist between the algorithms. Because significant difference exists, a post hoc procedure called Holm procedure is then performed and the three algorithms are compared to each other. This procedure can identify what are the pairs of the algorithms that have the significant difference. The result of the Holm procedure in Table 4.9 shows that there are significant differences exist between the performance of the MSPSO and the MSPSOER, and the MSPSOER and the BPSO because $P <$ Holm. Summary of the Holm procedure on these three algorithms is provided in Table 4.10.

Table 4.8
Friedman Test based on Results Computed in Table 4.4.

TSP Instance	**Item**	**MSPSO**	**MSPSOER**	**BPSO**
Burma14	Friedman Rank	3	1	2
Ulysses16	Friedman Rank	3	1	2
Ulysses22	Friedman Rank	2	1	3
Bays29	Friedman Rank	2	1	3
Eil51	Friedman Rank	2	1	3
Berlin52	Friedman Rank	3	1	2
Average Friedman rank		2.50	1.00	2.50

Table 4.9
Holm Procedure based on Average Friedman Rank in Table 4.8.

Algorithm	***P***	**z**	**Holm**
MSPSO vs. MSPSOER	0.0093	2.5981	0.0167
MSPSOER vs. BPSO	0.0093	2.5981	0.0250
MSPSO vs. BPSO	1.0000	0.0000	0.0500

Table 4.10
Performance of the MSPSO, the MSPSOER, and the BPSO Measurement using Friedman Test for Small Size of the TSP Benchmark Instances.

Summary	**Performance**
1	MSPSOER > MSPSO
2	MSPSOER > BPSO
3	MSPSO = BPSO

Note: ">" means better than and "="means similar

4.2.2 The Performance of the MSPSO, the MSPSOER, and the BPSO for Bigger Size of the TSP Benchmark Instances

The best, worst, average, and standard deviation of solutions recorded for each algorithm on medium size of the TSP benchmark instance is shown in Table 4.11.

Table 4.11
Performance of the MSPSO, the MSPSOER, and the BPSO on the Medium Size of the TSP Benchmark Instance.

TSP instance	Algorithm	Min	Mean	Max	SD
Eil76	MSPSO	1953.00	2030.68	2089.00	27.90
	MSPSOER	1890.00	**2025.20**	2111.00	58.27
	BPSO	1907.00	2036.86	2092.00	36.22
Rd100	MSPSO	43217.00	**45565.84**	47038.00	1102.53
	MSPSOER	44398.00	45802.72	47239.00	818.47
	BPSO	43060.00	45849.82	47147.00	802.57
Eil101	MSPSO	2715	2822.60	2878.00	38.65
	MSPSOER	2124	**2753.62**	2920.00	243.05
	BPSO	2687	2834.80	2936.00	53.46
Bier127	MSPSO	437345.00	529237.58	550213.00	31720.33
	MSPSOER	393989.00	**524965.40**	551755.00	49116.56
	BPSO	486034.00	534565.08	550759.00	12835.19
Ch130	MSPSO	37042.00	**38862.52**	39767.00	533.25
	MSPSOER	38216.00	39197.12	40460.00	549.64
	BPSO	37638.00	39113.46	40098.00	467.77
Ch150	MSPSO	44637.00	45973.06	46904.00	505.66
	MSPSOER	43973.00	**45928.84**	47349.00	910.80
	BPSO	44732.00	46159.44	47103.00	539.80

Note: Bold signifies the best mean result for each instance

Table 4.12

Performance of the MSPSO, the MSPSOER, and the BPSO on the Big Size of the TSP Benchmark Instance.

TSP instance	Algorithm	Min	Mean	Max	SD
Rd400	MSPSO	188763.00	**192819.56**	195336.00	1860.90
	MSPSOER	190669.00	192952.82	194679.00	1281.22
	BPSO	187569.00	192940.06	195418.00	1496.72
Rat783	MSPSO	72134.00	114856.78	168076.00	42277.63
	MSPSOER	72134.00	**109569.34**	168422.00	40881.85
	BPSO	71639.00	110450.06	127634.00	23239.69
Pr1002	MSPSO	349403.00	2733522.90	6111659.00	2611368.15
	MSPSOER	349403.00	**2031444.24**	6105917.00	2419601.40
	BPSO	349403.00	2687373.84	4222032.00	1838944.82
Rl1304	MSPSO	3231694.00	7216410.76	8950765.00	2386503.85
	MSPSOER	3231694.00	**5167684.32**	8958771.00	1747172.41
	BPSO	3231694.00	5765121.82	5963679.00	377814.96
Rl1323	MSPSO	3088190.00	7361480.44	9353923.00	2422943.71
	MSPSOER	3088190.00	**5555627.94**	9341337.00	2207204.58
	BPSO	3088190.00	5612973.50	6014060.00	652337.70
Rl1889	MSPSO	6601280.00	7344841.26	14207858.00	1710603.51
	MSPSOER	6601280.00	**7070631.30**	10262037.00	802824.10
	BPSO	6601280.00	8877423.56	12073899.00	2608097.78

Note: Bold signifies the best mean result for each instance

Meanwhile, the best, worst, average, and standard deviation of solutions recorded for each algorithm on big size of the TSP benchmark instance is shown in Table 4.12. Wilcoxon's Signed-Rank test is initially performed to the MSPSO and the BPSO that use medium and big size of the TSP benchmark instance in order to investigate either the quality of results between these two algorithms are identical or not; the null hypothesis is that the quality of results of these two algorithms are identical.

Table 4.13
Wilcoxon's Signed-Rank Test Table.

TSP Instance	MSPSO	BPSO	Diff	Rank
Eil76	2030.68	2036.86	-6.18	5
Rd100	45565.84	45849.82	-283.98	10
Eil101	2822.60	2834.80	-12.2	6
Bier127	529237.58	534565.08	-5327.5	11
Ch130	38862.52	39113.46	-250.94	9
Ch150	45973.06	46159.44	-186.38	8
Rd400	192819.56	192940.06	-120.5	7
Rat783	114856.78	110450.06	4406.72	4
Pr1002	2733522.90	2687373.84	46149.06	3
Rl1304	7216410.76	5765121.82	1451288.94	2
Rl1323	7361480.44	5612973.50	1748506.94	1
Rl1889	7344841.26	8877423.56	-1532582.30	12

Table 4.13 provides rank information that is used to execute Wilcoxon's Signed-Rank test. The level of significance is set at 5% ($p < 0.05$) and sample size is 12. The critical value for two-sided test is 14 and the decision rule is as follows: Reject the null hypothesis if the test statistic, $W \leq 14$. With regard to the Wilcoxon's Signed Rank Test, the test statistic is $W = 10$, where $W^+ = [4 + 3 + 2 + 1] = 10$ and $W^- = [5 + 10 + 6 + 11 + 9 + 8 + 7 + 12] = 68$. Therefore, the null hypothesis can be rejected because 10 < 14. The result is statistically significant at $p < 0.05$, where the MSPSO performs better than the BPSO.

To study the performance of the MSPSO, the MSPSOER, and the BPSO, the Friedman test with significance level 0.05 is performed using the experimental results from Table 4.11 and Table 4.12. The test begins with sorting the three algorithms based on their average performance, as presented in Table 4.14. The average rank is then utilized to measure the Friedman statistic value. It seems that the Friedman statistic value $\chi^2{}_F$ is greater than the critical value $\chi^2{}_\alpha$, (7.1664 > 5.9910). Thus, significant differences exist between the algorithms.

Holm procedure is then performed, and the three algorithms are compared to each other. The result of the Holm procedure in Table 4.15 presents that there is significant difference exists between the performance of the MSPSOER and the BPSO. Meanwhile, there are no significant differences exist between the MSPSO and the MSPSOER, and the MSPSO and the BPSO because P < Holm.

Table 4.14

Friedman Test based on Results Computed in Table 4.11 and Table 4.12.

TSP instance	Item	MSPSO	MSPSOER	BPSO
Eil76	Friedman Rank	2	1	3
Rd100	Friedman Rank	1	2	3
Eil101	Friedman Rank	2	1	3
Bier127	Friedman Rank	2	1	3
Ch130	Friedman Rank	1	3	2
Ch150	Friedman Rank	2	1	3
Rd400	Friedman Rank	1	3	2
Rat783	Friedman Rank	3	1	2
Pr1002	Friedman Rank	3	1	2
Rl1304	Friedman Rank	3	1	2
Rl1323	Friedman Rank	3	1	2
Rl1889	Friedman Rank	2	1	3
Average Friedman rank		2.08	1.42	2.50

Table 4.15

Holm Procedure based on Average Friedman Rank in Table 4.14.

Algorithm	*P*	z	Holm
MSPSOER vs. BPSO	0.0080	2.6536	0.0167
MSPSO vs. MSPSOER	0.1025	1.6330	0.0250
MSPSO vs. BPSO	0.3074	1.0206	0.0500

Table 4.16

Performance of the MSPSO, the MSPSOER, and the BPSO Measurement using Friedman Test for Bigger Size of the TSP Benchmark Instances.

Summary	Performance
1	MSPSOER > BPSO
2	MSPSOER = MSPSO
3	MSPSO = BPSO

Note: ">" means better than and "="means similar

To sum up, the performance between each algorithm is offered in Table 4.16.

4.3 The Performance of the MSGSA, the MSGSAER, and the BGSA for solving the TSP

This section offers the comparisons of the performance between the MSGSA, the MSGSAER, and the BGSA problems with small and bigger size of the TSP benchmark instance. The small and bigger size of the TSP benchmark instances are considered with the number of city ranged from 14 to 52 and 76 to 1889, respectively. Due to sensitivity of algorithmic parameters, G_0 is set to 100 and β is set to 20 according to the authors of GSA (Rashedi, Nezamabadi-pour, & Saryazdi, 2009a). The number of iteration T for the small and bigger size of the TSP benchmark instances are set to 10000 and 1000, respectively. T for the bigger size of the TSP benchmark instances is set smaller in order to speed up the computation in obtaining results. The total number of runs is 50 for the both two size categories.

In order to evaluate the performance of the algorithms in consideration, the indicators used are the quality of results and speed of convergence, and the superiority of results on individual runs. With regard to the speed of convergence, each agent of each algorithm records the objective value of the best solution found in every iteration. Convergence patterns for each algorithm are then constructed using the best solution of each algorithm.

Table 4.17
Parameter Settings of the MSGSA, the MSGSAER, and the BGSA on the Small Size of the TSP Benchmark Instances.

Parameter	**Algorithm** **MSGSA**	 **MSGSAER**	 **BGSA**
Number of run	50	50	50
Number of iteration	10000	10000	10000
Number of agent	30	30	30
G_0	100	100	100
β	20	20	20

Table 4.18
Parameter Settings of the MSGSA, the MSGSAER, and the BGSA on the Bigger Size of the TSP Benchmark Instances.

Parameter	**Algorithm** **MSGSA**	 **MSGSAER**	 **BGSA**
Number of run	50	50	50
Number of iteration	10000	10000	10000
Number of agent	30	30	30
G_0	100	100	100
β	20	20	20

In this study, the parameters and their respective value for the small and bigger size of the TSP benchmark instances of the MSGSA, the MSGSAER, and the BGSA are listed in Table 4.17 and Table 4.18, respectively.

4.3.1 The Performance of the MSGSA, the MSGSAER, and the BGSA for Small Size of the TSP Benchmark Instances

A summary of the performance of the MSGSA, the MSGSAER, and the BGSA for small size of the TSP benchmark instances is given in Table 4.19. The values reported are best, worst, average, and standard deviation of solutions over 50 independent runs. The MSGSAER yields smaller values for the minimum for all the six sets of the TSP benchmark instance compared to the MSGSA and the BGSA, thus verifying that the MSGSAER produces higher quality solutions. On the other hand, the best result found from these 50 independent runs of each algorithm for each TSP benchmark instance algorithm are selected and then portrayed in several convergence patterns. Best fitness is recorded in the convergence patterns of each iteration.

Table 4.19
Performance of the MSGSA, the MSGSAER, and the BGSA on the Small Size of the TSP Benchmark Instances.

TSP instance	Algorithm	Min	Mean	Max	SD
Burma14	MSGSA	3567.00	3827.88	4023.00	112.01
	MSGSAER	3323.00	**3368.00**	3411.00	21.60
	BGSA	3893	4437.58	4857.00	192.39
Ulysses16	MSGSA	7278.00	7938.58	8376.00	219.29
	MSGSAER	6802.60	**6905.00**	24.76	7481.00
	BGSA	7481.00	8800.20	9355.00	389.83
Ulysses22	MSGSA	9411.00	10351.62	10859.00	317.05
	MSGSAER	6971.00	**7117.18**	7302.00	87.92
	BGSA	10099.00	11157.12	11835.00	389.31
Bays29	MSGSA	3673.00	3993.46	4168.00	104.58
	MSGSAER	2076.00	**2144.54**	2239.00	52.17
	BGSA	3819.00	4235.85	4483.00	108.70
Eil51	MSGSA	1175.00	1229.50	1278.00	24.23
	MSGSAER	473.00	**483.20**	490.00	4.94
	BGSA	1191.00	1269.16	1326.00	22.86
Berlin52	MSGSA	20859.00	21993.80	22848.00	451.47
	MSGSAER	7751.00	**7914.76**	8300.00	144.09
	BGSA	21533.00	22830.96	24056.00	522.24

Note: Bold signifies the best mean result for each instance

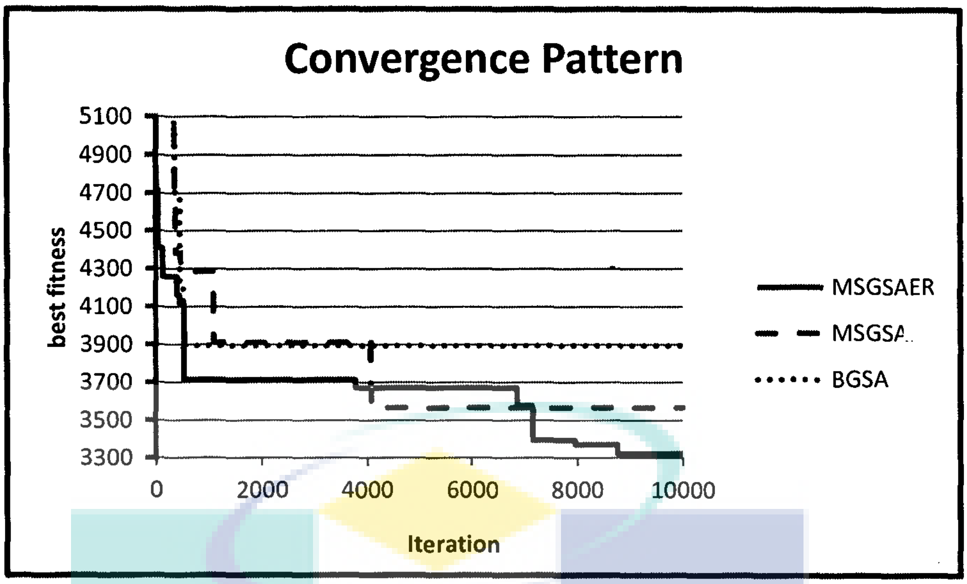


Figure 4.8. Comparison of the convergence pattern of the MSGSA, the MSGSAER, and the BGSA for Burma14 benchmark instance.

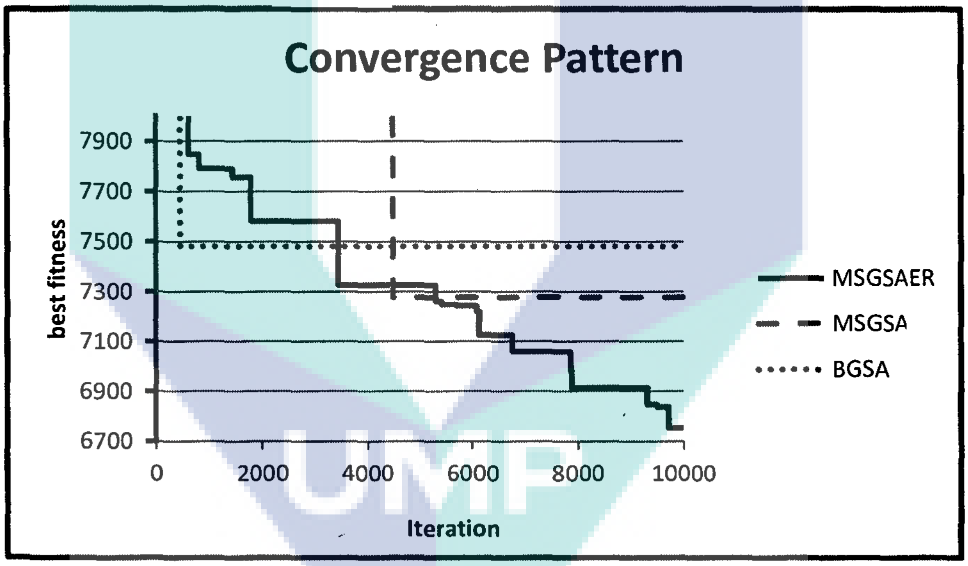


Figure 4.9. Comparison of the convergence pattern of the MSGSA, the MSGSAER, and the BGSA for Ulysses16 benchmark instance.

Figure 4.8 and Figure 4.9 show that the MSGSAER converges slower.

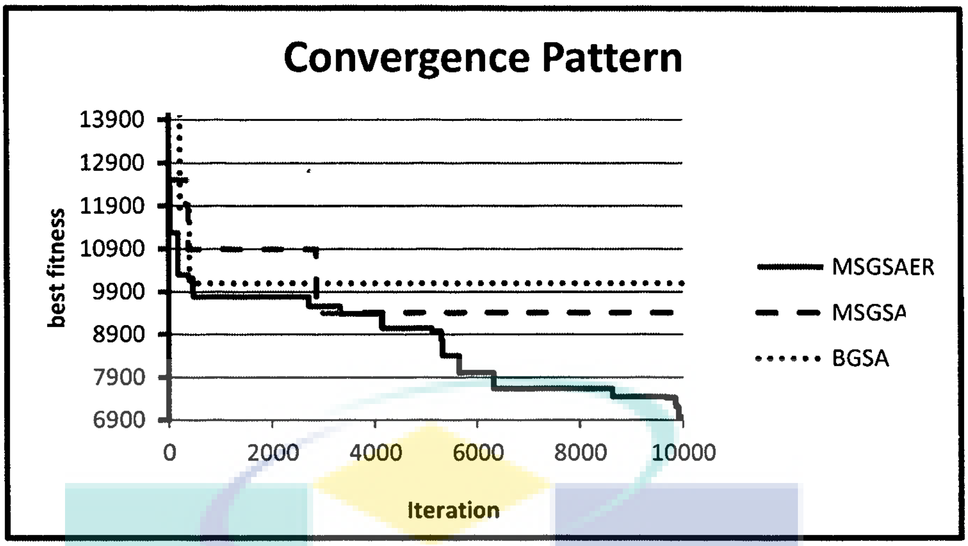


Figure 4.10. Comparison of the convergence pattern of the MSGSA, the MSGSAER, and the BGSA for Ulysses22 benchmark instance.

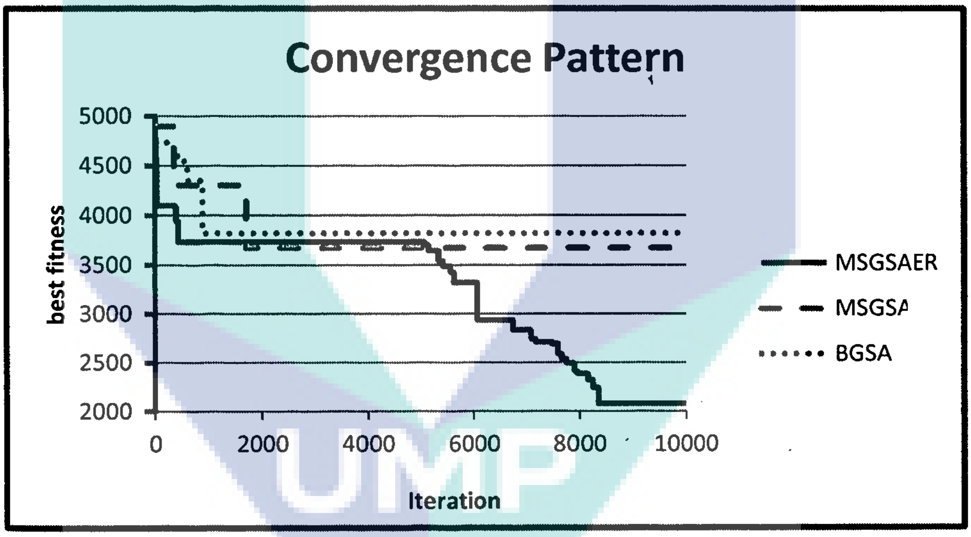


Figure 4.11. Comparison of the convergence pattern of the MSGSA, the MSGSAER, and the BGSA for Bays29 benchmark instance.

Figure 4.11 and Figure 4.12 also show that the MSGSAER converges slower.

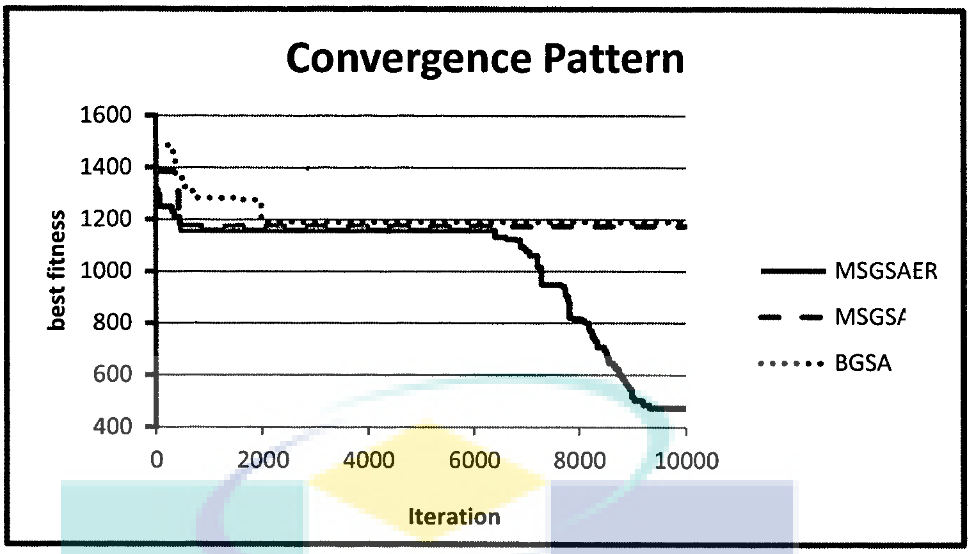


Figure 4.12. Comparison of the convergence pattern of the MSGSA, the MSGSAER, and the BGSA for Eil51 benchmark instance.

