INVESTIGATION INTO GA AND GSOA OPTIMISATION APPROACHES FOR SOLVING ASSEMBLY SEQUENCE PROBLEMS

By

FAWAZ SAAD T. ALHARBI

The thesis submitted for the degree of Doctor of Philosophy

To

School of Mechanical and Design Engineering

Faculty of Technology

University of Portsmouth

United Kingdom

July 2020

Table of Contents

DECLARATION

I hereby declare that this Ph.D. thesis entitled "Investigation into GA and GSOA Optimisation Approaches for Solving Assembly Sequence Problems" was carried out by me for the degree of Doctor of Philosophy. I also declare that, to the best of my knowledge, my thesis does not contain any materials previously published or written by other people except where due reference is made in the text.

Name: Fawaz Saad Alharbi

Signature: *FAWAZSAAD*

Date: 03/07/2020

ACKNOWLEDGMENT

First, I praise God, the almighty, merciful and passionate, for providing me this opportunity and granting me the capability to proceed successfully. In addition, may peace and salutation be given to the prophet Muhammad (peace be upon him) who taken all human being from the darkness to the lightness.

It would not have been possible to write this doctoral thesis without the support and help of the kind people around me.

Above all, I am grateful to my father, mother, brothers, sisters and wife, for their moral and emotional support in my life, also for their prayers, caring and sacrifices. I am also grateful to all my friends, precisely to Khalaf Mones Alharbi, Ibrahim Mohammed Aloyayna and Dr. Ziad Hunaiti who have supported me along the way. I am indebted to them for their support.

I would like to thank the Saudi Government and the Ministry of Higher Education, for their financial support under a scholarship, not only for providing the funding which allowed me to do this degree, but also for giving me the opportunity to attend conferences and meet so many important people in the research field.

I would like to express my sincere gratitude to my supervisors Dr. Qian Wang, Prof. Nick Bennett and Prof. David Sanders for the continuous support of my PhD study and related research for their patience, motivation and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisors and mentors for my PhD study.

Beside my advisors, I would like to express my deepest thanks and appreciation to the rest of my thesis examining committee: Prof. Ahmed Al-Ashaab and Dr. Andrea Bucchi, for their encouragement, insightful comments and hard questions.

I would like to take this opportunity to immense gratitude to Prof. Hom Dhakal, Dr. Jovana Radulovic and Dr. Sarinova Simandjuntak (School of Mechanical and Design Engineering), also to Linda Janes (Student and Academic Administrator) and to all persons who have given their invaluable support and recommendations.

Last but not least, I would like to express my sincere appreciation to Miss Valentina Vuntsova and Mrs. Donna Crighton from the research degrees and for their prompt reply, and for their valuable assistance and guidance.

LIST OF PUBLICATIONS & DISSEDMINATIONS

JOURNAL

- Alhrbi. F. S., Nujoom. R. and Wang. Q. (2020). A hybrid optimisation approach for energy-saving assembly systems. (Under review in Journal of Production Economics).
- Alharbi. F. S., Wang, Q., Sanders. D. and Bennett. N. (2019). A case study of using the genetic algorithm and the glowworm swarm algorithm for solving an assembly sequence optimisation problem, Robotics and Computer-Integrated Manufacturing, under Major review.
- Alharbi, F. S. and Wang, Q. (2017). A genetic algorithm for solving assembly sequence problem, Applied Mechanics and Materials, Vol. 872, pp. 420-424, published.

CONFERENCE PAPER

- Alharbi, F. S. and Wang, Q. (2018). Solving an assembly sequence optimisation problem using the genetic algorithm, Proc. of the IEEE 18th International Conference on Electronics, Control, Optimisation and Computer Science (ICECOCS), published.
- Alharbi, F. S. and Wang, Q. (2018) Applying a glowworm swarm algorithm for optimising the assembly sequence of a car engine pump valve, Proc. of the IEEE 18th International Conference on Electronics, Control, Optimisation and Computer Science (ICECOCS), published.

BOOK CHAPTER

- Alharbi F. S. (2017). An algorithm to optimise the assembly sequence of a product, Journal of computing in systems and engineering, Vol 18, pp: 399-404. ISSN 1472-9083, published.

CONFERENCE POSTER

- Alharbi. F. S. (2016). Using a genetic algorithm for solving assembly sequence problem, 9th SSC Conference, University of Birmingham, United Kingdom.
- Alharbi. F. S. (2015). Applying a genetic algorithm for solving assembly sequence problem of a product, 8th SSC Conference, Imperial College, London, United Kingdom.

I certify that I have obtained a written permission from the copyright owner(s) to include the above-published material(s) in my thesis. I certify that the above material describes work completed during my registration as a graduate student at the University of Portsmouth.

ABSTRACT

Assembly sequence planning (ASP) is a vital part in reduction of cost and lead time of a product that needs to be assembled. It involves a determination of assembly process often coupled with constraints that need also to be addressed. In order to resolve ASP optimisation problems, it was reported that genetic algorithms (GA) were used for gaining an optimal solution for sequence-dependent or non-sequence-dependent job scheduling of product assembly in order to maximise production volume and minimise production delay. A latest development through a literature review indicates that glowworm swarm optimisation algorithm (GSOA) can also be used effectively and efficiently for solving system engineering optimisation problems in terms of such as non-linear equation scheduling. This thesis presents an investigation of using the GA and the GSOA approaches, respectively to seek an optimal solution from possible assembly sequences of a car engine pump valve and a ball pen as a case studies. The research work was conducted based on a comparative result of minimal assembly time by searching an optimal assembly sequence using these two algorithms, which were implemented in a JAVA program. The research outcomes show that the GSOA outperforms the GA in generating an optimal assembly sequence with a minimal assembly time. It also demonstrates that the GSOA can be a useful decision-making tool for searching an optimal or near-optimal assembly sequence of a product for product designers.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACOA - Ant Colony Optimisation algorithms;

AOF - Aggregative Objective Function;

AP - Assembly Planning;

APSO - Adaptive Particle Swarm Optimisation;

ASP - Assembly Sequence Planning;

CE - Combinatorial explosion;

CEng - Concurrent Engineering;

CSG - Constructive Solid Geometry;

CX - Cycle Crossover;

DFA - Design for Assembly;

EA - Evaluation Function;

FF - Fitness Function/s;

GA - Genetic Algorithm/s;

GACIE - Genetic Algorithm-Based Approach to Color Image Enhancement;

GL - Graph of Liaisons;

GO - Genetic Operator/s;

GS - Guided Search;

GSAA - Genetic Simulated Annealing Algorithm;

GSOA - Glowworm Swarm Optimisation Algorithm;

HPCIE - Hue-Preserving Color Image Enhancement;

JSSP - Job Shop Scheduling Problem;

LVQ - Learning Vector Quantization;

MC - Meta-Component;

OX - Order Crossover;

PM - Pseudo-Mutation;

PMX - Partial-Mapped Crossover;

PSGA - Particle Swarm Genetic Algorithm;

PSOA - Particle Swarm Optimisation Algorithm;

PWOA - Particle Warm Optimisation Algorithms;

SA - Simulated Annealing;

SEng - Serial Engineering;

SI - Social Insects;

SME - Small-Medium Enterprises.

CHAPTER 1

1.1. INTRODUCTION

When a product needs to be assembled, complexity of assembling a product may lead to possible assembly sequences in various forms that usually need to be pre-defined by product designers at the early design stage aimed at a reduction of assembly time and therefore production costs. This is particularly crucial for many small-medium enterprises (SME) that rely on assembly of products to survive in the fierce competitions of the global market. Apart from the effect of product design, assembly time is largely subject to its assembly precedence, accessibility, constrains, geometry and number of assembly components. It is helpful to seek an optimal assembly sequence for a product that has the shortest assembly time. However, it can be difficult to find a quick solution using heuristic approaches. For instance, although genetic algorithms (GA) were reported as a cost-effective way for solving manufacturing optimisation problems in machining or assembly sequences, a recent literature review shows a latest development of the glowworm swarm optimisation algorithm (GSOA) that may also be used effectively and efficiently for resolving some system engineering optimisation problems on such as non-linear equations and scheduling.

The glowworm swarm optimisation algorithm (GSOA) was introduced by Krishnanand and Ghose (2006a). GSOA was aiming to solve engineering optimisation problems, its name was derived from the courtship behaviour of an insect called a glowworm. In nature, these glowworms are able to modify the amplitude of their light emission (Luciferin) and use the bioluminescence glow for different purposes. GSOA is involved in a deployment of glowworms, luciferin-update, movement and local-decision domain. Th location and movement direction of these glowworms can be deremind by the luciferin value. The GSOA is useful for a simultaneous search of multiple optimal values usually based on different

objective functions (Huang and Zhou 2012, He et al. 2013(a)(b), Marinaki and Marinakis 2016, Yang et al. 2010 and Yu and Yang 2013).

1.2. RESEARCH RATIONALE

In an attempt to provide solutions to assembly sequence problems, some optimisation algorithms have been developed. Notwithstanding, some of the existing common limitations of these algorithms include long computational time, cost, complexity.

This research study seeks to solve these limitations through a development of the following importance:

- A suitable optimisation algorithm that can be used to solve a problem of assembly sequence optimisation for a specified product with a flexible constraint degree that can be specified according to user needs will be developed.
- The users' desirable characteristics of products include portability, ease of maintenance and good durability increase as manufacturers tend to improve their products. Consequently, these properties often result into product complexity. In an attempt to solve this problem of product complexity as a main contribution to knowledge, products have been categorised into three basic types based on the number of their expected assembly parts: very simple, simple and complex, with the assigned components constraints for better programming.
- Due to the exponential increase in the world population resulting into high products demand. Hence, there is need for GA and a new optimisation approach that has not been used for solving assembly sequence problem and that could carry out huge assembly sequence assignments within micro-seconds efficiently. Thus, in this research GSOA will be used beside GA for solving assembly sequence problem which is the assembly sequence time.

- Furthermore, computational time is a function of cost. The computational time increases with increasing cost.
- Therefore, a more flexible GA and GSOA are hereby anticipated within the scope and focus of this research study.

1.3. RESEARCH OBJECTIVES

The aim of the present thesis involves an investigation of using the GA and the GSOA approaches, respectively to seek an optimal assembly time from possible assembly sequences of a specified product with a flexible constraint degree that can be specified according to user needs. In this research, two products will be used as case studies; 1) a car engine pump valve and 2) a ball pen. Thus, the research objectives were proposed as follows:

1. Understand the nature of product assembly, assembly sequence (AS) techniques, related issues and carry out a comprehensive literature study in optimisation methods with the focus on GA and GSOA in relevance to assembly sequence of products.
2. To develop a novel optimisation algorithm that can be used to reduce assembly sequence time for a specified product with a flexible constraint degree that can be specified according to user needs.
3. Defining the effectiveness by implementing GA and GSOA, respectively into a Java used for generating an assembly sequence optimisation of a specified product as a case study or experiments.
4. Clarify efficiency by comparing GSOA with GA in terms of comparative results through experiments.
5. Test and validate research outcomes using feasible case studies.

Within the boundary of the Research Objectives, seven research questions are raised and highlighted below:

1. What are the most effective factors that may impact on assembly sequence of products?

2. What the optimal solutions that can be used in solving assembly sequence problem?
3. What is the appropriate optimisation tools that can be used in the current research to solve the assembly sequence time?
4. How to employ the GA and GSOA approaches as an aid for solving assembly sequence problem?
5. What a suitable programming language that can works with GA and GSOA?
6. Are the GA and GSOA models valid?
7. Is the best approach suitable to solve another product problem?

1.4. SCOPE OF THE RESEARCH

This research was carried out based on the following hypothesis;

1. Investigating GA and GSOA algorithms in solving the AS optimisation problems based on a comprehensive literature review.
2. Implement the proposed optimisaton algorithms in programming.
3. Apply the developed optimsation algorithms into case studies.
4. Analysis of comparative results using these two methods with the focus on the latest development of the GOSA approach.

1.5. RESEARCH METHODOLOGY

Some methods could be used in solving assembly sequence problem such as GA and GSOA. Both optimisation approaches will be implemented into JAVA as an effective research tool to carry out this research work. In this research, the development of three steps approach is an anticipated methodology.

Basically, this approach involves the following steps.

i. Representations. This is broadly categorised into implicit and explicit. The implicit representations display precedence relations between the assembly parts implicitly, while the explicit depicts products assembly comprising precedence constraints such as graphical representation using liaison graphs.

ii. Assembly sequences generation. The most important issue here is the suitability of the generated sequences.

iii. Evaluation and optimisation. This is will be done by using GA and GSOA approaches.

Before proves its effectiveness and efficient, some set of assumptions are required, these assumptions are considered within this research work, which are the following:

a) Assembly product parts are inflexible.

b) Establishment of all the component contacts during assembly.

c) Assembly procedure is monotone, in order and sequentially well arranged.

The testing and validation of GA and GSOA techniques and research outcomes through case studies approach, using a car engine pump valve and a ball pen, respectively will be conducted.

1.6. ROADMAP OF CHAPTERS

The thesis is structured in 7 chapters and 3 appendices:

Chapter 1: Introduction

This chapter provides an introduction, the aim and objectives of this research work and the research questions.

Chapter 2: ASSEMBLY SEQUENCE PLANNING AND OPTIMISATION

The chapter focuses on the state of the art in assembly sequence planning. It critically studies various methods developed to solve and optimise the AS, as well as point out the limitation of each method.

Chapter 3: GENETIC ALGORITHMS FOR THE OPTIMISATION OF ASSEMBLY SEQUENCES

This chapter introduces GAs and their applications as optimisation tools for solving engineering problems. A special attention is reserved to combinatorial problems to handle constraints. Methods, techniques and particular issues used in the GA designed for solving the ASP are presented and justified.

Chapter4: THE REPRESENTATION OF ASSEMBLY SEQUENCES AS CHROMOSOMES

This chapter is dedicated to the modelling and representation of assembly sequences using chromosomes. Generally, the literature in this field encodes assembly sequences under constraints within the same representation, the literature review of those topics is presented in this chapter.

Chapter 5: GLOWWORM SWARM ALGORITHM FOR THE OPTIMISATION OF ASSEMBLY SEQUENCE

This chapter presented GSOA for the optimisation of AS. GSOA is suitable for a concurrent search of a number of solutions. A number of researchers utilised GSOA in different areas, for example; clustering and various optimisation problems. In addition, it has been observed that the literature showed that GSO is better than PSO, ACO and GA.

Chapter 6: A CASE STUDIES USING A GENETIC ALGORITHM AND A GLOWWORM SWARM ALGORITHM FOR SOLVING AN ASSEMBLY SEQUENCE OPTIMISATION PROBLEM

This chapter applied GA and GSOA approaches for solving an assembly sequence optimisation problem for a car engine pump valve and a ball pen.

Chapter 7: DISCUSSION, CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

The final chapter includes a discussion and conclusions of the research work, with an overview of research rationales, aims, research method used and the overall findings of this study. It also suggests recommendations for future work and enhancement.

1.7. SUMMARY

This chapter presents an outline by addressing the ASP problem and optimisation techniques through a literature review. Assembly sequence needs to be optimised partially because of reduction of lead time and manufacturing costs. The chapter also outlines a scope of the proposed research work with aims and objectives to be provided as the direction and methods used for this study.

CHAPTER 2

ASSEMBLY SEQUENCE PLANNING AND OPTIMISATION

2.1. INTRODUCTION

This chapter presenting product assembly and optimisation, also, addresses issues by examining the different methods for solving the assembly sequence planning problems, discuss their limitations and other issues related, representation, modelling and optimisation are researched in subsequent chapters. Section 2.3 addresses types of assembly plans. Section 2.4 demonstrates assembly sequence optimization. Section 2.5 describes the solution space and character of the assembly sequence problem. Section 2.6 details approaches for solving assembly sequence planning problems and their analysis and justification being carried out in this research. Section 2.7 focuses on the illustration of various optimisation approaches for the assembly sequence planning to be analysed critically, the limitations are also discussed.

ASP optimisation is, in this research, is the main issue to investigate in order to determine the near optimum or optimum sequence of assembling a product. The research methodology used in this research will be based on a comprehensive literature study to identify the suitable optimisation techniques for solving the assembly sequence of a product.

2.2. PRODUCT ASSEMBLY AND OPTIMISATION

If a product has more than one component, then it must be assembled. Product assembly is often involved in final operations of manufactured products before being shipped to either the next manufacturing phase or directly to the consumer. Figure 2.1. shows a schema related to issues of assembly.

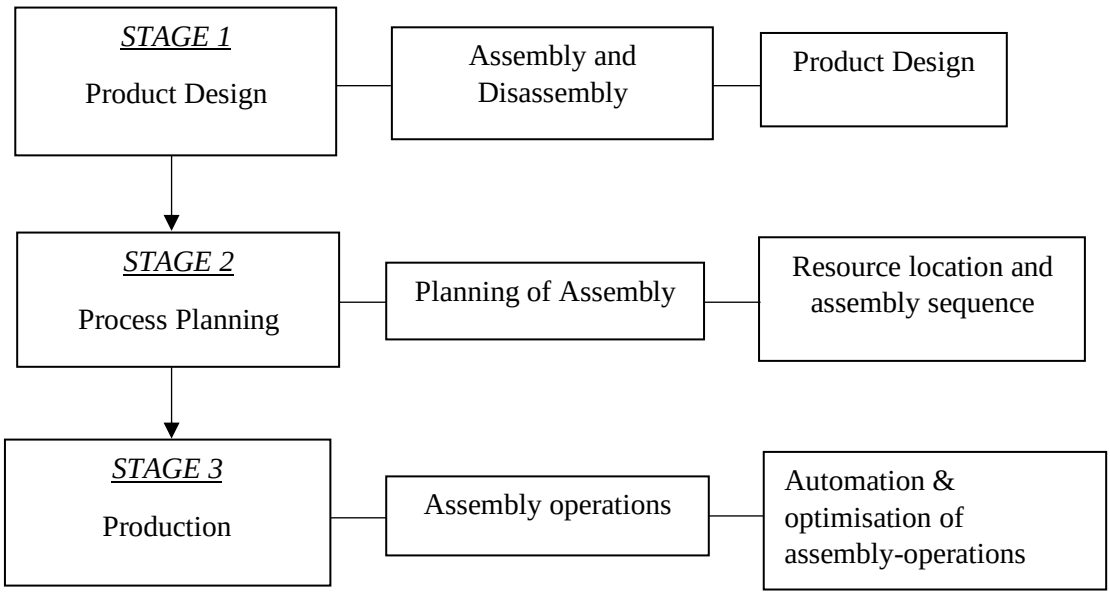


Figure 2.1. Product development, production planning and assembly

Optimisation of a product design can be made during the conceptual design stage. The key principle from the perspective of assembly is to ensure product assembly can be achieved easier by reducing complexity in terms of number of parts and operations that are needed to complete assembly tasks. In production planning, optimisation involves a determination of locations and allocations of resources of assembly lines/cell plans attempting to identify an optimal assembly sequence of a product with maximizing efficiency or productivity with minimal costs. Optimisation can be carried out via concurrent engineering (CEng) or serial engineering (SEng) as shown in Figure 2.2. When a product's development is fragmented or when there is little clarity for determining assembly facilities, SEng often finds favor.

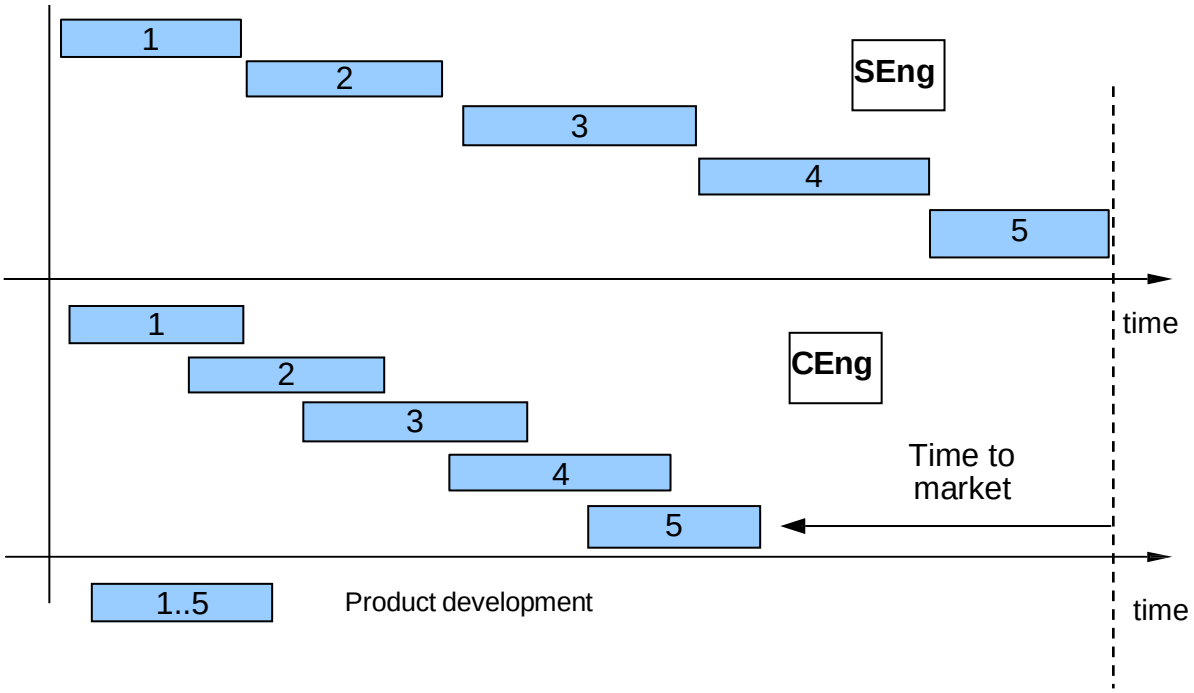


Figure 2.2. Optimisation in Concurrent Engineering and Serial Engineering (Marian at al. 2006)

CEng, on the other hand, provides more direct and definitive optimisation for a well specified and identified assembly or manufacturing environment. Also, there's a likelihood of co-evolutionary optimisation for either a single-criteria or multi-criteria optimisation. Co-evolution occurs because the different design problems do not have rigid specifications due to the fluidity of the design environment as well as people changing their minds. Co-evolutionary optimisation, Figure 2.3. has in optimisation criterion a moving and target states (both a problem and solution). Optimisation continues to evolve once a criterion optimisation is concluded, thus demanding further or another optimisation. When optimisation for different criteria cannot be linked together, then optimisation for once criterion might affect other criteria, therefore requiring a number of iterations. The major difference between CEng and SEng is that in CEng, several iterations and a number of smaller optimisations needs considering while in SEng, optimisation is done once representative of all known criteria. Therefore, optimising in assembly can be carried out for either a composite of single criterion. Moreover, 'freezing' the optimisation conditions for CEng during the process of optimisation

results in no difference between SEng and CEng in the application of co-evolutionary optimisation.

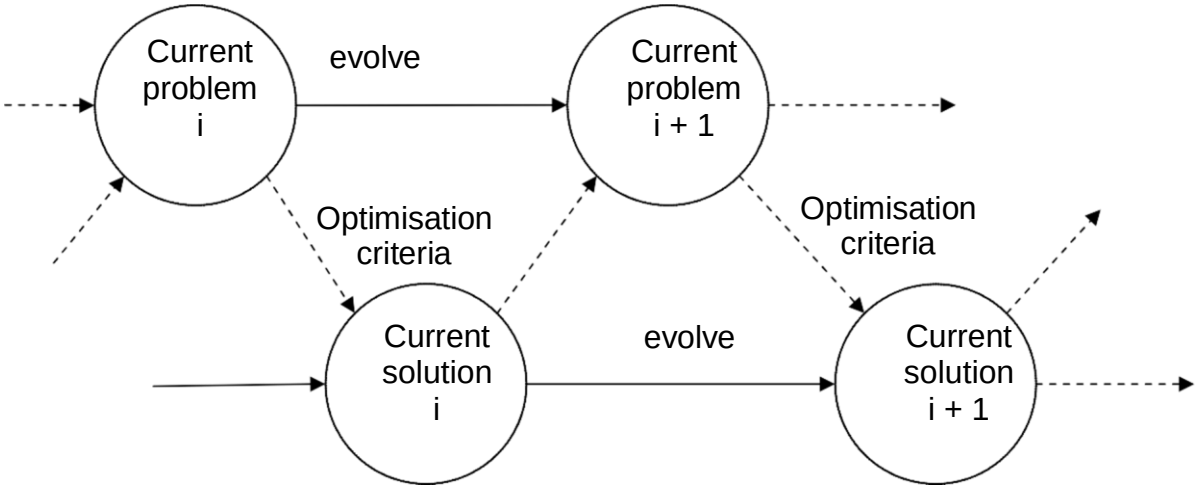


Figure 2.3. Co-evolutionary optimisation

2.2.1. Design for assembly

According to Boothroyd, Dewhurst et al. (1994), there is a widespread acceptance that a product design determines over 70% of a product final cost. Thus, Design for Assembly (DFA) is an important process in reduction of assembly costs and lead time Molloy and Tilley (1998) and Nof et al. (1997) list the principles associated with DFA:

1. Reduction of the number of part: fewer number of parts help reduce assembly operations and, in many cases, reduces the cycle time as well.
2. Design for easy insertion: for example, using suitable chamfers and tolerances on parts used for mating.
3. Ease of handling: parts are designed for ease of handling during assembly processes.
4. Standardized processes: promotion of usage of standard parts.

2.2.2. Assembly sequence planning

Wolter (1988) defined the assembly planning as pre-specification of assembly tasks and

identification of the optimal sequence. It is important to develop a proper sequence as it affects different aspects of the product design and its process of assembly. It is also important to define assembly sequence because failure to do so can prove to be costly and it affects productivity. Operation sequence is a vital factor to consider in the determination of the cost of assembly (Nof et al. 1997). In other terms, an assembly sequence problem (ASP) can be translated as the presence of a *n*-part product problem under assembly constrains. ASP can be classified by type and level of detail in both the output (plan of assembly) and input (product description).

2.3. TYPES OF ASSEMBLY PLANS

In Figure 2.4., for simplicity purposes, let us consider assemblies in 2D that are impossible to build by 2D monotone, linear and sequential assembly sequences and these properties are best described below (Wolter 1991, Jones et al. 1997 and Jones et al. 1998):

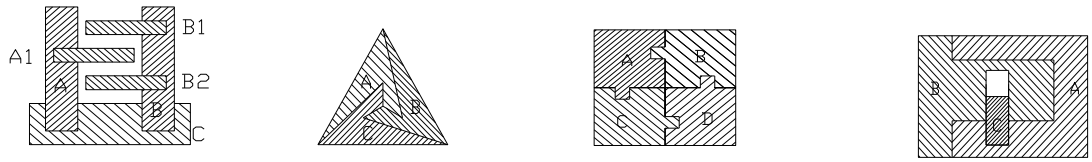
Monotone (M): one of the properties of assembly sequence whereby each component is inserted into its final location relative to the assembly. In such an instance, the n-part assembly is executed in n-1 operations. An example of an assembly that cannot be assembled using a monotone assembly sequence is illustrated in Figure 2.4d.

Linear (L): To the partial assembly, all parts are added one at a time, meaning that it does not form subassemblies. A product that cannot be assembled by a linear assembly sequence as shown in Figure 2.4c.

Sequential (S): If the plan can be broken down into the two-handed plan (where only one element can be added at each step). Below is Figure 2.4b. which demonstrates an assembly, which in 2D must be assembled through a coordinated movement of the three parts, therefore, not requiring a sequence.

Coherent (C): is a property of assembly sequence whereby each component inserted

effectively touches other components inserted earlier as shown in Figure 2.4a.



a b c d

Figure 2.4. Assemblies which cannot be assembled by: (a) a contact-coherent plan; (b) a sequential plan; (c) a linear plan; (d) a monotone plan (Jones et al. 1998)

An assembly plan can be coherent or non-coherent, sequential or non-sequential, linear or non-linear, monotone or non-monotone, or any combination of the above situations. Table 2.1 shows an example of 16 possible assembly sequence.

Table 2.1. Types of assembly sequences plans (Marian et al. 2003)

	Coherent	Sequential	Linear	Monotone
1	NO	NO	NO	NO
2	NO	NO	NO	YES
3	NO	NO	YES	NO
4	NO	NO	YES	YES
5	NO	YES	NO	NO
6	NO	YES	NO	YES
7	NO	YES	YES	NO
8	NO	YES	YES	YES
9	YES	NO	NO	NO
10	YES	NO	NO	YES
11	YES	NO	YES	NO
12	YES	NO	YES	YES
13	YES	YES	NO	NO
14	YES	YES	NO	YES
15	YES	YES	YES	NO
16	YES	YES	YES	YES

An assembly sequence planner has to consider all these situations. As most planners tend to limit the sequences, C-S-L-M is then perceived as a significant limitation of an assembly planner's capabilities.

2.4. ASSEMBLY SEQUENCE OPTIMISATION

An engineer assembly sequence is often subject to constraints of time and production cost. It becomes increasingly important to structure the optimisation of assembly sequences by overlooking certain assembly sequences that can prove to be expensive and considering the factors that may affect an assembly sequence including the structure and nature of the product. Any change in any of the above factors can also alter the assembly plan and require appropriate adjustment (Kavraki et al. 1993).

It is favourable to have a problem-oriented approach whenever solving or optimising the problem of ASP in all its diversity, generality and complexity. Solution-based methods are used in solving ASP problems through the incorporation of artificially limiting hypothesis. The advantage of the artificially limiting hypothesis is that the solution is representative. This hypothesis can be impossible to generalize if it requires a change in the problem.

2.5. SOLUTION SPACE AND CHARACTER OF THE ASSEMBLY SEQUENCE PLANNING PROBLEM

ASP can be a highly constrained, large scale and combinatorial problem. The difficulty of identifying an optimal solution is bound to the problem that is solved via an exhaustive search proportionate to the size of the solution space. When a tree search that divides the solution space is implemented, it is possible that the complexity might increase roughly with the increase in size of the solution space (Wolter, 1988). Solution space is identified by the number of potential assembly sequence in the ASP where the components of the solution space encompasses all the possibilities through which assembly of an n-part product can be made possible.

According to Wolter (1988), components might be moved via different temporary positions, potential sequential non-monotone plans are infinite. It is wise to note that ASP can be a combinatorial problem with a large scale:

- Not every component will be connected to any other component;
- The sequence cannot begin with any component;
- No dignified connection can be done between two components at a single time.

2.6. APPROACHES USED TO SOLVE THE ASP PROBLEM

Several methodologies and techniques have been developed to solve ASP problems. There are four general methods, which were presented by Delchambre (1992) aimed at generating assembly sequences (as illustrated in Figure 2.5.):

- The first method contains the three-step approach by definition of precedence constraints of assembly sequences.
- The second method requires the product to receive division into subassemblies which are in turn generated by the use of simple enough rules.
- The third method inloves the Expert Systems for generating specific assemblies.
- The fourth method is the Case-based reasoning approach.

The third and fourth methods and sometimes their combination cover virtually all methodologies incorporated in the solution of ASP problems.

Figure 2.5. shows the general methods for solving ASP problems, and all these methods are explained through the next 3 sub-sections.

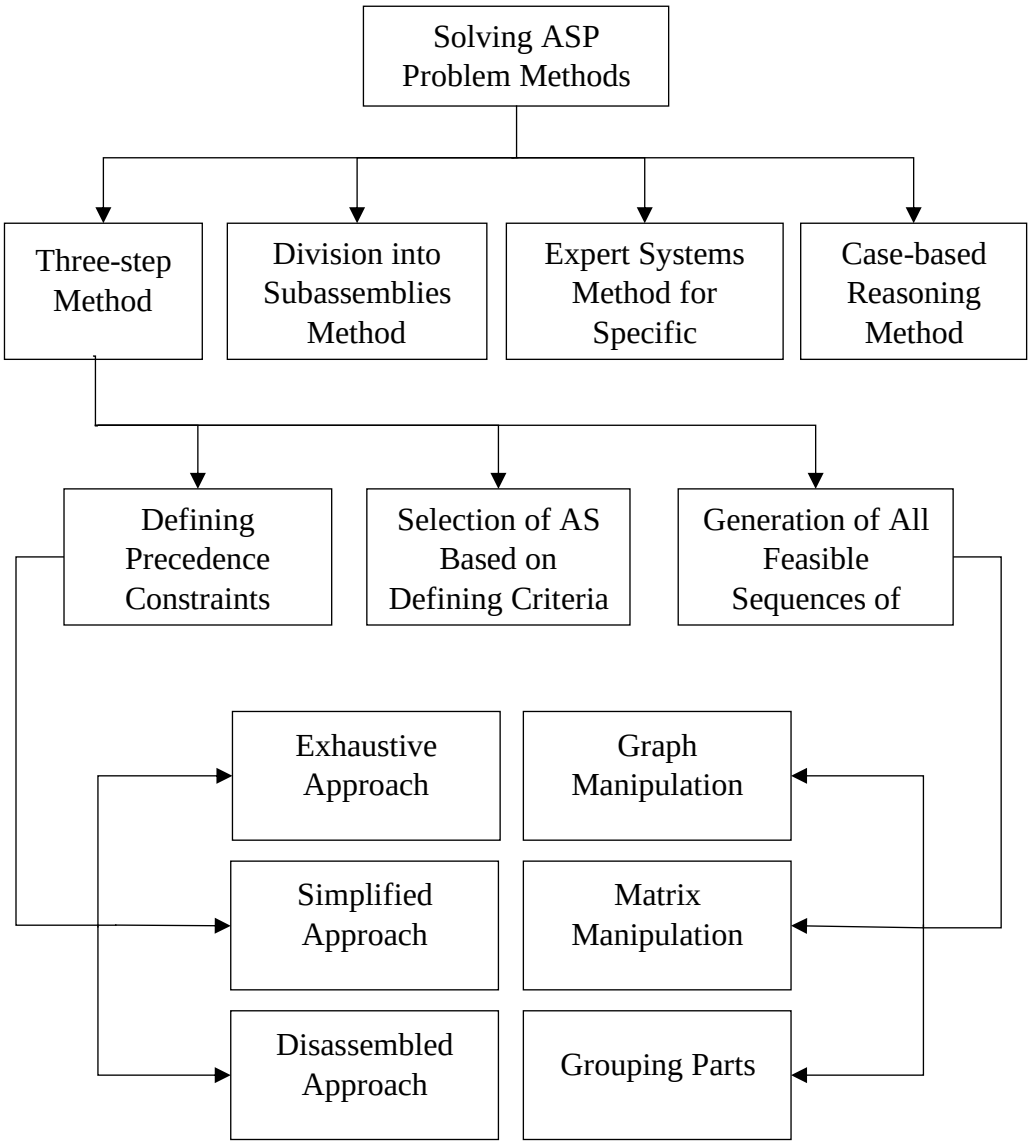


Figure 2.5. General methods for solving ASP problems

2.6.1. The Three-Step Approach

This approach often precludes some of feasible assembly sequences as it must satisfy precedence and constraints. The three steps include;

i. Defining the precedence constraints;

ii. Generation of all feasible sequences of assembly;

iii. Selection of an assembly sequence based on defined criteria.

The three-step approach is the most widely used method based on the following assumptions

(Golabi 1996, Jones et al. 1997, Wolter 1990a, Wolter 1990b):

a) The process of assembly is sequential;

b) The process of assembly is monotone;

c) The components are rigid;

d) All contacts between two components are established.

The details for the Three Steps are discussed below:

Defining precedence constraints - infeasible assembly sequences are a result of the determination of precedence constraints;

- **The exhaustive approach** – a methodology to identify all precedence relations in the midst of various assembly connections based on the connection graph of the assembly (Bourjault 1984). By considering two sets of questions (what's the possibility of establishing a connection L_i when the connection L_j has already been established? And what's the possibility of establishing a connection L_i when the connection L_j has not already been established?). Bourjault eliminated the prohibited partial states via a purely combinatorial computation. The operator needs answer the $2x$ (L^2 $-L$) questions for L connections within an assembly. However, this approach faces two distinct disadvantages; the first is due to the constant increase in number of questions, the number of parts which can be applied to such assemblies is limited. The second disadvantage is that it is prone to errors since the operator can mix a subjective analysis with a geometric analysis.

- **The simplified approach** – this approach identified by De Fazio and Whitney (1987) utilises two questions to be asked; "Which connection that cannot be established

before connection L?" and "Which connection must be established before connection L_i?". In this case, 2L questions may be asked, where precedence relations do not take alternative constraints into account, thus omitting some interesting assembly sequences (De Fazio and Whitney 1987).

- **The disassembly approach** - many researchers use the subassembly approach to identify the precedence constraints. Contains information of product parts as well as the relations between the two parts. A product assembly or disassembly directly implicates the satisfaction of precedence relationships. Precedence constraints might fail to be identified until the complete exhaustion of the search has been done (Lee 1992a; Lee 1992b). Assembly directions and proper relations of contact cannot be identified through forward planning. Huang and Lee (1988 and 1991) introduced two distinct precedence relations: 'No Later Than' (NL) as well as 'Must Precede' (MP). Based on the component's geometry, they developed an automatic procedure for disassembly to aid in the generation of precedence relations. Wolter (1988) explained precedence constraints of a single component that is added during a single operation. Homem de Mello and Sanderson (1990) break down the product into subassemblies whereby each subset of components (that only have fixed positions) is split into equal halves by all means in a feasible disassembly operation. Hyper-archs with three nodes represent the initial results; two represent subassemblies derived at via decomposition while the other only represents subassemblies. The hyper-archs

represent the precedence relations. Despite this method being the most preferred to identify all precedence relations for the product's assembly, there are setbacks associated with it at least in its usage thus far. This method only considers geometric and mechanical relations among elements.

Generation of all feasible sequences of assembly - a number of methods were developed for the generation of assembly sequences; this includes graph manipulation, matrix manipulation and grouping parts as presented below;

- **Graph manipulation -** the purpose of graph manipulation was to capture and store the data from connections between components. According to Golabi (1996), graphs are used to represent assemblies based on the graph theory techniques that were used to determine assembly sequences. For instance, connectivity data is represented by connection graphs whereas precedence data regarding the assembly is done by AND/OR graphs (Ben-Arieh 1994b), (Bourjault 1984, Homem de Mello and Sanderson 1989, Homem de Mello and Sanderson 1990, Gottipolu and Ghosh 1997). Manipulation of graphs can be both straightforward and simple especially-to model feasible assembly sequences.
- **Matrix manipulation -** data in connection relation of a product components can be stored and expressed in matrix form. For instance, the adjacency matrix of a product connection graph can be directly translated in the form of a matrix (n x n matrix for an n-component product) (Wilson and Watkins 1990).

A 2 x 2matrix can be used to record the mating kind of each individual mating pair, where both the row and column are named after the mating pairs. The output matrix can then be transformed and analysed by the use of linear algebra techniques and can also step up to represent an assembly sequence, according to Gairola (1986). Dini and Santochi (1992) explain that precedence constraints of assembly sequences can be found through the manipulation of interference and connection matrices.

- **Grouping parts -** A group of researchers proposed a simplified approach that may avoid sieving through feasible subassemblies as well as their decompositions. Their proposal involved classifying a set of components with unique characteristics in a subassembly that can be treated as an independent entity during the analysis. The use of subassemblies allows the reduction of search space through the early pruning of the links considered unnecessary while explicitly defining spatial and temporal parallelisms during assembly (Golabi, 1996).

  Lee (1992, 1994)) and Lee, Kim et al. (1993) introduced a new method to evaluate and generate assembly plans by a cut-set of liaison graphs. The procedure was aimed at determining the assembly partial order. From the graph representing the assembly, they extracted preferred subassemblies. The recursive extraction provides the basis for the extraction of subassemblies alongside the simultaneous verification of disassemblability. A preferred subassembly is where a cluster of components that can be disassembled from the original assembly. However, the problem with applying this

sort of method is the amount of data that is to be supplied and stored, and there might be the impossibility of its automatic extraction. Categorizing components into subassemblies is one of the important characteristics of the assembly planner as it cannot be overlooked at the expense of the assembly.

Assembly sequence under certain criteria - Assembly sequences are evaluated using optimisation criteria in quantitative terms and the sequence with either the lowest or highest value to be chosen. The criterion for one manufacturing company might not be the best fit for another. It is also difficult to distinguish between qualitative and quantitative terms and vice versa. The operation and construction costs are seen to have significant differences, which are associate with weight factors such as assembly costs, difficulty degree and assembly task time in the assembly sequence graph (Gottipolu and Ghosh 1997).

2.6.2. Division into subassemblies

Akagi, Osaki et al. (1980) proposed the classification of the end-product in terms of functional units 'f_i.'. All components that constitute the product are categorized into functional units. These units are responsible for categorizing components as fastening methods involved in their assembly, e.g., riveted and bolted joints, shrinking and pressing fits. The assembly operation is divided into work elements responsible for fastening the components in each category. Generation of assembly sequences follows three principle rules;

1. If all elements of f_i are included in f_j, then f_i must precede f_j.
2. If f_i and f_j share common elements, then f_i and f_j cannot be assembled simultaneously (in a non-linear assembly sequence).
3. If f_i and f_j have no common element, then f_i and f_j can be assembled simultaneously.

2.6.3. Expert systems and Case-based reasoning

The expert and knowledge-based systems was byHuang and Lee (1991) to define the relation among a pair of components, requiring relationship between the locative configuration regarding these two components. This method has a major disadvantage which is the search mechanism, the search mechanism only performs to find a local optimisation without a global optimum. Another obstacle in using knowledge-based and expert systems is hardly to get data about the assembly automatically, also translate the knowledge from a case to another.

2.2. APPROACHES USED TO OPTIMISE THE ASP

There are a number of approaches that can determine a near optimum or optimum assembly plan (Golabi 1996):

(a) Identifying the most suitable assembly sequences based on specified weighting criteria.

(b) Identifying the best assembly sequence by either disassembling or assembling the product. This method provides a best local solution for a task of assembly but with no guarantee of a global optimum.

(c) Generation of an assembly sequence using the knowledge-based system. Usually considering the base criteria by beginning with the base part, other components are added until all components can be assembled. This method determines the next best assembly task but also cannot guarantee a global best.

(d) A population search has to be conducted by beginning with the number of assembly sequences by identifying the best global solution but there is usually no guarantee to achieve this.

2.2.7. Exhaustive Search

In the ASP optimisation, the exhaustive search is merely a theoretical method that can be applied to decide the optimal assembly sequence by creating all assembly solution and assessing and selecting the best one. The assessment is done by utilising improvement criteria or weighting for each assembly sequence. The best assembly sequence may be identified by correlation of the estimation of an assembly solution to others (Homem de Mello 1989).

2.2.8. Simulated Annealing

Simulated Annealing (SA) is an effective stochastic pursuit technique appropriate to an extensive variety of issues for which minimal earlier learning is accessible. It may deliver solutions for hard combinatorial streamlining issues. The disadvantage is the long computational time required by SA (Yao 1991). The essential thought of SA originates from reduced matter physics. To minimise energy states, called ground states, of complex system, for example, solids. The system (solid) is initially warmed to high temperature, then gradually chilled off. The system will achieve a ground state if the cooling rate around the point of solidification of the framework is adequately moderate. At each condition of the reproduction, another condition of the system is produced from the present state by giving an irregular relocation to an arbitrarily chose molecule. The new state will be acknowledged as the present one if the vitality of the new state is no more prominent than that of the present state, else, it may be acknowledged with likelihood (Yao 1991).

Local optimisation of $f(x)$ begins with an initial solution x, $x \in s$. At that point, y, a neighbour of x is chosen, and if $f(y) < f(x)$, y is a downhill move, and is accepted. The procedure proceeds until no further downhill movements are found (a local minimum is found).

SA gives the possibility to avoid being caught in a local minimal by sometimes allowing an uphill move. The probability of uphill moves is higher at the beginning of the optimisation and decreases as the optimisation approaches to the end, to the optimum value.

SA has been utilised for the selection of the probable minimum cost assembly sequence (Milner Graves et al. 1994, Park and Asada 1994). The issues are addressed: given a product design, determine the minimum cost assembly system for the product. Their way to deal with select the minimum cost assembly sequence is include three stages:

First, all the probable assembly solutions must be given. De Fazio and Whitney (1987) created the Diamond Graph to represent to assembly states (by nodes) and tasks to the following assembly state (arcs). Any descending way from the highest point of the diagram (completely disassembled unit) to the base (completely assembled unit) represents a unique and valid assembly sequence.

Second, a technique, by which the cost of an assembly system for a given sequence of tasks is assessed. The equipment for workstations is selected, then tasks are assigned to workstations for a given solution in order to minimise the annual cost to produce the required number of assemblies every year. The presumption is that the cost is not added over steps in a sequence. The cost of an assembly operation is not a constant and depends on the previous operations and based on the production volume.

Finally, a search heuristic which can proficiently produce the least cost sequence.

The real disadvantages of the approach and optimisation method when connected to ASP are the following:

- Because of CE, the technique is restricted to reduced search spaces (assemblies with a reduced number of components or heavily artificially constrained).

- Because of randomly selecting another sequence, this would not provide an idea of appropriate neighbour of the ideal solution. Thus, the neighbour point is difficult to be appropriately described. Generally, it is just a matter of preferences and representation that characterises two points as neighbours.

2.2.9. Genetic Algorithms

Some researchers tried to optimise the ASP using Genetic Algorithms (GA). Sebaaly et al. (1996a) used a genetic planner for assembly automation. The data for assembly is kept in an implicit state, in a reference and a connectivity matrix (Sebaaly and Fujimoto 1996b, Sebaaly and Fujimoto 1996c). If a connection exists between two specific parts a_i and a_j, then the elements with the same rows and columns assume all non-zero values, otherwise they are zero. At the production of the chromosome, a gene is produced from the rules with the highest value which encodes the precedence constraints. It acts a population-based search rather than a part-based one and can produce linear and non-linear sequences.

Lazzerini and Dini (1999) and Dini, Failli et al. (1999) brought up another genetic algorithm to optimise the AS. The optimisation criteria are:

- Reductions of object orientations – reduction of assembly time and cost of assembly line.
- Reductions of gripper changes – reduction of assembly time.
- Placing as much as possible technologically similar assembly operations, e.g. screwing, pressing, that can be done with the same mechanical tool.
- Through a specific software module, Feasibility Evaluator (Santochi and Dini 1992) the evaluation of feasibility of a chromosome is carried out, depending on matrix measure which normally computes the feasibility degree of an assembly sequence defined as the length of the longest feasible subsequence in the chromosome.

As a weighed sum of the length longest feasible subsequence the fitness associated with a chromosome is calculated. The following are also weighed, the number of orientation changes of the assembly, the gripper changes and the number of the same assembly operations placed together. The Genetic operators are specifically assembled.

The limitations of the algorithm are the following:

- The algorithm cannot be used as it is if the assembly has a component assembled from a random direction because the Feasibility Evaluator works only on the major axes (x, y, and z).
- The algorithm can only detect sequential, monotone and coherent assembly sequences.

Marian, Luong and Abhary (2003) used the Three Step Approach to solve the AS problem. To optimise the AS, a population-based search is utilised as development of (d) approach (see Section 2.7.). Figure 2.6. illustrates the mechanism of the GA approach for solving the AS problem.

An assembly sequence demonstrates the progression of operations to amass the item from its parts. The assembly sequence is characterised by the attributes of the item (geometry of components, relations between components, materials of components, tolerances and so forth). The assembly solutions, absolute and enhancement constraints are characterised in the solution space. The genetic operators work in the model space with chromosomes. Assembly sequences are demonstrated and presented as chromosomes. There ought to be, a by-unique mapping between an assembly sequence and a chromosome. Not all assembly sequences are feasible unless if it satisfies a class of constraints (absolute constraints).

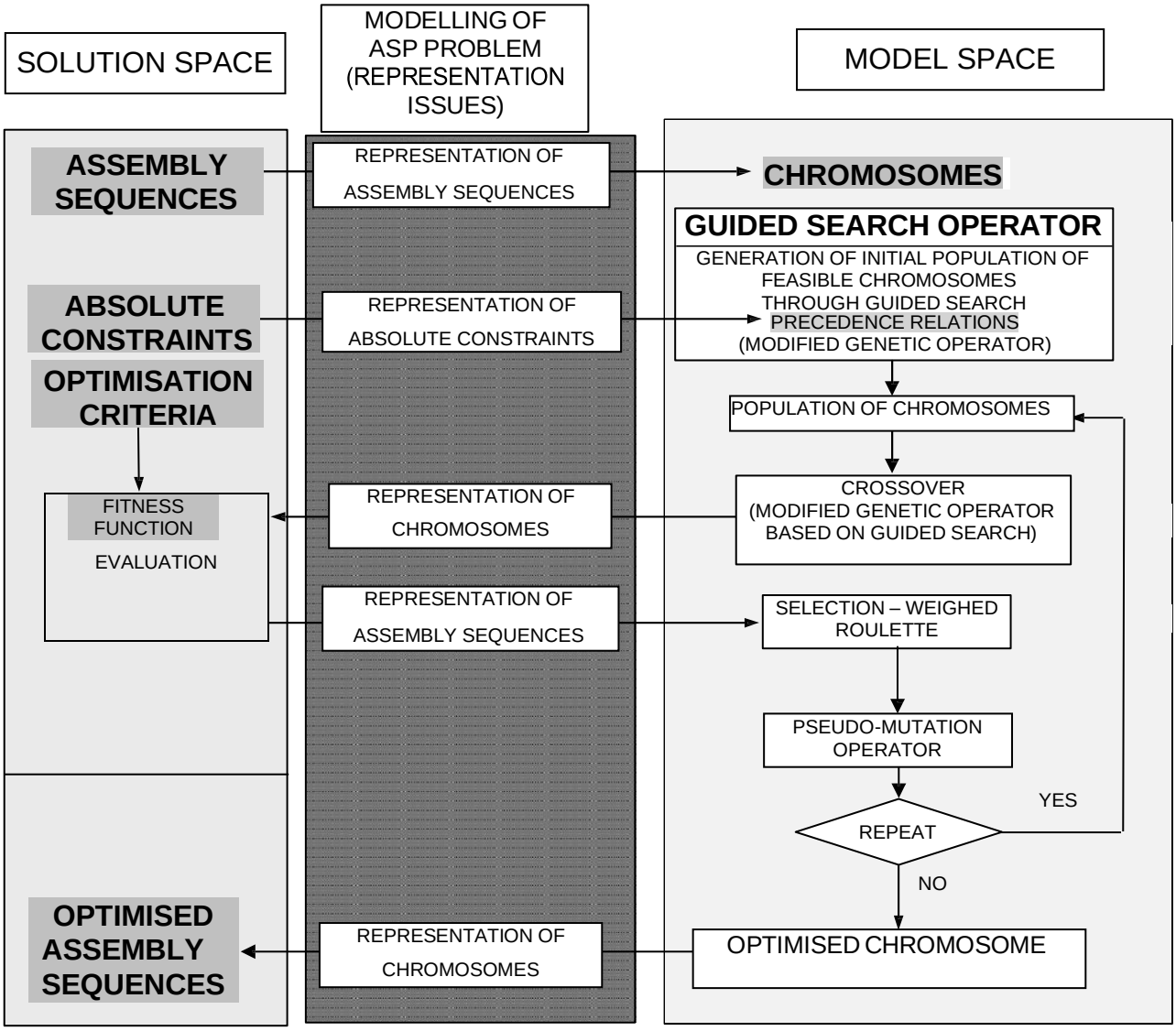


Figure 2.6. The GA approach for solving the ASP problem (Marian et al. 2003)

Figure 2.7. illustrates Modelling and representation issues in ASP. The constraints of the ASP, characterised in the solution space, are inferred as precedence relations. The assembly table incorporates the availability data from the table of liaisons and precedence relations that encode constraints. More precedence relations might be encoded as Boolean relations. Such a calculation needs to consider the scale, the intricacy and sweeping statement of the issue with the capacity to produce an attainable assembly sequence in any sensible mechanical setting.

The guided-seek calculation depends on a diagram-look system. It creates doable assembly solutions by arbitrarily selecting, in each stage, one of assembly operations that can be performed at this specific step. To accomplish this, the components for each progression are chosen by utilising the priority relations.

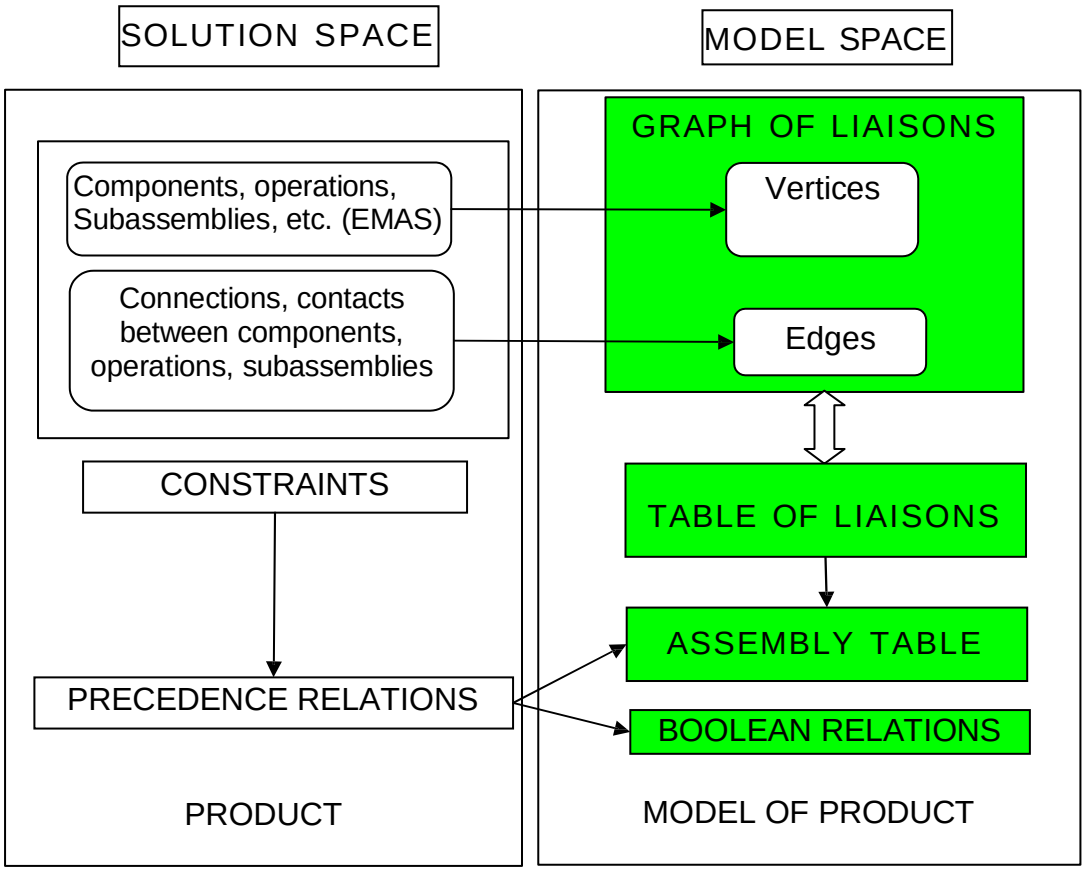


Figure 2.7. Modelling and representation issues in ASP (Marian et al. 2003)

The guided search operator is an adjusted genetic operator intended to beat the combinatorial blast by changing the combinatorial issue in a polynomial one (by producing and working just with achievable sequences). GA has capacity and adaptability to deal with expansive scale issues. The structure of the proposed GA depends on a great GA calculation (Gen and Cheng 1997) and consolidates the guided inquiry. Different methodologies (punishment, dismiss and repairing methodology) were endeavoured by Marian et al. (1999a); Marian et al. (1999b); Marian et al. (1999c) and ended up being successful just for assembly with a decreased

number of components (<10) and therefore the solution space was moderately restricted. After crossover, the chromosomes were made an interpretation of solution space to be are assessed utilising a fitness function based on pre-defined criteria for generating a suitable assembly sequence. Once these assembly sequences have been assessed by weighting the fitness value from which the one with the highest fitness value is chosen through a weighed roulette calculation (Gen and Cheng 1997).

2.3. STATEMENT OF PROBLEMS

During assembly planning, it is always difficult to make the selection of an assembly sequence due the presence of increasingly large and small parts coupled with minor variations in design. This has an effect on the selection of required assembly choices (De Fazio and Whitney, 1987). The possibility of feasible solutions from traditional genetic algorithms becomes a mirage due to the increase in complexity (Yu and Wang, 2013). In addition, the time spent and huge costs incurred in the assembly of products, there are other problems that need solutions and optimisation. There is a need for the reduction in the assembly planning cost and time. Moreover, these go along with the computing time and cost which could be reduced using optimised genetic algorithms (Ou and Xu, 2013).

2.4. RESEARCH GAPS

Tseng et al. (2010a) observed that the combinatorial sequence number increases with an increasing number of components. This implies that a larger number of product components may result in longer times taken during computation. As a result of the geometric complexity of components, coupled with the precedence complexity that is characteristic of assembly operations, it is not clear whether the time complexity can be exactly computed. The GA method is preferred to other methods because it has a shorter computational time. Chang et al. (2009) argue that one problem with ASP is that an increase in the number of components

implies that more constraints will occur during its assembly, which in turn makes the assembly problem complex. Thus, researchers have worked with the objective of finding alternative and suitable methods for getting feasible solutions in the solution space. These include the traditional GA that uses the method of random searching. It was reported by Tseng et al. (2010b) that the combinatorial sequence number is capable of increasing as the numbers of components grow. An increase in components of the products leads to an increase in the time used in the computation. In general, results show that the GA method has an advantage in cases where the computational time is shorter. Even though the methods discussed can prove useful in generating and evaluating useful sequences that have good solutions, a lot remains to be done for managing complicated products that have many components. According to Marian et al. (2006) there still exists a need to come up with a new methodology in order to be able to withstand the extraordinary varied character of the ASPP in large scale because real-life products have challenging constraints and sizes. To the best knowledge of the authors, no assembly sequence planner has yet been developed that is capable of reliably solving and optimising, as well as retaining the possibility of exploring various regions within the search space, an assembly problem that has 25 elements. Previous attempts only seem to deal with simplified problems whose components have been significantly reduced with search spaces that are severely limited. Zeng et al. (2013) stated that the search space, which is associable with assembly sequence planning, is usually proportional to both the component numbers and their assembly relationships. It takes a long computation period in cases where the assembly is complex. When the component numbers are above the set threshold, it is difficult to accomplish assembly sequence planning.

According to Tseng (2006) the Genetic Algorithms have limited applications as a result of the fact that the associated algorithms usually take exponential time as they run in relation to the component numbers. When there are large numbers of components, the assembly product will

have more constraints leading to an increase in the complexity of the procedure that is utilized in solving the assembly problem. According to the studies conducted previously, authors suggested that large numbers of components result in more complex assembly. In this research, product nature is among the main problems that face assembly sequences in contemporary industries. In an attempt to find a solution to this challenge, this research groups products into three disparate parts; complex products, very simple products, and simple products. The aforementioned classification has been based on ease of assembly of products, geometry of products and the time taken to complete the assembly process.