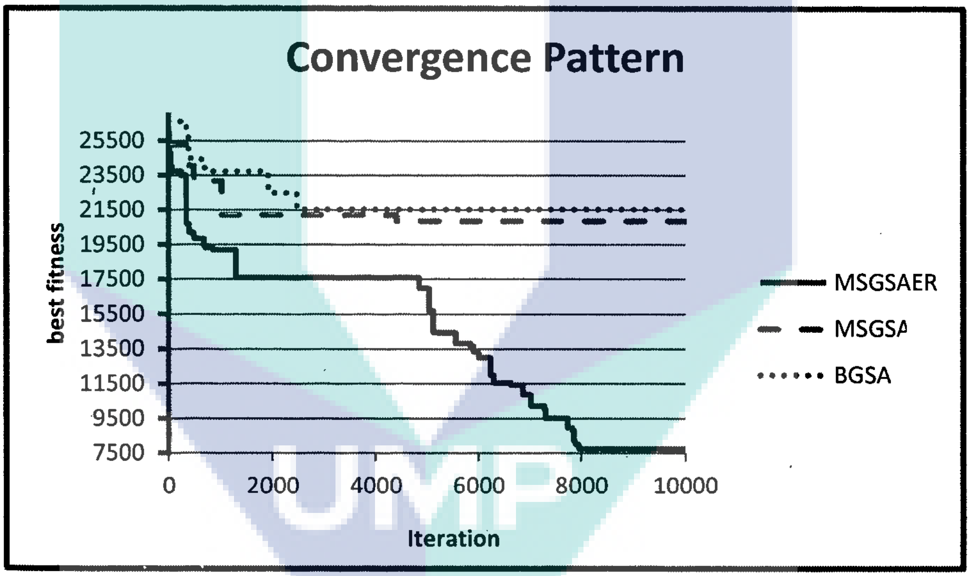


Figure 4.13. Comparison of the convergence pattern of the MSGSA, the MSGSAER, and the BGSA for Berlin52 benchmark instance.

Figure 4.12 and 4.13 also show that the MSGSAER converges slower.

Table 4.20

Number of Times of the MSGSA Performs Better, Similar, or Worse compared to the BGSA.

TSP instance	MSGSA vs. BGSA		
	Better	Similar	Worse
Burma14	49	0	1
Ulysses16	48	0	2
Ulysses22	47	0	3
Bays29	48	0	2
Eil51	44	0	6
Berlin52	47	0	3

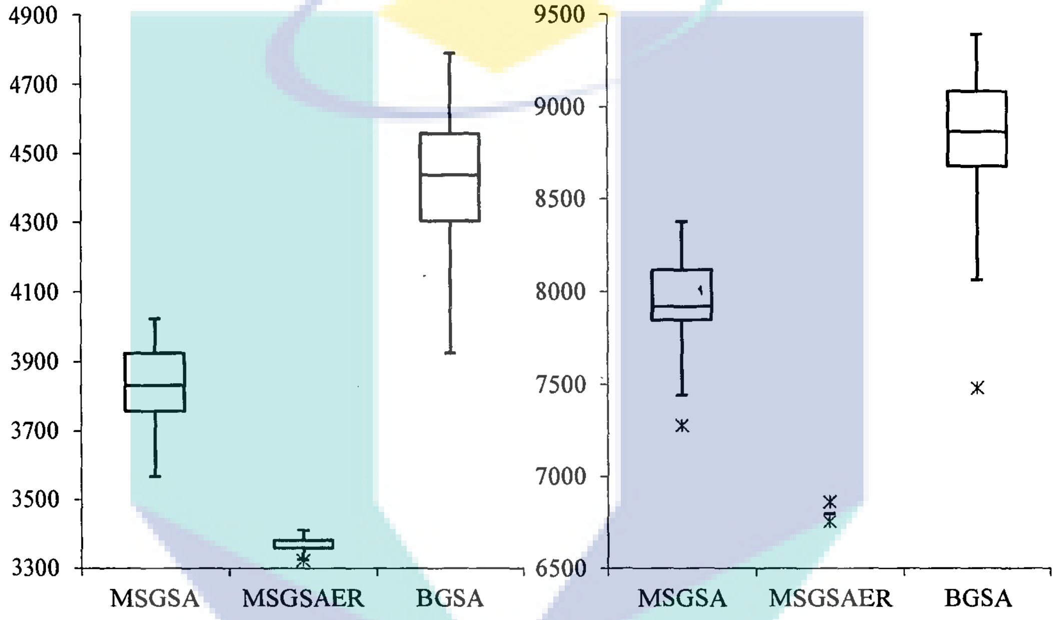
Table 4.21

Number of Times of the MSGSAER Performs Better, Similar, or Worse compared to the BGSA.

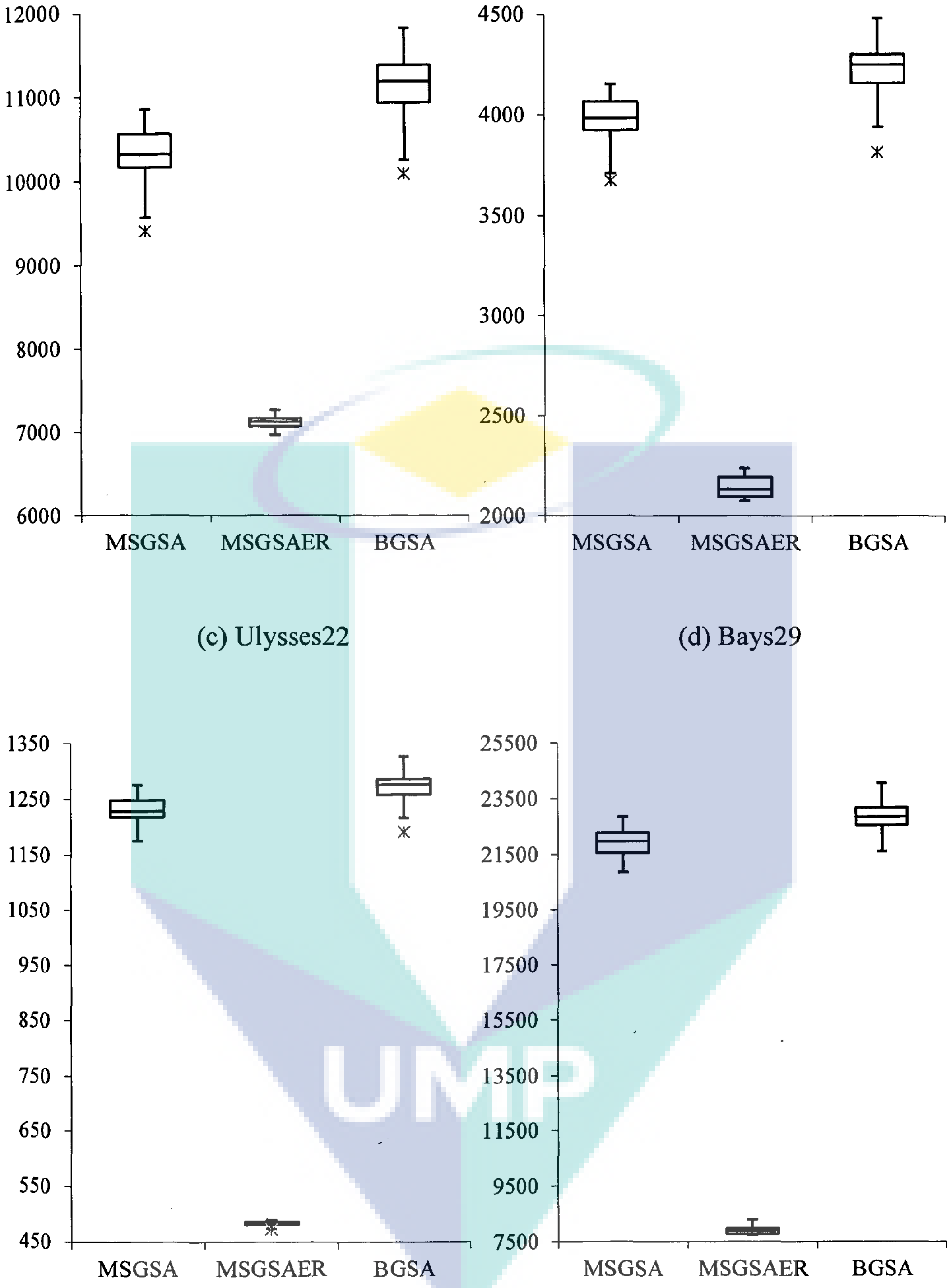
TSP instance	MSGSAER vs. BGSA		
	Better	Similar	Worse
Burma14	50	0	0
Ulysses16	50	0	0
Ulysses22	50	0	0
Bays29	50	0	0
Eil51	50	0	0
Berlin52	50	0	0

Two comparisons between the individual runs for the MSGSA and the BGSA, and the MSGSAER and the BGSA are shown in Table 4.20 and Table 4.21. In these two tables, the number of times of the MSGSA and the MSGSAER perform better, similar, or worse in compare to the BGSA are recorded. With regard to the individual runs conducted, it is observed that the MSGSA and the MSGSAER performs better in compare to the BGSA in obtaining the best solution in each benchmark instance.

The difference between the solutions obtained by these four algorithms is also analysed using boxplots, as shown in Figure 4.14. In this figure, each rectangle represents one of the six benchmark instances (ranging from (a) to (f)). Inside each rectangle, boxplots representing the distribution of the best solution value for the MSGSA, the MSGSAER, and the BGSA are drawn. In each boxplot, the minimum and maximum values are the lowest and highest lines, the upper and lower ends of the box are the upper and lower quartiles, a line within the box shows the median, and the isolated points are the outliers of the distribution.



(a) Burma14 (b) Ulysses16



(c) Ulysses22

(d) Bays29

(e) Eil51

(f) Berlin52

Figure 4.14. Boxplots of the MSGSA, the MSGSAER and the BGSA for (a) Burma14 (b) Ulysses16 (c) Ulysses22 (d) Bays29 (e) Eil51 and (f) Berlin52 benchmark instances.

Each boxplot offers the information about the quality and the performance of the MSGSA, the MSGSAER, and the BGSA. The size of the box represents how varies the results. A smaller box indicates a constant performance of the parameters. In some occasions, the results obtained by the three algorithms contain outliers. The outliers should not be casted from measurement because they are feasible solutions. The pure outliers are actually produced with an actual measurement from the experiments and without clerical errors. The unusual marks are generated by the stochastic behaviour of the algorithms. Because the TSP is a minimization problem, a boxplot that its lower line closes to the minimum value is covetable since the boxplot demonstrates better quality of the solutions. It can be observed from Figure 4.14 that the MSGSAER gives good performance on the six sets of TSP benchmark instances.

It seems that nonparametric test should be used to compare the algorithms because the solutions are not normally distributed. Wilcoxon's Signed-Rank test is performed to the MSGSA and the BGSA in order to investigate either the quality of results between these two algorithms are identical or not; the null hypothesis is that the quality of results of these two algorithms are identical. Table 4.22 provides rank information that is used to execute Wilcoxon's Signed-Rank test.

The level of significance is set at 5% ($p < 0.05$) and sample size is 6. The critical value for two-sided test is 1 and the decision rule is as follows: Reject the null hypothesis if the test statistic, $W \leq 1$. With regard to the Wilcoxon's Signed Rank Test, the test statistic is $W = 0$, where $W^{+} = [0] = 0$ and $W^{-} = [3 + 6 + 5 + 2 + 1 + 4] = 21$. Therefore, the null

Table 4.22
Wilcoxon's Signed-Rank Test Table.

TSP instance	MSGSA	BGSA	Diff	Rank
Burma14	3827.88	4437.58	-609.70	3
Ulysses16	7938.58	8800.20	-861.62	6
Ulysses22	10351.62	11157.12	-805.50	5
Bays29	3993.46	4235.85	-242.39	2
Eil51	1229.50	1269.16	-39.66	1
Berlin52	21993.80	22830.96	-837.16	4

Table 4.23
Friedman Test based on Results Computed in Table 4.19.

TSP Instance	Item	MSGSA	MSGSAER	BGSA
Burma14	Friedman Rank	2	1	3
Ulysses16	Friedman Rank	2	1	3
Ulysses22	Friedman Rank	2	1	3
Bays29	Friedman Rank	2	1	3
Eil51	Friedman Rank	2	1	3
Berlin52	Friedman Rank	2	1	3
Average Friedman rank		2.50	2.00	1.00

hypothesis can be rejected because 0 > 1. The result is statistically significant at $p < 0.05$, where the performance of the MSGSA is better than the BGSA.

To investigate the performance of the MSGSA, the MSGSAER, and the BGSA, other nonparametric test is performed; the Friedman test with significance level 0.05 is performed using the experimental results from Table 4.19. The test begins with sorting the three algorithms based on their average performance, as presented in Table 4.23. The average rank is then utilized to measure the Friedman statistic value. It seems that the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (12.000 > 5.9910). Thus, significant differences exist between the algorithms.

Because significant difference exists, a post hoc procedure called Holm procedure is then performed and the three algorithms are compared to each other. This procedure can identify what are the pairs of the algorithms that have the significant difference. The result of the Holm procedure in Table 4.24 shows that there are significant difference exists between the performance of the MSGSAER and the BGSA because $P <$ Holm.

Table 4.24
Holm Procedure based on Average Friedman Rank in Table 4.23.

Algorithm	*P*	z	Holm
MSGSAER vs. BGSA	0.0005	3.4641	0.0167
MSGSA vs. BGSA	0.0833	1.7321	0.0250
MSGSA vs. MSGSAER	0.0833	1.7321	0.0500

Table 4.25

Performance of the MSGSA, the MSGSAER, and the BGSA Measurement using Friedman Test for Small Size of the TSP Benchmark Instances.

Summary	Performance
1	MSGSAER > BGSA
2	MSGSA = BGSA
3	MSGSA = MSGSAER

Note: ">" means better than and "="means similar

Summary of the Holm procedure on these three algorithms is provided in Table 4.25.

4.3.2 The Performance of the MSGSA, the MSGSAER, and the BGSA for Bigger Size of the TSP Benchmark Instances

The best, worst, average, and standard deviation of solutions recorded for each algorithm on medium size of the TSP benchmark instance is shown in Table 4.26.

Table 4.26

Performance of the MSGSA, the MSGSAER, and the BGSA on the Medium Size of the TSP Benchmark Instance.

TSP instance	Algorithm	Min	Mean	Max	SD
Eil76	MSGSA	1927.00	2062.06	2133.00	56.87
	MSGSAER	1927.00	**2049.12**	2143.00	63.46
	BGSA	1958.00	2059.60	2121.00	40.14
Rd100	MSGSA	44578.00	47007.08	48472.00	756.97
	MSGSAER	43387.00	46597.72	48224.00	1389.86
	BGSA	44677.00	**46345.06**	47410.00	635.08
Eil101	MSGSA	2802.00	2893.08	2967.00	39.59
	MSGSAER	2750.00	2889.14	2953.00	41.46
	BGSA	2519.00	**2694.38**	2906.00	81.69
Bier127	MSGSA	518604.00	559991.00	571026.00	9027.67
	MSGSAER	541522.00	560471.54	572171.00	6427.68
	BGSA	483443.00	**510137.90**	533830.00	10528.60
Ch130	MSGSA	38376.00	40011.24	41032.00	737.36
	MSGSAER	37723.00	**39507.28**	40995.00	944.54
	BGSA	38326.00	39533.30	40457.00	591.38
Ch150	MSGSA	45824	46915.86	48187	652.281
	MSGSAER	45774	46942.14	47980	669.8379
	BGSA	44922	**46660.14**	47570	592.7121

Note: Bold signifies the best mean result for each instance

Table 4.27

Performance of the MSGSA, the MSGSAER, and the BGSA on the Big Size of the TSP Benchmark Instance.

TSP instance	Algorithm	Min	Mean	Max	SD
Rd400	MSGSA	189342.00	**193251.48**	196089.00	2395.96
	MSGSAER	192113.00	194776.78	197788.00	1444.07
	BGSA	190485.00	193274.98	196162.00	1290.79
Rat783	MSGSA	72134.00	150589.8.00	169459.00	37140.86
	MSGSAER	72134.00	152614.60	169634.00	33911.23
	BGSA	71869.00	**86997.04**	102835.00	13437.60
Pr1002	MSGSA	349403.00	4926156.00	6363234.00	2473434.00
	MSGSAER	349403.00	5397462.00	6354110.00	2088434.00
	BGSA	349317.00	**1698504.00**	3767664.00	1599321.00
Rl1304	MSGSA	3231694.00	8879502.00	9071746.00	816257.80
	MSGSAER	3231694.00	8758477.00	9050009.00	1140379.00
	BGSA	3226247.00	**5967482.00**	6371747.00	706717.70
Rl1323	MSGSA	3088190.00	9238533.00	9456165.00	889042.60
	MSGSAER	3088190.00	9026843.00	9443225.00	1367142.00
	BGSA	6004770.00	**6391573.00**	6729587.00	151284.6.00
Rl1889	MSGSA	6601280.00	14061828.00	14302746.00	1078312.00
	MSGSAER	6601280.00	13906016.00	14313826.00	1507092.00
	BGSA	6601280.00	**11189187.00**	11963460.00	1377345.00

Note: Bold signifies the best mean result for each instance

Meanwhile, the best, worst, average, and standard deviation of solutions recorded for each algorithm on big size of the TSP benchmark instance is shown in Table 4.27. Wilcoxon's Signed-Rank test is initially performed to the MSGSA and the BGSA that use medium and big size of the TSP benchmark instance in order to investigate either the quality of results between these two algorithms are identical or not; the null hypothesis is that the quality of results of these two algorithms are identical.

Table 4.28
Wilcoxon's Signed-Rank Test Table.

TSP Instance	MSGSA	BGSA	Diff	Rank
Eil76	2062.06	2059.60	2.46	11
Rd100	47007.08	46345.06	662.02	7
Eil101	2893.08	2694.38	198.70	10
Bier127	559991.00	510137.90	49853.10	6
Ch130	40011.24	39533.30	477.94	8
Ch150	46915.86	46660.14	255.72	9
Rd400	193251.48	193274.98	-23.50	12
Rat783	150589.8	86997.04	63592.76	5
Pr1002	4926156.00	1698504.00	3227652.00	1
Rl1304	8879502	5967482	2912020.00	2
Rl1323	9238533	6391573	2846960.00	4
Rl1889	14061828	11189187	2872641.00	3

Table 4.28 provides rank information that is used to execute Wilcoxon's Signed-Rank test. The level of significance is set at 5% ($p < 0.05$) and sample size is 12. The critical value for two-sided test is 14 and the decision rule is as follows: Reject the null hypothesis if the test statistic, $W \leq 14$. With regard to the‘Wilcoxon's Signed Rank Test, the test statistic is $W = 12$, where $W^+ = [11 + 7 + 10 + 6 + 8 + 9 + 5 + 1 + 2 + 4 + 3] = 66$ and $W^- = [12] = 12$. Therefore, the null hypothesis can be rejected because 12 < 14. The result is statistically significant at $p < 0.05$, where the BGSA performs better than the MSGSA.

To study the performance of the MSGSA, the MSGSAER, and the BGSA, the Friedman test with significance level 0.05 is performed using the experimental results from Table 4.26 and Table 4.27. The test begins with sorting the three algorithms based

Table 4.29

Friedman Test based on Results Computed in Table 4.26 and Table 4.27.

TSP instance	Item	MSGSA	MSGSAER	MSGSA
Eil76	Friedman Rank	3	1	2
Rd100	Friedman Rank	3	2	1
Eil101	Friedman Rank	3	2	1
Bier127	Friedman Rank	2	3	1
Ch130	Friedman Rank	3	1	2
Ch150	Friedman Rank	2	3	1
Rd400	Friedman Rank	1	3	2
Rat783	Friedman Rank	1	3	2
Pr1002	Friedman Rank	3	1	2
Rl1304	Friedman Rank	3	2	1
Rl1323	Friedman Rank	3	2	1
Rl1889	Friedman Rank	3	2	1
Average Friedman rank		2.50	2.08	1.42

Table 4.30

Holm Procedure based on Average Friedman Rank in Table 4.29.

Algorithm	*P*	z	Holm
MSGSA vs. BGSA	0.0022	3.0619	0.0167
MSGSAER vs. BGSA	0.0662	1.8371	0.0250
MSGSA vs. MSGSAER	0.2207	1.2247	0.0500

on their average performance, as presented in Table 4.29. The average rank is then utilized to measure the Friedman statistic value. It seems that the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (29.0418 > 5.9910). Thus, significant differences exist between the algorithms.

Holm procedure is then performed, and the three algorithms are compared to each other. The result of the Holm procedure in Table 4.30 presents that there are significant differences exist between the performance of the MSGSA and the BGSA, and the MSGSA and the MSGSAER because P < Holm.

Table 4.31
Performance of the MSGSA, the MSGSAER, and the BGSA Measurement using Friedman Test for Bigger Size of the TSP Benchmark Instances.

Summary	Performance
1	BGSA > MSGSA
2	MSGSAER = BGSA
3	MSGSA = MSGSAER

Note: “>” means better than and “=”means similar

To sum up, the performance between each algorithm is offered in Table 4.31.

4.4 The Performance Comparison between the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER for solving the TSP

This section presents the performance comparison between the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER for the TSP using the Friedman test. The null hypotheses is each ranking of the four algorithms within each benchmark instance is equally likely; there are no significant differences between the algorithms.

To study the performance of the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER, the Friedman test with significance level 0.05 is performed.

The test begins with sorting the three algorithms based on their average performance, as presented in Table 4.32. The average rank is then utilized to measure the Friedman statistic value. It seems that the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (127.0670 > 7.8150). Thus, significant differences exist between the algorithms.

Table 4.32
Friedman Test based on the Experimental Results of the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER.

TSP instance	Item	MSPSO	MSPSOER	MSGSA	MSGSAER
Burma14	Friedman Rank	3	2	4	1
Ulysses16	Friedman Rank	3	2	4	1
Ulysses22	Friedman Rank	3	2	4	1
Bays29	Friedman Rank	3	2	4	1
Eil51	Friedman Rank	3	2	4	1
Berlin52	Friedman Rank	4	2	3	1
Eil76	Friedman Rank	2	1	4	3
Rd100	Friedman Rank	1	2	4	3
Eil101	Friedman Rank	2	1	4	3
Bier127	Friedman Rank	2	1	3	4
Ch130	Friedman Rank	1	2	4	3
Ch150	Friedman Rank	1	2	3	4
Rd400	Friedman Rank	2	1	3	4
Rat783	Friedman Rank	2	1	3	4
Pr1002	Friedman Rank	2	1	4	3
Rl1304	Friedman Rank	2	1	4	3
Rl1323	Friedman Rank	2	1	4	3
Rl1889	Friedman Rank	2	1	4	3
Average Friedman rank		2.22	1.50	3.72	2.56

Table 4.33
Holm Procedure based on Average Friedman Rank in Table 4.32.

Algorithm	***P***	**z**	**Holm**
MSGSA vs. MSPSOER	0.0000	5.1588	0.0083
MSPSO vs. MSGSA	0.0005	3.4857	0.0100
MSGSA vs. MSGSAER	0.0070	2.6956	0.0125
MSPSOER vs. MSGSAER	0.0138	2.4632	0.0167
MSPSO vs. MSPSOER	0.0944	1.6731	0.0250
MSPSO vs. MSGSAER	0.4295	0.7901	0.0500

Table 4.34
Performance of the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER Measurement using Friedman Test for Bigger Size of the TSP Benchmark Instances.