CHAPTER 3

GENETIC ALGORITHMS FOR THE OPTIMISATION OF ASSEMBLY SEQUENCES

3.1. INTRODUCTION

GA is a search technique that can be used for solving the ASP issue. GA are a class of universally useful search techniques joining coordinated and stochastic search. Genetic Algorithms was created and presented by Holland in the 1960's and 1970's (Holland 1975) and it was promoted by David Goldberg (1989). GA is an inquiry-based system with common determination of "survival of the fittest" and therefore, GA is an Evolutionary Algorithm (EA), which likewise incorporate evolutionary programming and evolution strategies.

GA was reported to be effectively applied in engineering design and planning (Gen and Cheng 1997, Falkenauer and Delchambre 1992, Karr and Freeman 1999), cell fabricating (Kazerooni 1997), machine learning (Goldberg 1989; Michalewicz 1992; Michalewicz 1994; Michalewicz 1996), image processing (Pal and Wang 1996), robotics (Davidor 1991), Job Shop Scheduling Problem (JSSP) (Cheng, Gen et al. 1999), graph matching (Krcmar and Dhawan 1994).

Chang et al. (2009) stated that one of the problems in assembly sequence planning (ASP) is that an increase in the number of components often leads to more constraints, which in turn make the assembly process more complex. Ou and Xu (2013) adopted a matrix approach for analysing the information derived from a CAD model to obtain the assembly sequence for a two-stroke engine aiming to reduce both assembly time and cost. Rashid et al. (2011) provided a review on ASP using the soft computing approach. Three popular soft computing algorithms have been used in their studies, which are GA, ACOA (ant colony optimisation algorithms) and PWOA (particle warm optimisation algorithms). Xing et al. (2012) proposed a crossover particle swarm genetic algorithm (PSGA) to generate the optimised assembly sequence. They

compared the generated assembly sequence using a GA. Hongbo et al. (2006) developed a genetic simulated annealing algorithm (GSAA) for solving an ASP optimisation problem. Zhou W. et al. (2013) presented the imperialist competitive algorithm used for seeking an optimal or near-optimal solution of an ASP.

3.1.1. Structure and method of GA

The general structure and method of GA (Gen and Cheng 1997), illustrated in Figure 3.1., can be condensed as takes after:

- The search begins with an underlying random population of solutions (population-based pursuit);
- Every person in the population is a chromosome and is a representation of an answer of the issue;
- A chromosome is a series of images (twofold, whole number, and so forth);
- The chromosomes develop under determined determination runs through progressive cycles – generations;
- Amid every generation, the chromosomes develop through crossover and additionally transformation.

Crossover includes mating randomly shaped sets of chromosomes. The new chromosomes came about because of crossover – offspring - hold a portion of the parents' characteristics (correspondence and data trade between parents characteristics).

Mutation includes changes inside a chromosome. The new chromosome comes about because of the parent through a trade of qualities.

At this stage, another generation is shaped by selecting, as per the fitness value, a portion of the parents and offspring and dismissing others, in order to keep population, measure consistent. Fitter chromosomes have higher probabilities of being chosen. After various

generations, the calculation meets to a population of chromosomes, which, optimally, represents to the optimal or close optimal answer for the issue.

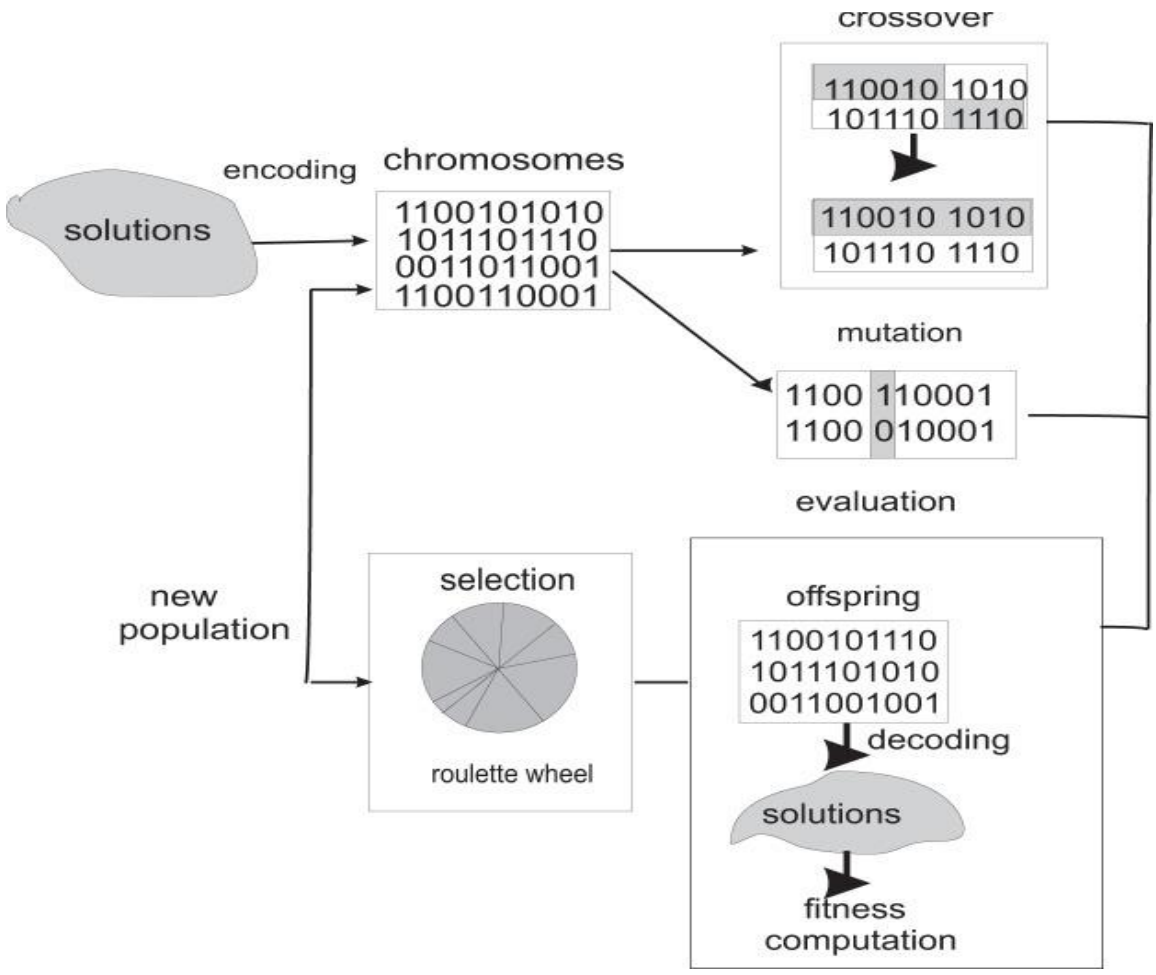


Figure 3.1. General structure of Genetic Algorithms (Gen and Cheng 1997)

3.2. GENETIC ALGORITHMS AS AN OPTIMISATION TOOL

Hong and Cho (1999) applied the GA to generate the optimal solution for a robotic assembly sequence aiming to minimise the assembly cost. Development of the GA used for assembly sequence optimisation generally involves Three Steps: representation, generation, and optimisation, as appeared in Figure 3.2. Representation can be categorised as two types: implicit and explicit. Implicit representation refers to precedence between two mating assembly parts, while explicit representation is involved in encoding possible assembly

sequences with constraints. In this study, a population of possible assembly sequences was initially generated in a random manner. Such a generation refers to a creation of assembly sequences allowing a little perturbation during the crossover stage. Within one generation, the GA is able to select a subset of chromosomes (often two) from the current population, called parents. These were used for mating to create a new chromosome called a child or offspring. Optimisation is carried out by executing user-defined criteria to seek an optimal solution among generated assembly sequences.

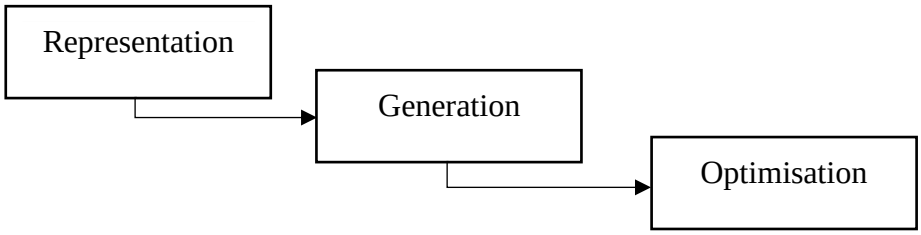


Figure 3.2. The steps of Genetic Algorithm approach

3.2.1. Termination of the GA Optimisation Process

Termination of the GA optimisation process occurs after the entire search space is completed. The solution space is classified into families whereby a single family represented a single valid assembly sequence (Senin 2000). A chromosome that contains a solution (i.e., parent assembly sequence) is probabilistically selected based on an evaluation of fitness relating to the current population. In particular, A chromosome with a higher fitness value has a greater chance to be selected for mating with another chromosome with a higher fitness value to produce a new chromosome. A genetic operator is subsequently applied leading to a new generation of offspring of assembly sequence.

Generally, there are three types of operators, which have crossover, mutation and selection, respectively, based on some forms of objective function known as a fitness function. Crossover is used in this case as a process that carries out an exchange of parental genes to

create a new chromosome. Further, genetic diversity can be introduced into the chromosomes of a population or family using crossover and mutation to generate a family of new chromosomes, and the GA repeatedly compares the fitness value of one chromosome with another until the optimal chromosome is formed. The use of GA to solve assembly sequence optimisation problems often produces a population of infeasible solutions because of optimisation problem constraints. Constraints in assembly have number of types, but the most important are the absolute constraints and optimisation constraints. Absolute constraints (hard constraints) as geometrical, precedence, accessibility is limiting the number of feasible assembly sequences.

On the other hand, the optimisation condition (weak constraints) is differentiate the quality of the assembly sequences (Sebaaly and Fujimoto 1996a) and (Jones et al. 1998). With respect to the constrained optimisation problem, GA searches the feasible solutions that satisfy the constraint conditions with the objective function over the entire genetic space. The solutions that do not satisfy the constraint conditions are referred to as infeasible solution whose encoding referred to as chromosomes (Zhang et al. 2014).

3.2.2. Evolutionary Algorithms

As one of Evolutionary Algorithms, GA have two conspicuous components:

- Population.
- There is communication and information exchange between individuals in a population.

Other particular features of GA are (Goldberg 1989, Haupt and Haupt 1998, Marian, Luong and Abhary 2003):

- GA work with a coding of parameter sets.
- GA utilise result (target work) data, not subordinates or other assistant knowledge;

- GA utilise probabilistic, not deterministic.

Thus of those features, GA have various real favourable advantages when contrasted with other enhancement methods:

1- GA do not have much scientific necessities about the enhancement issues and, because of their transformative nature:
    - GA will look for solutions without regard to the precise internal working of the problem.
    - GA can deal with constraints for parallel PCs, where each processor can assess a different capacity in the meantime.
    - GA work in discrete, constant or blended search spaces.
2- The capability of evalution operators makes GA exceptionally successful at implementing a probabilistic global search. An algorithm is appropriate in if it is conceivable to achieve any state from some other state in a limited number of iterations. Other conventional methodologies perform nearby pursuit by a combined step-by-step strategy, which analyses values of nearby points and moves to the relative optimal points. Global optima can be discovered just if the issue has certain convexity properties that basically ensure that any nearby optimal is a global optimal.
3- GA offer an extraordinary adaptability to hybridise with domain dependent heuristics to make an effective implementation for a particular issue.
4- Being population based:
    - Altogether search from a wide inspecting of the search space;
    - GA optimise parameters with to a great degree of complex cost surfaces and can skip local optima;
    - Provide a number of optimal solutions not only one solution.

GA can bargain effectively with an extensive variety of issue ranges, including those which are hard to comprehend with different strategies (Kazerooni 1997).

3.3. GENETIC ALGORITHMS AND COMBINATORIAL PROBLEMS

Finite problems are dealt by combinatorial optimisation, although there are often vast number of solutions (Gen and Cheng 1997). Everyday such issues abound, particularly in engineering: the knapsack, quadratic 0-1 integer programming, machine scheduling, vehicle routing, travelling salesman problem and so on. Combinatorial explosion (CE) is the most challenging aspect in combinatorial optimisation. The quantity of answers for a combinatorial issue is normally a component of the factorial or exponential of the quantity of components of the issue. For combinatorial problems the robustness of the algorithm becomes paramount.

3.4. GENETIC OPERATORS

A straightforward GA represents solutions utilising string of bits (0-1) that may encode whole numbers, genuine numbers, sets, and so forth. This all-inclusive representation has the upside of utilising a uniform solution of basic operators and streamlines the examination of GA properties hypothetically. Nevertheless, bitwise operators are regularly improper for generally issues. Today, most useful GA frameworks utilise issue particular representations (integers to represent whole integers, character strings to represent sets, etc.), and modified genetic operators for those representations (Kazerooni 1997).

This section quickly and thoughtfully reviews the operators that make a genetic algorithm and find out related issues that are to be considered preceding outline those operations (see Figures 3.1., and 3.2.):

1- Meaning of an underlying generation of chromosomes (Chromosomes generation). It is critical to note that:
   - A representation for chromosomes;

- A structure to represent imperatives must be created earlier.

2- Advancement of the parent generation of chromosomes through mating (crossover operator)

3.4.1. Chromosome Representation

A chromosome represents a solution of the issue, and it is a string of genes that can be coded. The double strings utilised by Holland, despite the fact that they require exceptionally straightforward genetic operator, are less reasonable for most complex applications, particularly for issues from engineering world (Gen and Cheng 1997).The optimal representation for an issue is pointless if it cannot be produced or are excessively unpredictable. Because ASP is a combinatorial issue accordingly, a non-string representation is looked for. Three basic issues rise while considering non-string approaches for the mapping amongst solutions and chromosomes:

- The legality of a chromosome: whether a chromosome represents to an answer for a given issue. As illustrated in Figures 3.3and 3.4, respectively, the wrongdoing of chromosomes begins from the way of encoding strategies. For some combinatorial issues, an illicit chromosome cannot be decoded to a solution (regardless of the possibility that incomplete chromosomes may relate to fractional solutions), and, thusly, it cannot be assessed. Accordingly, punishment techniques cannot be or are hard to be connected for this situation. For the most part, repair systems are connected for infeasible and unlawful chromosomes.

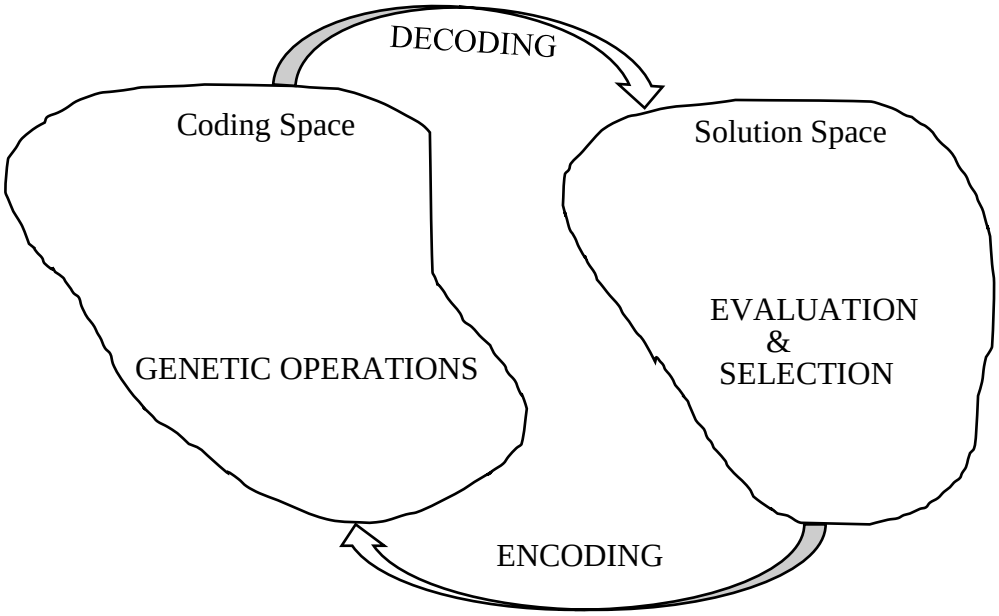


Figure 3.3. Coding space and solution space (Gen and Cheng 1997)

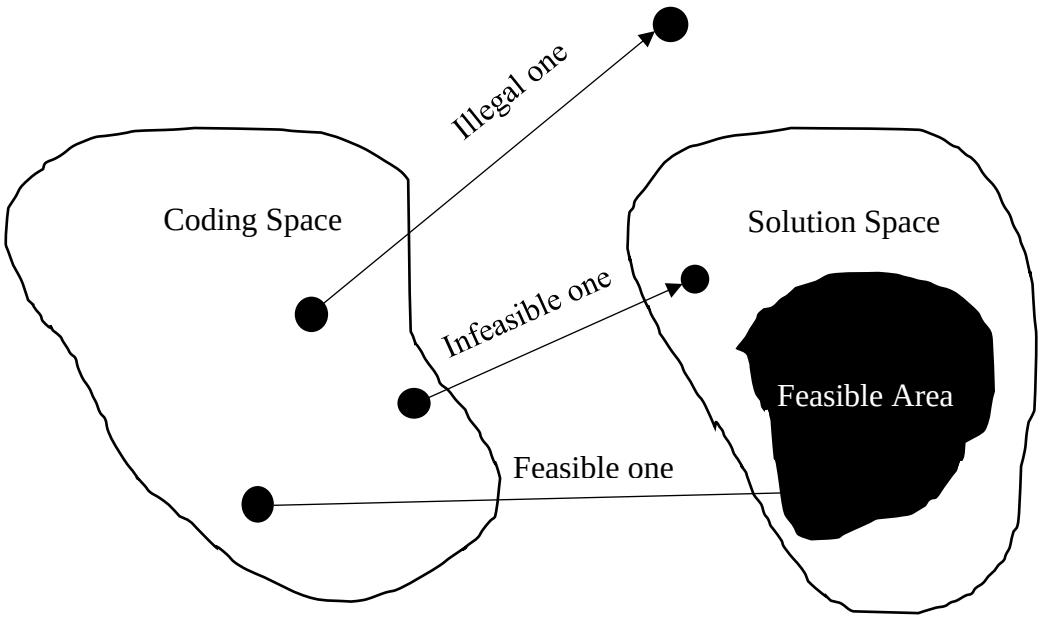


Figure 3.4. Feasibility and legality (Gen and Cheng 1997)

- The plausibility of a chromosome: Whether an answer decoded from a chromosome lies in the attainable district of a given issue. The infeasibility of the chromosome starts from the way of the obliged advancement issue. All GA must have the capacity to deal with

constraints. For the most part, penalty approaches are utilised to drive the genetic search to approach the optimal from both feasible and infeasible areas.

- The uniqueness of mapping: the mapping from chromosomes to solutions (decoding) may have a place with one of the accompanying three cases:
  - 1-to-1 mapping.
  - n-to-1 mapping.
  - 1-to-n mapping, as appeared in Figure 3.5.

The 1-to-1 mapping guarantees a bi-special correspondence between a chromosome and an answer. The other two mappings require supplementary operators to segregate between the helpful solutions and chromosomes.

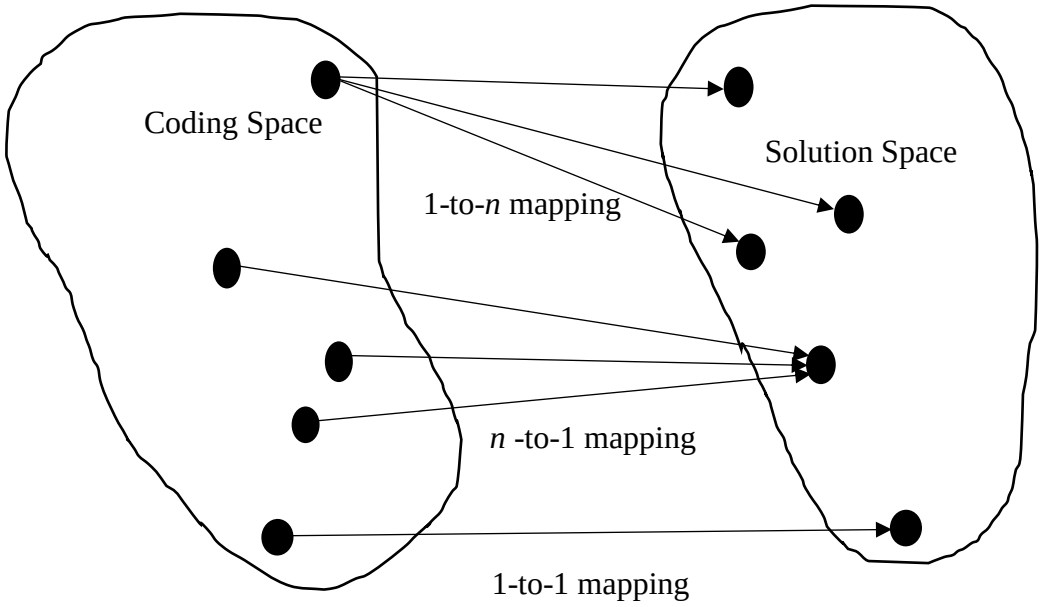


Figure 3.5. The mapping from chromosomes to solutions (Gen and Cheng 1997)

A decent representation of solutions into chromosomes for solving the ASP issue requires the accompanying qualities:

- Be conservative, basic and instinctive: the GA infers an iterative procedure in an inquiry space that is of combinatorial size. The span of the space requires a minimal representation of answers for chromosomes to empower the capacity and control of populations of chromosomes in today's computers in a sensible time.
- Be a 1-to-1 mapping: this property keeps away from the need for supplementary calculations that need to segregate amongst great and parasite solutions/chromosomes amid the encoding/translating process.
- Enable the coding of all important and helpful assembly plans to the level of detail required by reasonable applications.

3.4.2. Constraints

The constraints that depict the ASP are of differing nature and source and different effects on the optimisation procedure. The constraints are detailed in Section 3.5. In the present work, the genetic operators are custom fitted to tackle the ASP issue and the limitations are intensely utilise d as a part of the generation of chromosomes and in the crossover operator.

A representation structure for limitations must be produced to make the important requirement accessible when required. This structure needs to empower the encoding of important constraints to be straightforward and instinctive.

3.4.3. Fitness function

The fitness estimation of a person in the population is a measure of the nature of that person. The fitness capacity is connected in the solution space to an answer. A fitness value $f(i)$ is assigned to every individual element i in the population. In this proposition, a high fitness value means solid match. The purpose behind this in characterising the fitness is that the GA just needs an estimation of the fitness allocated to every person, not the way this value changes from a person to its neighbour or how it is characterised/acquired.

3.4.4. Chromosome generation

In established GA with double strings encoding, the chromosomes are produced randomly. In combinatorial issues, the generation of the chromosome needs to consider the way the solutions are characterised. The random key representation allows the random generation of a chromosome to decipher the solution. The generation of chromosomes by utilising the random key representation is impossible for the ASP issue. This is because of a number of supplementary limitations (Marian, Luong and Abhary 2003).

3.4.5. Crossover

Crossover is the main genetic operator. Thoughtfully, its input is a couple of randomly chosen parent chromosomes and the output is a couple of offspring chromosomes that join the couple' features. The crossover swaps a part of the couple' genetic data to create the new offspring (Holland 1975, Marian et al. 2000b).

As illustrated in Figure 3.6., two parent chromosomes P1 and P2 if the crossover point is between loci 6 and 7, the end bits are swapped. The result of this operation is a solution of two offspring chromosomes, C1 and C2.

This straightforward crossover is appropriate for paired strings, for this situation a1...a10 and b1...b10 have the qualities 0 or 1. Be that as it may, when combinatorial issues are included, integer representations are utilised, and the issues are constrained. Various crossover operators have been created for combinatorial enhancement: PMX (Partial-Mapped Crossover), OX (Order Crossover), CX (Cycle Crossover), and position-based crossover, order-based crossover, heuristic crossover, et cetera. They are not suitable for the AS problem because of its degree of constraint.

PARENTS

P1= (a1 a2 a3 a4 a5 a6 | a7 a8 a9 a10)

P2= (b1 b2 b3 b4 b5 b6 | b7 b8 b9 b10)

Crossover Point

OFFSPRING

C1= (a1 a2 a3 a4 a5 a6 b7 b8 b9 b10)

C2= (b1 b2 b3 b4 b5 b6 a7 a8 a9 a10)

Figure 3.6. A case of crossover operator

3.4.6. Mutation

Mutation is a foundation operator which produces unconstrained irregular changes in chromosomes. In GA, mutation serves the part of either:

- Changing the genes lost from the population through the selection procedure so they can be attempted in another specific circumstance, or
- Giving the genes that were not present in the initial population.

A straightforward approach to create a mutation is to modify at least one genes (Gen and Cheng 1997).

3.4.7. Evaluation

Every individual of the population is assessed by utilising the fitness function the selection procedure. To assess a chromosome, it must be decoded back to the assembly sequence. The assessment relies on upon how the fitness function is characterised.

3.4.8. Selection

The guideline behind genetic algorithms is basically Darwinian natural selection. The selection in GA is an artificial version of natural selection and it leads a GA towards likely in the search space. Selection gives the main impetus in a GA, and the selection pressure is

critical in it. At outrageous, if the selection pressure is very high, the pursuit will end rashly. At the other extraordinary, a low selection pressure indicates a slower than needed progress (Gen and Cheng 1997).

Three essential issues are included in the choice stage:

- Sampling space: selection may make another population for the following generation in view of either part of parents and offspring or all of them. The regular sampling space contains all offspring however simply part of the parents (Figure 3.7.) and different replacement techniques to abstain from offspring of lower quality than parents to be methodically selected (Holland 1975) were designed. When utilising an enlarged sampling space (as shown in Figure 3.8.), both parents and offspring have a similar possibility of going after survival. In addition, the expanded sampling space allows the utilisation of a high rate of randomness presented by the crossover and mutation. To maintain a strategic distance from an untimely merging the selection utilised for solving the ASP issue depends on the expanded sampling space.

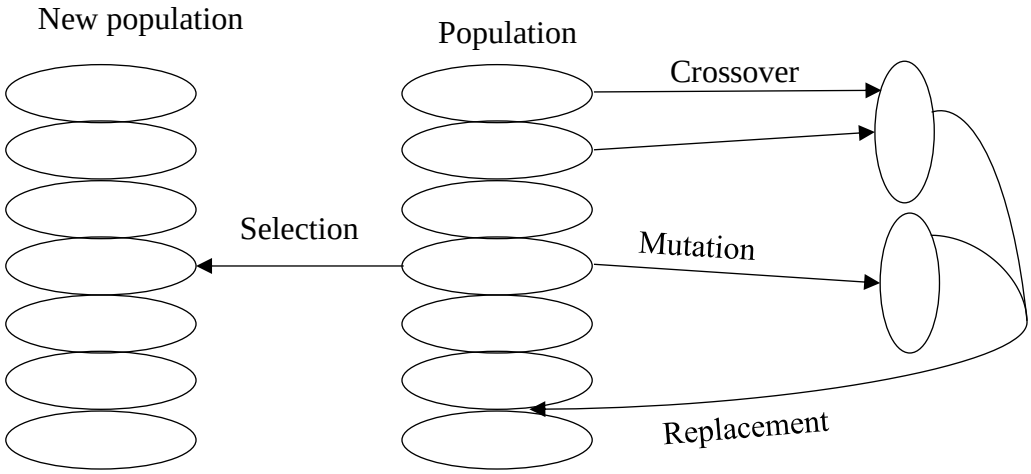


Figure 3.7. Selection performed on regular sampling space (Gen and Cheng 1997)

- Selection probability: concerns how to decide selection probability for every chromosome. Scaling and positioning mechanisms are utilised in order to keep up a sensible differential between relative fitness evaluations of chromosomes and to avert as well fast takeover by some super chromosomes. A static scaling is utilised as a part of the GA created for the ASP.
- Sampling mechanism: concerns how to choose chromosomes from sampling space. Three fundamental mechanisms are accustomed to sampling chromosomes:
    a) Stochastic sampling.
    b) Deterministic sampling.
    c) Mixed sampling.

The sampling system is utilised as a part of this theory is the stochastic sampling, related with the Holland's proportionate choice or roulette wheel choice. The selection probability for every chromosome is proportionate to its fitness value: a chromosome with fitness value f_i and with average fitness value of the population f_m is assigned f_i/f_m offspring. A string with a fitness value higher than the normal has a superior possibility of an offspring, while a string with a fitness value less than average has a lower opportunity to allow in the next generation. The proportionate selection assigns fractional number of offspring to strings.

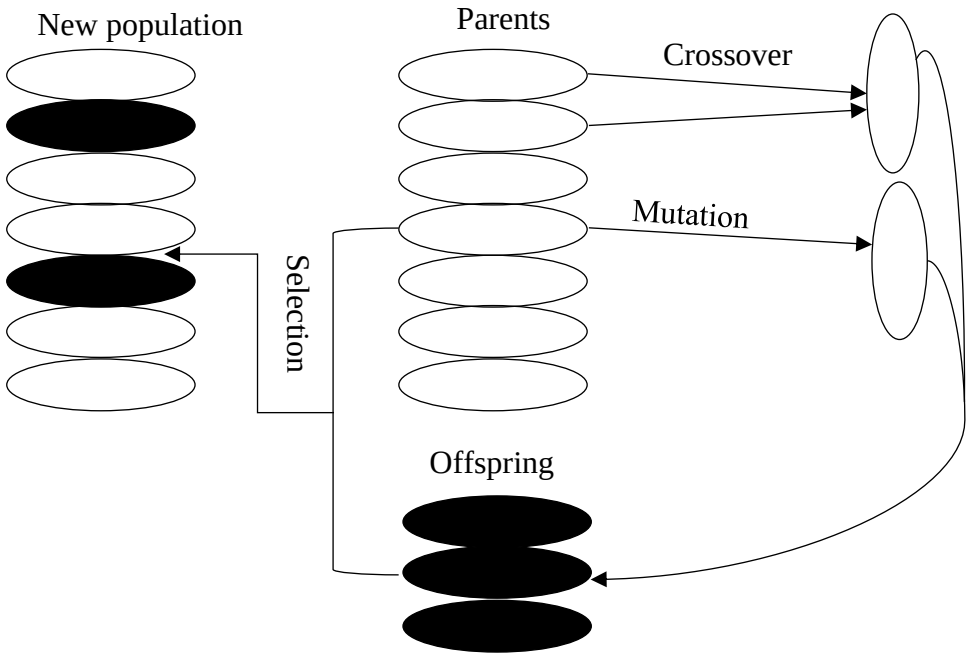


Figure 3.8. Selection performed on enlarged sampling space (Gen and Cheng 1997)

The weighed roulette shown below is a graphical representation as shown Figure 3.9. Every string is assigned an area of $2\pi f_i/f_m$. A string is assigned a posterity if a randomly produced number in the range 0 to 2π falls in the division relating to the string. The calculation rehashes the portion of posterity until all the cutting edge is made (Kazerooni 1997).

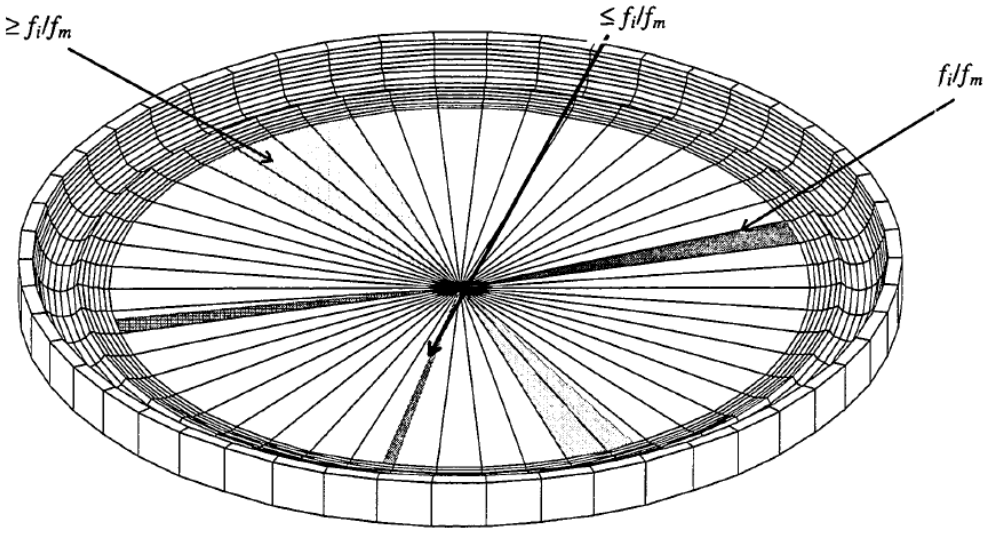


Figure 3.9. Simple reproduction allocates offspring strings using a roulette wheel

(Kazerooni 1997)

3.5. CONSTRAINTS IN ASSEMBLY

When attempting to solve the ASP problem constraints are essential. Erroneous approaches and, consequently, erroneous or incomplete results are brought up by any attempt to artificially simplify the problem by, sometimes, even partially ignoring constraints that are meaningful and important. This section will concisely present the concept of constraints, categories of constraints and their effect on the assembly process.

Constraints are relations in aspects of a product. The increase of the infeasible portion of the search space and the number of feasible solutions are limited by constraints. The search area which have more complex geometric shape yet often will become discontinuous and/or no longer convex. Consequently, the infeasible solutions are likely to be generated by the standard genetic operators, probably wasting the computational effort of the GA (Kowalczyk 1997).

Constraint can be illustrated in an example as in Figure 3.10. It shows an assembly of a ball pen product. The components are: c1-cap, c2-head, c3- tube, c4-ink (fluid), c5-body, then c6-button.

Let us consider the assembly plan is SLMC (Sequential, Linear, Monotone, and Coherent) then up to expectation c2 and c3 then c4 are to be assembled before c5.

Starting with c2 and, both components have to assembled together before c4 (ink). There is, consequently, a priority relation between those three components: c2, c3 and c4. On the other hand, starting assembly together with c1 is also impossible. c1(assembly 1) is in contact only with c5 the tube. Assembling c1 with c5 precludes the access because of c2, c3 and c4. As a result, c1-c5 can be done only after c5, c2, c3 and c4 were assembled (coherence condition).

If the first component to be assembled is c3, the next cannot be c1, c5 and c6, as there is no connection between them.

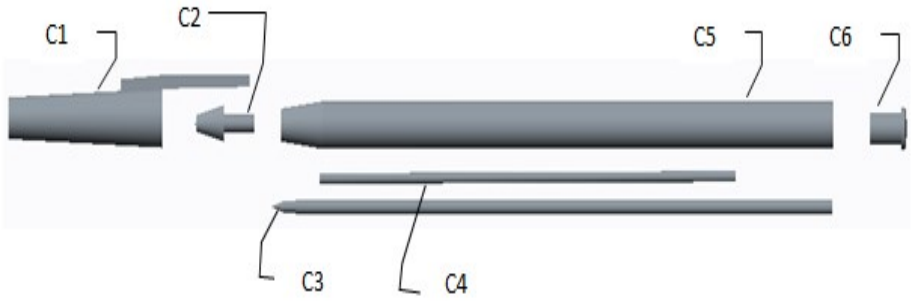


Figure 3.10. The ball pen assembly components (Fawaz and Qian 2017)

As the constraints above the assembly process may possibly start with c2, followed by c3, then the ink c4 is squeezed, the body c5 and then the button c6 to be inserted, and the cap concludes the assembly. Thus, assembly sequence is: c2, c3, c4, c5, c6, c1 Another feasible assembly sequence is c2, c3, c4, c5, c1, c6, which requires two changes in the assembly direction: c1 to c5 and c6 after c5.

From the above observations, conclusions can be drawn:

- There are a wide variety on constraints as this routinely leads to infeasible assembly sequences. Those constraints are acknowledged namely absolute or difficult constraints (Sebaaly and Fujimoto 1996c, Jones et al. 1998).
- Absolute constraints depend on accessibility over parts, and the assembly system used.

3.5.1. Absolute Constraints

Absolute constraints had been labelled into exceptional classes by a range of authors. Jones and Wilson (1996) and Jones at al. (1998) surveyed that associated constraints are categorized in accordance attributes:

1. **Obligation:** Constraints are absolute, both requiring (REQ) and prohibiting (PRH) certain services about assembly plans. Quality measures both maximise (MAX) then minimise (MIN) a scalar function.
2. **Scope:** The scope is concerned with a standard in conformity with the diagram on the assembly plan.
3. **Information required:** The relevant information need to be supplied to a standard for calculation at any given time. Some criteria want an entire plan to calculate, others require only provincial data – single assembly states or actions.

Internal constraints result from the geometry of the assembly components while external constraints emerge if the chosen plan should be completed in a robot assembly cell. (e.g. constraints prompted by way of the gravitational force, applied gripper, and many others.). The internal constraints taken into consideration are local geometrical feasibility, which defines the reparability of the involved subassemblies components, and global geometrical feasibility, derived in dexterity.

External constraints derived in grasp ability of the target component, task compatibility, grasp ability of the energetic subassembly, and assembly stability.

3.5.2. Handling Constraints in Genetic Algorithms

In this section and followed sub-sections will briefly explain optimisation constraints only during using GA, and these constraints are different of assembly sequence constraints of sloid components as shown in Figure 3.10. When using of GA to solve an optimisation issue under constraints, classic operators often yield infeasible offspring. This trouble turns into a scenario where the opportunity to reap a feasible assembly sequence via random generation of the assembly sequence is reduced rapidly as the number of components increase. This implies that genetic operator can probably produce an illegal offspring. Some strategies must be utilised to decrease the number of operations that required to be done within the genetic operators and simultaneously to keep the stochastic character of the GA. These strategies are rejecting, repairing and penalty.

3.5.2.1. Rejecting strategy

The Rejecting approach is a famous approach in GA. It discards all infeasible chromosomes generated at some point of the evolutionary manner. The approach may go reasonably properly when the feasible area is convex and constitutes a reasonable part of the whole seek space (Gen and Cheng 1997). In ASP, the search space is not always convex, viable solutions are scattered among non-feasible ones, and the ratio between feasible and infeasible solutions is extremely decreased, of the order of up to 10^{-18} for a 25 components product (Gen M. and Cheng R. 1996, Marian et al. 2003, Marian et al. 2006).

As a result, the utility of this approach is to maximise the chance to discard all infeasible chromosomes to be restricted to landscapes with populations of unrealistic dimensions. Using the reject approach become useful in an early level of the research, which is especially

beneficial for products exceeding 6-10 components (relying at the degree of connectivity among components).

3.5.2.2. Repairing strategy

The repairing strategy implies beginning with an infeasible chromosome and generating a feasible one out of it through a repairing process. The repairing method depends on the life of a deterministic restore process to transform an infeasible offspring right into a feasible one. The repairing approach is problem-dependent, and a specific repair algorithm must be evolved for every trouble. For some troubles the repairing technique is as complex as fixing the unique trouble.

The ASP is one of these problems and a repairing strategy to be carried out could be extremely complex. Because of the opportunity that any gene may be very likely to make a chromosome illegal or infeasible, the repairing method might have to be carried out again and again for each chromosome. There are also valid questions of what to do in a specific case: observe the repair strategy for a gene or for a group of genes, and in this example for what number of. Any solution has to be taken into consideration and tested, and the behavior and overall performance of a restore set of rules would be affected by the nature of the product and the scale of the problem.

The usage of a repairing approach may be considered at an earlier level of the research. Because of the severe collateral troubles implied and emerging while the approach turned into to be advanced, and due to the complexity of those troubles, this course became deserted.

3.5.2.3. Penalty Strategy

The penalty method from the rejecting and repairing techniques, which only consider points within the feasible areas. For vastly constrained problems infeasible areas take an incredibly important portion of the legal solutions and constraint management techniques that allow

movements through infeasible areas of the quest space may also produce most fulfilling outcomes faster.

The penalty method, in essence, transforms a limited trouble into an unconstrained one by penalizing infeasible solutions. The penalty term is brought to the objective function for any violation of the constraints. Basically, penalty is a feature of the distance from the feasibility area to the chromosome. The principal situation is how to determine the penalty time period if you want to strike a balance among the information preservation (retaining a few infeasible solutions) and the selective stress (rejecting some possible solutions) and void each underneath-penalty and over-penalty.

The problem of using the penalty method for the ASP is the impossibility to correctly define a penalty time period or function. In ASP, viable solutions are generally grouped in small clusters amongst the infeasible ones. It is, consequently, hard to define a penalty term that would discriminate between infeasible solutions.

This approach works with possible chromosomes with the aid of using custom-tailor-made genetic operators. Thus, this strategy is a whole lot greater reliable than another GA based totally on the penalty method (Michalewicz 1994). These methods of genetic algorithms are subjective on the amount of realization of the issue; well-known issues often have better, more unique approaches.

3.6. THE IDEA OF USING GA FOR SOLVING AS

GA is an optimisation method for solving assembly sequence optimisation problems due to its ability to offer a flexible way of defining constraints (Whitley 2014).

This research considers the idea of GA and their utilization to solve the AS problem. Specifically, it focuses on:

- a review of GA and explanation of their use to solve combinatorial issues of the AS;

- an investigation of different GA strategies and operators, evaluation and justification of their appropriateness and why they have been selected for solving the AS issues;
- types of constraints experienced in solving the AS problem and their impact on the search space;
- procedures to deal with constraints and the need to design a specific GA to solve the AS problem.

Genetic algorithms are appropriate to solve the AS problem. GA were selected in this research to solve the AS problem based on their classifications (Section3.2.), and especially because:

- they can simply deal with substantial search spaces;
- they are flexible in defining the constraints and arise them in a fitness function. This is mainly useful for AS where a quality function is hard to define;
- they are eligible algorithms to reach from a current solution in the search space to any further solution.

CHAPTER 4

THE REPRESENTATION OF ASSEMBLY SEQUENCES AS CHROMOSOMES

4.1. INTRODUCTION

The main aim of modeling assembly is to facilitate in developing a framework capable of representing and encoding any possible solution of the ASP difficulty as a chromosome. However, In GA, a chromosome can be used to represent an assembly plan that can be coherent or non-coherent, sequential or non-sequential, monotone or non-monotone and linear or non-linear. In addition to it, the likelihood to encode mechanisms with variable geometry and/or volume can be taken under consideration. Hence, the subsequent chromosomes should have a format that can be considered directly by the Genetic Operators.

Due to the presence of unexpected variety of possibilities observed in assembly, the issue is required to be analysed in detail, modelled and structured, so that all the required aspects of an assembly plan can be apprehended. Because of this reason, a modest representation of assembly sequences as chromosomes is required to be completed with relatively prior, extensive and modelling activity.

Preferably, it has been observed that for extensive search spaces, specifically for combinatorial issues, there needs to be a bi-unique mapping among the present entities within the spaces. On the other hand, the effective and influencing solution to a problem should present a sound and detailed demonstrations for problem states and transformations from one state to another for goal achievements. Hence, the identification of a problem has a considerable impact on the efforts required to find the solution (Nilsson 1980).

A good demonstration for combinatorial issues is required to have a little or at least controllable state space, which is not simple and easy for combinatorial issues as shown in

Chapter 3. Therefore, it is always important to focus on the respective stage that main obstacle in developing efficient algorithms to solve the AS optimisation problem for assembly.

4.1.1. Representation and modelling problems

It has been observed that there are two main representation and modelling problems linked with optimisation of AS with the use of GA: modelling of assembly sequences (indicated as chromosomes) and product modelling for assembly. Moreover, additional information is also important in the overall procedure of generating potential assembly sequence along with restrictions in assembly as priority relations.

Furthermore, it has also been observed that two models and the associated representations are interlinked, even though they are different issues but are interrelated closely. A feasible assembly plan identified as a chromosome is a sequence that satisfies all assembly constraints involved in assembly (indicated as precedence relations). However, the two indications are not completely separated as presented in the literature. There are several representations which are used in the planning of assembly in terms of (explicit representations) to be encoded as both assembly constraints and assembly sequences. Moreover, it has been found from the literature that there are several types of assembly plans (see Section 1.3. and Appendix 1). By taking into consideration in terms of the identification of assembly sequences, the assembly plans can be distributed in both sequential and non-sequential assembly plans. On the other hand, an assembly plan can also consist of a non-sequential aspect which may restrict the entire scope of the issues of ASP.

This chapter is structured as follows: the following section provides an overview of the state of the art in the representations used in assembly sequence planning, for both the constraints and assembly sequences. Then the representation and modeling of assembly sequences is indicated for SLMC sequences. The last sections simplify this representation for non-SLMC sequences are trying to integrate any identified assembly plan/sequence.

4.2. ASSEMBLY SEQUENCE PLANNING AND OPTIMISATION

It has been observed that a product may have a number of potential assembly sequences, and combinatorial explosions can intensify greatly with components involved. However, it is not possible to identify each sequence. Therefore, it is important to develop an efficient and systematic process to the possible solutions in an effort to select the best and effective solution based on the resources available.

4.2.1. Assembly for products

Conducted by Ben-Arieh (1994a) and Choi et al. (1998), there are three main categories to depict assembly for products: language-based approach, graph-based approach, and advanced data structure approach:

Language-based approach - Language- based approach refers to part assembly description language mainly oriented towards identifying the parts including the assembly and the necessary assembly operations. For example, the assembly is described by both its physical and geometric properties. The assembly instructions may be divided into three types: tools statement, state change instructions and fastener statements.

Graph-based approach - graph based approach is used for extended assembly analysis for more in-depth derivation of information with a focus on the assembly process and little on the properties of the components or assembly operations. In addition to it, the graph-based approach is based on informative of such as CAD-database or information specified by user. There are several graph based approaches: directed graphs AND/OR graphs (Homem de Mello and Sanderson 1990c), and connectivity graphs (Shpitalni et al. 1989), Petri Nets (Thomas et al. 1996), and hierarchical partial order graphs (Shin et al. 1995, Lee 1994), liaison diagrams (De Fazio and Whitney 1987), precedence diagrams, assembly constraint graphs (Wolter 1988 and Wolter 1990a) and interference graphs (DeFloriani and Nagy 1991).

Advanced Data Structure approach - Advanced data structure approach utilises designed data structure in an effort to capture a detailed assembly data using a hierarchical data structure. This sums up the geometric and topological information considering the connections that lead to generate the complete assembly.

4.2.2. Assembly Plans

This section presents several different descriptions and definitions that are available for assembly plans and certainly implied on different representations of the resulting assembly plans. The definitions determine the system and the method of presenting the assembly plans and the representations consists precedence relations among assembly operations. They are demonstrated using the example presented in Figure 4.1. and Figure 4.2. Understanding the system of assembly plans and all presented figures in all sections will play a key role in determing and solving the assembly sequence problem of the research case studies (see Chapter 6).

Assembly plans are to gain feasible assembly sequences and assembly operations. Figure 4.1 shows a four-part assembly (A) and a graph of liaisons (B). It demonstrates an assembly sequence as C - Cap, S - Stick, R - Receptacle and H – Handle:

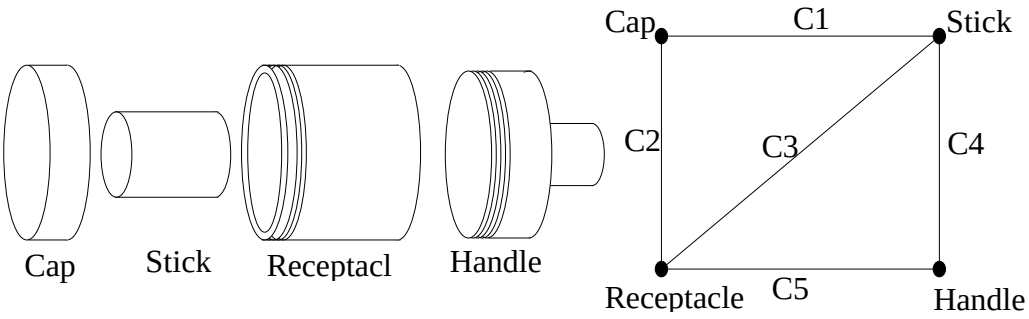
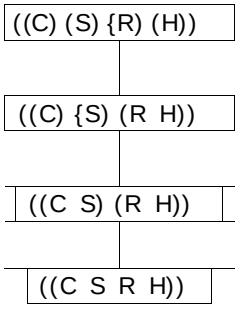


Figure 4.1. A four-part assembly (A) and a graph of liaisons (B) (Homem de Mello and Sanderson 1990, Homem de Mello and Sanderson 1991a, Homem de Mello and Sanderson 1991b)

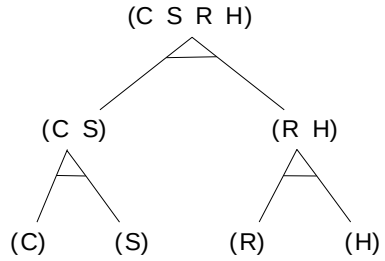
A State Sequence- consider an assembly plan as an entire sequence of join operations, each of which is combined in two specific assemblies as demonstrated in the Figure 4.2A. Assembly states (concerning about monotone plans) are to be identified by the partition of the specific part set combining sets of parts that are assembled already. For the assembly of four parts, as an example, the basic state would be {{C}, {S}, {R}, and {H}}, and therefore the final position will be {{C S R H}}, along with all the parts leading towards one assembly. Moreover, the assembly plan can be indicated as a sequence of such states in which each operation is combined with two partial assembles into one (n-1 operations). The state sequence indicates the operation sequencing in parallel sub-assemblies.

A Partial Assembly Tree- considers an assembly plan as a recursive decomposition of the assembly into two main subsets that continues until the entire parts have been separated as demonstrated in the Figure 4.2B. Each node indicates a partial plan of the assembly. The root node of the tree indicates the entire assembly and the leaves represent single parts, where each node leads to two children that indicates the two sub-assemblies and the components that are combined together to construct the product/assembly demonstrated by the node.

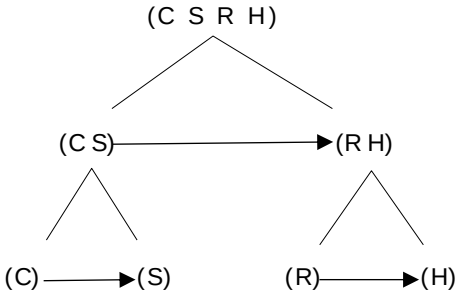
A Sub-Assembly Tree- takes into consideration an assembly plan with regards to a sequence of operations which eventually leads to insert subassemblies or parts into a base part of a fixture as exhibited in the Figure 4.2C. In a sub-assembly tree, each node leads to a sub-assembly where each lead to a part. Moreover, the children of a sub-assembly node include all the subassemblies and parts that are inserted with the subassembly, in the respective order within which they are inserted.



A. (a formal sequence)



B. (a partial assembly tree)



C. (a subassembly tree)

Figure 4.2. A plan for structure the four-part assembly (shown in Figure 4.1.) (Wolter 1991, Golabi 1996))

Wolter (1991) conducted a study that grouped the approaches considering the identification of sets of assembly plans in:

- Constraints Based Representations- that leads to identify each and every details that cannot be done, for example part A cannot be considered as being mate to part B after parts B and C are already mated;
- Enumerative Representations that indicated every minute details that can be possible, for example the assembly {A, B, C} can be constructed from the partial assemblies {A, B} and {C} or fom the {A, C} and {B} or from {A} and {B, C}.

From the understanding of the entire process, it is observed that constraints-based representations grow smaller and enumerative representations grow larger. However, some of the systems considered in optimising and solving the ASP issues operate completely with

constraint-based representations (Wolter 1988), while some undertakes with enumerative indications (Homem de Mello and Sanderson 1989), and few other systems undertakes both the representations at the same time. However, the study conducted by the researcher's grouped representations of mechanical assembly sequence in both implicit and explicit (Homem de Mello and Sanderson, 1991a, Homem de Mello and Sanderson 1991b). The next two sections review critically and analysis in detail the present representations that are grouped in two characteristics, such as implicit and explicit representations. However, the particular constraints and qualities are indicated.

4.2.3. Explicit Representations in Assembly Planning

Explicit representation leads to a direct mapping referring to the assembly tasks of components. An integrated form of state sequence was developed by Bourjault (1984) indicating as a tree. Figure 4.3. demonstrate the Bourjault's representation state of sequences. The root node indicates the unassembled and initial state. The nodes on the other hand indicate the established links and the edges demonstrate the transformation from ones stated to another state from rank n to n+1 or assembly. However, any path originating from a root and leading to leaf node points towards a feasible assembly sequence.

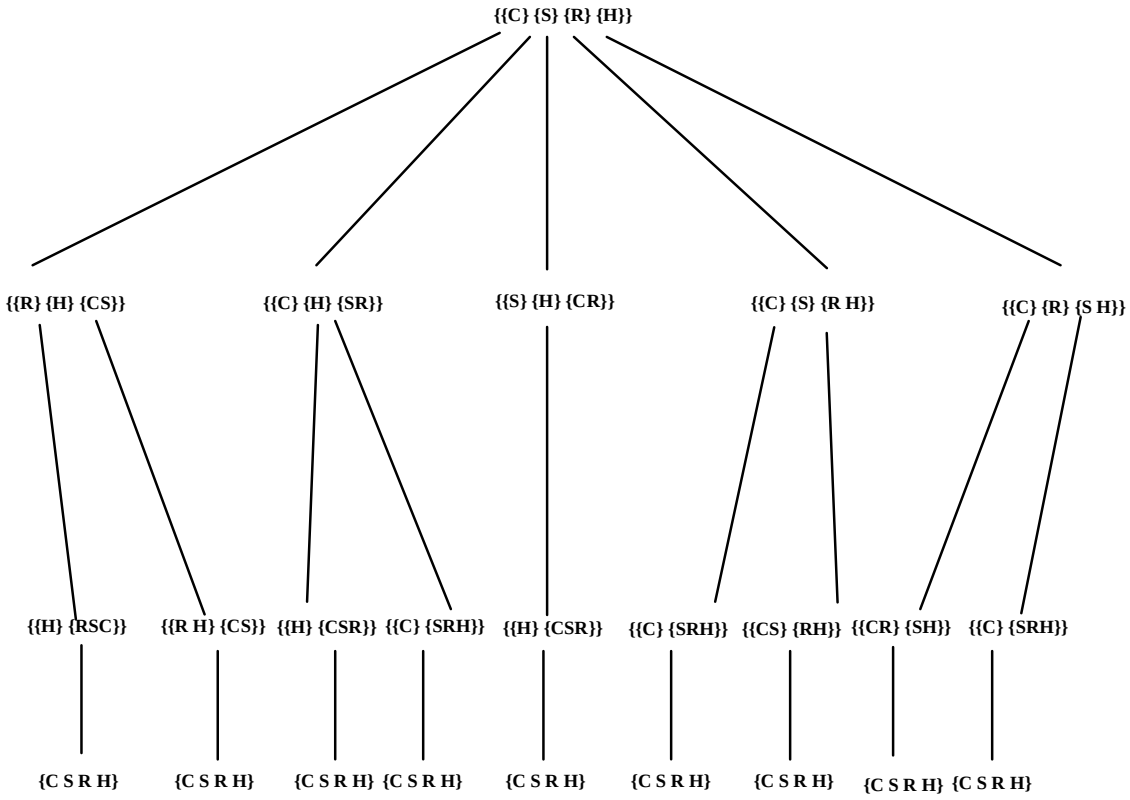


Figure 4.3. Bourjault's representation of all assembly sequences (Golabi 1996)

The directed graph was initially suggested by De Fazio and Whitney (1987), in an effort to explicitly indicate the assembly sequence. Provided with an assembly whose connection graph is (P, C)- in which P refers to the set of nodes and C points towards the set of edges. A directed graph can be taken under consideration to indicate the set of all the possible assembly sequences (Homem de Mello and Sanderson 1991a, Homem de Mello and Sanderson 1991b). Figure 4.4. leads to stable state partitions of the set P. The edges representing in the directed graphs are reflected as ordered pair of nodes leading to feasible state transformations.

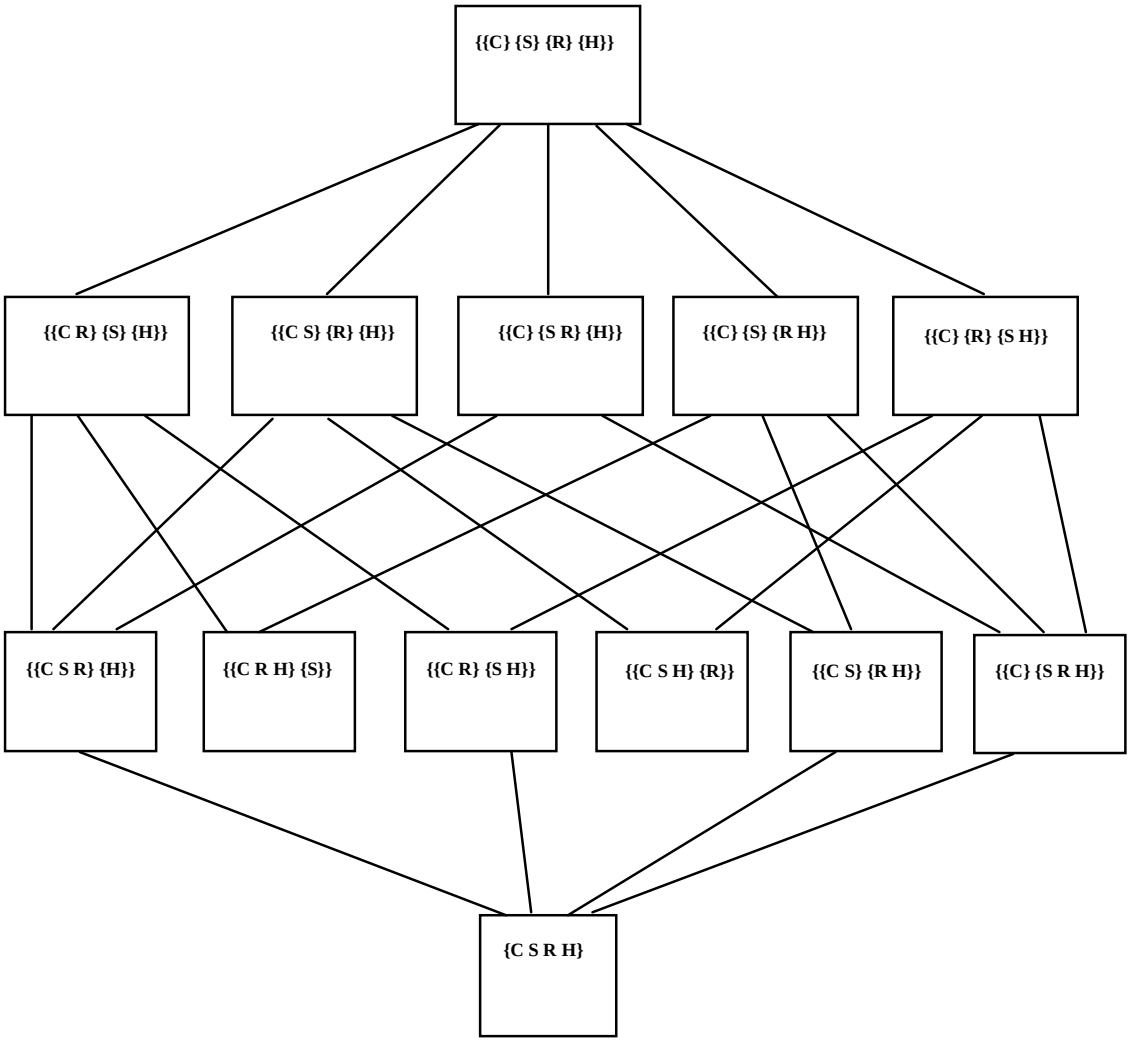


Figure 4.4. Directed graph of feasible assembly sequences using parts (for the assembly shown in Figure 4.1.) (Golabi 1996)

Furthermore, a path represented in the directed graph concerning feasible assembly sequences starting from the first node {{C} {S} {R} {H}} towards the terminal node {{C S R H}} leadings to a feasible assembly sequence. In the same way, Figure 4.5 illustrates the direct graph of feasible assembly sequences in relation to the product shown in Figure 4.1. The state of assembly indicates identified connections and each connection is identified by a black rectangle. Edges join every state to all the states that are reachable from it.