Summary	**Performance**
1	MSPSOER > MSGSA
2	MSPSO > MSGSA
3	MSGSAER > MSGSA
4	MSPSOER > MSGSAER
5	MSPSO = MSPSOER
6	MSPSO = MSGSAER

Note: “>” means better than and “=”means similar

Holm procedure is then conducted to know which pairs of algorithms are significantly difference. The result of the Holm procedure in Table 4.33 shows that there are significant differences exist between the performance of the MSGSA and the MSPSOER, the MSPSO and the MSGSA, the MSGSA and the MSGSAER, and the MSPSOER and the MSGSAER because *P* < Holm.

Meanwhile, there are no significant differences exist between the MSPSO and the MSPSOER, and the MSPSO and the MSGSAER because *P* > Holm. To sum up, the performance between each algorithm is offered in Table 4.34.

4.5 The Performance of the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER for solving the ASP

This section provides'the relative efficiency of the performance of the proposed approach based on the MSPSO, the proposed approach based on the MSPSOER, the proposed approach based on the MSGSA, and the proposed approach based on the MSGSAER. Each proposed algorithm is then compared to the approach based on the SA (Motavalli & Islam, 1997), the GA (Choi, Lee, & Cho, 2008), and the BPSO (Mukred et al., 2012). In all comparisons, the quality of the results of each proposed approach is measured based on the fitness values of the best solutions in minimizing the total assembly time.

Subsequently, this section offers results of a series of experiments to tune the best parameters for the four proposed algorithms for the assembly sequence planning problem due to the success of these algorithms are heavily depend on setting of control parameters. For the proposed approach based on the MSPSO and the proposed approach based on the MSPSOER, the control parameters namely; inertia weight ω, coefficient factors c_1 and c_2, number of particles *NOP*, and number of iteration *T*. Meanwhile, for the proposed approach based on the MSGSA and the proposed approach based on the MSGSAER, the control parameters namely; constant β, initial gravitational constant G_0, number of agents *NOA* and number of iteration *T*. These control parameters should be carefully selected when using the algorithms in order to know the best parameters, so a successful implementation of these algorithms can be achieved.

The solutions for each variation of the parameters is presented using boxplot. Each boxplot offers the information about the quality and the performance of the particular parameters. The size of the box represents how varies the results. A smaller box indicates a constant performance of the parameters. In some occasions, the results obtained by the three algorithms contain outliers. The outliers should not be casted from measurement because they are feasible solutions. The pure outliers are actually produced with an actual measurement from the experiments and without clerical errors. The unusual marks are generated by the stochastic behaviour of the algorithms. Because the TSP is a minimization problem, a boxplot that its lower line closes to the minimum value is covetable since the boxplot demonstrates better quality of the solutions.

A nonparametric test is used to compare the variation of the parameters because the solutions are not normally distributed. In this study, the Friedman test with significance level 0.05 is used. This test is advisable for comparison that is more than two sets of parameters (Dieterich, 2012). The test begins with sorting the variation of the parameters based on their average performance. The average rank is then utilized to measure the Friedman statistic value. If the Friedman statistic value χ^2_F is smaller than the critical value χ^2_α, the performance of the proposed approach based on the four algorithms that use the parameters is on par with others; otherwise, the differences are significantly exist. If significant differences exist, the various sets of parameters are then compared each other using the Holm procedure. This procedure can identify what are the pairs of the parameters that have the significant differences.

4.5.1 Results of the Proposed Approach based on the MSPSO compared to the SA

In the experiments, the parameters used for the proposed approach based on the MSPSO and the approach based on the SA are presented in Table 4.35. Based on these parameters, the quality of the results of the proposed approach based on the MSPSO for the 50 runs is presented in Table 4.36.

Table 4.35
Experimental Parameters for the Proposed Approach based on the MSPSO and the Approach based on the SA.

Parameters	MSPSO	SA
Iteration	500	500
Number of particle	30	NAP
ωInitial	0.9	NAP
ωFinal	0.4	NAP
Coefficient factor, c_1 and c_2	2	NAP
Initial temperature (°C)	NAP	100
Cooling rate	NAP	0.95
Number of run	50	NA

NA = not available from the source, NAP = not applicable.

Table 4.36
Quality of the Results for the Proposed Approach based on the MSPSO.

Experiment	Min	Mean	Max	SD
1	514.0	530.0	538.5	4.5

Table 4.37
Best Results and Associated Assembly Sequences of the Proposed Approach based on the MSPSO and the Approach based on the SA.

Approach based	**Total assembly time**	**Assembly sequence**
MSPSO	514.0	2-4-3-1-9-12-5-13-15-18-16-11-6-7-8-10-14-17-19
SA	528.7	2-1-4-9-3-12-13-16-5-15-18-6-11-7-8-10-14-17-19

Table 4.37 shows a comparison of the best results of the proposed approach based on the MSPSO and the approach based on the SA with their assembly sequences. The best values which are the minimum of total assembly time obtained by the proposed approach based on the MSPSO and the approach based on the SA are given in Table 4.37. It shows that the proposed approach based on the MSPSO outperformed the approach based on the SA.

4.5.2 Results of the Proposed Approach based on the MSPSO compared to the GA

Table 4.38 shows the parameters used for the proposed approach based on the MSPSO and the approach based on the GA in three different experiments. The quality of the results of the proposed approach based on the MSPSO for 50 runs is presented in Table 4.39.

Table 4.38
Experimental Parameters for the Proposed Approach based on the MSPSO and the Approach based on the GA.

Parameters	**Experiment 1**		**Experiment 2**		**Experiment 3**	
	MSPSO	GA	MSPSO	GA	MSPSO	GA
Iteration	100	100	100	100	100	100
Number of particle	20	20	40	40	100	100
ωInitial	0.9	NAP	0.9	NAP	0.9	NAP
ωFinal	0.4	NAP	0.4	NAP	0.4	NAP
Coefficient factor, c_1 and c_2	2	NAP	2	NAP	2	NAP
Mutation rate	NAP	0.05	NAP	0.05	NAP	0.05
Crossover	NAP	0.5	NAP	0.5	NAP	0.5
Number of run	50	NA	50	NA	50	NA

NA = not available from the source, NAP = not applicable.

Table 4.39

Quality of the Results for the Proposed Approach based on the MSPSO.

Experiment	Min	Mean	Max	SD
1	517.6	536.1	548.7	6.4
2	524.6	536.6	548.6	5.9
3	522.0	537.1	548.3	6.2

Table 4.40

Best Results of the Proposed Approach based on the MSPSO and the Approach based on the GA for Each Experiment.

Experiment	Total assembly time (MSPSO)	Total assembly time (GA)
1	517.6	535.1
2	524.6	527.9
3	522.0	524.1

Table 4.41

Best results and the Associated Assembly Sequences of the Proposed Approach based on the MSPSO and the Approach based on the GA.

Approach based	Total assembly time	Assembly sequence
MSPSO	517.6	2-1-4-9-15-3-5-6-18-12-7-13-16-8-11-14-10-17-19
GA	524.1	2-18-3-12-1-13-16-5-11-15-4-6-9-7-8-10-14-17-19

Table 4.40 shows the results of the approach based on the GA for the three different experiments. In all three experiments, the proposed approach based on the MSPSO surpassed the approach based on the GA in minimizing the total assembly time for the respective parameters. The best value which is the minimum of total assembly time obtained by the proposed approach based on the MSPSO in the three different experiments is then selected to be compared against the best value produced by the approach based on the GA, as presented in Table 4.40. Referring to the results given in Table 4.41, the proposed approach based on the MSPSO also outperformed the approach based on the GA in obtaining the minimum total assembly time of the ASP problem.

4.5.3 Results of the Proposed Approach based on the MSPSO compared to the BPSO

Parameters used for the proposed approach based on the MSPSO and the approach based on the BPSO are listed in Table 4.42 for three different experiments. The quality of the results of the proposed approach based on the MSPSO and the approach based on the BPSO for 10 runs is presented in Table 4.43. As shown in Table 4.43, the proposed approach based on the MSPSO yields bigger values for the minimum and mean for each experiment, thus verifying that the proposed approach based on the MSPSO produces lower quality solutions compared to the approach based on the BPSO.

Table 4.42
Experimental Parameters for the Proposed Approach based on the MSPSO and the Approach based on the BPSO.

Parameters	**Experiment 1**		**Experiment 2**		**Experiment 3**	
	MSPSO	BPSO	MSPSO	BPSO	MSPSO	BPSO
Iteration	500	500	500	500	500	500
Number of particle	40	40	50	50	60	60
ωInitial	0.9	0.9	0.9	0.9	0.9	0.9
ωFinal	0.4	0.4	0.4	0.4	0.4	0.4
Coefficient factor, c_1 and c_2	2	2	2	2	2	2
Number of run	50	50	50	50	50	50

Table 4.43
Quality of the Results for the proposed approach based on the MSPSO and the Approach based on the BPSO.

Ex	**Min**		**Mean**		**Max**		**SD**	
	MSPSO	BPSO	MSPSO	BPSO	MSPSO	BPSO	MSPSO	BPSO
1	518.5	514.4	528.8	520.3	522.6	526.2	5.2	3.8
2	519.5	515.8	529.0	520.8	522.2	523.4	5.5	2.3
3	514.7	516.9	529.5	521.1	522.4	527.9	6.2	3.2

Ex = experiment.

Table 4.44

Best Results and their Associated Assembly Sequences of the Proposed Approach based on the MSPSO and the Approach based on the BPSO.

Approach based	Total assembly time	Assembly sequence
MSPSO	514.7	3-1-2-4-12-9-16-13-5-15-18-11-6-7-8-14-10-17-19
BPSO	514.4	16-2-13-4-1-15-11-9-6-5-18-7-8-14-12-10-3-17-19

The best values which are the minimum of total assembly time obtained by the proposed approach based on the MSPSO and the approach based on the BPSO are given in Table 4.44. It shows that the proposed approach based on the BPSO outperformed the approach based on the MSPSO.

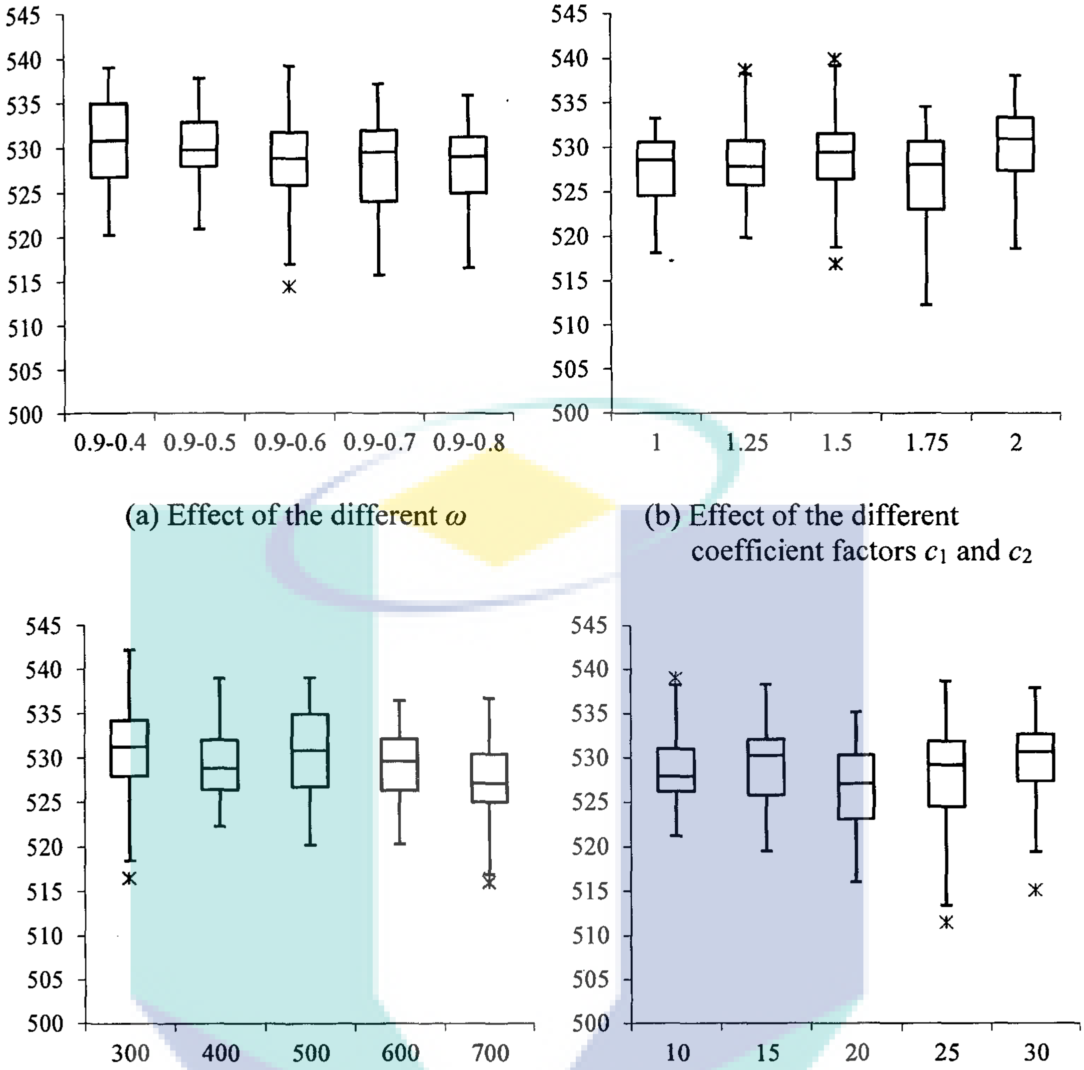
4.5.4 Effect of the MSPSO Parameters

To quantify the results, the 50 runs are performed for each parameter variation, so that the combination of the best parameter settings can be found. Table 4.45 offers the MSPSO outcomes as a result of varying its parameters. Figure 4.15 illustrates the results in box plots. According to Table 4.45, the best parameters for inertia weight ω, coefficient factors c_1 and c_2, number of particles, and number of iteration T are ω = 0.9 to 0.4, c_1 and c_2 = 2, number of particles NOP = 25, T = 500 respectively. The best objective value obtained for these parameters is 511.5. The assembly sequence

Table 4.45

Study of Tuning the MSPSO Parameters.

Parameter	Min	Mean	Max	SD	Other parameters
ω= 0.9 to 0.4	520.2	530.3	539.0	4.9	T = 500, c_1 and c_2 = 2.00, NOP = 30
ω= 0.9 to 0.5	520.9	529.8	537.8	3.7	T = 500, c_1 and c_2 = 2.00, NOP = 30
ω = 0.9 to 0.6	514.4	528.4	539.2	5.7	T = 500, c_1 and c_2 = 2.00, NOP = 30
ω = 0.9 to 0.7	520.9	529.8	537.8	3.7	T = 500, c_1 and c_2 = 2.00, NOP = 30
ω = 0.9 to 0.8	520.2	530.3	539.0	4.9	T = 500, c_1 and c_2 = 2.00, NOP = 30
c_1 and c_2 = 1.00	518.7	530.1	539.3	5.1	T = 500, ω= 0.9 to 0.4, NOP = 30
c_1 and c_2 = 1.25	512.3	526.9	534.7	5.0	T = 500, ω= 0.9 to 0.4, NOP = 30
c_1 and c_2 = 1.50	517.0	529.4	540.0	4.1	T = 500, ω= 0.9 to 0.4, NOP = 30
c_1 and c_2 = 1.75	519.9	528.1	538.7	4.4	T = 500, ω= 0.9 to 0.4, NOP = 30
c_1 and c_2 = 2.00	518.2	527.5	537.3	4.3	T = 500, ω= 0.9 to 0.4, NOP = 30
NOP = 10	521.2	529.0	539.0	3.4	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, T = 500
NOP = 15	519.5	529.9	538.3	4.3	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, T = 500
NOP = 20	516.0	527.2	535.2	4.5	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, T = 500
NOP = 25	**511.5**	**528.1**	**538.7**	**5.4**	**c_1 and c_2 = 2.00, ω= 0.9 to 0.4, T = 500**
NOP = 30	515.1	529.1	537.9	5.2	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, T = 500
T = 300	516.5	530.7	542.2	5.4	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, NOP = 30
T = 400	522.3	529.3	538.9	4.0	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, NOP = 30
T = 500	520.2	530.3	539.0	4.9	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, NOP = 30
T = 600	520.3	528.7	536.4	4.0	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, NOP = 30
T = 700	516.0	527.4	536.7	4.7	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, NOP = 30



(a) Effect of the different ω

(b) Effect of the different coefficient factors c_1 and c_2

(c) Effect of the different number of iteration

(d) Effect of the different number of particle

Figure 4.15. Effect of the usage of the different parameters (MSPSO).

generated for the best objective value using these parameters is 15-1-2-4-12-9-3-13-5-18-6-16-11-7-8-14-10-17-19. The result clearly shows that the MSPSO is an efficient approach compared to the SA, the GA, and the BPSO.

Table 4.46
Friedman Test on the Effect of the Inertia Weight ω, the Coefficient Factors c_1 and c_2, Number of Particle NOP, and Number of Iteration T.

Items	**Parameter settings**				
Inertia weight ω	0.9.to 0.4	0.9 to 0.5	0.9 to 0.6	0.9 to 0.7	0.9 to 0.8
Average Friedman rank	4.50	2.50	1.00	2.50	4.50
Coefficient factors c_1 and c_2	1.00	1.25	1.50	1.75	2.00
Average Friedman rank	5.00	1.00	4.00	3.00	2.00
Number of iteration *NOP*	10	15	20	25	30
Average Friedman rank	3.00	5.00	1.00	2.00	4.00
Number of iteration *T*	300	400	500	600	700
Average Friedman rank	5.00	3.00	4.00	2.00	1.00

Based on Table 4.46, the preferences of inertia weight ω, coefficient factors c_1 and c_2, the number of iteration *NOP*, and the number of iteration *T* on the performance of the MSPSO are investigated. The Friedman statistic indicates that using different ω values cause significant difference to the MSPSO because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.0000 > 9.4880), hence demonstrating that the performance of the MSPSO is extensively affected by the preference of ω.

Table 4.47

Holm Procedure on the Initial Weight ω.

Dataset	*P*	z	Holm
$\omega = 0.9$ to 0.4 vs. $\omega = 0.9$ to 0.6	0.0000	4.9497	0.0053
$\omega = 0.9$ to 0.6 vs. $\omega = 0.9$ to 0.8	0.0000	4.9497	0.0053
$\omega = 0.9$ to 0.4 vs. $\omega = 0.9$ to 0.5	0.0047	2.8284	0.0077
$\omega = 0.9$ to 0.4 vs. $\omega = 0.9$ to 0.7	0.0047	2.8284	0.0077
$\omega = 0.9$ to 0.5 vs. $\omega = 0.9$ to 0.8	0.0047	2.8284	0.0077
$\omega = 0.9$ to 0.7 vs. $\omega = 0.9$ to 0.8	0.0047	2.8284	0.0077
$\omega = 0.9$ to 0.5 vs. $\omega = 0.9$ to 0.6	0.0339	2.1213	0.0143
$\omega = 0.9$ to 0.6 vs. $\omega = 0.9$ to 0.7	0.0339	2.1213	0.0143
$\omega = 0.9$ to 0.4 vs. $\omega = 0.9$ to 0.8	1.0000	4.9497	0.0333
$\omega = 0.9$ to 0.5 vs. $\omega = 0.9$ to 0.7	1.0000	4.9497	0.0333

The Holm procedure is then performed and its statistical values are portrayed in Table 4.47. The result of the Holm procedure shows that significant differences exist between the performances of the MSPSO if comparing the results between these two ω values namely, $\omega = 0.9$ to 0.4 and $\omega = 0.9$ to 0.6, $\omega = 0.9$ to 0.6 and $\omega = 0.9$ to 0.8, $\omega = 0.9$ to 0.4 and $\omega = 0.9$ to 0.5, $\omega = 0.9$ to 0.4 and $\omega = 0.9$ to 0.7, $\omega = 0.9$ to 0.5 and $\omega = 0.9$ to 0.8, and $\omega = 0.9$ to 0.7 and $\omega = 0.9$ to 0.8.

Meanwhile, the Friedman test is also performed on the effect of the different coefficient factors c_1 and c_2. The Friedman statistic indicates that different coefficient factors c_1 and c_2 cause significant difference to the MSPSO because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6667 > 9.4880), hence indicating that the performance of the MSPSO is extensively affected by the preference of coefficient factors c_1 and c_2.

The Holm procedure is then performed and the results are shown in Table 4.48. The results demonstrate that significant differences exist between the performances of the MSPSO if comparing the results between these two coefficient factors c_1 and c_2 namely, c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 1.25$, c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 2.00$, c_1 and $c_2 = 1.25$ vs. c_1 and $c_2 = 1.50$, c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 1.75$, c_1 and $c_2 = 1.50$ vs. c_1 and $c_2 = 2.00$, and c_1 and $c_2 = 1.25$ vs. c_1 and $c_2 = 1.75$.

The Friedman statistic also provides a finding on the performance that uses different number of particle values; there are significant differences to the MSPSO, because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6667 > 9.4880), thus presenting that the performance of the MSPSO is influenced by the preference of number of particle.

Table 4.48
Holm Procedure on the Coefficient Factors c_1 and c_2.

Dataset	*P*	z	Holm
c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 1.25$	0.0000	5.6569	0.0050
c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 2.00$	0.0000	4.2426	0.0059
c_1 and $c_2 = 1.25$ vs. c_1 and $c_2 = 1.50$	0.0000	4.2426	0.0059
c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 1.75$	0.0047	2.8284	0.0083
c_1 and $c_2 = 1.50$ vs. c_1 and $c_2 = 2.00$	0.0047	2.8284	0.0083
c_1 and $c_2 = 1.25$ vs. c_1 and $c_2 = 1.75$	0.0047	2.8284	0.0083
c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 1.50$	0.1573	1.4142	0.0200
c_1 and $c_2 = 1.50$ vs. c_1 and $c_2 = 1.75$	0.1573	1.4142	0.0200
c_1 and $c_2 = 1.25$ vs. c_1 and $c_2 = 2.00$	0.1573	1.4142	0.0200
c_1 and $c_2 = 1.75$ vs. c_1 and $c_2 = 2.00$	0.1573	1.4142	0.0200

The Holm procedure is then performed and its statistical values are displayed in Table 4.49. The results of Holm procedure demonstrate that significant differences exist between the performances of the MSPSO if comparing the results between these two number of particle namely, *NOP* = 15 vs. *NOP* = 20, *NOP* = 15 vs. *NOP* = 25, *NOP* = 20 vs. *NOP* = 30, *NOP* = 10 vs. *NOP* = 15, *NOP* = 25 vs. *NOP* = 30, and *NOP* = 10 vs. *NOP* = 200.