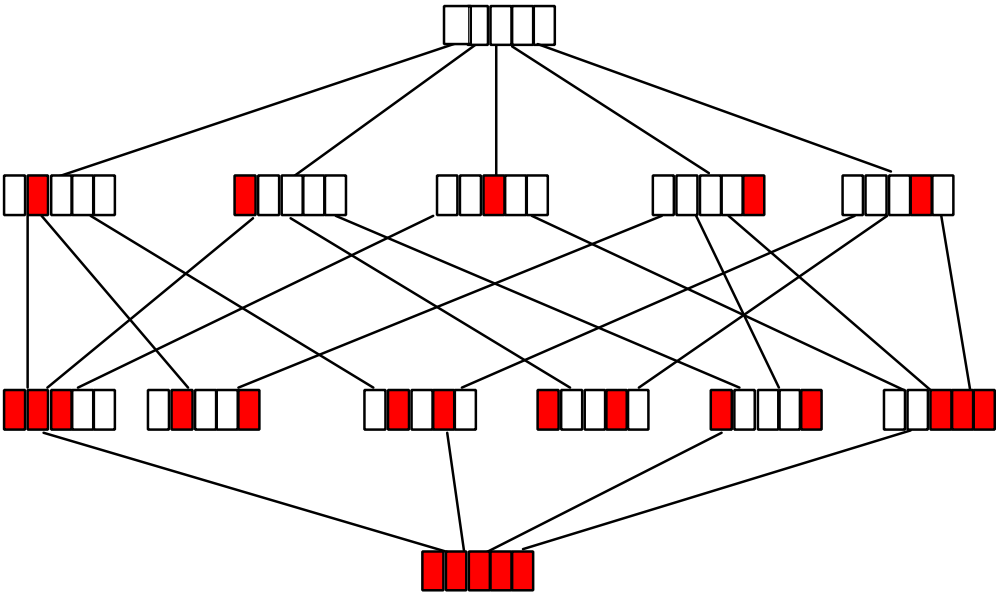


Figure 4.5. Directed graph of feasible assembly sequences using liaisons (for the assembly shown in Figure 4.1.) (Arthur et al. 1990)

AND/OR graphs are possibly the most widely used in representing the assembly sequences in an explicit manner (Homem de Mello 1989, Homem de Mello and Sanderson 1990, Homem de Mello and Sanderson 1991a, Homem de Mello and Sanderson 1991b). The nodes in the AND/OR graph as indicated in the Figure 4.6. are linked with the subsets of parts that lead to a stable subassembly. Nevertheless, the root node (node 1, Figure 4.6.) is linked with the group of parts that leads to the entire assembly. Among the four hyper-arcs, each of them corresponds to a particular way within which the entire assembly be taken apart and points towards the two nodes that are linked with the sets of parts that explains the subsequent subassemblies. In the same way, the remaining nodes in the graph leave a hyper-arc for every possible way through which their subsequent subassembly can be taken into parts. Path in the AND/OR graph {{C S R H}} as its initial node and {C}, {S}, {R}, {H} as terminal nodes are a feasible assembly tree of that specific assembly. An assembly tree consists of partial order within its hyper-arcs: where hyper-arc h_i is considered to be preceding hyper-arc h_j, if there is a node *nk* in the assembly tree considering the fact that h_i is incident from *nk* and h_j is incident

to *nk*. Furthermore, it is observed that one sequence of the hyper-arc from an assembly tree is persistent with this fractional order. Moreover, each sequence of the hyper-arcs which is persistent with the fractional order leads to a potential assembly sequence as mentioned in the study by Homem de Mello and Sanderson (1991a). On the other hand, any stable subassembly that is linked can be made up of the components that are found to be only once in the AND/OR graph, even at the stage where it is found to be an outcome of different disassembly operations. AND/OR the graphical representations are the main foundations for other derived and related representations.

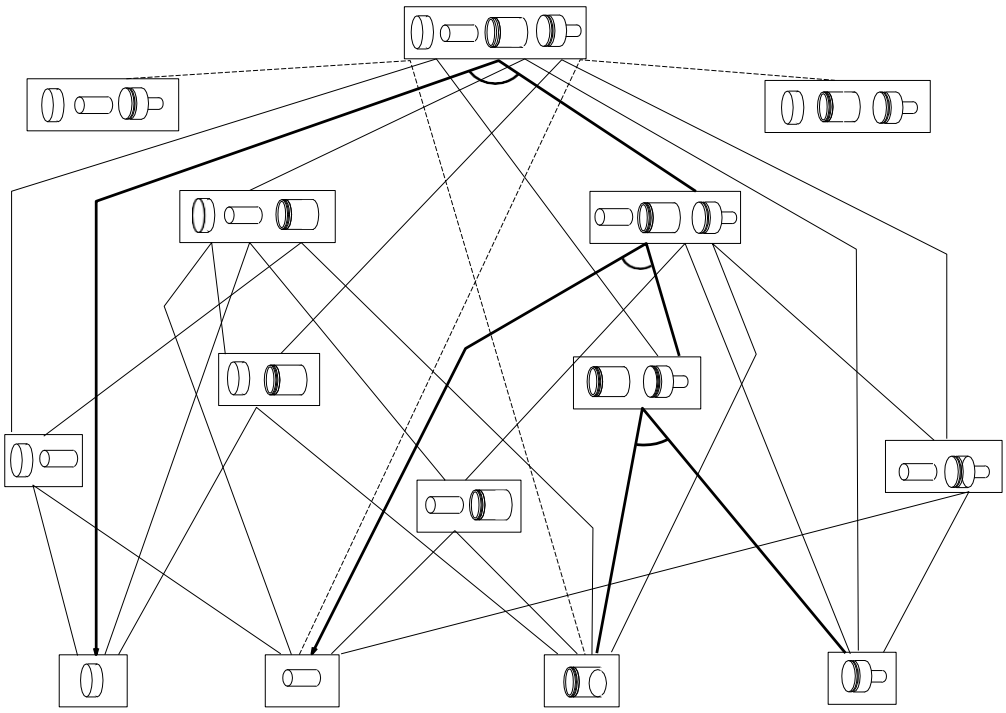


Figure 4.6. AND/OR graph of assembly sequences of Figure 4.1. using AND/OR graph (Homem de Mello and Sanderson 1990, Homem de Mello and Sanderson 1991a, Homem de Mello and Sanderson 1991b)

Gottipolu and Ghosh (1997) developed an Assembly Sequence Graph (ASG), which is found to be an explicit graph representation linked to the AND/OR graphs and the Liaison Sequence Graphs (LSG), in an effort to sustain the benefits of the schemes. However, the nodes in the

ASG are considered to be the subsets of the parts set P that categorises the possible subassemblies. Thus, each node leads to a subassembly. Moreover, the nodes are indicated by the boxes where each box represents N cells leading to the N parts in the assembly. In addition to it, the blank cell entails that the leading part is not directly assembled whereas a marked (hatched) cell signifies that the leading part is already been assembled. At the top, which is the first level, there are N boxes in which each box represents one marked cell indicating all the individual parts of the assembly within an unassembled state, Figure 4.7., for the product in Figure 4.1.

The disadvantage of AND/OR graph that it cannot represent all feasible assembly sequences for real size problem, hence, the method is restricted to reduced size or heavily constrained issues. For instance, through this research case studies, AND/OR graph cannot be used for the first case study (engine pump valve) due to number of components, while in the second case study (ball pen) AND/OR graph is possible to be used (see Chapter 6).

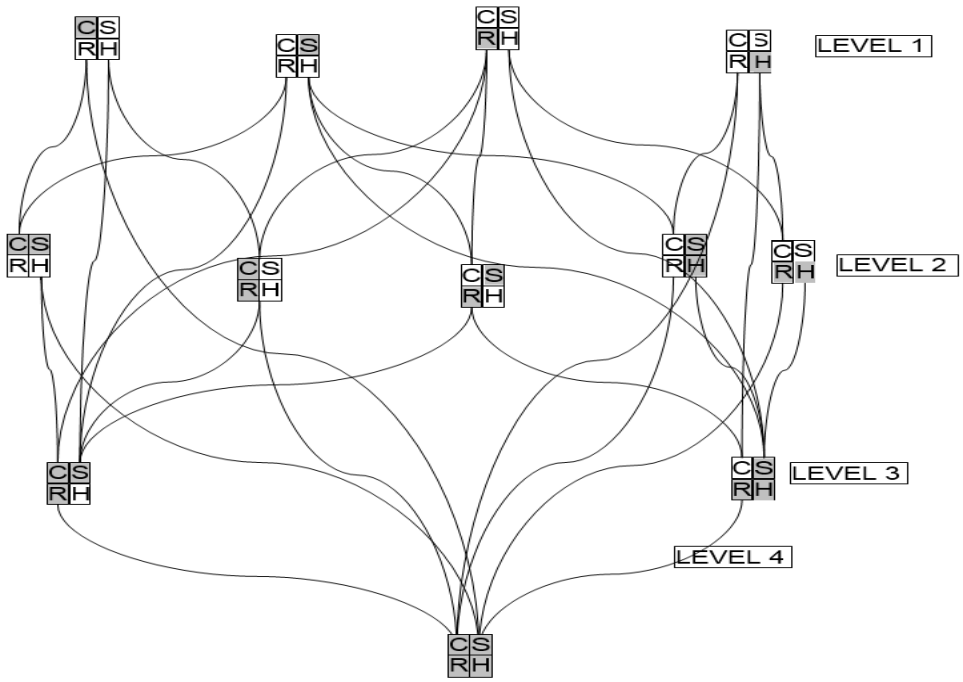


Figure 4.7. Assembly sequences graph (ASG) from Figure 4.1. (Marian, Luong and Abhary 2003)

At the bottom box, it represents the entire assembly. At level L, the box includes all the "L" marked cells, i.e. level L consists of all the subassemblies containing L components. The lines linked to the boxes indicate all the feasible assembly state transformations- assembly tasks. One assembly task links to two subassemblies holds two arcs corresponding to the subsequent subassemblies, one from every constituent subassembly. The pair of arcs can be stated as a hyperarcs leading to an assembly task. The hyper-arcs within the ASG can be linked with the weight elements for instance assembly task time, degree of difficulty of assembly functions, assembly costs and subassembly stability etc. The weighted ASG can be utilised for the assessment of assembly plans (Gottipolu and Ghosh 1997). A mutual disadvantage of explicit representations is their size. Even though the most compact of representations, the AND/OR graph has among n*(n+1)/n and 2^n -1 nodes based on the level and degree of connectivity to be identified, stored and linked. They linked to 120 to 32767 nodes for a 15-part assembly, correspondingly 205 to 1048575 for a 20-part assembly that turns to be difficult or incredible to store and manage. Considering assemblies with huge number of parts, the AND/OR graph is quite large. In such cases, parts can be clustered hierarchically into subassemblies. However, affectedly clustering parts, the size of the AND/OR graphs would be reduced. According to Homem de Mello and Sanderson (1990) it would not be considered all the divergent ways within which the parts in clusters could be assembled.

4.2.4. Implicit Representations in Assembly Planning

The implicit identifications include a combination of conditions that needs to content by the assembly sequences. According to Homem de Mello and Sanderson (1991a) (1991b) that if the states of the assembly process are indicated by L-dimensional binary vectors, then a combination of logical expressions can be utilised to encode the directed graph of possible assembly sequences.

If $\Xi_i = \{\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_{Ki}\}$ represents the sets of states due to which the *i-th* connection can be

recognised without impeding the completion of the assembly, the identified condition for the *i-th* connection is found to be the logical function:

$$F_i(x) = F_i(x_1, x_2, x_3, ..., x_L) = \sum_{k=1}^{K} \prod_{i=1}^{L} \gamma kl \tag{4.1}$$

where the product and the sum are the logical operations AND/OR respectively, *L* is found to be the number of liaisons in the liaisons graph and γkl is either the symbol *xl* if the *l-th* element of $\underline{x_k}$ is true (T) or the symbol x_l if the *l - th* component of $\overline{x_l}$ is false (F).

$$F_i(x_k) = T \quad \text{only if } \underline{x_k} \text{ is an element of } \Xi_i \tag{4.2}$$

Any assembly sequence that is represented as an ordered sequence of state is ($\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_N$) and whose identification as an ordered sequence o subsets of connections is ($\gamma_{1,}\ \gamma_{2,}\ \ldots,\ \gamma_{N-1}$) is possible if it is such that if the *i-th* connection is identified in the k-th task (i.e. $c_i \in \gamma_k$), then $F_i(x_k) = T$.

Therefore, the set of establishment conditions is a correct and complete representation of assembly sequences (Marian et al. 2003). The establishment conditions obtained from the AND/OR graph for the assembly shown in Figure 4.1. are:

A)
$$F_1(\underline{x}) = \bar{x}_1.\bar{x}_2.x_3.x_4.x_5 + \bar{x}_1.x_2.\bar{x}_3.x_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.x_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.x_2.\bar{x}_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.\bar{x}_3.\bar{x}_4 + \bar{x}_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.\bar{x}_5$$

B)
$$F_2(\underline{x}) = \bar{x}_1.\bar{x}_2.x_3.x_4.x_5 + x_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.x_3.\bar{x}_4.\bar{x}_5 + x_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.\bar{x}_3.\bar{x}_5 + \bar{x}_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.\bar{x}_5$$

C)
$$F_3(\underline{x}) = x_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.x_5 + \bar{x}_1.x_2.\bar{x}_3.x_4.\bar{x}_5 + x_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.x_2.\bar{x}_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.x_5 + \bar{x}_1.\bar{x}_2.\bar{x}_3.x_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.\bar{x}_5$$

D)
$$F_4(\underline{x}) = x_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.x_5 + x_1.x_2.x_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.x_5 + \bar{x}_1.\bar{x}_2.x_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.\bar{x}_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.\bar{x}_5$$

E)
$$F_5(\underline{x}) = \bar{x}_1.x_2.\bar{x}_3.x_4.\bar{x}_5 + x_1.x_2.x_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.\bar{x}_3.x_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.x_3.\bar{x}_4.\bar{x}_5 + \bar{x}_2.\bar{x}_3.\bar{x}_4.\bar{x}_5 + \bar{x}_1.\bar{x}_2.\bar{x}_3.\bar{x}_4.\bar{x}_5$$

For example, the first establishing state ($F_1(x)$) matches to the actuality that the only conditions in which assembly c_1 (i.e., the assembly among the cap and the stick) can be established without prevent the completion of the assembly are either the condition in which no assembly has been determined (node 1 in Figure 4.4.), or the condition in which only assembly c_2 is determined (node 2), or the condition in which only assembly c_3 is determined (node4), or the condition in which only assembly c_5 is determined (node 5), or the condition in which only assembly c_2 and c4 are determined (node 9), or the condition in which only assembly c_1 and c_2 are determined (node 12). It should be noticed that there is no term matching to the condition in which only assembly c_4 is determined (node6); but while it is feasible to determine assembly c_1, the resulting condition (node 10) is a dead-end from which the assembly cannot be completed.

De Fazio and Whitney (1987) used priority relationships as an intermediate representation in their processes for the generation of all the assembly sequences. However, two main kinds of precedence relationships can be taken under consideration to identify assembly sequences (Homem de Mello and Sanderson 1991a).

4.2.4.1. Precedence Relationships between the Establishment of One Connection and States of the Assembly Process

Considering an assembly sequence whose representation is done as an ordered sequence of subsets of connections is ($\gamma_1, \gamma_2, \ldots, \gamma_{N-1}$) that actually satisfies the precedence relationship

$$c_i \rightarrow S(x) \tag{4.3}$$

If

$$S(\underline{x}_k) \Rightarrow \exists l \left[(l < k) \wedge (c_i \in \gamma_l) \right], \quad \text{for k=1, 2, ...N} \tag{4.4}$$

Where $c_i \rightarrow S(\underline{x})$ points towards the establishment of the i-th connection that must precede any

state *s* of the process of assembly used for what the value of the logical function *S*(*x*) is true.

4.2.4.2. Precedence Relationships between the Establishment of One Connection and the Establishment of another Connection

An assembly sequence that is identified as an ordered sequence of binary vector is $(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_N)$ and it is represented as an ordered sequence of connection's subsets as $(\gamma_{1,} \gamma_{2,} \ldots, \gamma_{N-1})$ that actually signifies the precedence relationship $c_i > c_j$ if $c_i \in \gamma_a$, $c_j \in \gamma_b$ and $a \leq b$. Here, $c_i > c_j$ indicates that the establishment of connection c_i must precede the establishment of connection c_j.

However, each possible assembly sequence of a given assembly is uniquely categorised by the logical expressions based on the conjunction of precedence relationships among the establishments of the connections with one another. Moreover, the disadvantages of such approach originate from the identification of the establishment conditions which is neither straightforward nor it is easy to use. In relation to this, Shpitalni and Elber (1989) has represented a structure that is consisting of four bodies, each body is represented by a CSG (Constructive Solid Geometry). Also, provide such connectivity or supportive graphs to identify relations among the structure's bodies (components) that are required to be assembled. The main emphasis of the support graph can be considered as a directed graph that indicates internal connectivity relations between the K components B(1)..B(K) of the related structure. The support graph can be explained as follows:

- Each component is linked with the structure that is indicated by a single node in the graph.
- A directed arc from B(j) to B(i) is present only and if B(i) is directly sustained by B(j) (i.e. B(i) and B(j) do not intersect each other).

A structure and its Z+ connectivity graph (i.e. connectivity throughout the +Z axis) is indicated in the Figure 4.8. a and b. moreover, three kinds of nodes are taken under consideration in the connectivity graph.

- A regular node along with both incoming and outgoing arrows signifies a regular component that also supports other components and is also supported by several other components.
- A sink node refers to a node with only incoming arrows and not the outgoing arrows. The sink node indicates a component that is supported by other components, but it is not supported by any other components such as B4.

It can also be seen that a source node refers to a node that is only represented as outgoing arrows. The source node actually supports other components but itself is not supported by other components. Therefore, to generate a disassembly sequence, the connectivity graph throughout the +Z axis is established. In the same manner, the graphs for remaining directions can also be established as required. Taking in view of the disassembly along with +Z axis, the significant candidate considered to be disassembled is a factor whose node in the graph of connectivity is a sink node. i.e., it does not provide any support to any other structure of the graph. In case where component can be removed if a collision-free path can be identified for it and its removal would not lead instability, it is removed, and the graph of connectivity is also updated. Figure 4.8.c-e. The breaking lines represent the body to be removed at every stage. If the selected candidate is not possible to remove, the system makes an effort to opt for any other candidate.

The disadvantage of the approach is based on the representation of the connectivity throughout the axis. It is not easy to work even throughout the triple axes of coordinates, and the identification can turn out to be unusable.

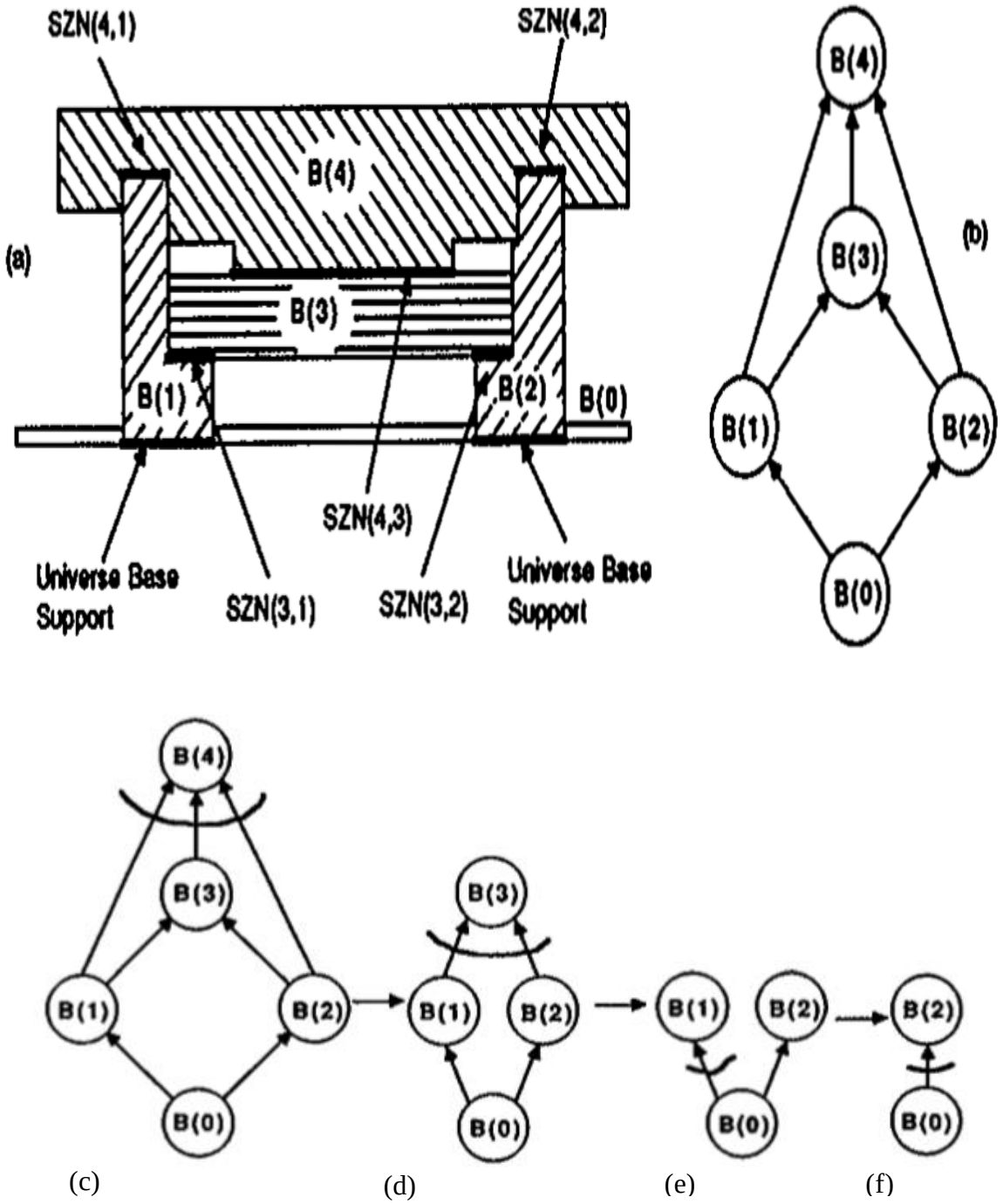
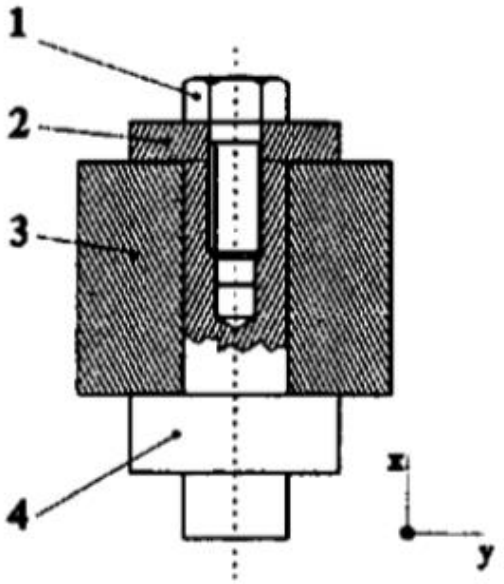


Figure 4.8. (a) Structure, (b) its +Z connectivity graph and (c, d, e and f) and the representation of assembly states (Shpitalni and Elber 1989)

Sebaaly and Fujimoto (1996c) stored the information regarding the product in a very compact or an implicit form. In an effort to overcome the constrained character of the ASP, the complete place of search involves all the feasible combination of parts, i.e. both the feasible and non-feasible sequences, which are grouped into families of same sequences, in which each family

consists of only one feasible sequence satisfying the problem constraints.

Dini and Santochi (1992) have developed a mathematical model for a product used for the automatic identification of disassembly/assembly sequences and also for the identification of subassemblies. They utilised the contact, interference and connection matrices where each one evaluates along with the 3 Cartesian directions, x, y and z of the CAD space, hence requiring 9 matrices. Moreover, the interference matrix A_k is considered to be the square matric of order *n*-considering an *n*-element product in which a_i, j=1, if the element e_i interfered with e_j element while the translation with +k (k= x, y, z), otherwise a_i, j=0. Traditionally always a_i, j=0. However, the contact matrix is considered to be the square matrix of order *n*, in which b_i, j=1 if e_i is in connection with e_j along +k, on the other hand b_i, j=0. Traditionally, b_i, j=0. The connection matrix is considered to be the square matrix of an order *n*, where c_i, j consider a numerical code which is the process of the types of the connection among e_i and e_j with k (e.g. c_{ij}=1 for a looped connection, where e_i can be considered as disassembled, c_{ij}=-1 for threaded connection, in which e_i cannot be considered as disassembled as per Dini and Santochi (1992). Throughout the generation of a right disassembly sequence, the code can be separated from the element that can be disassembled. However, a process of utilising the information given in this model among an element and other elements is able enough to present feasible assembly sequences. Figure 4.9. shows an example of a product, along with its interference, connectivity and contact matrices. This identification is restricted by the number of directions through which the disassembly can be taken under consideration, i.e. 1. In addition, it also takes into consideration disassembly gained from the linear translation of components.



$$A_x = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$ Interference Matrix

$$B_x = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$ Contact Matrix

$$C_x = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$ Connection Matrix

Figure 4.9. Example of a product with its matrices (Dini and Santochi 1992)

4.3. SLMC ASSEMBLY SEQUENCES

In a SLMC (Sequential, Linear, Monotone and Coherent) manner (the number of assembly operations (m) equals the number of components (n)), an assembly sequence can be encoded in a chromosome which is a permutation of product components. A gene in locus *j*, *j*=1 and n computed typically to the right from the left encodes the addition of the leading components in the *j*-th step and any partial chromosome with k genes, *k*=1, ... n indicated an assembly state, in which the first k components are assembled in a partial assembly and all their liaisons are developed. A component that is encoded with a gene will come out in the chromosome only for once. The remaining constraints apply, an n-term sequence of components of the assembly that can be infeasible or illegal chromosome. However, a simple n-term sequence is considered to be an illegal chromosome even when it is not a permutation. In this case a component number is found to be more than once, which does not include all the components that exists, for instance the torch in Figure 4.1. a1-a1- a3-a4-a4-a6-a7-a8-a9 is an illegal chromosome: a1 and a4 appear

twice and a2 and a5 do not appear at all). However, a permutation is found to be legal chromosome and assigns a tentative assembly sequence, where all the components exists and is possible that that assembly cannot be realised due to the existence of the constraints. (e.g. geometry, unreachable positions – a1-a2-a3-a4-a5-a6-a7-a8-a9 is infeasible because a3 cannot be assembled to a2). In addition, a feasible chromosome encodes a feasible assembly sequence. It is found to be constrained permutation which compiles all the assembly particular conditions (e.g. a4-a3-a2-a1-a5-a6-a7-a8-a9 is a legal and feasible chromosome, it complies with all constraints).

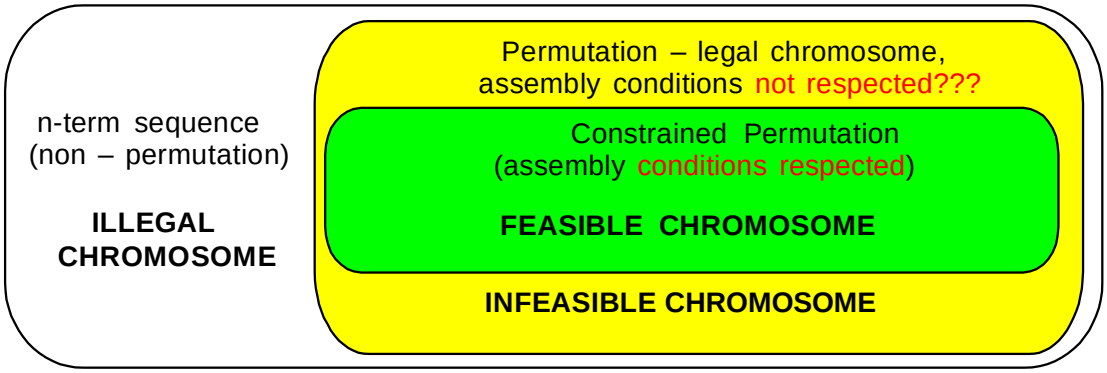


Figure 4.10. Relations between chromosomes and assembly sequences (Marian et al. 2006)

4.4. MODELLING AND REPRESENTATION OF NON-LINEAR ASSEMBLY SEQUENCES

Under this section, the main emphasis is on representation and modeling of non-linear assembly sequences, subassemblies and assembly plan including components. However, artificially, a gene can encode the inclusion of subassembly to the partial assembly. Therefore, considering this case the chromosome indicates a non-linear assembly sequence. As shown in Figure 4.11, a1, a2, a3 and a4 can be considered as a subassembly. These can be considered as a group in a subassembly that can be known as A. However, the subassembly A can be considered as a complex component. Taking into the consideration, the assembly process regarding the

components in the assembly A have been considered before the assembly process of the flashlight.

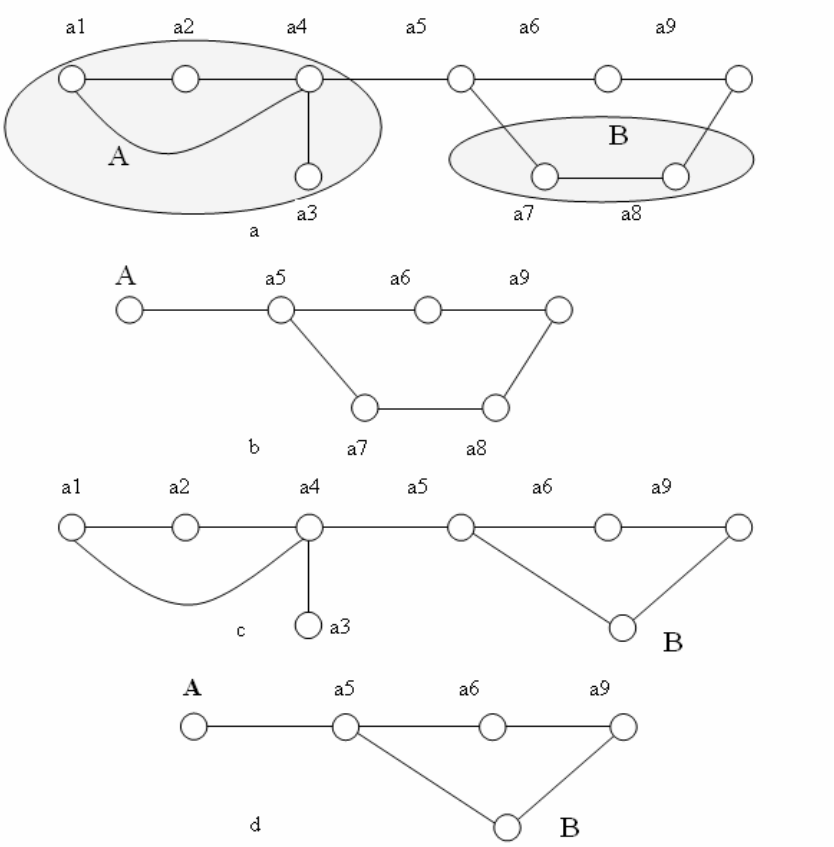


Figure 4.11. The graph of liaisons of the flashlight in Figure 4.1.

The relationship among the components in A are built when subassembly A is done. Therefore, they are not significant at the initial phase of adding A to the rest of the product. Only the related links between components in A and components outside A are considered. The edges a1-a2, a2-a4, a3-a4 are internal to A, i.e. among components within the assembly A. However, liaisons are built when A is assembled, therefore they are no longer related when A is added to the remaining torch. In addition to it, the edge among a4 and a5 is between component within A (i.e. a4) and component outside A (i.e. a5) and its identified when A is added to the remaining product. In fact, this edge is between the rest of the product and the subassembly. Two examples related to assembly sequences for the flashlight includes A as a subassembly which is made earlier are: - A-a5-a6-a7-a8-a9 and - a5-A-a6-a7-a8-a9, if a7 and a8 are assembled and added to the

flashlight as a subassembly B, the edge a7-a8 is internal to the subassembly B and edges a5-a7 and a8-a9 are between a component in B and a component outside B.

There are two examples of assembly sequences in terms of flashlight, consisting of B as a subassembly made earlier, which are:

- a4-a3-a2-a1-a5-a6-B-a9 and - a5-a6-B-a9-a4-a3-a2-a1.

Moreover, an assembly sequence can be consisting of two or more subassemblies. The two instances of assembly sequences for the flashlight includes A and B as subassemblies made earlier are:

- B-a5-a6-a9-A and - a5-A-a6-B-a9.

However, the modelling of an assembly sequence that consists of non-linear component includes:

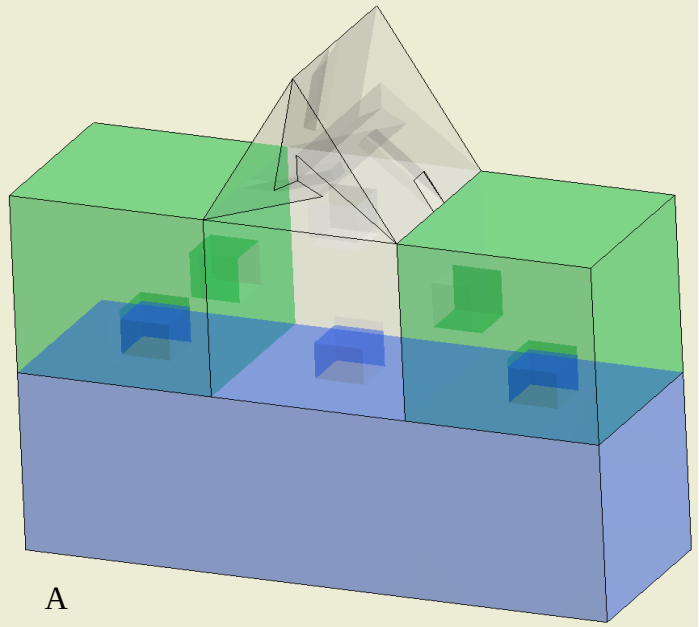
- Selection of the components comprised in each subassembly that is to be assembled *as is*.
- Encoding of each subassembly as a complex component that received new name.
- Encoding of each subassembly as a vertex in the graph of liaisons.
- Encoding of each subassembly as a gene in the chromosome.

4.5. MODELLING AND REPRESENTATION OF NON-SEQUENTIAL ASSEMBLY PLANS

A non-sequential assembly sequence is considered as contradictions with regards to assembly plans with non-sequential operations. To consider the non-sequential assembly plans in the operation of optimisation, the non-sequential operations set are aggregated and isolated into critical components or subassembly. Moreover, a non-sequential operations set cannot be viewed as assembly sequence optimisation for the critical components because of the fact that there is only one possible way to assemble it at the same time for diverse directions.

Figure 4.12a. shows a product that can be assembled only with a non-sequential assembly plan and a cross-section through it (b). Its graph of liaisons is presented in Figure 4.12c. Components a1, a2 and a3 are the components that are required to be assembled at the same time coordinated set of movements. The remaining components a4, a7 can be assembled in a sequential manner. The components a1, a2 and a3 are combined in a subassembly A (Figure 4.12d.). By taking a view at A as a subassembly, the assembly plan can be considered as encoding in a chromosome such as an assembly sequences as a subassembly. In Figure 4.12., the product signifies the liaison between a3 and a5 is considered when the liaison is taken into account between A and the remaining components. In Figure 4.12., three examples of possible assembly plans for the product are identified including A as a subassembly made earlier and using a non-sequential plan, which are:

- A-a5-a4-a6-a7.
- a5-a6-A-a4-a7.
- a4-a5-a6-a7-A.



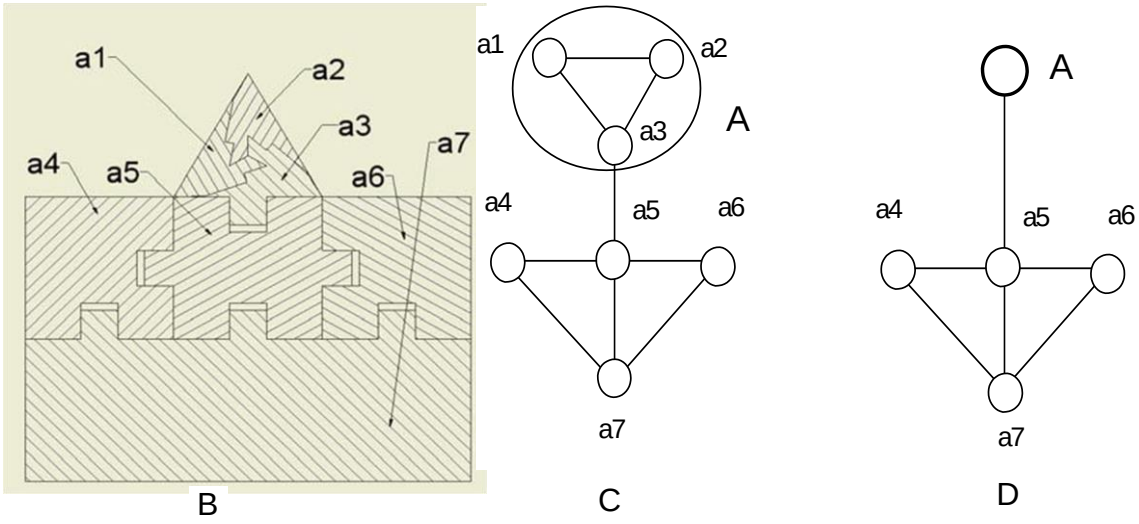


Figure 4.12. A product that can be assembled only with a non-sequential assembly plan (A), a cross-section (B), the graph of liaisons (C) and the simplified graph of liaisons (D)

The modelling of a non-sequential assembly plan includes:

- Selection of each set of components in a synchronized sequence of movements.
- Encoding of each of those sets as a complex component receiving a new name.
- Encoding of each complex component as a vertex in the graph of liaisons.
- Encoding of each intricate component as a gene in the chromosome.

4.6. MODELLING AND REPRESENTATION OF NON-MONOTONE ASSEMBLY SEQUENCES

An assembly sequence can be considered as a non-monotone where a component is included to the partial assembly and not in its final stage. This case will need a specific stage down towards the track and a position to the respective component is transformed and it is moved to its final and specific position with regards to the rest of the product components. Therefore, an assembly sequence consists of non-monotone operations signifies that:

- The component (e.g. $a_{n,}$ where n= 1,2,3, …, h) needs at least two main sets of assembly

operations, and

- The two sets of the respective assembly operations are parted by at least one assembly operation, not taking into account the component a_n.

In an effort to represent and model non-monotone assemble sequences in a chromosome, a gene is required to be perfect enough to encode particular operations and not including the additional part. In the Figure 4.13. it can be seen that there is a graph of liaison and product, where the product can be assembled with the monotone assembly sequence consisting of additional components c1 (a1), c2 (a2) and c3 (a3) along with the assembly operation, a4. In relation to this case, the chromosome a2-a3-a1-a4 encodes within homogeneous notation and non-homogeneous information. The components c2 is assembled initially and after which c3 is included then c1. At the final stage, c3 is pulled in the c1 slot. However, each substring a2, a2-a3, a2-a3-a1 and a2-a3-a1-a4 indicates an assembly stated, the last one encodes more advanced stage of assembly plan than the prior one.

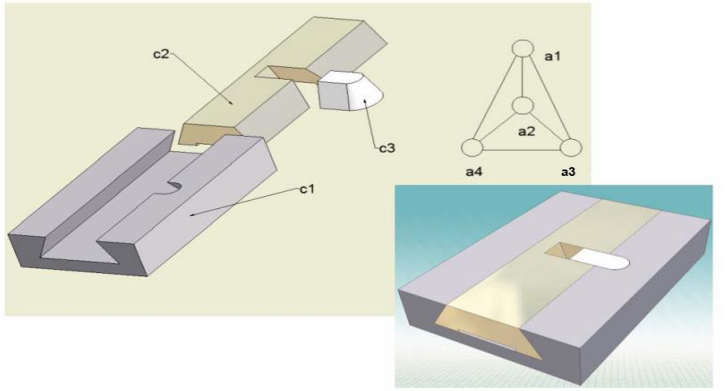


Figure 4.13. A product realised with a non-monotone assembly sequence (Marian et al. 2006)

Modelling non-monotone assembly plan includes:

- Selection of appropriate assembly-like operations to be considered in the assembly sequence.
- Encoding of each of those operations as a pseudo-mutation (PM) or meta-

component (MC) which receives a new name.

- Encoding of each pseudo-component as a vertex in the graph of liaisons.
- Encoding of each pseudo-component as a gene in the chromosome.

4.7. MODELLING AND REPRESENTATION OF PSEUDO-NON-COHERENT ASSEMBLY PLANS

It is observed that assembly plans are coherent in which each part is actually inserted and touch other placed part. However, the two different situations can be taken as exceptions. In relation to this, first situation happens in the non-linear plans and in this case the assembly process is coherent at each level of subassembly. On the other hand, the whole assembly process of the product is coherent subassembly. In relation to this, each subassembly can be viewed as complex component made previously and indicated as a vertex along with its leading external liaisons the process of assembly. Considering this the plan can be treated and encoded as mentioned above, specifically for non-linear plans. The other situation happens when auxiliary fixture is utilised temporarily in the initial phase of the assembly process. For modelling, for instance assembly process, the auxiliary tool or fixture can be taken as auxiliary component to be included to the product then removed. Therefore, the assembly sequence is changed to form a non-coherent one into a coherent and non-monotone sequence which can also be encoded.

Figure 4.14 indicates a product that can be considered as assembling with a pseudo-non-coherent assembly plan. The product is based on two vertical poles a1 and a2 in a horizontal bar a3. Figure 4.14. indicates the liaison graph of product with three components a1, a2 and a3. However, it can be seen that assembly sequence is coherent. In the graph of liaisons, can be seen relating to the product and the ground as an auxiliary component a4 (upper surface). A negative component -a4 (bottom surface), is also added to the graph of liaisons which holds the same contacts as a4, respectively a1 and a2. The feasible assembly sequences in this case will be a4-a1-a2-a3-(-a4) and a4-a2-a1-a3-(-a4).

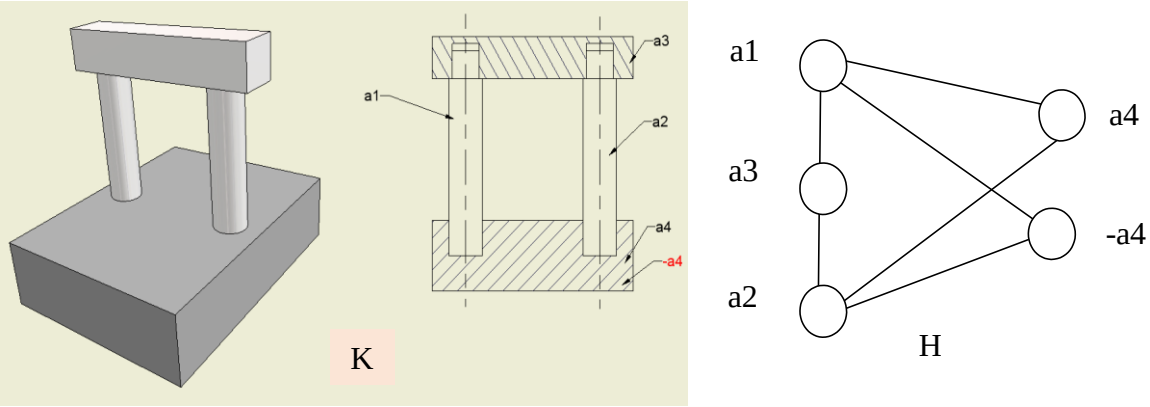


Figure 4.14. The graph of liaisons of the product (K and H) (Marian et al. 2003)

CHAPTER 5

GLOWWORM SWARM ALGORITHM FOR THE OPTIMISATION OF ASSEMBLY SEQUENCE

5.1. INTRODUCTION

Krishnanand and Chose (2006a) introduced GSOA aiming to solve engineering optimisation problems. It has been reported that the GSOA is effectively used for optimisation of multi-function wireless sensors, solving a number of analytical problems (Yu and Yang 2013) and (Pengzhen et al. 2014). Variants of such an algorithm namely particle swarm optimisation (PSOA) and niching particle swarm optimisation (NichePSOA) have the similar approach. The NichePSOA is a technique that extends the unimodal particle swarm optimizer for solving multimodal problems, i.e., multiple subswarms are grown from an initial particle swarm by monitoring the fitness of individual particles (Brits et al. 2002, Kennedy and Eberhart 1995). By comparing NichePSOA and GSOA, it was reported that a better performance of NichePSOA has been observed than GSOA in terms of acquiring an optimal solution for multimodal problems (Kennedy and Eberhart 1995), (Van den-bergh 2002) and (Yu and Wang 2013).

Glowworms, ants and bees behave differently, and their social behavior is impacted by the interactions of each other. The versatile behavior of social insects (SI) can be transformed into digital software solutions. In SI systems these behaviors can be imitated. The basis of these systems is that they focus on the behavior of local agents interacting with each other and behaving as a local swarm. The interaction of different swarms with each other is also considered. Their movement depends on the local sources placed in the simulation system.

5.1.1. General Collective Behavior of Swarms

The main properties of the collective behavior can be pointed out as follows and is summarised in Figure 5.1:

Homogeneity - Every agent in swarm has the same behavior. It may appear that different leaders are formed during the movement of swarm.

Locality - The locality is the influence of subgroups of agents affecting each other in the region (Krause and Ruxton 2002). Within the swarm organization, the most important quality of swarm is the ability of vision of each leader and their subordinates during the movement.

Swarm Centering - Due to this inherent ability of swarm, it is easy for the agents to stay close to each other. It is their ability that a specific distance can be maintained between them and other agents.

This is observed in a large swarm of animals that they give this the highest priority (Krause and Ruxton 2002).

Velocity Matching - Attempting to match the velocity with the nearby swarm mates.

Collision Avoidance - This ability is used by the stock mates to avoid the collision with nearby swarm mates. It is done by using the velocity matching technique which results in matched velocities (Krause and Ruxton 2002). They are attracted towards other members of swarm if they don't do the action of avoidance. It is not in their power to remain isolated as they are attracted towards other individuals and to align themselves with neighbours (Partridge BL 1982) and (Partridge and Pitcher 1980).

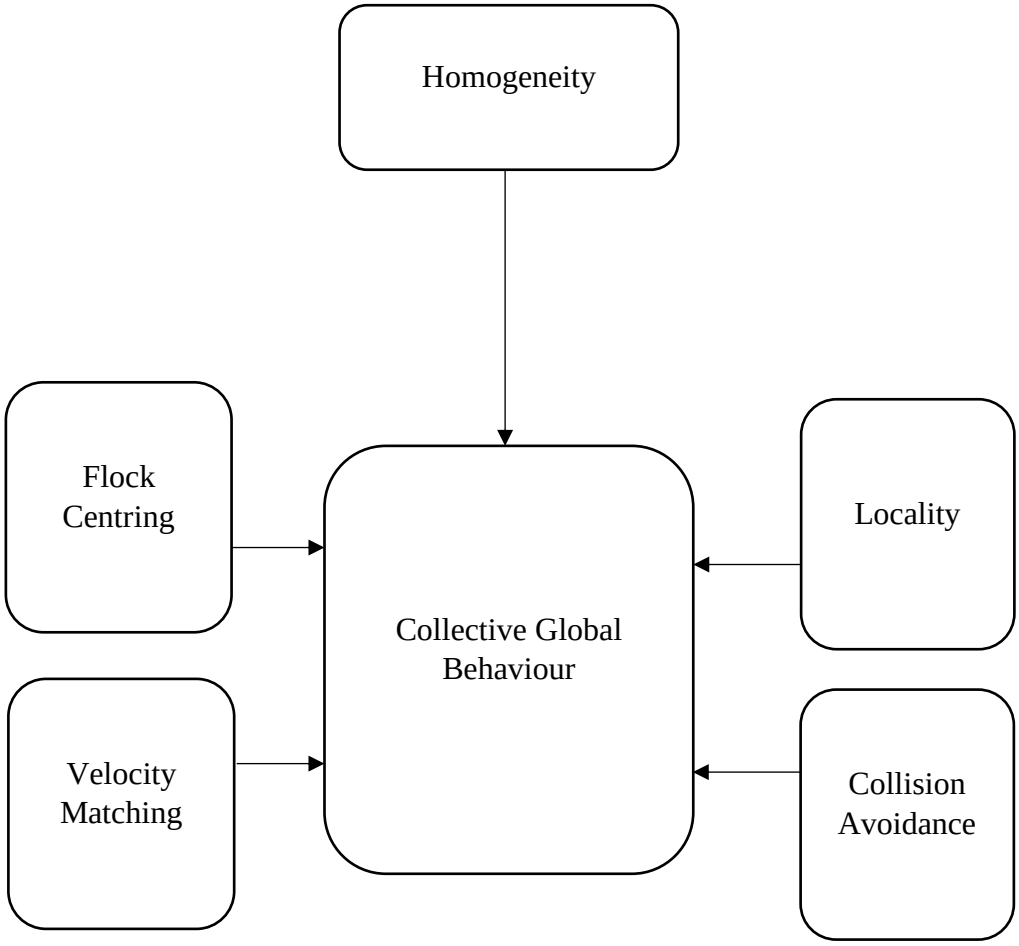


Figure 5.1. The character of collective behavior (Thiruvenkadam and Perumal 2017)

5.1.2. Collective Behavior of Glowworms

Krishnanand and Ghose (2009a) analysed the flashing behavior of glowworms. Each glowworm carries a luminescence amount called luciferin, which is decided by the function value of glowworms' current location. A range is defined for each glowworm and through this range, depending upon the level of luciferin, a glowworm moves towards another glowworm. The higher luciferin level of the glowworm leads to attraction to movement which is decided by a probability mechanism (Krishnanand and Ghose 2006a, Krishnanand and Ghose 2006b, Krishnanand and Ghose 2009a, Krishnanand and Ghose 2008, Liao, Kao and Li 2011, Wu et. al. 2012 and Jayakumar and Venkatesh 2014).

Zhang et al. (2011) used a methodology for limitation of scent sources with respect to an advanced GSO calculation. It has been observed that the far-reaching calculations of tuft following can be performed by the applications for utilisation independent robots. Tang et al. (2013) proposed the GSO solution that was developed on a global base using the mutation program in optimum conditions. This process is called the parallel crossover mutation glowworm swarm optimisation. Jayakumar and Venkatesh (2014) developed the optimal solution for resolving the problem of multiple objectives based on ecological and economic parameters using GSO algorithm.

Atheer and Nordin (2017) proposed GSO technique by increasing the population range using the mutation process. Diffusion solutions in space research are retained by way of mutation operation. Some solutions turn into infeasible following the operation of mutation and migration during the problems of optimisation. Multiple solutions can be added by the addition of other methods to verify the possibility of the solution in such cases (Pan and Xu 2016, Mo, Li and Zhang 2016).

5.1.3. Differential Methods in Terms of the Extensive Review

A multimodal optimisation problem can be formulated as the clustering problem using a GSOA method (Aljarah and Ludwig 2013). These methods have been known to provide better results compared to traditional clustering methods of such as the K-Means clustering, average linkage agglomerative hierarchal clustering, furthest first (FF) and learning vector quantization (LVQ). Gorai and Ghosh (2011) find the best enhancement setting of images which was based on PSO (Particle Swarm Optimisation Algorithm). The quality of intensity of image is enhanced by the parameterized transformation function which was a similar proposition to earlier. The rescaling method has also been utilised for solving scale problem. Table 5.1 shows a summary that distinguishes the differential methods in terms of GA, PSO, ACO and GSO, respectively (Zhan, Zhang, Li and Chung 2009).

Table 5.1. The differential methods in terms of the extensive review of the GA, PSO, ACO and GSO, respectively

Items	**Algorithm**			
	GA	**ACO**	**PSO**	**GSO**
Year	**1975**	**1999**	**1995**	**2005**
Author	John Holland	Dorigo & Di Caro	Kennedy & Eberhart	Krishnanand & Ghose
Optimisation	Discrete Optimisation	Meta heuristic Optimisation	Stochastic Optimisation	Meta heuristic Optimisation
Parameters	Reproduction, Crossover, Mutation.	Construct Ant Solutions, Daemon Actions (optional), Update Pheromones.	Current velocity, Personal Best, Neighbourhood Best.	Initialization, Updating Luciferin, Movement, Updating the Local Decision Range.
Purpose	Find the best among others.	Find the shortest path.	Reach target with minimal duration.	Find the local finest solution.
Advantages	1.Large combinatorial problems can be solved by means of efficient investigation 2. Exhaustive brute forces searches appear slow as compared to	1. Parallelism is present inherently. 2. Rapid discovery of goods solution are given as positive feedback. 3. Travelling salesman and other similar	1. Scientific and engineering problems can be accounted in this mechanism. 2. Mutation calculation and overlapping does not occur in this method.	1. Highly nonlinear and multimodal optimisation problems can be handled naturally and efficiently. 2. Velocities are not used in GSO. PSO also shows no problem

	many orders of magnitude	problems are efficiently solved. 4. Dynamic applications can be used (the changes in new distances can be formulated)	3. Speed of particle can help carry the search. 4. Real number code is adopted by PSO. The solution decides this directly.	associated with velocity. 3. Global optimised solution has a very high probability of reaching as the speed of convergence in GSO is very high.
Disadvantages	1. It is expensive computational 2. Weeks or days may be consumed to analyse the large problems. 3. It is faster than force. 4. Glowworm algorithm can be directed towards optimal solution but it is blind.	1. Difficulty has been observed in theoretical analysis. 2. Independent use of sequences of random decisions. 3. Iterations are changed by probability distribution. 4. The research has been experimental and not theoretical.	1. Mid optimum point can reach premature convergence have a fast tendency. 2. Scattering and optimisation problems cannot be solved by this method. 3. For each iterative process there is slow convergence.	1. High dimensional problems have a problem with GSO. 2. The conventional speed for the algorithm is slowed for glowworms moving as the dynamic change of decision domain is GSO. 3. Slow iteration process occurs as the local search ability is reduced.
Medical Field	The optimisation of artificial neural	The neural network has been optimised	1. Image segmentation (MRI) has been	1. The future selection problems can be

	networks among others seem slow as compared to genetic algorithm.	artificially in ACO. This has been used in the field of medical image processing.	used to detect the Brain tumour 2. The artificial neural networks have been optimised for medical image processing by using PSO.	optimised by using GSO.

5.2. GLOWWORM SWARM OPTIMISATION ALGORITHM

According to Krishnanand and Ghose (2009b) glowworm swarm s which contains of m glowworms, is distributed in the search space. A random position p_j is assigned to the glowworms g_j (j=1…m) in the search space. A specific luciferin level L_j is assigned to each glowworm g_j in the local decision range rd_j. A glowworm having a higher level of luciferin will be brighter. Within the neighbourhood range of the glowworms, they move towards the brighter glowworms that are having high luciferin level value within their restricted domain range. At multiple optimal locations in search space, compact groups are formed by most of glowworms. During the initial stages when the glowworms are placed in the search space, they have a luciferin level (L_0) which is equal for all. The r_s radial sensor range is also initialised with the condition of r_0. At a position of glowworm p_i the objective function is evaluated at luciferin level update. After that the luciferin level for the combined group is set to drive the new objective function values. For the glowworm, the luciferin level is L_j is defined as follows:

$$L_j(t) = (1-\rho)L_j(t-1) + \gamma F_i(p_j(t)) \tag{5.1}$$

Here ρ is the luciferin decay constant and L_j (t-1) shows the value of luciferin at the previous level. γ is the luciferin enhancement fraction.

At any current glowworm position p_j for any glowworm j, F (p_j (t)) represents the objective function. T is the current iteration for glowworm j. During iteration, the glowworm j explores its neighbourhood region for finding the highest luciferin level by applying the following rule.

$$z \in N_j(t) \text{ if } d_{jz} < rd_j(t) \text{ and } L_z(t) > L_j(t) \tag{5.2}$$

Where distance is represented by d. Glowworm j is closer to glowworm z. N_j (t) is defined as the neighbourhood set. d_{jz} is the distance between the glowworm z and glowworm j. Local decision range for the glowworm j is defined by r_{dj} (t). L_z (t) defines the luciferin value for glowworm z for time t while L_j (t) defines the glowworm j luciferin level for time t.

$$prob_{jz} = \frac{L_z(t) - L_j(t)}{\sum_{k \in N_j(t)} L_z(t) - L_j(t)} \tag{5.3}$$

This equation 5.3. describes the preference of glowworms to select the best neighbour in the neighbourhood. For this purpose, the equation drives test for each glowworm and analyze the probability for selecting best neighbours. Z is described as one of the many neighbourhoods set for glowworm j. A glowworm which has a high level of probability will have a higher chance of getting selected from the neighbourhood, while the direction is measured by the roulette wheel method. Previous glowworm is adjusted according to the new neighbour glowworm.

$$p_j(t) = p_j(t-1) + s\frac{p_z(t) - p_j(t)}{d_{jz}} \tag{5.4}$$

Distance jz is defined as the Euclidean distance between the glowworm j and z. At the end of the glowworm iterations, the range for the local decision domain with the new adjusted glowworms is given by,

$$r_d^i(t) = \min\left\{r_s, \max\{0, r_d^i(t-1) + \beta(n_t - \left|N_j(t-1)\right|)\}\right\} \tag{5.5}$$

β is a constant parameter that affects the rate of change of the neighbor

domain.

The neighbour set size is restricted by a constant parameter n_t. The actual neighbourhood set size is described by N_j (t).

The computational procedure for the GSOA is shown in the Figure 5.2.