Meanwhile, the Friedman statistic indicates that using different number of iteration values cause significant differences to the MSPSO because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6667 > 9.4880), hence presenting that the performance of the MSPSO is influenced by the preference of number of iteration.

Table 4.49
Holm Procedure on the Number of Particle NOP.

Dataset	***P***	**z**	**Holm**
NOP = 15 vs. *NOP* = 20	0.0000	5.6569	0.0050
NOP = 15 vs. *NOP* = 25	0.0000	4.2426	0.0059
NOP = 20 vs. *NOP* = 30	0.0000	4.2426	0.0059
NOP = 10 vs. *NOP* = 15	0.0047	2.8284	0.0083
NOP = 25 vs. *NOP* = 30	0.0047	2.8284	0.0083
NOP = 10 vs. *NOP* = 20	0.0047	2.8284	0.0083
NOP = 15 vs. *NOP* = 30	0.1573	1.4142	0.0200
NOP = 10 vs. *NOP* = 30	0.1573	1.4142	0.0200
NOP = 20 vs. *NOP* = 25	0.1573	1.4142	0.0200
NOP = 10 vs. *NOP* = 25	0.1573	1.4142	0.0200

Table 4.50
Holm Procedure on the Number of Iteration T.

Dataset	*P*	z	Holm
$T = 300$ vs. $T = 700$	0.0000	5.6569	0.0050
$T = 300$ vs. $T = 600$	0.0000	4.2426	0.0059
$T = 500$ vs. $T = 700$	0.0000	4.2426	0.0059
$T = 300$ vs. $T = 400$	0.0047	2.8284	0.0083
$T = 500$ vs. $T = 600$	0.0047	2.8284	0.0083
$T = 400$ vs. $T = 700$	0.0047	2.8284	0.0083
$T = 300$ vs. $T = 500$	0.1573	1.4142	0.0200
$T = 400$ vs. $T = 500$	0.1573	1.4142	0.0200
$T = 600$ vs. $T = 700$	0.1573	1.4142	0.0200
$T = 400$ vs. $T = 600$	0.1573	1.4142	0.0200

The Holm procedure is then performed and its statistical values are portrayed in Table 4.50. The results of Holm procedure demonstrate that significant differences exist between the performances of the MSPSO if comparing the results between these two number of iteration namely, $T = 300$ and $T = 700$, $T = 300$ and $T = 600$, $T = 500$ and $T = 700$, $T = 300$ and $T = 400$, $T = 500$ and $T = 600$, and $T = 400$ and $T = 700$.

4.5.5 Results of the Proposed Approach based on the MSPSOER compared to the SA

In the experiments, the parameters used for the proposed approach based on the MSPSOER and the approach based on the SA are presented in Table 4.51. Based on these

Table 4.51
Experimental Parameters for the Proposed Approach based on the MSPSOER and the Approach based on the SA.

Parameters	MSPSOER	SA
Iteration	500	500
Number of particle	30	NAP
ωInitial	0.9	NAP
ωFinal	0.4	NAP
Coefficient factor, c_1 and c_2	2	NAP
Initial temperature (°C)	NA	100
Cooling rate	NA	0.95
Number of run	50	NA

NA = not available from the source, NAP = not applicable.

Table 4.52
Quality of the Results for the Proposed Approach based on the MSPSOER.

Experiment	Min	Mean	Max	SD
1	517.9	528.6	538.4	4.3

Table 4.53
Best Results and Associated Assembly Sequences of the Proposed Approach based on the MSPSOER and the Approach based on the SA.

Approach based	Total assembly time	Assembly sequence
MSPSOER	517.9	2-1-3-15-4-12-5-13-18-9-6-16-11-7-8-14-10-17-19
SA	528.7	2-1-4-9-3-12-13-16-5-15-18-6-11-7-8-10-14-17-19

parameters, the quality of the results of the proposed approach based on the MSPSOER for the 50 runs is presented in Table 4.52. Table 4.53 shows a comparison of the best results of the proposed approach based on the MSPSOER and the approach based on the SA with their assembly sequences. Using the minimum values of the proposed approach based on the MSPSOER and the approach based on the SA given in Table 4.52, it seems that the proposed approach based on the MSPSOER outperformed the approach based on the SA in obtaining the minimum total assembly time of the ASP problem. The mean or average of the total assembly time yielded by the proposed approach based on the MSPSOER to solve the ASP problem is also better than the minimum of the assembly time produced by the approach based on SA.

4.5.6 Results of the Proposed Approach based on the MSPSOER compared to the GA

Table 4.54 shows the parameters used for the proposed approach based on the MSPSOER and the approach based on the GA in three different experiments. The quality of the results of the proposed approach based on the MSPSOER for 50 runs is presented in Table 4.55.

Table 4.54
Experimental Parameters for the Proposed Approach based on the MSPSOER and the Approach based on the GA.

Parameters	Experiment 1		Experiment 2		Experiment 3	
	MSPSOER	GA	MSPSOER	GA	MSPSOER	GA
Iteration	100	100	100	100	100	100
Number of particle	20	20	40	40	100	100
ωInitial	0.9	NAP	0.9	NAP	0.9	NAP
ωFinal	0.4	NAP	0.4	NAP	0.4	NAP
Coefficient factor, c_1 and c_2	2	NAP	2	NAP	2	NAP
Mutation rate	NAP	0.05	NAP	0.05	NAP	0.05
Crossover	NAP	0.5	NAP	0.5	NAP	0.5
Number of run	50	NA	50	NA	50	NA

NA = not available from the source, NAP = not applicable.

Table 4.55
Quality of the Results for the Proposed Approach based on the MSPSOER.

Experiment	Min	Mean	Max	SD
1	519.5	534.4	548.7	6.0
2	512.0	533.2	547.3	7.5
3	523.4	534.8	543.1	5.3

Table 4.56 shows the results of the approach based on the GA for the three different experiments. In all three experiments, the proposed approach based on the MSPSOER surpassed the approach based on the GA in minimizing the total assembly time for the respective parameters. The best value obtained by the proposed approach based on the MSPSOER in the three different experiments is then selected to be compared against the best value produced by the approach based on the GA, as presented in Table 4.56. Referring to the results given in Table 4.57, the proposed approach based on the MSPSOER also outperformed the approach based on the GA in obtaining the minimum total assembly time of the ASP problem.

Table 4.56
Best Results of the Proposed Approach based on the MSPSOER and the Approach based on the GA for Each Experiment.

Experiment	Total assembly time (MSPSOER)	Total assembly time (GA)
1	519.5	535.1
2	512.0	527.9
3	523.4	524.1

Table 4.57
Best results and the Associated Assembly Sequences of the Proposed Approach based on the MSPSOER and the Approach based on the GA.

Approach based	Total assembly time	Assembly sequence
MSPSOER	512.0	1-2-4-9-12-5-15-3-13-16-18-11-6-7-8-14-10-17-19
GA	524.1	2-18-3-12-1-13-16-5-11-15-4-6-9-7-8-10-14-17-19

4.5.7 Results of the Proposed Approach based on the MSPSOER compared to the BPSO

Parameters used for the proposed approach based on the MSPSOER and the approach based on the BPSO are listed in Table 4.58 for three different experiments. The quality of the results of the proposed approach based on the MSPSOER and the approach based on the BPSO for 10 runs is presented in Table 4.59. As shown in Table 4.59, the proposed approach based on the MSPSOER yields bigger values for the minimum and mean for each experiment, thus verifying that the proposed approach based on the MSPSOER produces lower quality solutions compared to the approach based on the BPSO.

Table 4.58
Experimental Parameters for the Proposed Approach based on the MSPSOER and the Approach based on the BPSO.

	Experiment 1		**Experiment 2**		**Experiment 3**	
Parameters	MSPSOER	BPSO	MSPSOER	BPSO	MSPSOER	BPSO
Iteration	500	500	500	500	500	500
Number of particle	40	40	50	50	60	60
ωInitial	0.9	0.9	0.9	0.9	0.9	0.9
ωFinal	0.4	0.4	0.4	0.4	0.4	0.4
Coefficient factor, c_1 and c_2	2	2	2	2	2	2
Number of run	50	50	50	50	50	50

Table 4.59
Quality of the Results for the proposed approach based on the MSPSOER and the Approach based on the BPSO.

	Min		**Mean**		**Max**		**SD**	
Ex	MSPSOER	BPSO	MSPSOER	BPSO	MSPSOER	BPSO	MSPSOER	BPSO
1	517.4	514.4	528.3	520.3	536.4	526.2	4.1	3.8
2	516.7	515.8	527.7	520.8	537.8	523.4	4.1	2.3
3	516.9	516.9	528.8	521.1	538.1	527.9	4.3	3.2

Ex = experiment.

Table 4.60
Best Results and their Associated Assembly Sequences of the Proposed Approach based on the MSPSOER and the Approach based on the BPSO.

Approach based	Total assembly time	Assembly sequence
MSPSOER	516.7	1-12-2-4-9-3-5-15-13-16-18-11-6-7-8-10-14-17-19
BPSO	514.4	16-2-13-4-1-15-11-9-6-5-18-7-8-14-12-10-3-17-19

Table 4.60 compares the best results of these approaches that represent the minimum of total assembly time and their associated assembly sequences. The BPSO outperforms the MSPSOER in all experiments.

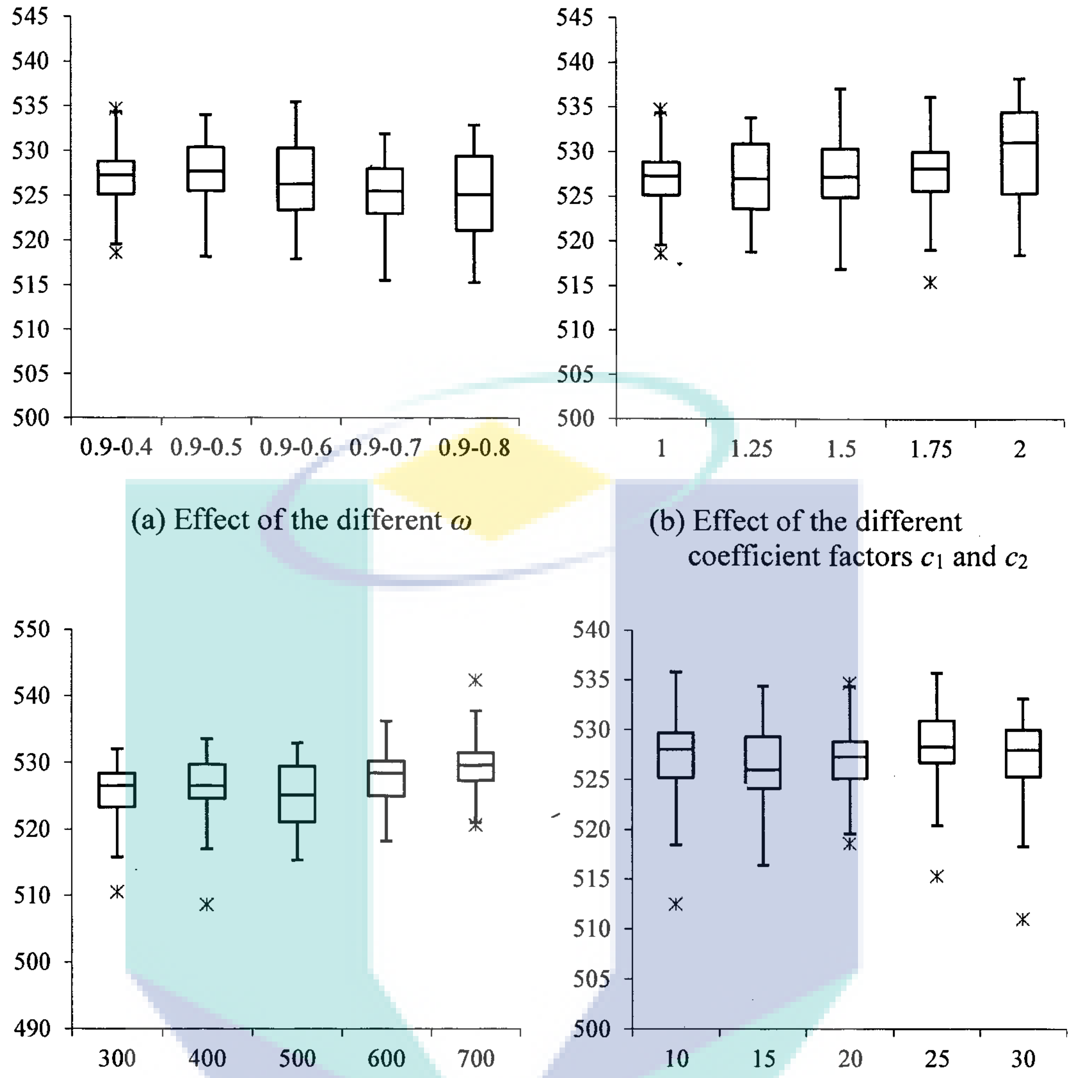
4.5.8 Effect of the MSPSOER Parameters

To quantify the results, the 50 runs are performed for each parameter variation, so that the combination of the best parameter settings can be found. Table 4.61 offers the MSPSOER outcomes as a result of varying its parameters. Figure 4.16 illustrates the results in box plots. It is clear from results shown in Table 4.61 that the best parameters for inertia weight ω, coefficient factors c_1 and c_2, number of particles, and number of iteration T are ω = 0.9 to 0.7, c_1 and c_2 = 2.00, number of particles NOP = 30, T = 500 respectively. The best objective value obtained for these parameters is 508.3. The assembly sequence generated for the best objective value using these parameters is 1-2-12-4-3-9-13-15-11-5-16-6-18-7-8-14-10-17-19. The result clearly shows that the MSPSOER is an efficient approach compared to the SA, the GA, and the BPSO.

Table 4.61

Study of Tuning the MSPSOER Parameters.

Parameter	Min	Mean	Max	SD	Other parameters
ω= 0.9 to 0.4	518.6	526.5	534.7	3.6	T = 500, c_1 and c_2 = 2.00, NOP = 30
ω= 0.9 to 0.5	513.7	527.0	534.0	4.3	T = 500, c_1 and c_2 = 2.00, NOP = 30
ω = 0.9 to 0.6	517.9	526.5	535.5	3.9	T = 500, c_1 and c_2 = 2.00, NOP = 30
ω = 0.9 to 0.7	**508.3**	**525.1**	**536.8**	**4.7**	**T = 500, c_1 and c_2 = 2.00, NOP = 30**
ω = 0.9 to 0.8	515.3	525.1	532.9	4.6	T = 500, c_1 and c_2 = 2.00, NOP = 30
c_1 and c_2 = 1.00	518.5	530.1	538.2	5.3	T = 500, ω= 0.9 to 0.4, NOP = 30
c_1 and c_2 = 1.25	515.5	528.2	536.2	4.7	T = 500, ω= 0.9 to 0.4, NOP = 30
c_1 and c_2 = 1.50	514.1	527.5	537.1	4.6	T = 500, ω= 0.9 to 0.4, NOP = 30
c_1 and c_2 = 1.75	518.8	527.1	533.8	4.2	T = 500, ω= 0.9 to 0.4, NOP = 30
c_1 and c_2 = 2.00	518.6	526.5	534.7	3.6	T = 500, ω= 0.9 to 0.4, NOP = 30
NOP = 10	512.5	527.1	535.8	4.9	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, T = 500
NOP = 15	516.4	526.9	534.4	4.1	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, T = 500
NOP = 20	518.6	526.5	534.7	3.6	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, T = 500
NOP = 25	515.3	527.3	535.7	4.8	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, T = 500
NOP = 30	511.0	527.1	533.1	4.2	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, T = 500
T = 300	513.9	523.0	531.3	3.8	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, NOP = 30
T = 400	518.2	527.7	538.0	4.4	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, NOP = 30
T = 500	518.6	526.5	534.7	3.6	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, NOP = 30
T = 600	508.6	526.8	533.5	4.4	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, NOP = 30
T = 700	510.5	525.4	532.0	4.4	c_1 and c_2 = 2.00, ω= 0.9 to 0.4, NOP = 30



(a) Effect of the different ω

(b) Effect of the different coefficient factors c_1 and c_2

(c) Effect of the different number of iteration

(d) Effect of the different number of particle

Figure 4.16. Effect of the usage of the different parameters (MSPSOER).

To study the effect of the inertia weight ω, the coefficient factors c_1 and c_2, number of particle *NOP*, and number of iteration *T* on the MSPSOER performance, Friedman test is performed on the experimental results shown in Tables 4.61. The average rank is presented in Table 4.62. Based on Table 4.62, the Friedman statistic indicates that using different ω values cause significant difference to the MSPSO because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.0000 > 9.4880), hence demonstrating that the performance of the MSPSO is extensively affected by the preference of ω.

Table 4.62
Friedman Test on the Effect of the Inertia Weight ω, the Coefficient Factors c_1 and c_2, Number of Particle NOP, and Number of Iteration T.

Items	Parameter settings				
Inertia weight ω	0.9 to 0.4	0.9 to 0.5	0.9 to 0.6	0.9 to 0.7	0.9 to 0.8
Average Friedman rank	3.50	5.00	3.50	1.50	1.50
Coefficient factors c_1 and c_2	1.00	1.25	1.50	1.75	2.00
Average Friedman rank	5.00	4.00	3.00	2.00	1.00
Number of iteration *NOP*	10	15	20	25	30
Average Friedman rank	3.00	5.00	1.00	2.00	4.00
Number of iteration *T*	300	400	500	600	700
Average Friedman rank	1.00	5.00	3.00	4.00	2.00

Table 4.63

Holm Procedure on the Initial Weight ω.

Dataset	*P*	z	Holm
ω = 0.9 to 0.5 vs. ω = 0.9 to 0.7	0.0000	4.9497	0.0053
ω = 0.9 to 0.5 vs. ω = 0.9 to 0.8	0.0000	4.9497	0.0053
ω = 0.9 to 0.4 vs. ω = 0.9 to 0.7	0.0047	2.8284	0.0077
ω = 0.9 to 0.4 vs. ω = 0.9 to 0.8	0.0047	2.8284	0.0077
ω = 0.9 to 0.6 vs. ω = 0.9 to 0.7	0.0047	2.8284	0.0077
ω = 0.9 to 0.6 vs. ω = 0.9 to 0.8	0.0047	2.8284	0.0077
ω = 0.9 to 0.4 vs. ω = 0.9 to 0.5	0.0339	2.1213	0.0143
ω = 0.9-0.5 vs. ω = 0.9-0.6	0.0339	2.1213	0.0143
ω = 0.9-0.4 vs. ω = 0.9-0.6	1.0000	0.0000	0.0333
ω = 0.9-0.7 vs. ω = 0.9-0.8	1.0000	0.0000	0.0333

The Holm procedure is then performed and its statistical values are portrayed in Table 4.63. The results of the Holm procedure record that significant differences exist between the performances of the MSPSOER if comparing the results between these two ω values namely ω = 0.9 to 0.5 vs. ω = 0.9 to 0.7, ω = 0.9 to 0.5 vs. ω = 0.9 to 0.8, ω = 0.9 to 0.4 vs. ω = 0.9 to 0.7, ω = 0.9 to 0.4 vs. ω = 0.9 to 0.8, ω = 0.9 to 0.6 vs. ω = 0.9 to 0.7, and ω = 0.9 to 0.6 vs. ω = 0.9 to 0.8.

Table 4.64
Holm Procedure on the Coefficient Factors c_1 and c_2.

Dataset	*P*	z	Holm
c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 2.00$	0.0000	5.6569	0.0050
c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 1.75$	0.0000	4.2426	0.0059
c_1 and $c_2 = 1.25$ vs. c_1 and $c_2 = 2.00$	0.0000	4.2426	0.0059
c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 1.50$	0.0047	2.8284	0.0083
c_1 and $c_2 = 1.25$ vs. c_1 and $c_2 = 1.75$	0.0047	2.8284	0.0083
c_1 and $c_2 = 1.50$ vs. c_1 and $c_2 = 2.00$	0.0047	2.8284	0.0083
c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 1.25$	0.1573	1.4142	0.0200
c_1 and $c_2 = 1.25$ vs. c_1 and $c_2 = 1.50$	0.1573	1.4142	0.0200
c_1 and $c_2 = 1.75$ vs. c_1 and $c_2 = 2.00$	0.1573	1.4142	0.0200
c_1 and $c_2 = 1.50$ vs. c_1 and $c_2 = 1.75$	0.1573	1.4142	0.0200

The Friedman test performed on the effect of the different coefficient factors c_1 and c_2 shows that the MSPSOER with different coefficient factors c_1 and c_2 is significantly different because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6667 > 9.4880). This observation is further studied using the Holm procedure as presented in Table 4.64. The outcomes of Holm procedure reveal that significant differences exist between the performances of the MSPSOER if comparing the results between these two coefficient factors c_1 and c_2 namely, c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 2.00$, c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 1.75$, c_1 and $c_2 = 1.25$ vs. c_1 and $c_2 = 2.00$, c_1 and $c_2 = 1.00$ vs. c_1 and $c_2 = 1.50$, c_1 and $c_2 = 1.25$ vs. c_1 and $c_2 = 1.75$, and c_1 and $c_2 = 1.50$ vs. c_1 and $c_2 = 2.00$.

Table 4.65

Holm Procedure on the Number of Particle NOP.

Dataset	***P***	**z**	**Holm**
NOP = 15 vs. *NOP* = 20	0.0000	5.6569	0.0050
NOP = 15 vs. *NOP* = 25	0.0000	4.2426	0.0059
NOP = 20 vs. *NOP* = 30	0.0000	4.2426	0.0059
NOP = 10 vs. *NOP* = 15	0.0047	2.8284	0.0083
NOP = 25 vs. *NOP* = 30	0.0047	2.8284	0.0083
NOP = 10 vs. *NOP* = 20	0.0047	2.8284	0.0083
NOP = 15 vs. *NOP* = 30	0.1573	1.4142	0.0200
NOP = 10 vs. *NOP* = 30	0.1573	1.4142	0.0200
NOP = 20 vs. *NOP* = 25	0.1573	1.4142	0.0200
NOP = 10 vs. *NOP* = 25	0.1573	1.4142	0.0200

The Friedman statistic also provides a finding on the performance that uses different number of particle values; there are significant differences to the MSPSOER because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6667 > 9.4880), thus presenting that the performance of the MSPSOER is affected by the choice of the number of particle. This result shown in Table 4.65 is then analysed using the Holm procedure. The results of Holm procedure demonstrate that significant difference exists between the performances of the MSPSOER if comparing the results between these two number of particle namely, *NOP* = 15 vs. *NOP* = 20, *NOP* = 15 vs. *NOP* = 25, *NOP* = 20 vs. *NOP* = 30, *NOP* = 10 vs. *NOP* = 15, *NOP* = 25 vs. *NOP* = 30, and *NOP* = 10 vs. *NOP* = 200.

Table 4.66
Holm Procedure on the Number of Iteration T.

Dataset	*P*	z	Holm
$T = 300$ vs. $T = 700$	0.0000	5.6569	0.0050
$T = 300$ vs. $T = 600$	0.0000	4.2426	0.0059
$T = 500$ vs. $T = 700$	0.0000	4.2426	0.0059
$T = 300$ vs. $T = 400$	0.0047	2.8284	0.0083
$T = 500$ vs. $T = 600$	0.0047	2.8284	0.0083
$T = 400$ vs. $T = 700$	0.0047	2.8284	0.0083
$T = 300$ vs. $T = 500$	0.1573	1.4142	0.0200
$T = 400$ vs. $T = 500$	0.1573	1.4142	0.0200
$T = 600$ vs. $T = 700$	0.1573	1.4142	0.0200
$T = 400$ vs. $T = 600$	0.1573	1.4142	0.0200

Meanwhile, the Friedman statistic shows that using different number of iteration values makes significant difference to the MSPSOER because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6667 > 9.4880), thus presenting that the performance of the MSPSOER is affected by the choice of number of iteration. This result is further studied using Holm procedure as shown in Table 4.66. The results of Holm procedure reveal that significant differences exist between the performances of the MSPSOER if comparing the results between these two number of iteration namely, $T = 300$ and $T = 700$, $T = 300$ and $T = 600$, $T = 500$ and $T = 700$, $T = 300$ and $T = 400$, $T = 500$ and $T = 600$, and $T = 400$ and $T = 700$.

4.5.9 Results of the Proposed Approach based on the MSGSA compared to the SA

In the experiments, the parameters used for the proposed approach based on the MSGSA and the approach based on the SA are presented in Table 4.67. Based on these parameters, the quality of the results of the proposed approach based on the MSGSA for the 50 runs is presented in Table 4.68. Table 4.69 shows a comparison of the best results of the proposed approach based on the MSGSA and the approach based on the SA with their assembly sequences. Using the minimum values of the proposed approach based on the MSGSA and the approach based on the SA given in Table 4.69, it seems that the proposed approach based on the MSGSA outperformed the approach based on the SA in obtaining the minimum total assembly time of the ASP problem. The mean or average of

Table 4.67
Experimental Parameters for the Proposed Approach based on the MSGSA and the Approach based on the SA.

Parameters	MSGSA	SA
Iteration	500	500
Number of agent	30	NAP
G_0	100	NAP
β	20	NAP
Initial temperature (°C)	NAP	100
Cooling rate	NAP	0.95
Number of run	50	NA
Number of run	50	NA

NA = not available from the source, NAP = not applicable.

Table 4.68
Quality of the Results for the Proposed Approach based on the MSGSA.

Experiment	Min	Mean	Max	SD
1	509.1	519.0	525.6	3.1

Table 4.69
Best Results and Associated Assembly Sequences of the Proposed Approach based on the MSGSA and the Approach based on the SA.

Approach based	Total assembly time	Assembly sequence
MSGSA	509.1	2-1-4-9-12-3-5-13-15-16-18-6-11-7-8-10-14-17-19
SA	528.7	2-1-4-9-3-12-13-16-5-15-18-6-11-7-8-10-14-17-19

the total assembly time yielded by the proposed approach based on the MSGSA to solve the ASP problem is also better than the minimum of the assembly time produced by the approach based on the SA.

4.5.10 Results of the Proposed Approach based on the MSGSA compared to the GA

Table 4.70 shows the parameters used for the proposed approach based on the MSGSA and the approach based on the GA in three different experiments. The quality of the results of the proposed approach based on the MSGSA for 50 runs are presented in Table 4.71.

Table 4.70
Experimental Parameters for the Proposed Approach based on the MSGSA and the Approach based on the GA.

Parameters	**Experiment 1**		**Experiment 2**		**Experiment 3**	
	MSGSA	GA	MSGSA	GA	MSGSA	GA
Iteration	100	100	100	100	100	100
Number of agent	20	20	40	40	100	100
G_0	100	NAP	100	NAP	100	NAP
β	20	NAP	20	NAP	20	NAP
Mutation rate	NAP	0.05	NAP	0.05	NAP	0.05
Crossover	NAP	0.5	NAP	0.5	NAP	0.5
Number of run	50	NA	50	NA	50	NA

NA = not available from the source, NAP = not applicable.

Table 4.71
Quality of the Results for the Proposed Approach based on the MSGSA.

Experiment	**Min**	**Mean**	**Max**	**SD**
1	515.5	524.6	531.1	3.9
2	513.0	523.0	530.2	4.1
3	515.5	524.6	531.1	3.9

Table 4.72

Best Results of the Proposed Approach based on the MSGSA and the Approach based on the GA for Each Experiment.

Experiment	Total assembly time (MSGSA)	Total assembly time (GA)
1	515.5	535.1
2	513.0	527.9
3	514.9	524.1

Table 4.73

Best results and the Associated Assembly Sequences of the Proposed Approach based on the MSGSA and the Approach based on the GA.

Approach based	Total assembly time	Assembly sequence
MSGSA	513.0	1-2-4-3-9-5-12-13-15-16-18-11-6-7-8-14-10-17-19
GA	524.1	2-18-3-12-1-13-16-5-11-15-4-6-9-7-8-10-14-17-19

Table 4.72 shows that each best value obtained by the proposed approach based on the MSGSA in the three different experiments are better than to each best value produced by the approach based on the GA. Meanwhile, Table 4.73 presents that the minimum total assembly time of the ASP obtained by the proposed approach based on the MSGSA is better than the value yielded using the approach based on the GA.

Table 4.72
Best Results of the Proposed Approach based on the MSGSA and the Approach based on the GA for Each Experiment.

Experiment	Total assembly time (MSGSA)	Total assembly time (GA)
1	515.5	535.1
2	513.0	527.9
3	514.9	524.1

Table 4.73
Best results and the Associated Assembly Sequences of the Proposed Approach based on the MSGSA and the Approach based on the GA.

Approach based	Total assembly time	Assembly sequence
MSGSA	513.0	1-2-4-3-9-5-12-13-15-16-18-11-6-7-8-14-10-17-19
GA	524.1	2-18-3-12-1-13-16-5-11-15-4-6-9-7-8-10-14-17-19

Table 4.72 shows that each best value obtained by the proposed approach based on the MSGSA in the three different experiments are better than to each best value produced by the approach based on the GA. Meanwhile, Table 4.73 presents that the minimum total assembly time of the ASP obtained by the proposed approach based on the MSGSA is better than the value yielded using the approach based on the GA.

4.5.11 Results of the Proposed Approach based on the MSGSA compared to the BPSO

Parameters used for the proposed approach based on the MSGSA and the approach based on the BPSO are listed in Table 4.74 for three different experiments. The quality of the results of the proposed approach based on the MSGSA and the approach based on BPSO for 10 runs is presented in Table 4.75. As shown in Table 4.75, the proposed approach based on the MSGSA yields smaller values for the minimum and mean for each experiment, thus verifying that the proposed approach based on the MSGSA produces higher quality solutions compared to the approach based on the BPSO.

Table 4.74
Experimental Parameters for the Proposed Approach based on the MSGSA and the Approach based on the BPSO.

Parameters	**Experiment 1**		**Experiment 2**		**Experiment 3**	
	MSGSA	BPSO	MSGSA	BPSO	GSA	BPSO
Iteration	500	500	500	500	500	500
Number of agent	40	40	50	50	60	60
G_0	100	NAP	100	NAP	100	NAP
β	20	NAP	20	NAP	20	NAP
Inertia weight, ω	NAP	0.9 to 0.4	NAP	0.9 to 0.4	NAP	0.9 to 0.4
Coefficient factor, c_1 and c_2	NAP	2	NAP	2	NAP	2
Number of run	50	50	50	50	50	50

NA = not available from the source, NAP = not applicable.

Table 4.75
Quality of the Results for the proposed approach based on the MSGSA and the Approach based on the BPSO.

Ex	**Min**		**Mean**		**Max**		**SD**	
	MSGSA	BPSO	MSGSA	BPSO	MGSA	BPSO	MSGSA	BPSO
1	508.4	514.4	517.3	520.3	522.6	526.2	4.1	3.8
2	510.8	515.8	518.0	520.8	522.2	523.4	3.7	2.3
3	509.5	516.9	517.7	521.1	522.4	527.9	4.3	3.2

Ex = experiment.

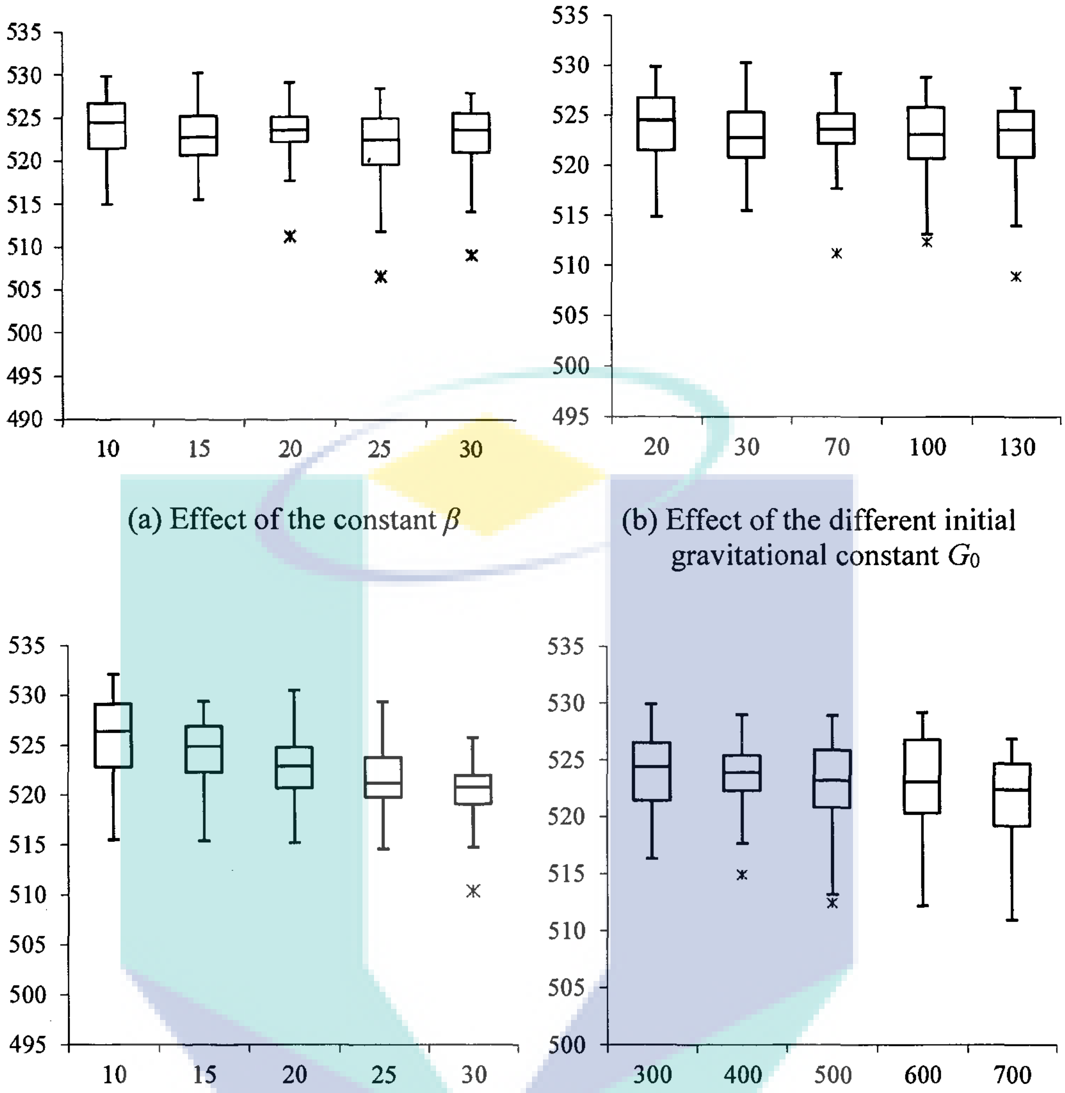
Table 4.76
Best Results and their Associated Assembly Sequences of the Proposed Approach based on the MSGSA and the Approach based on the BPSO.

Approach based	Total assembly time	Assembly sequence
MSGSA	508.4	1-2-4-3-9-5-12-13-15-16-18-11-6-7-8-14-10-17-19
BPSO	514.4	16-2-13-4-1-15-11-9-6-5-18-7-8-14-12-10-3-17-19

Table 4.76 compares the best results of these approaches that represent the minimum of total assembly time and their associated assembly sequences where the MSGSA outperforms the BPSO in all experiments.

4.5.12 Effect of the MSGSA Parameters

To quantify the results, the 50 runs are performed for each parameter variation, so that the combination of the best parameter settings can be found. Table 4.77 offers the MSGSA outcomes as a result of varying its parameters. Figure 4.17 illustrates the results in box plots. It is clear from results shown in Table 4.77 that the best parameters for constant β, initial gravitational constant G_0, number of agents *NOA* and number of iteration T are $\beta = 20$, $G_0 = 100$, $NOA = 30$, $T = 500$ respectively. The best objective value obtained.for these parameters is 508.3. The assembly sequence generated for the best objective value using these parameters is 1-2-4-3-9-12-13-5-16-15-18-11-6-7-8-14-10-17-19. The result clearly shows that the MSGSA is an efficient approach compared to the SA, the GA, and the BPSO.



(a) Effect of the constant β

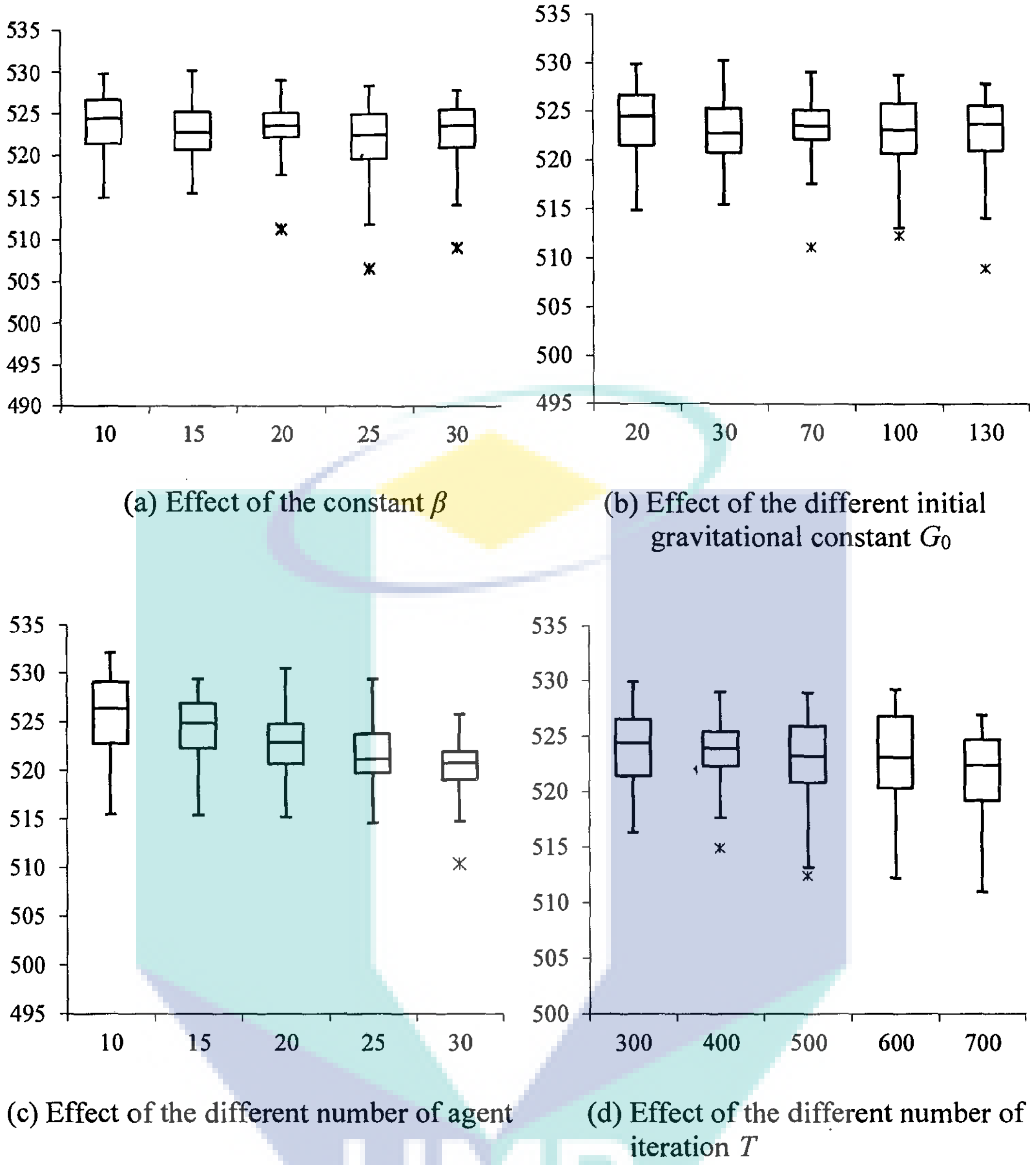
(b) Effect of the different initial gravitational constant G_0

(c) Effect of the different number of agent

(d) Effect of the different number of iteration T

Figure 4.17. Effect of the usage of the different parameters (MSGSA).

To study the effect of the constant β, the initial gravitational constant G_0, the number of agents NOA and the number of iteration T on the MSGSA performance, the Friedman test is performed on the experimental results presented in Table 4.77.



(a) Effect of the constant β

(b) Effect of the different initial gravitational constant G_0

(c) Effect of the different number of agent

(d) Effect of the different number of iteration T

Figure 4.17. Effect of the usage of the different parameters (MSGSA).

To study the effect of the constant β, the initial gravitational constant G_0, the number of agents *NOA* and the number of iteration T on the MSGSA performance, the Friedman test is performed on the experimental results presented in Table 4.77.

Table 4.77

Study of Tuning the MSGSA Parameters.

Parameter	Min	Mean	Max	SD	Other parameters
$\beta = 10$	511.8	522.0	527.7	3.6	$T = 500$, $NOA = 20$, $G_0 = 100$
$\beta = 15$	511.8	522.0	527.7	3.6	$T = 500$, $NOA = 20$, $G_0 = 100$
$\beta = 20$	512.4	522.8	528.9	3.9	$T = 500$, $NOA = 20$, $G_0 = 100$
$\beta = 25$	512.4	522.8	528.9	3.9	$T = 500$, $NOA = 20$, $G_0 = 100$
$\beta = 30$	512.4	522.8	528.9	3.9	$T = 500$, $NOA = 20$, $G_0 = 100$
$G_0 = 20$	514.9	523.8	529.9	3.8	$T = 500$, $NOA = 20$, $\beta = 20$
$G_0 = 30$	515.5	522.8	530.3	3.5	$T = 500$, $NOA = 20$, $\beta = 20$
$G_0 = 70$	511.2	522.9	529.2	3.3	$T = 500$, $NOA = 20$, $\beta = 20$
$G_0 = 100$	512.4	522.8	528.9	3.9	$T = 500$, $NOA = 20$, $\beta = 20$
$G_0 = 130$	509.0	522.7	527.9	3.9	$T = 500$, $NOA = 20$, $\beta = 20$
$NOA = 10$	515.5	525.8	532.1	4.2	$T = 500$, $G_0 = 100$, $\beta = 20$
$NOA = 15$	515.4	523.9	529.4	3.4	$T = 500$, $G_0 = 100$, $\beta = 20$
$NOA = 20$	515.2	522.6	530.5	3.0	$T = 500$, $G_0 = 100$, $\beta = 20$
$NOA = 25$	514.6	521.6	529.4	3.3	$T = 500$, $G_0 = 100$, $\beta = 20$
$NOA = 30$	510.4	520.5	526.4	3.3	$T = 500$, $G_0 = 100$, $\beta = 20$
$T = 300$	516.3	524.3	529.9	3.6	$NOA = 20$, $G_0 = 100$, $\beta = 20$
$T = 400$	514.9	523.6	529.0	3.2	$NOA = 20$, $G_0 = 100$, $\beta = 20$
$T = 500$	512.4	522.8	528.9	3.9	$NOA = 20$, $G_0 = 100$, $\beta = 20$
$T = 600$	512.2	522.9	530.2	4.0	$NOA = 20$, $G_0 = 100$, $\beta = 20$
$T = 700$	**508.3**	**521.9**	**529.0**	**3.7**	**$NOA = 20$, $G_0 = 100$, $\beta = 20$**

Table 4.78
Friedman Test on the Effect of the Inertia Weight ω, the Coefficient Factors c_1 and c_2, Number of Particle NOP, and Number of Iteration T.

Items	Parameter settings				
Constant β	.10	15	20	25	30
Average Friedman rank	1.50	1.50	4.00	4.00	4.00
Initial gravitational constant G	20	30 ·	·70	100	130
Average Friedman rank	5.00	2.50	4.00	2.50	1.00
Number of agents *NOA*	10	15	20	25	30
Average Friedman rank	5	4	3	2	1
Number of iteration T	300	400	500	600	700
Average Friedman rank	5	4	2	3	1

The average rank is presented in Table 4.78. The Friedman statistic shows that using different β values cause significant difference to the MSGSA because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (17.0000 > 9.4880), hence indicating that the performance of the MSGSA is affected by the preference of β.

Table 4.79

Holm Procedure on the Constant β.

Dataset	*P*	z	Holm
$\beta = 10$ vs. $\beta = 20$	0.0004	3.5356	0.0067
$\beta = 10$ vs. $\beta = 25$	0.0004	3.5356	0.0067
$B = 10$ vs. $\beta = 30$	0.0004	3.5356	0.0067
$\beta = 15$ vs. $\beta = 20$	0.0004	3.5356	0.0067
$\beta = 15$ vs. $\beta = 25$	0.0004	3.5356	0.0067
$\beta = 15$ vs. $\beta = 30$	0.0004	3.5356	0.0067
$\beta = 10$ vs. $\beta = 15$	1.0000	0.0000	0.0200
$\beta = 20$ vs. $\beta = 25$	1.0000	0.0000	0.0200
$\beta = 20$ vs. $\beta = 30$	1.0000	0.0000	0.0200
$\beta = 25$ vs. $\beta = 30$	1.0000	0.0000	0.0200

Table 4.80

Holm Procedure on the Initial Gravitational Constant G_0.