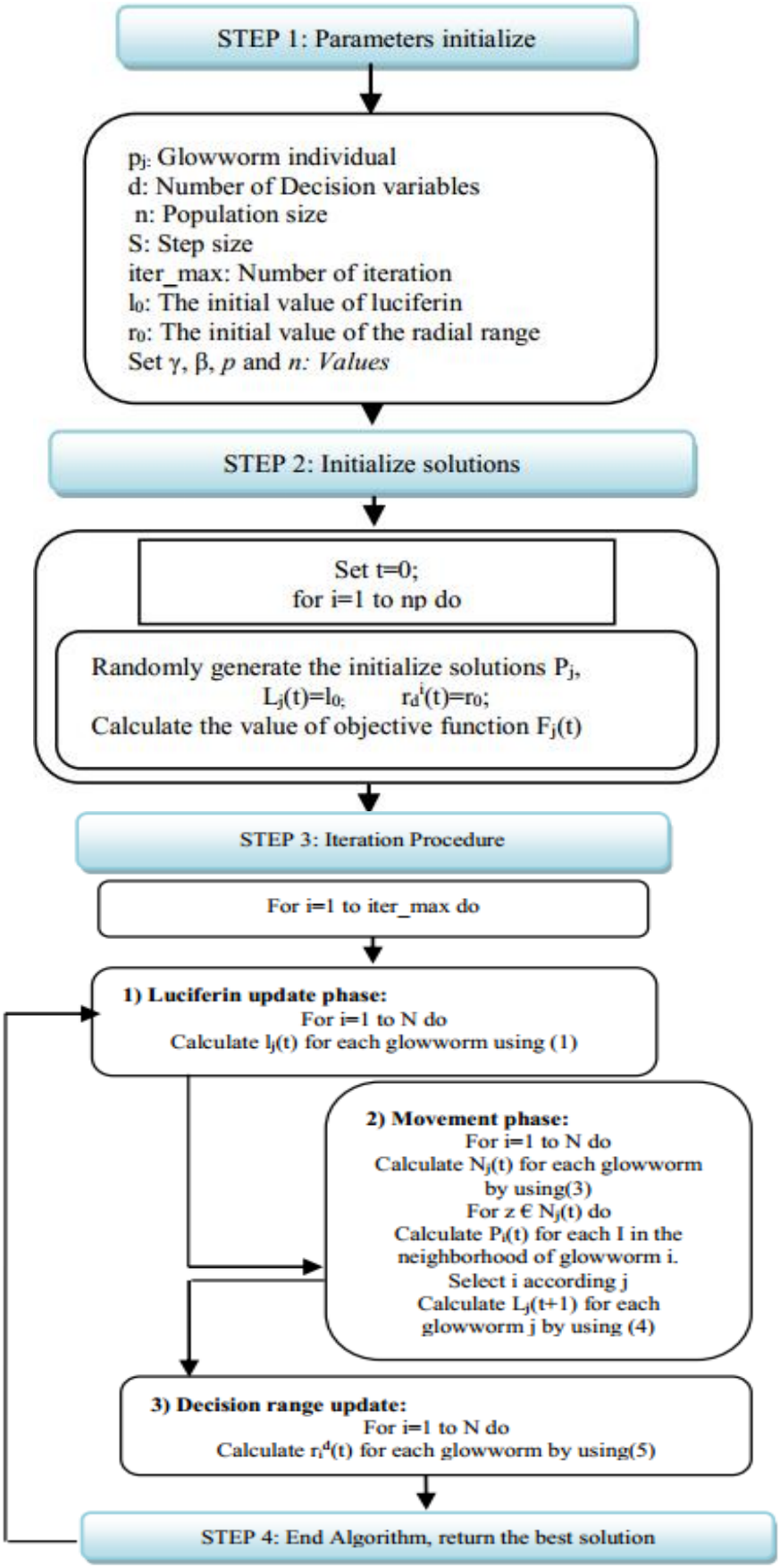


Figure 5.2. Flowchart of GSO (Thiruvenkadam and Perumal 2017)

5.3. GLOWWORM SWARM OPTIMISATION CLUSTERING ALGORITHM

GSO clustering algorithm turned to be a significant method in machine learning, pattern recognition and other engineering fields. The clustering algorithm aimed to identify and extract important groups in underlying data. Emerging Clustering with GSO based algorithm as an alternative to more classical clustering approaches.

In GSO clustering algorithm two processes has been added to the main GSO processes. First one is defining a cluster data object and the second process is defining the attraction data object.

5.3.1. GSO Clustering Process

GSO clustering algorithm has additional processes and defined as follows:

For A cluster data object x (x_1, x_2, …., x_m), the equation 5.6. describes the local space relative density:

$$d\left(x_i(t)\right) = \frac{\left|N_r\left(x_i(t)\right)\right|}{g} \tag{5.6}$$

Where r is the local space radius, N_r (x_i(t)) is the data set containing in local space within r of x at iteration t, g is the overall numbers of data object. The bigger d (x_i(t)) value, the more data object X (x_1, x_2, …., x_m).

For attraction data object x (x_1, x_2, …., x_m) is described by the next equation:

$$J\left(x_i(t)\right) = -\mathrm{In}\left(d\left(x_i(t)\right)\right) \tag{5.7}$$

Where ln() is the natural logarithm. Also, The bigger J (x_i(t)) value, the more data object X (x_1, x_2, …., x_m).

5.3.2. GSO Clustering Algorithm

GSO clustering algorithm is described as follows:

Input cluster data object;

Set maximum iteration number = *iter_max*;

Let *s* be the step size;

Let *r* be the local space radius;

Let L_i (0) be the initial luciferin;

Let r_d^i (0) be the initial dynamic decision domain radius

Set *t* =1.

While (*t* < = *iter_max*) do:

{

for *i* =1.

$N_r(x_i(t)) = \{j : \|x_j(t) - x_i(t)\| < r\}$; %Where $\|\bar{x}\|$ is the norm of $\bar{x}$

$$d(x_i(t)) = \frac{|N_r(x_i(t))|}{g}$$

$$J(x_i(t)) = -\mathrm{In}(d(x_i(t)))$$

$$L_j(t) = (1-\rho)L_j(t-1) + \gamma F_i(p_j(t))$$

For each glowworm *i* do: %movement-phase

{

$N_i = \{j : d_{i,j}(t) < r_i^d(t) and l_i(t) < l_j(t)\}$

Where $\|\bar{x}\|$ is the norm for $\bar{x}$

for every glowworm $j \in N_i(t)$ do:

$$p_{ij}(t) = \frac{L_j(t) - L_i(t)}{\sum_{k \in N_j(t)} L_k(t) - L_i(t)}$$

j= select glowworm ($\bar{p}$)

where $\overline{p}$ is the maximal element of P

$$x(t+1) = x_i(t) + s\left(\frac{x_j(t) - x_i(t)}{\left\|x_j(t) - x_i(t)\right\|}\right)$$

$$r_d^i(t+1) = \min\{r_s, \max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\}$$

}

$t \leftarrow t+1;$

}

Algorithm symbolic description: $x_i(t)$ is the glowworm i in t iteration location; $L_i(t)$ is the luciferin of the glowworm i in t iteration; $N_i(t)$ is the neighbourhood set of glowworm i in t iteration; $r_d^i(t)$ is the dynamic decision domain radius of glowworm i in t iteration; is the upper bound of the $r_d^i(t)$; $p_{ij}(t)$ is the probability of glowworm i selects neighbour j (Thiruvenkadam and Perumal 2017).

CHAPTER 6

A CASE STUDY USING A GENETIC ALGORITHM AND A GLOWWORM SWARM ALGORITHM FOR SOLVING AN ASSEMBLY SEQUENCE OPTIMISATION PROBLEM

6.1. INTRODUCTION

An assembly sequence must usually be pre-defined when a product needs to be assembled. This is ideally considered at the early design stage and is aimed at a reduction of assembly time and therefore costs. That is particularly crucial for many small-medium enterprises (SME) that rely on assembly of products to survive in fierce competition. Apart from the effect of product design, assembly time is largely subject to its assembly precedence, accessibility, constraints, geometry and number of assembly components. Marian et al. (2006) suggested a GA for solving an ASP optimisation problem with an aid provided by a guided search effective algorithm. Choi et al (2009) developed an approach to optimise multi-criteria ASP based on a GA. Yasin et al (2010) investigated the application of GA in optimising product assembly sequences and the study concluded that GA can be used to obtain a near optimal solution for seeking a minimal process time of sequence assembly. Thus, GA is an efficient algorithm to find an optimal or a near optimal solution for assembly sequence time.

As presented earlier in the research literature, GSOA was introduced by Krishnanand and Chose (2006a) to solve engineering optimisation issues. To accomplish GSOA objective (engineering optimisation problems), a swarm must have an ability to be split into disjoint groups. During one program run, the GSOA is capable of determining the multiple optimal solutions in parallel. First, the algorithm involves a random deployment of a population in a specified size *n* glowworm in a search space at the inception and each carries a luminescence containing a quantity of luciferin as physical entity. Location of a glowworm is determined by an objective function calculating the strength of luciferin, i.e., the intensity of luciferin is

associated with the objective function of a glowworm's location. A greater luciferin intensity implies a better location associated with an objective function value. Each individual glowworm updates its luciferin level based on the objective function value of its recent position.

Unlike GA which are commonly used for solving assembly sequence optimisation problems, GSOA was not reported as being used for solving similar issues. This research presents two case studies that applies the GA approach and the GSOA approach to obtain the fastest solution for the assembly sequence of a car engine pump valve and a ball pen product. GSOA outperformed the GA in terms of reducing assembly time for an assembly sequence.

6.2. PROBLEM STATEMENT MODEL FORMULATION

It is widely understood that efficiency of assembling a product by reducing assembly times (therefore costs) is vital particularly for small manufacturing companies to survive in an increasingly competitive market. Optimally, it is helpful for determining an optimal assembly sequence of a product at the early design stage. The complexity of assembling a product is often subject to the number of assembly components and the relationship between mating parts. Products complexity can be divided to three types;

- **Large product**: That has more than 25 components, for example, a car engine (will be one of the research future work).

- **Medium product**: that has up to 25 components (Marian et al. 2006). For instance, a car engine pump valve (first case study).

- **Simple product**: that has a small number of components, for example, a ball pen product (second case study). The product assembly sequences can be determined at the early design stage.

Nevertheless, it may find inefficient using the heuristic approaches in acquisition of a quick solution in terms of an optimal assembly sequence with a minimal assembly time. It starts by selecting input parameters based on number of sequences, priority matrix and assembly sequence time.

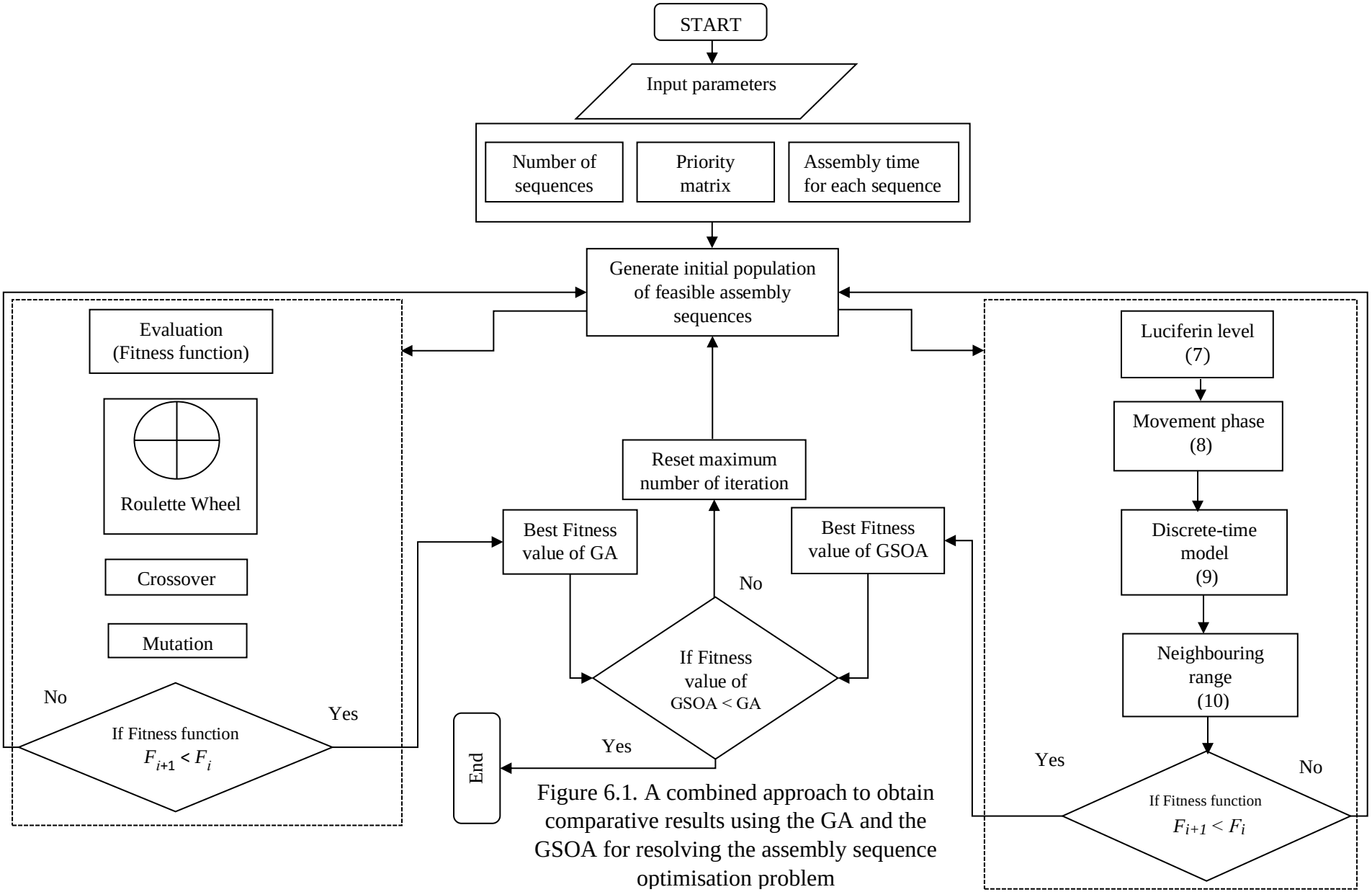


Figure 6.1. A combined approach to obtain comparative results using the GA and the GSOA for resolving the assembly sequence optimisation problem

6.2.1. Genetic Algorithm

Figure 6.2. illustrates the mechanism of the GA used in programming. It starts with the initial population that is usually generated randomly as a binary string of zeros and ones or as integers or real numbers; this is also known as a genetic representation or encoding. The next process is the evaluation stage, which involves a computation of a fitness value based on an objective (fitness) function. Thus, selection plays a key role in GA programming; only those representing a possible solution with either a highest or lowest fitness value are selected. The Roulette Wheel approach was used to ensure that a certain number of the population of chromosomes are retained in the next generation, which contains chromosomes with greater fitness. Crossover operates on pairs of chromosomes simultaneously with the aim of creating offspring that combines the features of both parental chromosomes. This is usually carried out via a random selection of parental chromosomes to produce new chromosomes. In this study, however, it was performed by crossing over the genes as illustrated in Figure 6.3. to generate possible assembly sequences for the car engine pump valve with assumption that the bits of chromosomes can be swapped freely without following the precedence required for assembly. Mutation is used to have a complete loss of a particular allele or bit, i.e, the mutation of swapped genes is utilised to prevent chromosomes from repeating the gene of a new offspring. This was performed by crossing over the genes in different sequences leading to various assembly paths and total time of assembly. Only the bits of chromosomes that do not have a successor or precedence are swapped as illustrated in Figure 6.4. and these chromosomes were used.

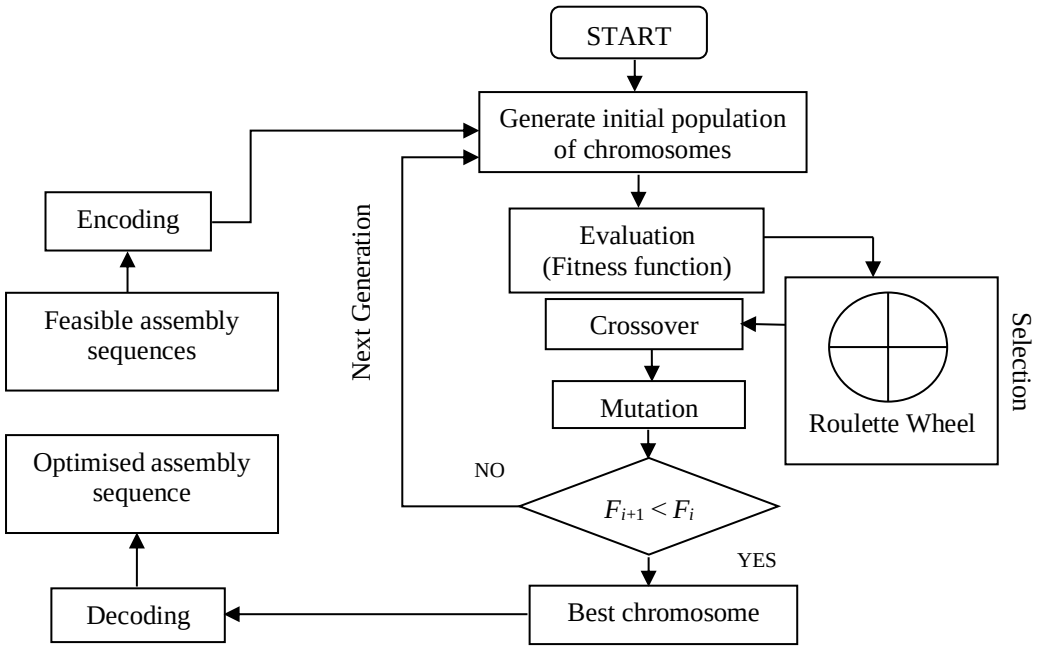


Figure 6.2. The GA programming approach

The following notations and parameters are used:

i: Number of a chromosome, i = 1, 2, 3,…, k;

f_i: Fitness of chromosome i;

t_i: Time taken of chromosome i;

F: Fitness of the population;

C_r: Crossover rate;

R_i: A Roulette Wheel probability;

P_i: The cumulative probability for chromosome i;

L: Total length of gene in a population;

e: Number of genes in a chromosome, e = 1, 2, 3, …, n;

mr: Mutation rate;

r: Random number;

M: Number of mutations.

The Fitness Function

In this study, the GA uses a single objective function as the fitness function for selecting a chromosome with a higher fitness value. The fitness f_i, which is a function of assembly time of an assembly sequence represented by chromosome *i*, is described as:

$$f_i = \frac{1}{\sum_{i=1}^{m} t_i} \tag{6.1}$$

Thus, the total fitness *F* is given:

$$F = \frac{1}{\sum_{j=1}^{m} f_j} \tag{6.2}$$

The for loop is used to compute the fitness value for each of the generations with the above-mentioned formula. The computed fitness values are stored in the array future usage. The pseudocode that is used to implement the fitness function is provided below:

```
int noGenerations = F_Obj.length;
 double Fitness[] = new double[noGenerations];
 for(int i=0;i<noGenerations;i++)
 {
  Fitness[i]=(1/(1+(double) F_Obj[i]));
 }
 return Fitness;
```

Selection of a chromosome

As illustrated in Figure 6.3., in the proportionate fitness selection, which is also known as the roulette wheel selection, fitness is calculated by assigning a fitness value to one of possible chromosomes or solutions. This fitness value is often associated with a probability of a selection with each of individual chromosomes. Only a chromosome with a high fitness value will be selected during a selection process.

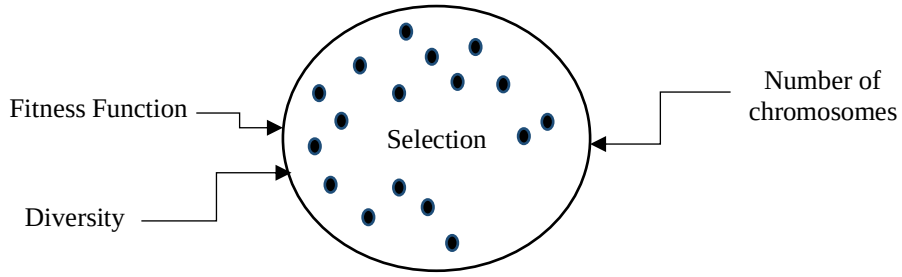


Figure 6.3. Selection of a better chromosome

Thus, only a chromosome, which is fittest with the greater roulette wheel probability, is selected. The roulette wheel probability R_i is given by:

$$R_i = \frac{f_i}{F} \tag{6.3}$$

The percentage of the chance for chromosome i is expressed as probability P_i where,

$$P_i = R_i \times 100\% \tag{6.4}$$

The probability is calculated with the use of Fitness value of the chromosome. However, before calculating the probability the total sum of the fitness values of the entire chromosomes should be calculated. The pseudocode used to compute the probability using the fitness function is provided below:

```
int noGenerations = Fitness.length;
 double Probability[] = new double[noGenerations];
 double sum =0;
 for(int i=0;i<noGenerations;i++)
  sum =sum+Fitness[i];
 for(int i=0;i<noGenerations;i++)
 {
  Probability[i]=Fitness[i]/sum;
 }
 return Probability;
```

Crossover

Figure 6.4. illustrates a crossover process where the first two genes of two different chromosomes are exchanged. The crossover process is controlled by a probabilistic operator. Repetition of the same gene number is strictly avoided during the crossover process, and each of the genes involved is thoroughly checked before completing the process.

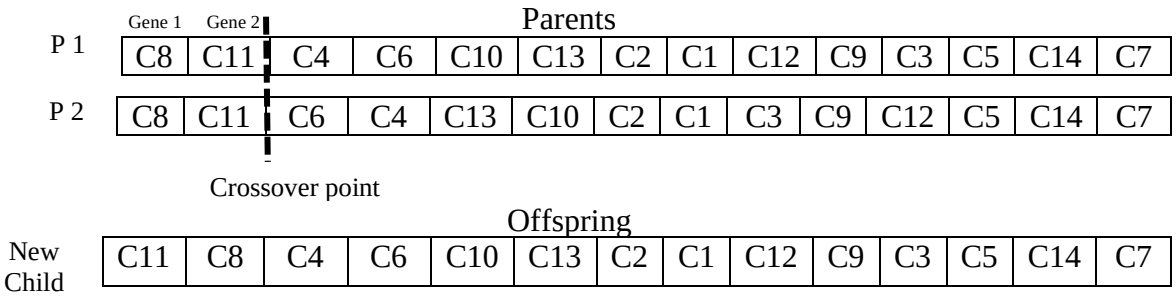


Figure 6.4. The crossover process of swapping genes

The two different genes that will be used for the crossover is selected using the random function. The pseudocode that is used for the random selection of the genes are provided below:

```
Random rn = new Random();
firstChromosome=chromosome[rn.nextInt(noGenerations)];
secondChromosome=chromosome[rn.nextInt(noGenerations)];
```

Mutation

A mutation is performed by a random replacement of a gene from its original state with a new quantity in other position or attributes, according to a user-defined mutation probability or mutation rate. The only thing this prevents is the taking of the fittest of the population in the next generation rather than randomly selecting those that are fitter. Parameters of C6 and C4 were used for the calculation of the mutated chromosomes in a particular population, as shown in Figure 6.5.

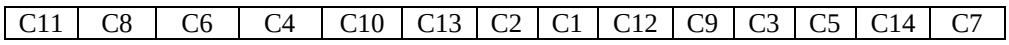


Figure 6.5. The mutation operator

Thus, the total length of genes L in a chromosome is thereby given by:

$$L = e\,\frac{i}{n} \tag{6.5}$$

Where $i = 1, 2, 3, \ldots, k$. L is a random integer ranging from 1 to 14 in this case. As a result of this, a probability of a mutation of a gene is $1/L$, If the mutation rate mr is greater than the selected random number r, i.e., $mr > r$, where is r a random number r in the *range* between (0, …1), $0 \leq r < n$, then the mutation should be performed. Hence, the number of mutations M is given by

$$M = r\,\frac{1}{L} \tag{6.6}$$

After a re-allocation of the suitable gene position of the chosen parent, a new child chromosome is established. This implies that the new child chromosome has a new identification which possibly makes it a new parent for the next generation of the continuous population.

The pseudocode that is used to perform the mutation of the genes in a chromosome is provided below:

```
for(int j=0;j<mutation.length;j++)
{
 for(int k=0;k<mutation[0].length;k++)
 {
  sum1=sum1 +mutation[j][k];
 }
 if (sum1!=0)
  counter++;
}
```

6.2.1.1. Acquisition of an assembly sequence time using the GA

The notation used in this study to summarise and highlight the proposed solutions to the assembly sequence problem is subsequently explained.

Indices

g: generation index (g = 1, ……G), where G represents the number of generations.

s: assembly sequence index (s = 1,….S_g), where S_g represents the number of assembly sequences in a specific generation g.

c: an assembly component index (c = 1, ….C_{sg}), where C_{sg} represents the number of assembly components in a particular sequence of assembly of a generation.

P: priority, $P = \begin{cases} 1 & \text{if priority exists} \\ 0 & \text{otherwise} \end{cases}$

Decision variables

$X_{csg} = \begin{cases} 1 & \text{if component } c \text{ is utilised on assembly sequence } s \text{ of generation } g \text{ with a priority com} \\ 0 & \text{otherwise} \end{cases}$

Parameters

HT_s: Handling time for assembly sequence $_S$

IT_s: Insertion time for assembly sequence $_S$

TT_s: Total time for assembly sequence s, where TT_s is a sum of HT_s and IT_s

r: Random number

CP: Cumulative probability

F_s: Fitness of assembly sequence s (s = 1, 2, 3, …S_g) in a generation

F_g: Fitness of generation g, F_g = (1, ….G), where the total number of fitness for a generation g

Indicator variables

t_{cg}^{s}: Starting time of component c on assembly sequence s in generation g

$Z_{csg} = \begin{cases} 1 & \text{if component } c \text{ is utilised on assembly sequence } s \text{ of generationg } g \\ 0 & \text{otherwise} \end{cases}$

The computation of cumulative probability is the sum of the probabilities computed for the chromosomes. The pseudocode used to compute the cumulative probability is provided below:

```
int noGenerations = Probability.length;
 double Cumulative[] = new double[noGenerations];
 double sum =0;
 for(int i=0;i<noGenerations;i++)
 {
  for (int j=0;j<=i;j++)
  {
   sum =sum+Probability[j];
  }
   Cumulative[i]=sum;
 }
 return Cumulative;
```

The objective function

The aim of seeking the minimum time taken for assembling a product associated with an assembly sequence of a specific generation can be described as the objective function where a minimal assembly time TT_s can be sought as follows:

$\{Min\ (TT_s)_g$, where $s = 1\ldots..S_g$ and $g = 1\ldots..G$

If a minimum assembly time is repeated over generations, then the most dominant assembly sequence will be selected with the minimum assembly time.

Constraints

In this study, the total assembly time TT_s was subject to a sum of handling assembly time HT_s and insertion assembly time IT_s, where

$$TT_s = HT_s + IT_s \tag{6.7}$$

Let us define the probability of an assembly sequence P_S,

The probability P of an assembly sequence s is:

$$P_s = F_s / F_g \tag{6.8}$$

Where F_s is the fitness of assembly sequence s and F_g is the fitness for the generation g. Thus,

the cumulative probability of *CP* is given by:

$$CP = \sum_{i=1}^{s} P(i) \quad \text{, Where } i = 1, 2, 3, \ldots S \tag{6.9}$$

Fitness Value

The value of fitness F_s for assembly sequence *s* can be expressed as the function of assembly time *TTs*:

$$F_s = 1 \ / \ TT_s \tag{6.10}$$

Stopping Criteria

Stopping criteria are the rules that govern the termination of the iteration are as follows:

Criteria 1: When $g = G$, where it occurs at the end of the generation, all preceding components are satisfactorily assembled, and there is no component remaining for assembly within a particular assembly sequence.

Criteria 2: In this case study, after 10 iterations, then the creation of a new generation will be terminated, i.e.,

$\{Min\ (TT_s)\}_{g\ -\ (n-10)} = \{Min\ (TT_s)\}_{g\ -\ (n-9)} = \ldots\ldots = \{Min\ (TT_s)\}_{g\ -\ (n-1)}$.

6.2.2. The Glowworm Swarm Optimisation Algorithm

Figure 6.6. illustrates the mechanism of the glowworm swarm optimisation algorithm (GSOA). In this work, a glowworm denotes a component and a swarm of components is a population that is initially distributed randomly in a search space. Like the natural world, each component also acts as if it is a luminescent or glowing glowworm emitting a light whose intensity is proportional to the associated luciferin interacting with other glowworms or components within a defined neighbourhood. The neighbourhood area is categorised as a local-decision domain that has a variable neighbourhood range r_d^i bounded by a radial luciferin sensor range r_s ($0 < r_d^i \leq r_s$). In nature, the neighbourhood range is a dynamic quantity.

In this study, assuming that component *i* considers another component *j* of its neighbour, if *j* is within the neighbourhood range of *i* and the luciferin level (in this case, it refers to the gap in dimensions between two mating components, i.e., mating component *i* with component *j* or parts based on the time taken to assemble) of *j* is higher than that of *i*. The decision domain allows a selective neighbour interaction. Each component is attracted by a suitable dimension of another glowworm in the neighbourhood. Components in a GSOA depend only on information accessible in their neighbourhood to make possible decisions. Thus, each component selects a probabilistic neighbour that has a higher suitable dimension and moves toward it. These movements, which are based only on local information and selective neighbour interactions, enable a swarm of components to partition into disjoint subgroups that steer toward and meet with a multiple optimum of a given multimodal function, whereby the functional integrity of the components is not compromised.