Dataset	*P*	z	Holm
$G_0 = 20$ vs. $G_0 = 70$	0.0000	5.6569	0.0050
$G_0 = 20$ vs. $G_0 = 30$	0.0000	4.2426	0.0056
$G_0 = 70$ vs. $G_0 = 130$	0.0004	3.5355	0.0067
$G_0 = 20$ vs. $G_0 = 100$	0.0004	3.5355	0.0067
$G_0 = 30$ vs. $G_0 = 130$	0.0339	2.1213	0.0111
$G_0 = 70$ vs. $G_0 = 100$	0.0339	2.1213	0.0111
$G_0 = 20$ vs. $G_0 = 130$	0.0339	2.1213	0.0111
$G_0 = 100$ vs. $G_0 = 130$	0.0339	2.1213	0.0111
$G_0 = 30$ vs. $G_0 = 70$	0.1573	1.4142	0.0250
$G_0 = 30$ vs. $G_0 = 100$	1.0000	0.0000	0.0500

The Holm procedure is then performed and the results are shown in Table 4.79. The results demonstrate that significant differences exist between the performances of the MSGSA if comparing the results between these two β namely, $\beta = 10$ vs. $\beta = 20$, $\beta = 10$ vs. $\beta = 25$, $\beta = 10$ vs. $\beta = 30$, $\beta = 15$ vs. $\beta = 20$, $\beta = 15$ vs. $\beta = 25$, and $\beta = 15$ vs. $\beta = 30$. On the other hand, the results of Friedman test show that significant difference also exists in the MSGSA performance for different G_0 values because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.3333 > 9.4880), hence indicating that the performance of the MSGSA is extensively affected by the preference of G_0. Holm procedure is then conducted and its statistical values are provided in Table 4.80. The results of the Holm procedure show that significant differences exist between the performances of the MSGSA if comparing the results between these two G_0 values namely $G_0 = 20$ vs. $G_0 = 70$, $G_0 = 20$ vs. $G_0 = 30$, $G_0 = 70$ vs. $G_0 = 130$, and $G_0 = 20$ vs. $G_0 = 100$.

Table 4.81
Holm Procedure on the Number of Agents NOA.

Dataset	*P*	z	Holm
NOA = 10 vs. *NOA* = 30	0.0000	5.6569	0.0050
NOA = 10 vs. *NOA* = 25	0.0000	4.2426	0.0059
NOA = 15 vs. *NOA* = 30	0.0000	4.2426	0.0059
NOA = 10 vs. *NOA* = 20	0.0047	2.8284	0.0083
NOA = 15 vs. *NOA* = 25	0.0047	2.8284	0.0083
NOA = 20 vs. *NOA* = 30	0.0047	2.8284	0.0083
NOA = 10 vs. *NOA* = 15	0.1573	1.4142	0.0200
NOA = 15 vs. *NOA* = 20	0.1573	1.4142	0.0200
NOA = 25 vs. *NOA* = 30	0.1573	1.4142	0.0200
NOA = 20 vs. *NOA* = 25	0.1573	1.4142	0.0200

The Friedman test performed on the effect of the different number of agents shows that the MSGSA with different number of agents is significantly different because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6667 > 9.4880). This observation is further studied using Holm procedure as shown in Table 4.81. The outcomes of the Holm procedure reveal that significant differences exist between the performances of the MSGSA if comparing the results between these two number of agents namely, *NOA* = 10 and *NOA* = 30, *NOA* = 10 and *NOA* = 25, *NOA* = 15 and *NOA* = 30, *NOA* = 10 and *NOA* = 20, *NOA* = 15 and *NOA* = 25, and *NOA* = 20 and *NOA* = 30.

Table 4.82

Holm Procedure on the Number of iteration T.

Dataset	***P***	**z**	**Holm**
$T = 300$ vs. $T = 600$	0.0000	5.6569	0.0050
$T = 300$ vs. $T = 700$	0.0000	4.2426	0.0059
$T = 500$ vs. $T = 600$	0.0000	4.2426	0.0059
$T = 300$ vs. $T = 400$	0.0047	2.8284	0.0083
$T = 500$ vs. $T = 700$	0.0047	2.8284	0.0083
$T = 400$ vs. $T = 600$	0.0047	2.8284	0.0083
$T = 300$ vs. $T = 500$	0.1573	1.4142	0.0200
$T = 400$ vs. $T = 500$	0.1573	1.4142	0.0200
$T = 600$ vs. $T = 700$	0.1573	1.4142	0.0200
$T = 400$ vs. $T = 700$	0.1573	1.4142	0.0200

The Friedman statistic shows that using different number of iteration values makes significant difference to the MSGSA because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6667 > 9.4880), thus presenting that the performance of the MSGSA is affected by the choice of number of iteration. This result is further studied using Holm procedure as in Table 4.82. The result of Holm procedure demonstrates that significant difference exists between the performances of the MSGSA if comparing the results between these two number of iteration namely, $T = 300$ and $T = 600$, $T = 300$ and $T = 700$, $T = 500$ and $T = 600$, $T = 300$ and $T = 400$, $T = 500$ and $T = 700$, and $T = 400$ and $T = 600$.

4.5.13 Results of the Proposed Approach based on the MSGSAER compared to the SA

In the experiments, the parameters used for the proposed approach based on the MSGSAER and the approach based on SA are presented in Table 4.83. Based on these parameters, the quality of the results of the proposed approach based on the MSGSAER for the 50 runs is presented in Table 4.84. Table 4.85 shows a comparison of the best results of the proposed approach based on the MSGSAER and the approach based on the SA with their assembly sequences. Using the minimum values of the proposed approach based on the MSGSAER and the approach based on the SA given in Table 4.85, it seems that the proposed approach based on the MSGSAER outperformed the approach based

Table 4.83
Experimental Parameters for the Proposed Approach based on the MSGSAER and the Approach based on the SA.

Parameters	MSGSAER	SA
Iteration	500	500
Number of agent	30	NAP
G_0	100	NAP
β	20	NAP
Initial temperature (°C)	NAP	100
Cooling rate	NAP	0.95
Number of run	50	NA

NA = not available from the source, NAP = not applicable.

Table 4.84
Quality of the Results for the Proposed Approach based on the MSGSAER.

Experiment	Min	Mean	Max	SD
1	511.9	519.0	526.4	3.2

Table 4.85
Best Results and Associated Assembly Sequences of the Proposed Approach based on the MSGSAER and the Approach based on the SA.

Approach based	Total assembly time	Assembly sequence
MSGSAER	511.9	2-15-4-1-9-12-3-5-13-18-6-16-11-7-8-14-10-17-19
SA	528.7	2-1-4-9-3-12-13-16-5-15-18-6-11-7-8-10-14-17-19

on the SA, obtaining the minimum total assembly time of the ASP problem. The mean or average of the total assembly time yielded by the proposed approach based on the MSGSAER to solve the ASP problem is also better than the minimum of the assembly time produced by the approach based on the SA.

4.5.14 Results of the Proposed Approach based on the MSGSAER compared to the GA

Table 4.86 shows the parameters used for the proposed approach based on the MSGSAER and the approach based on the GA in three different experiments. The quality of the results of the proposed approach based on the MSGSAER for 50 runs is presented in Table 4.87.

Table 4.86
Experimental Parameters for the Proposed Approach based on the MSGSAER and the Approach based on the GA.

Parameters	Experiment 1		Experiment 2		Experiment 3	
	MSGSAER	GA	MSGSAER	GA	MSGSAER	GA
Iteration	100	100	100	100	100	100
Number of agent	20	20	40	40	100	100
G_0	100	NAP	100	NAP	100	NAP
β	20	NAP	20	NAP	20	NAP
Mutation rate	NAP	0.05	NAP	0.05	NAP	0.05
Crossover	NAP	0.5	NAP	0.5	NAP	0.5
Number of run	50	NA	50	NA	50	NA

NA = not available from the source, NAP = not applicable.

Table 4.87
Quality of the Results for the Proposed Approach based on the MSGSAER.

Experiment	Min	Mean	Max	SD
1	514.1	524.0	533.0	4.0
2	508.6	521.8	529.1	3.8
3	513.5	521.9	528.7	3.5

Table 4.88
Best Results of the Proposed Approach based on the MSGSAER and the Approach based on the GA for Each Experiment.

Experiment	**Total assembly time (MSGSAER)**	**Total assembly time (GA)**
1	514.1	535.1
2	508.6	527.9
3	513.5	524.1

Table 4.89
Best results and the Associated Assembly Sequences of the Proposed Approach based on the MSGSAER and the Approach based on the GA.

Approach based	**Total assembly time**	**Assembly sequence**
MSGSAER	508.6	1-2-4-9-3-12-5-15-13-16-18-6-11-7-8-10-14-17-19
GA	524.1	2-18-3-12-1-13-16-5-11-15-4-6-9-7-8-10-14-17-19

Table 4.88 shows the results of the approach based on the GA for the three different experiments. In all three experiments, the proposed approach based on the MSGSAER surpassed the approach based on the GA in minimizing the total assembly time for the respective parameters. The best value obtained by the proposed approach based on the MSGSAER in the three different experiments is then selected to be compared against the best value produced by the approach based on the GA, as presented in Table 4.88. Referring to the results given in Table 4.89, the proposed approach based on the MSGSAER also outperformed the approach based on the GA in obtaining the minimum total assembly time of the ASP problem. In the three experiments, the mean of the total assembly time produced by the proposed approach based on the MSGSAER to solve the ASP problem is also better than the minimum of the assembly time produced by the approach based on the GA.

4.5.15 Results of the Proposed Approach based on the MSGSAER compared to the BPSO

Parameters used for the proposed approach based on the MSGSAER and the approach based on the BPSO are listed in Table 4.90 for three different experiments. The quality of the results of the proposed approach based on the MSGSAER and the approach based on the BPSO for 10 runs is presented in Table 4.91. Table 4.91 shows that the proposed approach based on the MSGSAER yields smaller values for the minimum and mean for each experiment, thus verifying that the

Table 4.90
Experimental Parameters for the Proposed Approach based on the MSGSAER and the Approach based on the BPSO.

	Experiment 1		**Experiment 2**		**Experiment 3**	
Parameters	MSGSAER	BPSO	MSGSAER	BPSO	MSGSAER	BPSO
Iteration	500	500	500	500	500	500
Number of agent	40	40	50	50	60	60
G_0	100	NAP	100	NAP	100	NAP
β	20	NAP	20	NAP	20	NAP
Inertia weight, ω	NAP	0.9 to 0.4	NAP	0.9 to 0.4	NAP	0.9 to 0.4
Coefficient factor, c_1 and c_2	NAP	2	NAP	2	NAP	2
Number of run	50	50	50	50	50	50

NA = not available from the source, NAP = not applicable.

Table 4.91
Quality of the Results for the proposed approach based on the MSGSAER and the Approach based on the BPSO.

	Min		**Mean**		**Max**		**SD**	
Ex	MSGSAER	BPSO	MSGSAER	BPSO	MSGSAER	BPSO	MSGSAER	BPSO
1	511.6	514.4	517.6	520.3	524.9	526.2	4.2	3.8
2	510.4	515.8	515.2	520.8	518.5	523.4	2.6	2.3
3	512.3	516.9	517.6	521.1	522.5	527.9	3.2	3.2

Ex = experiment.

Table 4.92

Best Results and their Associated Assembly Sequences of the Proposed Approach based on the MSGSAER and the Approach based on the BPSO.

Approach based	Total assembly time	Assembly sequence
MSGSAER	510.4	2-1-15-4-3-9-12-13-16-5-18-6-7-8-11-10-14-17-19
BPSO	514.4	16-2-13-4-1-15-11-9-6-5-18-7-8-14-12-10-3-17-19

proposed approach based on the MSGSAER produces higher quality solutions compared to the approach based on the BPSO. Table 4.92 compares the best results of these approaches that represent the minimum of total assembly time and their associated assembly sequences. The MSGSAER outperforms the BPSO in all experiments.

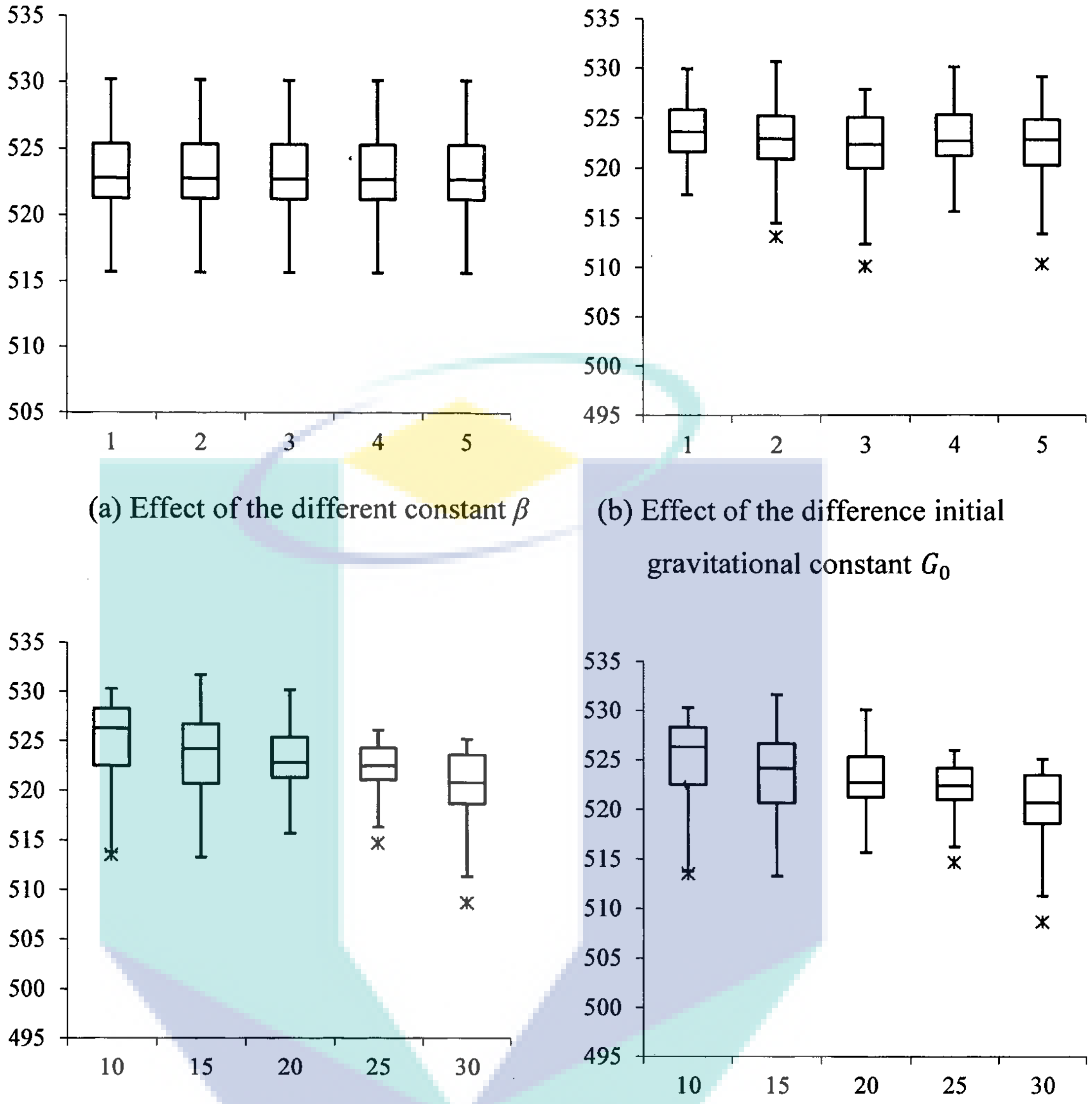
4.5.16 Effect of the MSGSAER Parameters

To quantify the results, the 50 runs are performed for each parameter variation, so that the combination of the best parameter settings can be found. Table 4.93 offers the MSGSAER outcomes as a result of varying its parameters. Figure 4.18 illustrates the results in box plots. It is clear from results shown in Table 4.93 that the best parameters for constant β, initial gravitational constant G_0, number of agents NOA and number of iteration T are $\beta = 20$, $G_0 = 100$, $NOA = 30$, $T = 500$ respectively. The best objective value obtained for these parameters is 508.7, which is shown in bold font. The assembly sequence generated for the best objective value using these parameters is 1-2-15-4-3-9-12-13-16-5-18-11-6-7-8-14-10-17-19. The result clearly shows that the MSGSAER is an efficient approach compared to the SA, the GA, and the BPSO.

Table 4.93

Study of Tuning the MSGSAER Parameters.

Parameter	**Min**	**Mean**	**Max**	**SD**	**Other parameters**
$\beta = 10$	515.7	522.4	530.2	3.6	$T = 500, NOA = 20, G_0 = 100$
$\beta = 15$	515.7	522.4	530.2	3.6	$T = 500, NOA = 20, G_0 = 100$
$\beta = 20$	515.7	522.4	530.2	3.6	$T = 500, NOA = 20, G_0 = 100$
$\beta = 25$	515.7	522.4	530.2	3.6	$T = 500, NOA = 20, G_0 = 100$
$\beta = 30$	515.7	522.4	530.2	3.6	$T = 500, NOA = 20, G_0 = 100$
$G_0 = 20$	517.3	524.2	531.5	3.1	$T = 500, NOA = 20, \beta = 20$
$G_0 = 30$	513.1	522.3	530.6	4.1	$T = 500, NOA = 20, \beta = 20$
$G_0 = 70$	510.1	522.0	527.9	3.9	$T = 500, NOA = 20, \beta = 20$
$G_0 = 100$	515.7	522.4	530.2	3.6	$T = 500, NOA = 20, \beta = 20$
$G_0 = 130$	510.4	522.5	529.3	3.4	$T = 500, NOA = 20, \beta = 20$
$NOA = 10$	513.5	525.0	530.6	3.9	$T = 500, G_0 = 100, \beta = 20$
$NOA = 15$	513.3	524.1	531.7	3.7	$T = 500, G_0 = 100; \beta = 20$
$NOA = 20$	515.7	522.4	530.2	3.6	$T = 500, G_0 = 100, \beta = 20$
$NOA = 25$	514.7	522.0	526.1	2.9	$T = 500, G_0 = 100, \beta = 20$
$\boldsymbol{NOA = 30}$	**508.7**	**520.2**	**525.2**	**3.5**	$\boldsymbol{T = 500, G_0 = 100, \beta = 20}$
$T = 300$	513.9	523.0	531.3	3.8	$NOA = 20, G_0 = 100, \beta = 20$
$T = 400$	508.8	522.2	529.4	4.3	$NOA = 20, G_0 = 100, \beta = 20$
$T = 500$	515.7	522.4	530.2	3.6	$NOA = 20, G_0 = 100, \beta = 20$
$T = 600$	509.9	521.4	526.9	3.6	$NOA = 20, G_0 = 100, \beta = 20$
$T = 700$	514.0	521.5	528.5	3.3	$NOA = 20, G_0 = 100, \beta = 20$



(a) Effect of the different constant β

(b) Effect of the difference initial gravitational constant G_0

(e) Effect of the different number of agent

(d) Effect of the different number of iteration

Figure 4.18. Effect of the usage of the different parameters (MSGSAER).

To study the effect of the constant β, the initial gravitational constant G_0, the number of agents *NOA* and the number of iteration *T* on the MSGSAER performance, Friedman test is performed on the experimental results in Tables 4.93.

Table 4.94
Friedman Test on the Effect of the Inertia Weight ω, the Coefficient Factors c_1 and c_2, Number of Particle NOP, and Number of Iteration T.

Items	**Parameter settings**				
Constant β	10	15	20	25	30
Average Friedman rank	3.00	3.00	3.00	3.00	3.00
Initial gravitational constant G	20	30	70	100	130
Average Friedman rank	5.00	2.00	1.00	3.00	4.00
Number of agents *NOA*	10	15	20	25	30
Average Friedman rank	5	4	3	2	1
Number of iteration T	300	400	500	600	700
Average Friedman rank	5	3	4	1	2

The average rank is presented in Table 4.94. The Friedman statistic shows that using different β values makes no significant difference to the MSGSAER, thus showing that the performance of the MSGSAER is not greatly affected by the choice of β. This result is confirmed by boxplots in Figure 4.18(a) where the size of the boxes is similar to each other. Occasionally, the results obtained by the MSGSAER contains multiple outliers, caused by the stochastic behaviour of the MSGSAER.

Table 4.95

Holm Procedure on the Initial Gravitational Constant G_0.

Dataset	*P*	z	Holm
$G_0 = 20$ vs. $G_0 = 70$	0.0000	5.6569	0.0050
$G_0 = 20$ vs. $G_0 = 30$	0.0000	4.2426	0.0056
$G_0 = 70$ vs. $G_0 = 130$	0.0004	3.5355	0.0067
$G_0 = 20$ vs. $G_0 = 100$	0.0004	3.5355	0.0067
$G_0 = 30$ vs. $G_0 = 130$	0.0339	2.1213	0.0111
$G_0 = 70$ vs. $G_0 = 100$	0.0339	2.1213	0.0111
$G_0 = 20$ vs. $G_0 = 130$	0.0339	2.1213	0.0111
$G_0 = 100$ vs. $G_0 = 130$	0.0339	2.1213	0.0111
$G_0 = 30$ vs. $G_0 = 70$	0.1573	1.4142	0.0250
$G_0 = 30$ vs. $G_0 = 100$	1.0000	0.0000	0.0500

On the other hand, the results of Friedman test show that significant difference also exists in the MSGSAER performance for different G_0 values because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6666 > 9.4880), hence indicating that the performance of the MSGSAER is extensively affected by the preference of G_0. Holm procedure is then conducted and its statistical values are provided in Table 4.95. The results of the Holm procedure show that significant differences exist between the performances of the MSGSAER if comparing the results between these two G_0 values namely $G_0 = 20$ vs. $G_0 = 70$, $G_0 = 20$ vs. $G_0 = 30$, $G_0 = 70$ vs. $G_0 = 130$, $G_0 = 20$ vs. $G_0 = 100$, $G_0 = 30$ vs. $G_0 = 130$, and $G_0 = 70$ vs. $G_0 = 100$.

Table 4.96

Holm Procedure on the Number of Agents NOA.

Dataset	***P***	**z**	**Holm**
NOA = 10 vs. *NOA* = 30	0.0000	5.6569	0.0050
NOA = 10 vs. *NOA* = 25	0.0000	4.2426	0.0059
NOA = 15 vs. *NOA* = 30	0.0000	4.2426	0.0059
NOA = 10 vs. *NOA* = 20	0.0047	2.8284	0.0083
NOA = 15 vs. *NOA* = 25	0.0047	2.8284	0.0083
NOA = 20 vs. *NOA* = 30	0.0047	2.8284	0.0083
NOA = 10 vs. *NOA* = 15	0.1573	1.4142	0.0200
NOA = 15 vs. *NOA* = 20	0.1573	1.4142	0.0200
NOA = 25 vs. *NOA* = 30	0.1573	1.4142	0.0200
NOA = 20 vs. *NOA* = 25	0.1573	1.4142	0.0200

The Friedman test performed on the effect of the different number of agents shows that the MSGSAER with different number of agents is significantly different because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6667 > 9.4880). This observation is further studied using the Holm procedure as shown in Table 4.96. The outcomes of the Holm procedure reveal that significant differences exist between the performances of the MSGSAER if comparing the results between these two number of agents namely, *NOA* = 10 vs. *NOA* = 30, *NOA* = 10 vs. *NOA* = 25, *NOA* = 15 vs. *NOA* = 30, *NOA* = 10 vs. *NOA* = 20, *NOA* = 15 vs. *NOA* = 25, and *NOA* = 20 vs. *NOA* = 30.

Table 4.97
Holm Procedure on the Number of iteration T.

Dataset	*P*	z	Holm
$T = 300$ vs. $T = 600$	0.0000	5.6569	0.0050
$T = 300$ vs. $T = 700$	0.0000	4.2426	0.0059
$T = 500$ vs. $T = 600$	0.0000	4.2426	0.0059
$T = 300$ vs. $T = 400$	0.0047	2.8284	0.0083
$T = 500$ vs. $T = 700$	0.0047	2.8284	0.0083
$T = 400$ vs. $T = 600$	0.0047	2.8284	0.0083
$T = 300$ vs. $T = 500$	0.1573	1.4142	0.0200
$T = 400$ vs. $T = 500$	0.1573	1.4142	0.0200
$T = 600$ vs. $T = 700$	0.1573	1.4142	0.0200
$T = 400$ vs. $T = 700$	0.1573	1.4142	0.0200

The Friedman statistic shows that using different number of iteration values makes significant difference to the MSGSAER because the Friedman statistic value χ^2_F is greater than the critical value χ^2_α, (18.6667 > 9.4880), thus presenting that the performance of the MSGSAER is affected by the choice of number of iteration. This result is further studied using Holm procedure as in Table 4.97. The results of Holm procedure demonstrate that significant differences exist between the performances of the MSGSA if comparing the results between these two number of iteration namely $T = 300$ vs. $T = 600$, $T = 300$ vs. $T = 700$, $T = 500$ vs. $T = 600$, $T = 300$ vs. $T = 4000$, $T = 500$ vs. $T = 700$, and $T = 400$ vs. $T = 600$.

4.6 The Performance Comparison between the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER for solving the ASP

With regard to the best result obtained using the best parameters, this section presents a comparison between the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER for the ASP.

Table 4.98

Best Results and their Associated Assembly Sequences of the Proposed Approaches based on the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER.

Approach based	**Total assembly time**	**Assembly sequence**
MSPSO	511.5	15-1-2-4-12-9-3-13-5-18-6-16-11-7-8-14-10-17-19
MSPSOER	508.3	1-2-12-4-3-9-13-15-11-5-16-6-18-7-8-14-10-17-19
MSGSA	508.3	1-2-4-3-9-12-13-5-16-15-18-11-6-7-8-14-10-17-19
MSGSAER	508.7	1-2-15-4-3-9-12-13-16-5-18-11-6-7-8-14-10-17-19

Table 4.98 shows that the MSPSOER and the MSGSA able to obtain the best result that represents minimum of total assembly time but with different assembly sequence, compared to two other approaches (the MSPSO and the MSGSAER).

4.7 SUMMARY

The purpose of this chapter is to summarize the experimental results, which are the outcomes of the research. The results reported in this chapter were obtained from several experiments on two categories of COPs; the TSP and the ASP. The chapter begin by discussing the experimental results of three algorithms for the TSP, namely the MSPSO, the MSPSOER, and the BPSO. Next, the experimental results of the three algorithms for the TSP, namely the MSGSA, the MSGSAER, and BGSA are analysed. In these two performance measures, the comparison focuses on the quality of results, speed of convergence, and the superiority of results on individual runs. The best solution, worst solution, average solution, and standard deviation for each algorithm on the small and bigger size of the TSP benchmark instances are recorded.

The difference between the solutions obtained by all algorithms is also analysed using boxplots. Wilcoxon's Sign-Ranked test is then performed to compare the performance between the MSPSO and the BPSO. Next, Friedman test is performed to investigate if there is significant improvement of the MSPSOER to the MSPSO and the BPSO. If significant differences are found, the MSPSO, the MSPSOER, and the BPSO are then compared each other using a post hoc procedure. In this study, the Holm procedure is chosen. This procedure can identify what are the pairs of the algorithms that have the significant differences. Later on, these two tests and the Holm procedure are also conducted to study the performance of the MSGSA, the MSGSAER, and BGSA. The experimental results obtained from the TSP benchmark instances used showed that the MSPSOER able to obtain better results, compared to the MSPSOER and the BPSO. Meanwhile, the BGSA have better results compared to the MSGSA and the MSGSAER.

Subsequently, the chapter discusses the experimental results of four approaches for an application in engineering problem (ASP), namely the proposed approach based on the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER and compared the performance of each algorithm to the SA, the GA, and the BPSO. BPSO. Next, the experimental results of three algorithms for the ASP, namely the MSGSA, the MSGSAER, and BGSA are analysed. The comparison focuses on the quality of results. The best solution, worst solution, average solution, and standard deviation for the ASP are recorded. The experimental results obtained shows that the proposed approach based on the MSPSO and the MSPSOER consistently outperforms the SA and the GA, but not the BPSO.

The success of any algorithm is heavily depend on setting of control parameters. Hence, a series of experiments are carried out to tune the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER best parameters for the ASP. Initially, the difference between the solutions obtained by all algorithms is analysed using boxplots. To study the performance of the proposed approaches based on the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER, Friedman test is performed. If significant differences are found, a post hoc procedure that is Holm procedure is conducted. This procedure can identify what are the pairs of the parameters that have the significant differences. With regard to the best parameters, the experimental results obtained shows that the proposed approaches based on the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER consistently outperforms the SA, the GA, and the BPSO. As a final conclusion, the proposed approaches based on the MSPSOER and the MSGSA able to obtain the best minimum of total assembly time but with different assembly sequence, compared to two other approaches (the MSPSO and the MSGSAER).

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Thesis Summary

In this thesis, the research are originated from two fundamental topics, which are treated in Chapter 2 and Chapter 3.

The first topic that is mentioned in Chapter 2 is the algorithms considered; particle swarm optimization and gravitational search algorithm. These two algorithms are optimization techniques which were developed from nature behaviour and are now effective alternative to more established methods, like those in the evolutionary computation field. Since the original PSO and GSA can only optimize problem in which the elements of the solution are continuous real numbers, a modification of the PSO and the GSA for problems with binary-valued solution elements were introduced. The two binary algorithms, BPSO and BGSA preserves the fundamental concept of their predecessor except that each particle (BPSO) and each agent (BGSA) in a swarm consists of binary string representing a particle's position vector (BPSO) and an agent's position vector (BGSA). This algorithm uses the concept of velocity as a probability that a bit flips to one or zero.

With regard to the literature review given in Chapter 2, it demonstrates that many previous studies have been proposed to solve discrete COPs that is naturally discrete either using the binary codification (i.e. Goldberg, 1989), integer codification (Tasgetiren, Evkli, Liang, & Gencyilmaz, 2004), or real-to-binary (Kennedy & Eberhart, 1997) or real-to-integer transformation (Wei & Hanning, 2007). There could be other approaches that search all possible solutions for solving discrete COPs in other type of search space rather than real-valued and binary search spaces, as provided by (Wei-Neng et al., 2010), where S-PSO makes use of set-based search space for searching all possible solutions. Two problem domains of COPs which are TSP and ASP were then provided and explained.

The second topic that is mentioned in Chapter 3 is related to the optimization of discrete COPs, and thus to the development of optimization algorithms which can solve discrete COPs without involving binary-valued solution elements. In particular, this research focused on the representation of multi-state model inspired from a sequential circuit in digital system. The multi-state concept was used in the context of evolutionary computation precisely with the intent to enable swarm intelligence algorithms to deal with discrete COPs. With regard to the existing of states transitions in the sequential circuit, a new approach of states representation called multi-state model is then introduced to optimize COPs. Up-to-date, the multi-state model is relatively a new type of discretization approach that involved transition between many pairs of two states. To begin, the original PSO and the original GSA were modified to multi-state PSO and multi-state GSA in which these two algorithms operated using the multi-state model. The characteristic of these two proposed algorithms is each solution's vector or dimension was represented as a collective of states; neither continuous nor discrete value. By using this multi-state model, next state for each current state can be selected randomly from the collective of states, depending on current velocity value. There are two primary features in the multi-state model; a current state and a radius. In this model, current state can be represented as a centroid of a circle. Meanwhile, velocity can be represented as radius of the circle.

Subsequently, the concept of multi-state model based on embedded rule was discussed. The concept was fundamentally introduced to solve the repetitive issue occurred in the MSPSO and the MSGSA that used the original multi-state model. The concept of embedded rule were implemented in the MSPSO and the MSGSA by designing such procedure to eventually produce unrepeated states in each solution. This procedure obeyed on an embedded rule which is "each state can only occur once in each solution" The introduction of the embedded rule in the multi-state model efficiently removed the limitation of the MSPSO and the MSGSA in which all updated solutions guaranteed forming by unrepeated states. Next, the chapter offers a description of the statistical tests to analyse either there are significant differences in term of performance between the algorithms in comparison. The chapter eventually elaborates the general process of this research step by step in research methodology.

Next, this thesis also presents the proposed approaches based on the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER applied to the TSP and the ASP with the outline of the proposed approaches which includes initialization, the evaluation, and the velocities and positions update. It is noticeable that there was an obvious difference between the proposed MSPSO and the MSGSA, and the proposed MSPSOER and the MSGSAER where the proposed MSPSO and the MSGSA should evolve their infeasible solutions to feasible solutions.

The four proposed algorithms was put to test on the TSP which were built in order to investigate the four proposed algorithms ability to produce results with higher quality. The results of the tests were then compared to their binary counterpart. The MSPSO and the MSPSOER were compared to the BPSO. On the other hand, the MSGSA and the MSGSAER were compared to the BGSA. The comparison focuses on the quality of results, speed of convergence, and the superiority of results on individual runs. The best solution, worst solution, average solution, and standard deviation for each algorithm on the eighteen sets of the TSP benchmark instances are recorded. The experimental results obtained from the TSP benchmark instances used showed that the MSPSOER able to obtain better results, compared to the MSPSO and the BPSO. However, the BGSA produced better results compared to the MSGSA and the MSGSAER.

Subsequently, this thesis discusses the experimental results of four approaches for an application in engineering called ASP, namely the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER and compared each algorithm to the SA, GA, and the BPSO. The experimental results obtained shows that the proposed approach based on the MSPSO and the MSPSOER consistently outperformed the SA and the GA, but not the BPSO. However, the usage of the best parameters improved experimental results where the proposed approaches based on the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER consistently outperformed the SA, the GA, and also the BPSO. The MSPSO, the MSPSOER, the MSGSA, and the MSGSAER that employed the representation of the multi-state can be eventually sighted in Figure 5.1.

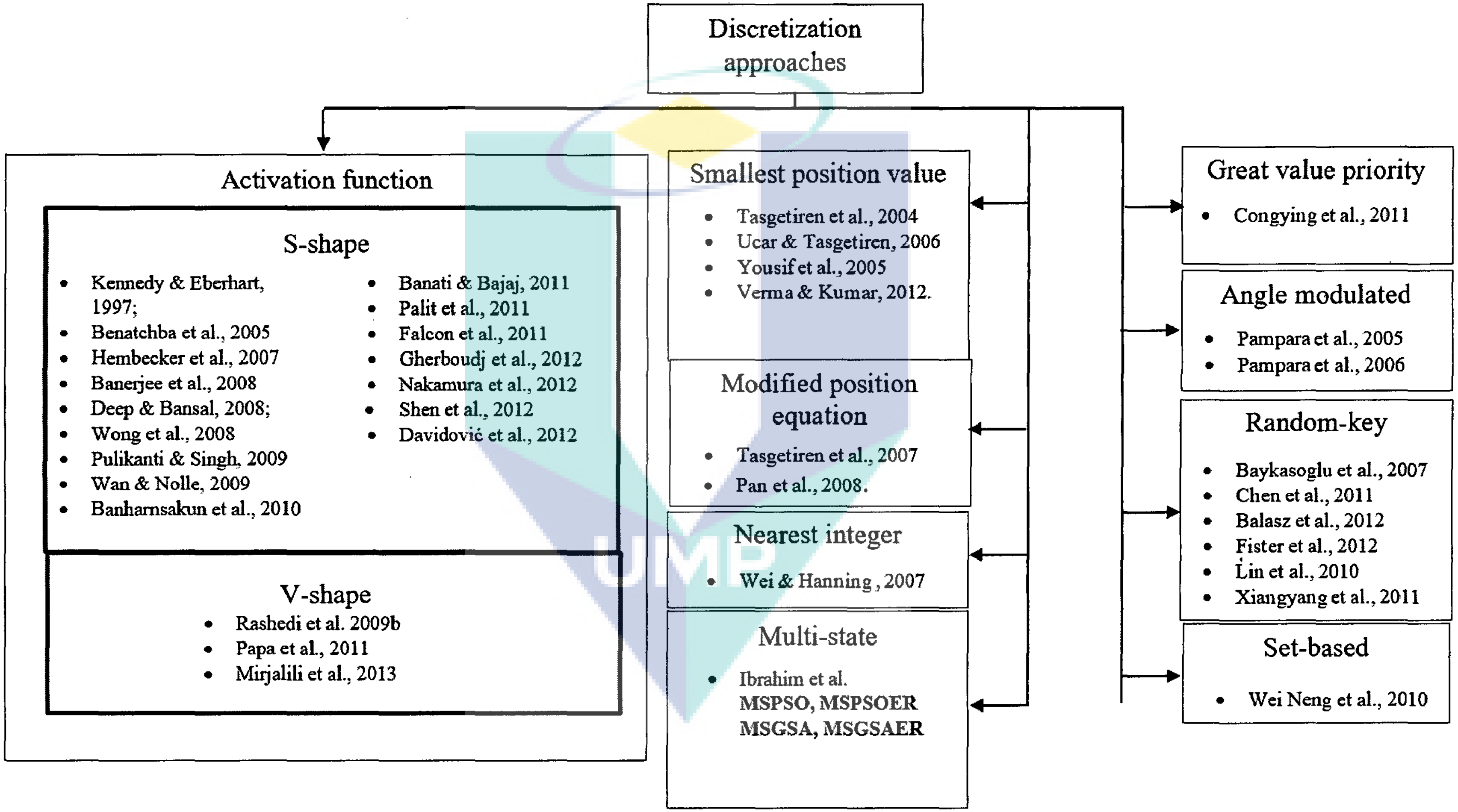


Figure 5.1. Discretization approaches with the proposed algorithms that used multi-state model. Bold signifies the proposed algorithms.

5.2 Recommendation for Future Research

In this thesis, four version of multi-state-based algorithms for solving the TSP and the ASP are proposed. The algorithms which are evaluated on the six sets of benchmark instances of TSP and the 19 components of ASP are successfully applied and showed their superiority performance with respect to the quality of results, compared to the BPSO and the BGSA. However, the promising performance that are marked on the four proposed algorithms certainly encourage the building of the motivation for further quest into the extension of the four multi-state-based algorithms as follows:

i. It is suggested that the concept of the multi-state representation and its improved concept can be applied to other meta-heuristics such as GA, DE, CS, and so forth in order to further investigate the effects of the multi-state representation on the quality of solutions.

ii. It is suggested that the MSPSO, the MSPSOER, the MSGSA, and the MSGSAER can be tested to other COPs, such as VLSI routing, DNA sequence design, PCB routing, and so forth in order to further investigate the effects of the multi-state representation on the quality of solutions on those problems.

iii. It is suggested that the multi-state model and its improved model with the embedded rule can be redesigned to solve multi-objective problems.

REFERENCES

Afshinmanesh, F., Marandi, A. and Rahimi-Kian, A. (2005). A novel binary particle swarm optimization method using artificial immune system. *Proceedings of the International Conference on Computer as a Tool*, *1*, 217–220.

Balaprakash, P., Birattari, M., Stützle, T. and Dorigo, M. (2010). Estimation-based metaheuristics for the probabilistic travelling salesman problem. *Computers and Operations Research*, *37*(11), 1939–1951.

Balasz, K., Horvath, Z. and Koczy, L. (2012). Hybrid bacterial iterated greedy heuristics for the permutation flow shop problem. *Proceedings of the IEEE Congress on Evolutionary Computation, 1–8.*

Banati, H. and Bajaj, M. (2011). Fire fly based feature selection approach. *International Journal of Computer Science*, *8*(4), 473–480.

Banerjee, S., Dangayach, G., Mukherjee, S. and Mohanti, P. (2008). *Metaheuristics for scheduling in industrial and manufacturing applications*. Germany: Springer Berlin.

Banharnsakun, A., Achalakul, T. and Sirinaovakul, B. (2010). A hybrid method for solving Travelling salesman problem. *Proceedings of the 2nd Congress on Nature and Biologically Inspired Computing*, 394–399.

Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P. and Vance, P.H. (1998). Branch-and-price: column generation for solving huge integer programs. *Operations Research*, *46*(3), 316–329.

Basalo, P.M.R., Laguna, I.R. and Rubio Díez, F. (2007). Using river formation dynamics to design heuristic algorithms. *Unconventional Computation*, 163–177.

Baykasoglu, A., Ozbakir, L. and Tapkan, P. (2007). *Swarm intelligence: Focus on ant and particle swarm optimization*. Istanbul, Turkey: I-tech Eduucation and Publishing.

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, *38*(8),716–719.

Benatchba, K., Admane, L. and Koudil, M.(2005). Using bees to solve a data-mining problem expressed as a max-sat one. *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, *3562*,75–86.

Bentley, J. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, *4*(4), 387–411.

Blum, C. and Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, *35*(3),268–308.

Bonneville, F., Perrard, C. and Henrioud, J.M. (1995). A genetic algorithm to generate and evaluate assembly plans. *Proceedings of the INRIA/IEEE Symposium on Emerging Technologies and Factory Automation,2*, 231–239.

Boros, E. and Hammer, P.L. (2003). *Discrete optimization: the state of the art*. Amsterdam: Elsevier Science.

Branke, J. (2002). *Evolutionary optimization in dynamic environments*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Burnwal, S. and Deb, S. (2012). Scheduling optimization of flexible manufacturing system using cuckoo search-based approach. *International Journal of Advanced Manufacturing Technology*, 1–9.

Chakrabarty, S. and Wolter, J. (1997). A structure-oriented approach to assembly sequence planning. *IEEE Transactions on Robotics and Automation*, *13*(1), 14–29.

Chen, H., Li, S. and Tang, Z. (2011). Hybrid gravitational search algorithm with random-key encoding scheme combined with simulated annealing. *International Journal of Computer Sciences, Network and Security, 11*(6), 208–217.

Chen, W.-C., Tai, P.-H., Deng, W.-J. and Hsieh, L.-F. (2008). A three-stage integrated approach for assembly sequence planning using neural networks. *Expert Systems with Applications*, *34*(3), 1777–1786.

Chiang, A.C. (1992). *Elements of dynamic optimization*. Prospect Height, IL: Waveland Press.

Choi, Y.-K., Lee, D.M. and Cho, Y. Bin. (2008). An approach to multi-criteria assembly sequence planning using genetic algorithms. *The International Journal of Advanced Manufacturing Technology*, *42*(1-2), 180–188.

Christofides, N. (1973). The optimum traversal of a graph. *Omega*, *1*, 719–732.

Chuang, L., Hsiao, C. and Yang, C. (2011). An improved binary particle swarm optimization with complementary distribution strategy for feature selection. *International Association of Computer Science & Information Technology*, *3*, 244–248.

Chuang, L.-Y., Chang, H.-W., Tu, C.-J. and Yang, C.-H. (2008). Improved binary PSO for feature selection using gene expression data. *Computational Biology and Chemistry*, *32*(1), 29–37.

Chuang, L.-Y., Tsai, S.-W. and Yang, C.-H. (2011). Improved binary particle swarm optimization using catfish effect for feature selection. *Expert Systems with Applications*, *38*(10), 12699–12707.

Cirasella, J., Johnson, D.S., Mcgeoch, L.A. and Zhang, W. (2001). The asymmetric travelling salesman problem : algorithms,instance generators , and tests.

Proceedings of the Third International Workshop on Algorithm Engineering and Experimentation, *2153*, 32–59.

Congram, R.K., Potts, C. N. and van de Velde, S.L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, *14*(1), 52–67.

Congying, L., Huanping, Z. and Xinfeng, Y. (2011). Particle swarm optimization for quadratic assignment problem. *Proceeding of the International Conference on Computer Science andNetwork Technology*, *3*, 1728–1731.

Chapleau, L., Ferland, J.A., Lapalme, G. and Rousseau, J.M. (1984). A parallel insert method for the capacitated arc routing problem. *Operations Research Letters*, *3*, 95-99.

Davidović, T., Šelmić, M., Teodorović, D. and Ramljak, D. (2012). Bee colony optimization for scheduling independent tasks to identical processors. *Journal of Heuristics*,*18*, 549–569.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. Chichester, UK: John Wiley & Sons.

Deep, K. and Bansal, J.C. (2008). A socio-cognitive particle swarm optimization for multi-dimensional knapsack problem. *Proceedings of the 1st International Conference on Emerging Trends in Engineering and Technology,* 355–360.

Dieterich, J.M. (2012). Empirical review of standard benchmark functions using evolutionary global optimization. *Applied Mathematics*, *3*(30), 1552–1564.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *26*(1), 29–41.

Doyen, L., Thomas, A.H., Legay, A. and Dejan, N. (2010) Robustness of Sequential Circuits. *Proceedings of the 10th International Conference on Application of Concurrency to System Design,* pp. 77–84.

Du, D.-Z. and Pardalos, P.M. (2005). *Handbook of Combinatorial Optimization: Supplementary Volume B*. Boston,: Springer Science and Business Media, Inc.

Ehrgott, M. and Gandibleux, X. (2002). *Multiple criteria optimization: state of the art annotated bibliographic surveys*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Evan, J.R. and Minieka, E. (1992). *Optimization Algorithms for Networks and Graphs.* New York: Marcel Dekker.

Falcon, R., Almeida, M. and Nayak, A. (2011). Fault identification with binary adaptive fireflies in parallel and distributed systems. *Proceedings of the IEEE Congress of Evolutionary Computation,* pp. 1359–1366.

Feng, X., Lau, F.C.M. and Yu, H. (2013). A novel bio-inspired approach based on the behavior of mosquitoes. *Information Sciences*, *233*, 87–108.

Feo, T.A. and Resende, M.G.C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, *6*(2), 109–133.

Fister, I., Yang, X.S., Brest, J. and Fister, I.J. (2012). Memetic firefly algorithm for combinatorial optimization. *Proceedings of the 5th International Conference on Bioinspired Optimization Methods and their Applications*, pp. 75–86.

Fleetwood, K. (1999). *New ideas in optimization*. UK: McGraw-Hill Ltd.

Floudas, C.A. (1995). *Nonlinear and mixed-integer optimization*. New York: Oxford University Press.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, *32*(200), 675–701.

Gao, L., Qian, W., Li, X. and Wang, J. (2009). Application of memetic algorithm in assembly sequence planning. *The International Journal of Advanced Manufacturing Technology*, *49*(9-12), 1175–1184.

Garey, M.R. and Johnson, D.S. (1979). *Computers and intractability: a guide to the theory of np-completeness*, W. H. Freeman.

Garrod, W.E. (1989). *A.S.A.P., Automated sequential assembly planner*. Texas A & M University.

Gerstman, B.B. (2006). Probability table (online). Retrived from http://www.sjsu.edu/faculty/gerstman/StatPrimer/z-two-tails.pdf. on 15 December 2015.

Gherboudj, A., Layeb, A. and Chikhi, S. (2012). Solving 0-1 knapsack problems by a discrete binary version of cuckoo search algorithm. *International Journal of Bio-Inspired Computation*, *4*(4), 229–236.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533–549.

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.

Golden, B.L. and Wong, R.T. (1981). Capacitated arc routing problems. *Networks*, *11*, 305–315.

Golden, B.L., DeArmon, J.S. and Baker, E.K. (1983). Computatiomal experiments with algorithms for a class of routing problem. *Computers & Operations Research*, *10*, 47–59.

Guo, Y.W., Li, W.D., Mileham, A.R. and Owen, G.W. (2009). Applications of particle swarm optimisation in integrated process planning and scheduling. *Robotics and Computer-Integrated Manufacturing*, *25*(2), 280–288.

Hansen, P. and Mladenovic, N. (2001). *Developments of variable neighborhood search, in essays and surveys in metaheuristics*. Dordrecht: Kluwer Academic Publishers.

Hansen, P. and Mladenovic, N. (2002). *Variable neighborhood search in handbook of applied optimization*. New York: Oxford University Press.

Hart, P.E., Nilsson, N.J. and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2),100–107.

He, Y. and Hui, C.-W. (2010). A binary coding genetic algorithm for multi-purpose process scheduling: a case study. *Chemical Engineering Science*, *65*(16), 4816–4828.

Hembecker, F., Lopes, H.S. and Godoy, J. (2007). Particle swarm optimization for the multi-dimensional knapsack problem. *Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms*, 358–365.

Hernández, H. and Blum, C. (2012). Distributed graph coloring: an approach based on the calling behavior of japanese tree frogs. *Swarm Intelligence*, *6*,117–150.

Holland, J.H. (1962). Outline for a logical theory of adaptive systems. *Journal of the ACM*, *9*(3), 297–314.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, *6*, 65–70.

Hong, D.S. and Cho, H.S. (1995). A neural-network-based computational scheme for generating optimized robotic assembly sequences. *Engineering Applications of Artificial Intelligence*, *8*(2),129–145.

Horst, R. and Pardalos, P.M. (1995). *Handbook of global optimization*. The Netherlands: Kluwer Academic Publishers.

Horst, R. and Tuy, J. (2003). *Global optimization: deterministic approaches*. Berlin: Springer-Verlag.

Hosseini, H.S. (2009). Intelligent water drops algorithm: a new optimization method for solving the multiple knapsack problem. *International Journal of Intelligent Computing and Cybernetics*, *1*(2), 193–212.

Hsin-Hao, H., Wang, M.H. and Johnson, M.R. (2000). Disassembly sequence generation using a neural network approach. *Journal of Manufacturing Systems*, *19*(2), 73–82.

Hui, C., Yuan, L. and Kai-fu, Z. (2008). Efficient method of assembly sequence planning based on GAAA and optimizing by assembly path feedback for complex product. *The International Journal of Advanced Manufacturing Technology*, *42*(11-12),1187–1204.

Jeong, Y.W., Park, J.B., Jang, S.H. and Lee, K.Y. (2010). A new quantum-inspired binary PSO: Application to unit commitment problems for power systems. *IEEE Transactions on Power Systems*, *25*(3), 1486–1495.

Johnson, D.S. and McGeoch, L.A. (2002). *The traveling salesman problem and its variation*. Boston: Kluwer Academic Publishers.

Kalashnikov, A. V. and Kostenko, V.A. (2008). A parallel algorithm of simulated annealing for multiprocessor scheduling. *Journal of Computer and Systems Sciences International*, *47*(3):455–463.

Karaboga, D. 2005. *An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University*.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *Proceedings of the IEEE International Conference on Neural Networks 1995*, *4*, 1942–1948.

Kennedy, J. and Eberhart, R.C. (1997). A discrete binary version of the particle swarm algorithm. *IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation 1997*,*5*, 4104–4108.

Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671–680.

Kong, M. and Tian, P. (2005). *Computational Intelligence and Security*. Springer Berlin Heidelberg.

Krause, J., Cordeiro, J., Parpinelli, R.S and Lopes, H.S. (2013). *Swarm Intelligence and Bio-Inspired Computation*. Elsevier.

Land, A.H. and Doig, A.G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, *28*(3), 497–520.

Lawler, E., Lenstra, J.K., Rinnooy, A.H.G. and Shmoys, D.B. (1985). *The Travelling salesman problem: a guided tour of combinatorial optimization*. Wiley.

Lee, S. and Shin, Y.G. (1990). Assembly planning based on geometric reasoning. *Computers & Graphics*, *14*(2), 237–250.

Leiserson, C.C.E., Rivest, R.R.L., Stein, C. and Cormen, T.H. (2009). *Introduction to algorithms. 3rd ed.* The MIT Press.

Li, M., Wu, B., Hu, Y., Jin, C. and Shi, T. (2013). A hybrid assembly sequence planning approach based on discrete particle swarm optimization and evolutionary direction operation. *The International Journal of Advanced Manufacturing Technology*, *68*(1-4), 617–630.

Lin, T.-L., Horng, S.-J., Kao, T.-W., Chen, Y.-H., Run, R.-S., Chen, R.-J., Lai, J.-L. and Kuo, I.-H. (2010). An efficient job-shop scheduling algorithm based on particle swarm optimization. *Expert Systems with Applications*, *37*, 2629–2636.

Lit, P. De, Latinne, P., Rekiek, B. and Delchambre, A. (2010). Assembly planning with an ordering genetic algorithm. *International Journal of Production Research, 39*(16), 3623–3640.

Lu, C., Wong, Y.S. and Fuh, J.Y.H. (2006). An enhanced assembly planning approach using a multi-objective genetic algorithm. *Journal of Engineering Manufacture*, *220*(2), 255–272.

Luenberger, D.G. (1989). *Linear and nonlinear programming*. Reading, MA: Addison-Wesley.

Mann, P. S. (2007). *Introductory statistics*. John Wiley & Sons.

Mano, M. M. and Ciletti, M. D. (2012). *Digital design, 5th edition*. Prentice Hall.

Marian, R.M., Luong, L.H.S. and Abhary, K. (2003). Assembly sequence planning and optimisation using genetic algorithms: Part I. Automatic generation of feasible assembly sequences. *Applied Soft Computing*, *2*(3), 223–253.

Mello, L.S.H. de and Sanderson, A.C. (1990). AND/OR graph representation of assembly plans. *IEEE Transactions on Robotics and Automation*, *6*(2),188–199.

Menhas, M.I., Wang, L., Pan, H. and Fei, M. (2010). *Life system modeling and intelligent computing*. Springer Berlin Heidelberg.

Menhas, M.I., Fei, M., Wang, L. and Fu, X. (2011). *Advances in swarm intelligence*. Springer Berlin Heidelberg.

Milner, J.M., Graves, S.C. and Whitney, D.E. (1994). Using simulated annealing to select least-cost assembly sequences. *Proceedings of the IEEE International Conference on Robotics and Automation*, *3,* 2058–3063.

Mirjalili, S., Mirjalili, S.M. and Yang, X.S. (2013). Binary bat algorithm. *Neural Computing and Applications*, 1–19.

Mitchell, J.E. (2002). *Handbook of Applied Optimization*. Oxford University Press.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100.

Modiri, A. and Kiasaleh, K. (2011). Modification of real-number and binary PSO algorithms for accelerated convergence. *IEEE Transactions on Antennas and Propagation*, *59*(1), 214–224.

Moore, K.E., Güngör, A. and Gupta, S.M. (2001). Petri net approach to disassembly process planning for products with complex AND/OR precedence relationships. *European Journal of Operational Research*, *135*(2), 428–449.

Motavalli, S. and Islam, A.-U. (1997). Multi-criteria assembly sequencing. *Computers & Industrial Engineering*, *32*(4), 743–751.

Mukred, J.A.A., Ibrahim, Z., Ibrahim, I., Adam, A., Wan, K., Yusof, Z.M. and Mokhtar, N. (2012). A binary particle swarm optimization approach to optimize assembly sequence planning. *Advanced Science Letters*, *13*, 732–738.

Nakamura, R.Y.M., Pereira, L.A.M., Rodrigues, D., Costa, K.A.P., Papa, J.P. and Yang, X.S. (2012). A binary bat algorithm for feature selection. *Proceedings of the conference on Graphics, Patterns and Images*, pp. 291–297.

Nocedal, J. and Wright, S.J. (1999). *Numerical Optimization*. New York: Springer.

Nozarian, S., Soltanpoor, H. and Vafaei, M. (2012). A binary model on the basis of cuckoo search algorithm in order to solve the problem of knapsack 1-0. *Proceedings of the third International Conference on Services in Emerging Markets 2012*, *34*, 67–71.

Osman, I. and Kelly, J. (1996). *Meta-heuristics: theory & applications*. Springer.

Palit, S., Sinha, S.N., Molla, M.A., Khanra, A. and Kule, M. (2011). A cryptanalytic attack on the knapsack cryptosystem using binary firefly algorithm. *Proceedings of the 2nd International Conference on Computer and Communication Technology*, 428–432.

Pampara, G., Franken, N. and Engelbrecht, A.P. (2005). Combining particle swarm optimisation with angle modulation to solve binary problems. *Proceedings of the IEEE Congress on Evolutionary Computation 2005, 1*, 89–96.

Pampara, G., Engelbrecht, A.P. and Franken, N. (2006). Binary Differential Evolution. *Proceedings of the IEEE International Conference on Evolutionary Computation 2006*, *1*, 1873–1879.

Pan, Q.-K., Fatih Tasgetiren, M. and Liang, Y.-C. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research*, *35*(9), 2807–2839.

Papa, J.P., Pagnin, A., Schellini, S.A., Spadotto, A., Guido, R.C., Ponti, M., Chiachia, G. and Falcão, A.X. (2011). Feature selection through gravitational search algorithm. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2052–2055.

Papadimitriou, C. and Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Dover.

Parsopoulos, K.E. and Vrahatis, M.N. (2010). *Particle swarm optimization and intelligence: advances and applications*. Information Science Publishing.

Pearn, W.L. (1989). Approximate solutions for the capacitated arc routing problem, *Computers & Operations Research*, *16*, 589–600.

Polak, E. (1997). *Optimization: algorithms and consistent approximations*. New York: Springer.

Porto, S.C.S. and Ribeiro, C.C. (1994). A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *International Journal of High-Speed Computing, 7*, 45-71.

Prins, C., Labadi, N. and Reghioui, M. (2009). Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, *47*, 507–536.

Pulikanti, S. and Singh, A. (2009). An artificial bee colony algorithm for the quadratic knapsack problem. *Proceedings of the 16th International Conference on Neural Information Processing*, pp. 196–205.

Qu, S., Jiang, Z. and Tao, N. (2013). An integrated method for block assembly sequence planning in shipbuilding. *The International Journal of Advanced Manufacturing Technology*, *69*(5-8),1123–1135.Yu

Rashedi, E., Nezamabadi-pour, H. and Saryazdi, S. (2009a.) GSA: a gravitational search algorithm. *Information Sciences*, *179*(13), 2232–2248.

Rashedi, E., Nezamabadi-pour, H. and Saryazdi, S. (2009b.) BGSA: binary gravitational search algorithm. *Natural Computing*, *9*(3), 727–745.

Reinelt, G. (1991). TSPLIB--A Travelling Salesman Problem Library. *INFORMS Journal on Computing*, *3*(4), 376–384.

Reiter, S. and Rice, D.B. (1966). Discrete optimizing solution procedures for linear and nonlinear integer programming problems. *Management Science*, *12*, 829–850.

Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M.(2009). *Fundamental problems in computing: essays in honor of professor daniel j. rosenkrantz*. Netherlands: Springer.

Rossi, F., Van Beek, P. and Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier.

Rothman, S. (2012). *Learning statistics with baseball*. Maryland: Johns Hopkins University Press.

Salzberg, S. I. (1997). *Data mining and knowledge discovery*. Boston: Kluwer Academic Publishers.

Santos, L., Coutinho-Rodrigues, J. and Current, J.R. (2009). An improved heuristic for the capacitated arc routing problem. *Computers & Operations Research*, *36*, 2632–2637.

Shen, Q., Jiang, J.-H., Jiao, C.-X., Shen, G.-L. and Yu, R.-Q. (2004). Modified particle swarm optimization algorithm for variable selection in MLR and PLS modeling: QSAR studies of antagonism of angiotensin II antagonists. *European Journal of Pharmaceutical Sciences : Official Journal . of the European Federation for Pharmaceutical Sciences*, *22*(2-3), 145–52.

Shen, X., Li, Y., Chen, C., Yang, J. and Zhang, D. (2012). Greedy continuous particle swarm optimization for the knapsack problems. *International Journal of Computation and Application Technology*, *2*(44), 137–144.

Sheskin, D.(2011). *Handbook of parametric and nonparametric statistical procedures*. Boca Raton, FL: Chapman & Hall/CRC.

Sorensen, K. and GLover, F. (2013). *Encyclopedia of Operations Research and Management Science*. New York: Springer.

Talbi, E.G. (2009). *Metaheuristics: from design to implementation*. John Wiley and Sons.

Tasgetiren, M., Suganthan, P. and Pan, Q. (2007). A discrete particle swarm optimization algorithm for the generalized Travelling salesman problem. *Proceedings of the 9th annual conference on Genetic and evolutionary computation. London, England,* 158-167

Tasgetiren, M.F., Evkli, M., Liang, Y.-C. and Gencyilmaz, G. (2004). Particle swarm optimization algorithm for single machine total weighted tardiness problem. In *Congress on Evolutionary Computation,* 1412–1419.

Torn, A. and Žilinskas, A. (1989). *Global optimization.* Berlin: Springer.

Tseng, Y.-J., Chen, J.-Y. and Huang, F.-Y. (2009). A multi-plant assembly sequence planning model with integrated assembly sequence planning and plant assignment using GA. *The International Journal of Advanced Manufacturing Technology*, *48*(1-4), 333–345.

Tseng, Y.-J., Yu, F.-Y. and Huang, F.-Y. (2011). A green assembly sequence planning model with a closed-loop assembly and disassembly sequence planning using a particle swarm optimization method. *International Journal of Advanced Manufacturing Technology*, *57*(9-12), 1183–1197.

Tumeo, A., Pilato, C., Ferrandi, F., Sciuto, D. and Lanzi, P.L. (2008). Ant colony optimization for mapping and scheduling in heterogeneous multiprocessor systems. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation 2008*, pp. 142–149.

Ucar, H. and Tasgetiren, M.F. (2006). A particle swarm optimization for permutation flow shop sequencing problem with the number of tardy jobs criterion.

Proceedings of the 5th International Symposium on Intelligent Manufacturing Systems, pp. 237–250.

Ulusoy, G. (1985). The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, *22*, 329–337.

Upton, G. and Cook, I. (2002). *Oxford dictionary of statistics*. UK: Oxford University Press.

Verma, R.S. and Kumar, S. (2012). DSAPSO: DNA sequence assembly using continuous particle swarm optimization with smallest position value rule. *Proceedings of the 1st International Conference on Recent Advances in Information Technology,* pp. 410–415.

Voudouris, C. and Tsang, E. (1999). Guided local search and its application to the Travelling salesman problem. *European Journal of Operational Research*. *113*(2), 469–499.

Voudouris, C., Tsang, E.P.K. and Alsheddy, A. (2010). *Handbook of metaheuristics*. US: Springer.

Wan, N.F. and Nolle, L. (2009). Solving a multi-dimensional knapsack problem using hybrid particle swarm optimization algorithm. *Proceedings of the 23rd European Conference on Modelling and Simulation*.

Wang, L. and Yu, J.S. (2005). *Advances in Natural Computation*. Springer Berlin Heidelberg.

Wang, J.F., Liu, J.H. and Zhong, Y.F. (2004). A novel ant colony algorithm for assembly sequence planning. *The International Journal of Advanced Manufacturing Technology*, *25*(11-12), 1137–1143.

Wang, L., Xu, Y., Mao, Y. and Fei, M. (2010). *Life system modeling and intelligent computing*: Springer Berlin Heidelberg.

Watkins, P.R. (1990). Integer and combinatorial optimization. *Journal of the Operational Research Society*, *41*(2), 177–178.

Wei, L. and Hanning, C. (2012). BABC: A binary version of artificial bee colony algorithm for discrete optimization. *International Journal of Advancements in Computing Technology*, *4*(14), 307–314.

Wei-Neng, C., Jun, Z., Chung, H.S.H., Wen-Liang, Z., Wei-Gang, W. and Yu-hui, S. (2010). A novel set-based particle swarm optimization method for discrete optimization problems. *Evolutionary Computation, IEEE Transactions, 14*(2), 278–300.

Wen, Y., Xu, H. and Yang, J. (2011). A heuristic-based hybrid genetic-variable neighborhood search algorithm for taskscheduling in heterogeneous multiprocessor system. *Information Sciences*, *181*(3), 567–581.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, *1*, 80–83.

Wohlk, S. (2005). *Contributions to arc routing*. Ph.D. Dissertation. University of Southern Denmark, Denmark.

Wong, L.-P., Low, M.Y.H. and Chong, C.S. (2008). A bee colony optimization algorithm for Travelling salesman problem. *Proceedings* of *The 2nd Asia International Conference on Modelling & Simulation,* pp 818–823.

Xiangyang, S., Minghao, Z. and Yamin, Z. (2011). Improved artificial fish swarm algorithm to solve knapsack problem. *Computation, Engineerings and Applications*, *41*(21), 43.

Yang, X.S. (2009). Harmony search as a metaheuristic algorithm. *Studies in Computational Intelligence*, *191*,1–14.

Yang, X.-S. (2013). Bat algorithm: literature review and applications. *International Journal of Bio-Inspired Computation*, *5*(3),10.

Yang, X.S. and Deb, S. (2009). Cuckoo search via Lévy flights. *Proceedings of the World Congress on Nature and Biologically Inspired Computing 2009*, pp. 210–214.

Yousif, A., Abdullah, A.H., Nor, S.M. and Abdelaziz, A.A. (2011). Scheduling jobs on grid computing using firefly algorithm. *Journal of Information Technology Theory and Application*, *33*(2), 155–164.

Yu, H. (2008). Optimizing task schedules using an artificial immune system approach. *Proceedings of the Genetic And Evolutionary Computation Conference*, 151–158.

Yuan, L.Y.L. and Zhao, Z.-D.Z.Z.-D. (2007). A Modified Binary Particle Swarm Optimization Algorithm for Permutation Flow Shop Problem. *Proceedings of the International Conference on Machine Learning and Cybernetics 2007*, *2*, 902–907.

Zar, J. (2009). *Biostatistical analysis*. Upper Saddle River, NJ: Prentice Hall.

Zha, X.F. (2000). An object-oriented knowledge based Petri net approach to intelligent integration of design and assembly planning. *Artificial Intelligence in Engineering*, *14*(1), 83–112.

Zhang, W. (1989). Representation of assembly and automatic robot planning by petri net. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 19(2), 418–422.

Zhigljavsky, A. and Zhilinskas, A. (2002). *Stochastic and Global Optimization*. New York: Springer.

Zhou, W., Zheng, J., Yan, J. and Wang, J. (2010). A novel hybrid algorithm for assembly sequence planning combining bacterial chemotaxis with genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, *52*(5-8), 715–724.

APPENDICES

List of Publications

Published Journal Papers

1. Ibrahim, I., Ibrahim, Z., Ahmad, H. and Yusof, Z.M. (2016). An improved multi-state particle swarm optimization for discrete combinatorial optimization problems. *International Journal of SIMULATION: Systems, Science & Technology (IJSSST)*, *15*(14), 1–8.

2. Ibrahim, I., Ibrahim, Z., Ahmad, H. and Yusof, Z.M. (2016). A novel multi-state gravitational search algorithm for discrete optimization problems. *International Journal of SIMULATION: Systems, Science & Technology (IJSSST)*, *15*(15), 1–8.

3. Ibrahim, I., Ibrahim, Z., Ahmad, H. and Yusof, Z.M. (2015). An assembly sequence planning approach with a multi-state gravitational search algorithm, *ARPN Journal of Engineering and Applied Sciences*, *10*(22), 2015.

4. Ibrahim, I., Ibrahim, Z., Ahmad, H., Yusof, Z.M., Nawawi, S.W. and Mubin. M. (2015). An assembly sequence planning approach with a rule-based multi-state gravitational search algorithm. *International Journal of Advanced Manufacturing Technology (IJAMT)*, *79*(5), 2015.

5. Ibrahim, I., Ibrahim, Z., Ahmad, H., Yusof, Z.M., Nawawi, S.W., Mohamad, M.S. and Mokhtar, N. (2014). Multi-state particle swarm optimization primitive for discrete optimization problem. *International Journal of SIMULATION: Systems, Science & Technology (IJSSST)*, *15*(1), 15–25.

Published Conference Papers

1. Ibrahim, I., Ibrahim, Z., Ahmad, H. and Yusof, Z.M. (2016). An assembly sequence planning approach with a multi-state particle swarm algorithm. *Proceedings of the 29th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE 2016).*

2. Ibrahim, I., Ibrahim, Z., Ahmad, H. and Yusof, Z.M. (2015). A multi-state gravitational search algorithm for combinatorial optimization problems. *Proceedings of the 7th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN 2015)*, pp. 9–14.

3. Ibrahim, I., Ibrahim, Z., Ahmad, H. and Yusof, Z.M. (2015). An improved multi-state particle swarm optimization for discrete optimization problems. *Proceedings of the 7th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN 2015)*, pp. 3–8.

4. Ibrahim, I., Ibrahim, Z., Ahmad, H. and Yusof, Z.M. (2015). Rule-based multi-state gravitational search algorithm for discrete optimization problems. *Proceedings of the 4th International Conference on Software Engineering and Computer Systems (ICSECS 2015)*, pp. 142–147.

5. Ibrahim, I.; Ibrahim, Z., Ahmad, H. and Yusof, Z.M. (2015). An assembly sequence planning approach with a multi-state gravitational search algorithm. *Proceedings of the International Conference on Software Engineering and Computer Systems (InECCE 2015).*

6. Ibrahim, I., Ibrahim, Z., Ahmad, H. and Yusof, Z.M. (2014). An assembly sequence planning approach with binary gravitational search algorithm. *Proceedings of the 13th International Conference on Intelligent Software Methodologies, Tools, and Techniques (SOMET 2014)*, pp.179–193.

7. Ibrahim, I., Ibrahim, Z., Ahmad, H., Yusof, Z.M., Nawawi, S.W., Mohamad, M.S. and Mubin. M. (2014). Gravitational search algorithm for assembly sequence planning approach. *Proceedings of the Colloquium on Robotics, Unmanned Systems and Cybernetics (CRUSC 2014)*, pp.32–36.

8. Ibrahim, I., Yusof, Z.M., Nawawi, S.W., Rahim, M.A.A., Khalil, K., Ahmad, H. and Ibrahim, Z. (2012). A novel multi-state particle swarm optimization for discrete combinatorial optimization problems. *Proceedings of the Fourth International Conference on Computational Intelligence, Modelling and Simulation (CIMSiM 2012)*, pp.18–23.