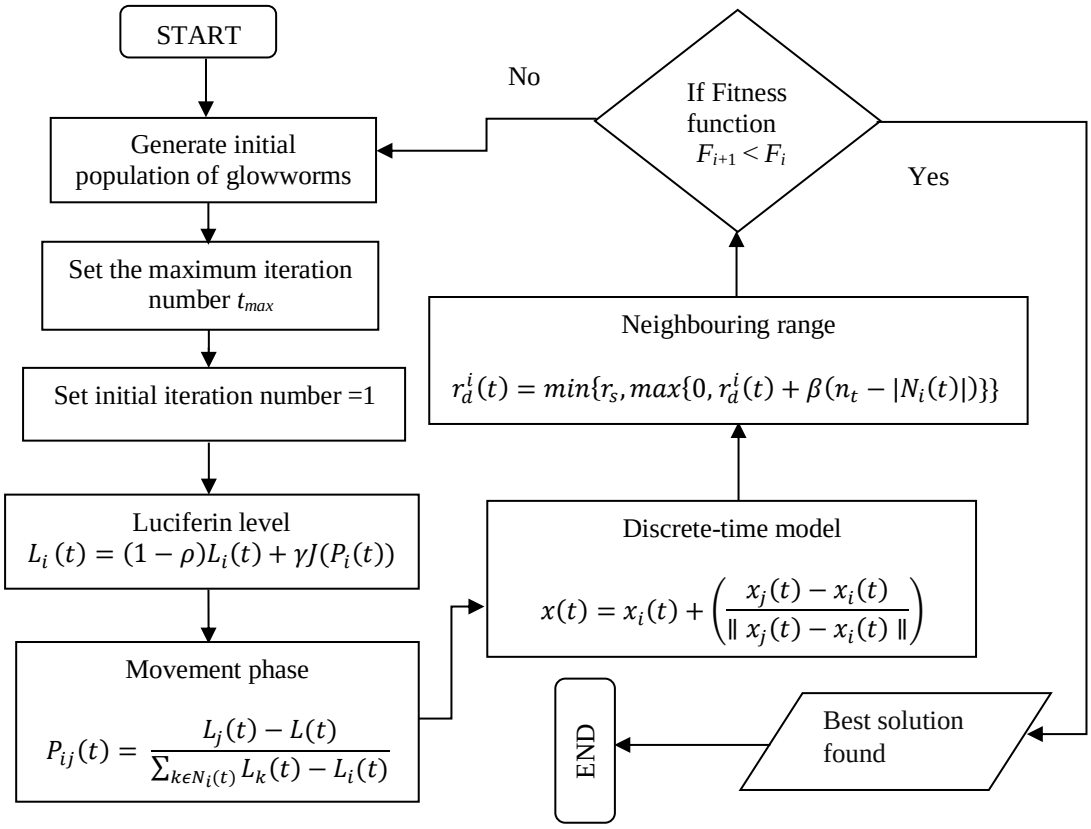


Figure 6.6. Mechanisme of the glowworm swarm optimisation algorithm

The following variables are used:

L_0 : quantity of luciferin

n: random population of n glowworms ($1 \leq n \leq 14$ in this study)

r_d^i: neighbourhood range

r_s: radial sensor range

γ: luciferin enhancement constant

ρ: luciferin decay constant

6.2.2.1. The Luciferin Level

At the inception of the initial iteration, all the glowworms begin with the same value of luciferin L_0 , these values change depending on the function value at a glowworm position. During the luciferin-update phase; each glowworm adds its previous luciferin level, i.e., a luciferin quantity proportional to the fitness of its current location based on the objective function. Also, a fraction of the luciferin value is subtracted due to the decay in luciferin over time. Thus, the objective function value for a glowworm at iteration t is calculated using the luciferin update rule as follows:

$$L_i\,(t) = (1-\rho)L_i(t) + \gamma J(P_i(t)) \tag{6.11}$$

Where $L_i(t)$ denotes the luciferin level of glowworm i at time t; $J\,(x_i\,(t))$ denotes the objective function value of glowworm i at time t; x_i represents the luciferin's location of a glowworm i; ρ represents the luciferin decay constant ($0 < \rho < 1$), and γ is enhancement constant of the luciferin.

6.2.2.2. The Movement Phase

During the movement phase, the probability of the location of a glowworm moves towards a neighbour that has a luciferin value higher than its own value. The glowworm tends to gain more attraction as its luciferin level increases. This is derived from the fact that glowworms

are attracted to neighbours that glow brighter. The probability p of glowworm i that moves towards j at time t is given below:

$$P_{ij}(t) = \frac{L_j(t) - L_i(t)}{\sum_{k \epsilon N_i(t)} L_k(t) - L_i(t)} \tag{6.12}$$

Where, $j \in N_i(t)$ and $N_i(t) = \{j : d_{ij}(t) < r_d^i(t); L_i(t) < L_j(t)\}$, which is a set of neighbour of glowworm i at time t, $d_{ij}(t)$ denotes the Euclidean space, usually from glowworms i and j at time t, and $r_d^i(t)$ denotes the variable neighbourhood difference related to glowworms i and at time t. Let glowworm i select a glowworm $j \in N_i(t)$ with $p_{ij}(t)$ given by Eq. 6.12. Then, the discrete-time of the glowworm movements can be stated as:

$$x(t) = x_i(t) + \left(\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}\right) \tag{6.13}$$

Where, $x_i(t)$ represents glowworm i location at time t, $\|.\|$ denotes the norm operator in an Euclidean space.

The pseudocode used to compute the Euclidean space is provided below:

```
return Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2));
```

6.2.2.3. The Neighbourhood Range

There is an association between glowworm i and j within a neighbourhood range. The term r_d^i of glowworm i is a dynamic radial range at initial iteration, providing $0 < r_d^i \leq r_s$. When the glowworms depend only on local information to decide their movements, it is expected that the number of peaks-captured may be a function of the radial sensor range. In fact, if the sensor range of each agent covers the entire search space, all the agents move to the global optimum and the local optima are ignored. Since assuming a priori information about the objective function (e.g., number of maximum and minimum) is not available, it is difficult to fix the

neighborhood range at a value that works well for different function landscapes. For instance, a chosen neighborhood range *rd* may work relatively better on objective functions where the minimum agent distance is more than *rd* rather than on those where it is less than *rd* (Krishnanand and Ghose 2009a). There is an improvement in capability of GSOA to set the peaks-captured as a function of agents by substituting a constant neighbourhood range with a variable function, where the number of peaks captured is a strong function of the radial sensor range (Krishnanand and Ghose 2006b, Krishnanand and Ghose 2009b). Hence, the GSOA applies an adaptive local-decision domain, which is used effectively to detect the multiple optimum locations of the multimodal function. Therefore, the neighbourhood range can be updated as:

$$r_d^i(t) = min\{r_s, max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\} \tag{6.14}$$

The pseudocode that is used to select the glowwarm from the neighbourhood based on the probability is provided below:

```
int index = rouletteSelect(probabilities);
if(neighborhood.size() > 0) {
        return neighborhood.get(index);
}
return null;
```

Table 6.1. shows the constant values of the parameters used in this study using the GSOA approach.

Table 6.1. The constant values of parameters used the GSOA approach

Parameters	ρ	γ	β	L_0
Constant values	0.4	0.6	0.08	5

Table 6.2. shows part of the programming approach based on the GSOA. It starts with a random population of glowworms, which generates a new population of glowworms by updating the position of glowworms and terminates when the stopping conditions or criteria are met.

Table 6.2. Part of the programming approach based on the GSOA

Set number of glowworms = n

Let $x_i(t)$ be the location of glowswam i at time t

delay_components_randomly

for i=1 to n do $L_i(0) = L_0$

$r_d^i(0) = r_0$

set maximum iteration number =

set $i_n = 1$

while ($i_n < t_{max}$) do:

{

for each glowworm i do:

$$L_i(t+1) = (1-\rho)L_i(t) + \gamma J(P_i(t+1))$$

for each glowworm i do:

{

$N_i(t) = \{j : d_{ij}(t) < r_d^i(t); L_i(t) < L_j(t)\}$;

for each glowworm $j \in N_i(t)$ do:

$$P_{ij}(t) = \frac{L_j(t) - L_i(t)}{\sum_{k \epsilon N_i(t)} L_k(t) - L_i(t)}$$

J= select_glowworm(p)

$$x(t) = x_i(t) + \left(\frac{x_j(t) - x_i(t)}{\| x_j(t) - x_i(t) \|} \right)$$

$$r_d^i(t+1) = min\{r_s, max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\}$$

}

$t \leftarrow t+1$;

}

6.3. A CAR ENGINE PUMP VALVE CASE STUDY

The engine pump valve is a real product and the University of Portsmouth has the entire details of this product, also the university make it possible for students who want to do their research

specially the Manufacturing and Formula Racing Team. The current issues with the engine pump valve are described below:

- The original number of feasible assembly sequences that provided by the designer at the early design stage (this information based on the product details obtained from the university) shows less than expected number of feasible sequences (five feasible assembly sequences) due to the number of components. Thus, the time of assembly sequence of a product can be optimise.
- The number of components can be reduced, for example, number of screws.
- The size of components can be resized.

This experiment will focus on the first issue which is the number of feasible assembly sequences and that to define the optimal assembly sequence time of the engine pump valve by using GA and GSOA and comparing the results from each algorithm to find the optimal result.

Figure 6.1. illustrates the integrated programming approach used in this work. The GA and the GSOA are used to obtain an optimal solution in terms of assembly sequence with a minimal assembly time. It starts by selecting input parameters based on number of sequences, priority

In order to examine the applicability and the validation of GA (Figure 6.2.) and GSOA models (Figure 6.6.), a real case study was applied. Table 6.3a. shows a list of components used for assembly of a car engine pump valve as a case study of this work. Figure 6.7. (also see Appendix 2) shows the drawing of assembly parts of the pump valve to be used. The drawing has been done by using CAD. Table 6.3b. (also see Appendix 2) shows the feasible assembly sequences for the engine pump valve. As clarified above, there are five feasible assembly sequences that have been provided with the entire details of this product from the University of Portsmouth (Table 6.3b. (A, B, C, D and E)). The database for the generation of the rest of

feasible assembly sequences was constituted by the Liaison graph, Figure 6.8., the table of liaisons, Table 6.4. and the table of assembly, Table 6.5.

The liaison graph is very conjectural for humans but is complicated to be managed by a computer, while it can easily handle the data in matrix method. To operate data about the product (possible assembly between components), the table of liaison will be linked to the graph of liaison.

$$L_{ij} = \begin{cases} 1 & \text{if there is a liaison between component } ai \text{ and component } aj \\ 0 & \text{otherwise} \end{cases}$$

The table of liaisons is the description of the abutment matrix of the graph of liaisons (Wilson and Watkins 1990).

Figure 6.8a. illustrates the liaison graph of the engine pump valve assembly sequences. Figure 6.8b. shows an example of the feasible assembly sequences in (A):

(A): 11,8,6,4,13,10,2,1,12,3,9,5,14,7

As described below, will start connecting component 11 with component 8, and so on, until all components assembled together.

{11,8},{6},{4},{13},{10},{2},{1},{12},{3},{9},{5},{14},{7}

{11,8,6},{4},{13},{10},{2},{1},{12},{3},{9},{5},{14},{7}

{11,8,6,4},{13},{10},{2},{1},{12},{3},{9},{5},{14},{7}

{11,8,6,4,13},{10},{2},{1},{12},{3},{9},{5},{14},{7}

{11,8,6,4,13,10},{2},{1},{12},{3},{9},{5},{14},{7}

{11,8,6,4,13,10,2},{1},{12},{3},{9},{5},{14},{7}

{11,8,6,4,13,10,2,1},{12},{3},{9},{5},{14},{7}

{11,8,6,4,13,10,2,1,12},{3},{9},{5},{14},{7}

{11,8,6,4,13,10,2,1,12,3},{9},{5},{14},{7}

{11,8,6,4,13,10,2,1,12,3,9},{5},{14},{7}

{11,8,6,4,13,10,2,1,12,3,9,5},{14},{7}

{11,8,6,4,13,10,2,1,12,3,9,5,14},{7}

{11,8,6,4,13,10,2,1,12,3,9,5,14,7}

Table 6.4. shows the liaisons between two possible assembly components. The binary numbers 0 and 1 indicate the impossibility and possibility, respectively. Table 6.5. shows the average time taken for assembly between two possible components.

Table 6.3a. Assembly components of the car engine pump valve

Component Number	Component Names
1	Arm
2	Body
3	Bolt
4	Bolt-Shaft
5	Key
6	Nut-Shaft
7	Nut3
8	Plate
9	Retainer
10	Shaft
11	Sleeve1
12	Sleeve2
13	Washer-shaft
14	Washer3

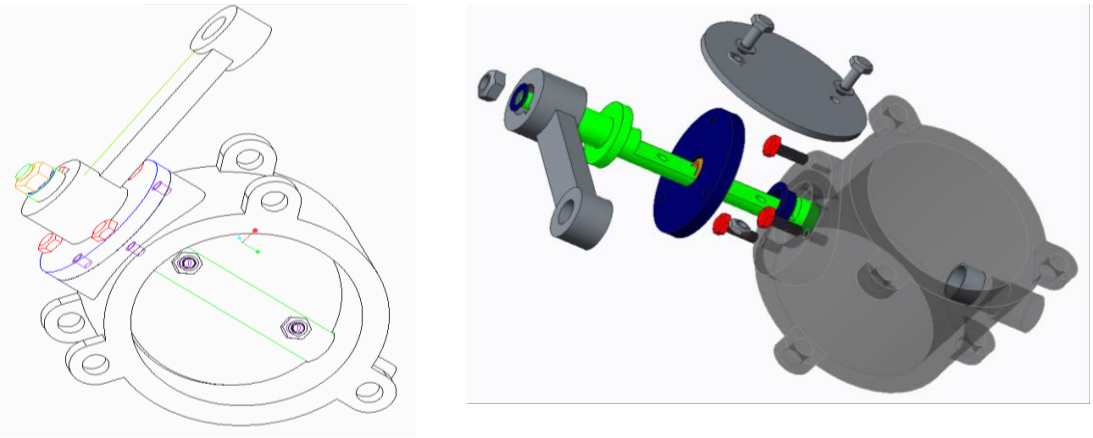


Figure 6.7. Components of the car engine pump valve

Table 6.3b. The feasible assembly sequences of the car engine pump valve

Create (A ➡ M) of the feasible assembly sequences of an automobile engine pump valve
(A): 11,8,6,4,13,10,2,1,12,3,9,5,14,7
(B): 8,11,10,6,13,4,2,1,12,9,3,5,14,7
(C): 8,11,6,10,13,4,2,1,12,9,3,5,14,7
(D): 11,8,6,4,13,10,2,1,9,12,3,5,14,7
(E): 8,11,4,6,10,13,2,1,12,9,3,5,14,7
(F): 11,8,4,6,13,10,2,1,12,3,9,5,14,7
(G): 8,11,6,4,13,10,2,1,12,9,3,5,14,7
(H): 11,8,6,4,13,10,2,1,3,9,12,5,14,7
(J): 8,11,4,10,13,6,2,1,12,9,3,5,14,7
(K): 8,11,10,6,13,4,2,1,12,9,3,5,14,7
(L): 11,8,6,4,13,10,2,1,3,9,12,5,14,7
(M): 8,11,4,10,13,6,2,1,12,9,3,5,14,7

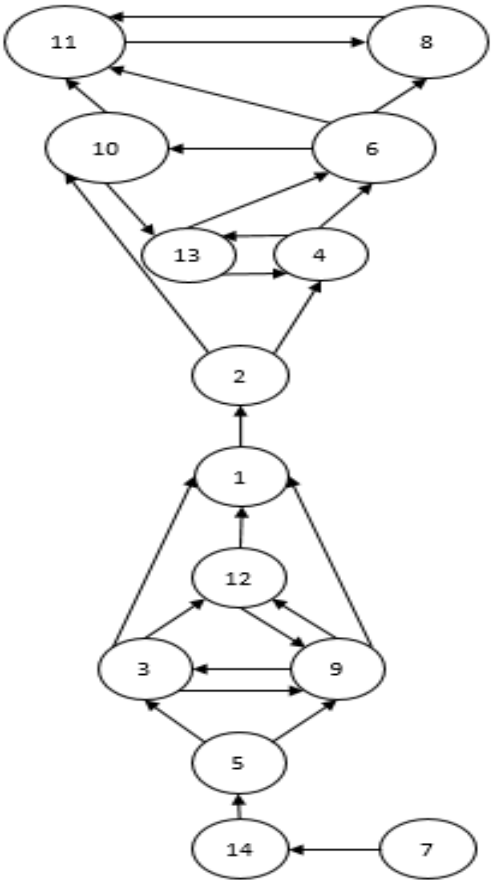


Figure 6.8. The Liaison graph for the car engine pump valve

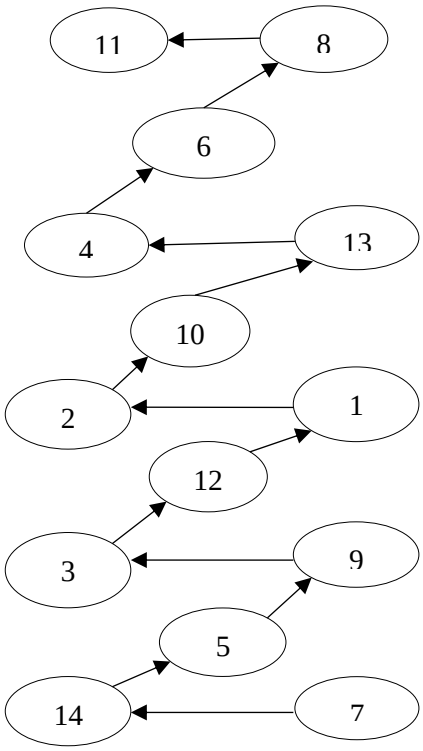


Figure 6.8b. The Liaison graph of assembly sequence A

Table 6.4. The priority matrix showing liaisons between two possible assembly components of the car engine pump valve

Component (name & number)	**Sleeve (11)**	**Plate (8)**	**Nut-Shaft (6)**	**Bolt-Shaft (4)**	**Washer-Shaft (13)**	**Shaft (10)**	**Body (2)**	**Arm (1)**	**Sleeve (12)**	**Retainer (9)**	**Bolt (3)**	**Key (5)**	**Washer (14)**	**Nut (7)**
Sleeve (11)	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Plate (8)	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Nut-Shaft (6)	1	1	1	0	0	0	0	0	0	0	0	0	0	0
Bolt-Shaft (4)	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Washer-Shaft (13)	1	1	1	1	1	0	0	0	0	0	0	0	0	0
Shaft (10)	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Body (2)	1	1	1	1	1	1	1	0	0	0	0	0	0	0
Arm (1)	1	1	1	1	1	1	1	1	0	0	0	0	0	0
Sleeve (12)	1	1	1	1	1	1	1	1	1	0	0	0	0	0
Retainer (9)	1	1	1	1	1	1	1	1	1	1	0	0	0	0
Bolt (3)	1	1	1	1	1	1	1	1	1	1	1	0	0	0
Key (5)	1	1	1	1	1	1	1	1	1	1	1	1	0	0
Washer (14)	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Nut (7)	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 6.5. Average assembly time between two possible components of the car engine pump valve

Component (name & number)	Sleeve (11)	Plate (8)	Nut-Shaft (6)	Bolt-Shaft (4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm (1)	Sleeve (12)	Retainer (9)	Bolt (3)	Key (5)	Washer (14)	Nut (7)
Sleeve1 (11)	0	2	2	1	1	3	4	2	3	1	4	5	5	4
Plate (8)	2	0	5	2	2	6	6	3	10	3	2	2	5	2
Nut-Shaft (6)	3	3	0	2	2	3	3	1	3	4	5	3	4	5
Bolt-Shaft (4)	2	5	5	0	11	15	4	4	4	4	4	5	8	2
Washer-Shaft (13)	4	4	10	10	0	7	13	2	5	6	5	4	6	3
Shaft (10)	3	5	2	7	7	0	2	13	7	8	6	6	4	5
Body (2)	4	8	1	3	3	4	0	3	18	7	7	7	6	8
Arm (1)	6	7	2	8	8	5	6	0	6	6	4	3	5	6
Sleeve (12)	8	6	4	5	5	8	7	17	0	52	2	4	7	4
Retainer (9)	9	8	6	2	2	7	18	7	4	0	42	2	8	7
Bolt (3)	7	6	8	8	8	13	6	5	3	3	0	5	5	5
Key (5)	4	14	18	7	7	4	4	3	6	4	5	0	4	4
Washer (14)	2	6	6	2	9	6	2	4	1	5	6	4	0	1
Nut (7)	4	3	5	4	4	5	3	4	8	7	3	4	1	0

Note: Assembly time is calculated in seconds.

Output from the Genetic Algorithm Implementation for Pump

The Genetic Algorithm was implemented and in Java and it is continuously iterated for 5 generations by creating new chromosome. Each result shows the assembly time in response to each of 10 chromosomes, of which each depicts a possible assembly sequence for the car engine pump valve. In the end of each generation assembly time in seconds were computed of the chromosomes were generated to plot the graphs. The Java implementation of the GA algorithm is provided in appendix A. The figure 6.9a provides the generation 1 from the GA where the highest assembly time was 690s and the smallest assembly time was 567s. The figure 6.9b provides the generation 2 from the GA where the highest assembly time was still 690 and the smallest assembly time was 510s. This indicates that the assembly time from generation 1 to generation 2 was reduced by 57s. The figure 6.9c provides the generation 3 from the GA where the highest assembly time was 580 and the smallest assembly time was 500s. This indicates that the assembly time was further reduced in 3rd generation by 10s. The figure 6.9d provides the generation 4 from the GA where the highest assembly time was 530s and the smallest assembly time was 500s. The figure 6.9e provides the generation 5 from the GA where there is no highest or smallest assembly time where all the iteration got the same results which is 500s. Therefore, it is clear that the lowest assembly time taken in the GA was 500s which came in the 3rd and 4th generation but got prevalence in 5th generation. Moreover, from the observation it is possible to state that the 2nd generation of GA had more fluctuation in the assembly times compared to other generations.

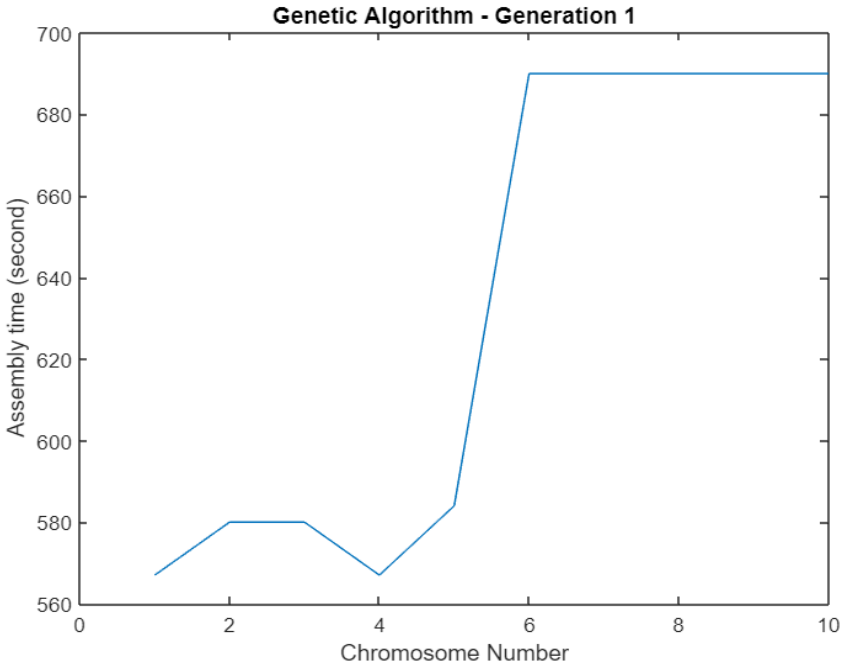


Figure 6.9 a. Assembly time obtained using the GA in response to each of chromosomes in generation 1

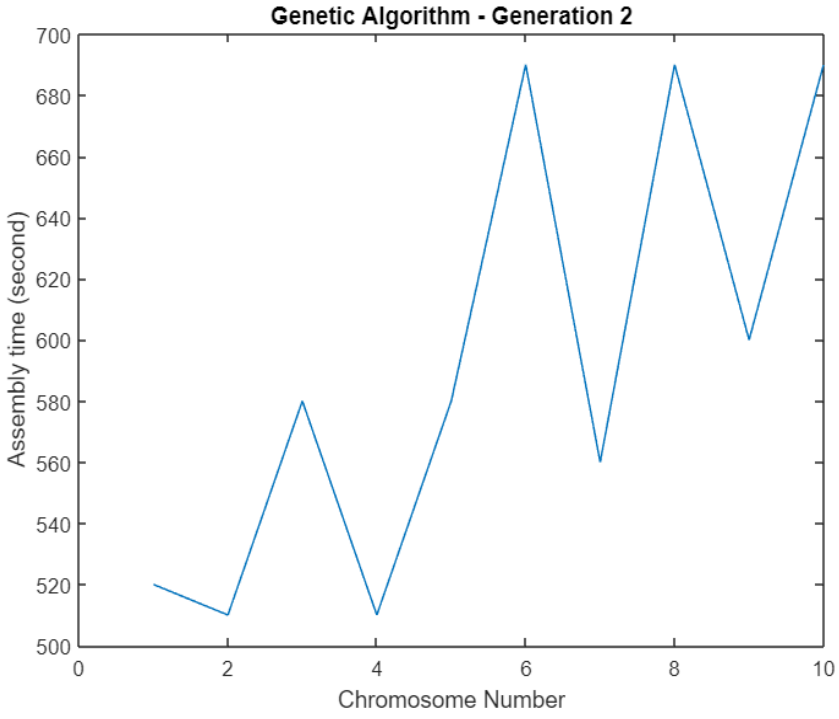


Figure 6.9 b. Assembly time obtained using the GA in response to each of chromosomes in generation 2

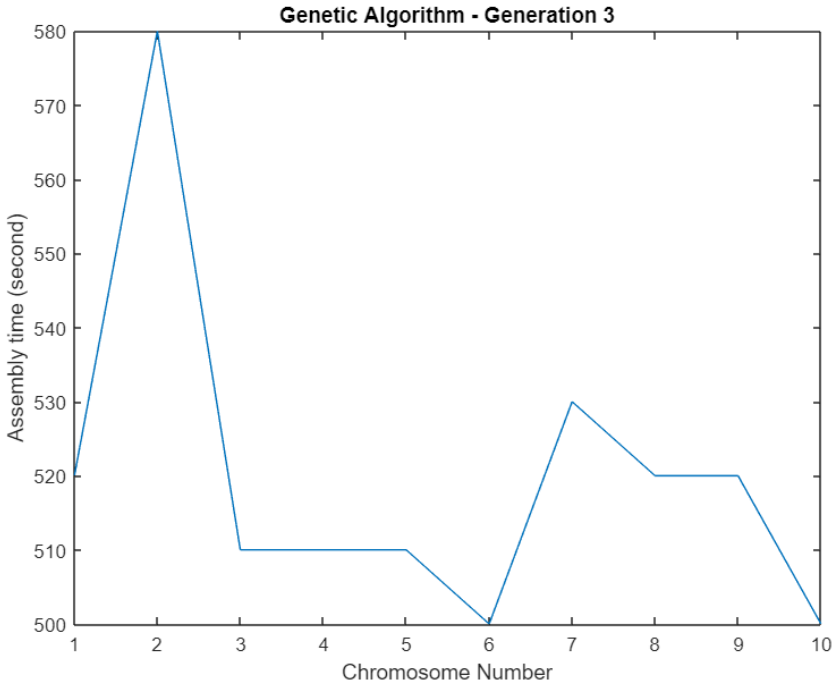


Figure 6.9 c. Assembly time obtained using the GA in response to each of chromosomes in generation 3

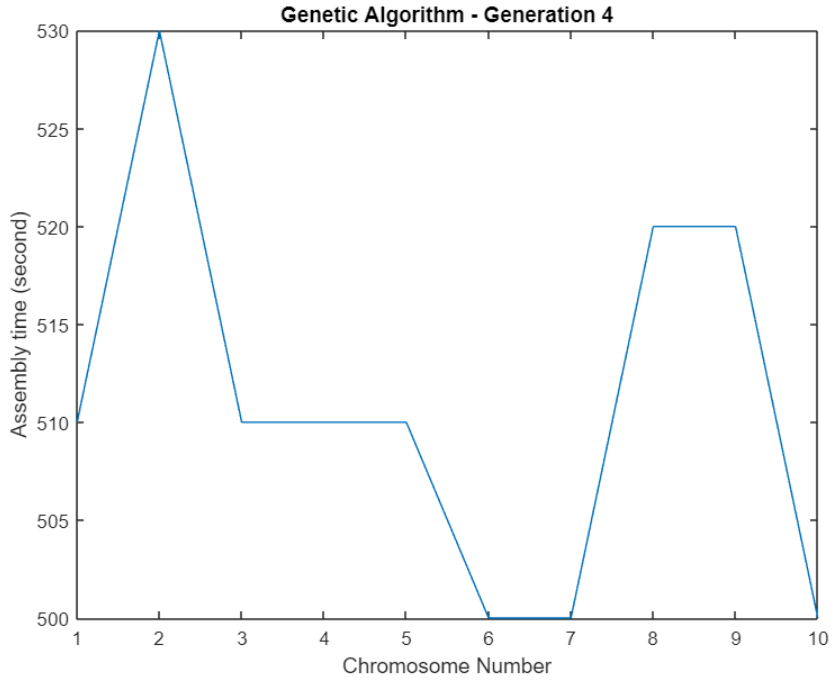


Figure 6.9 d. Assembly time obtained using the GA in response to each of chromosomes in generation 4

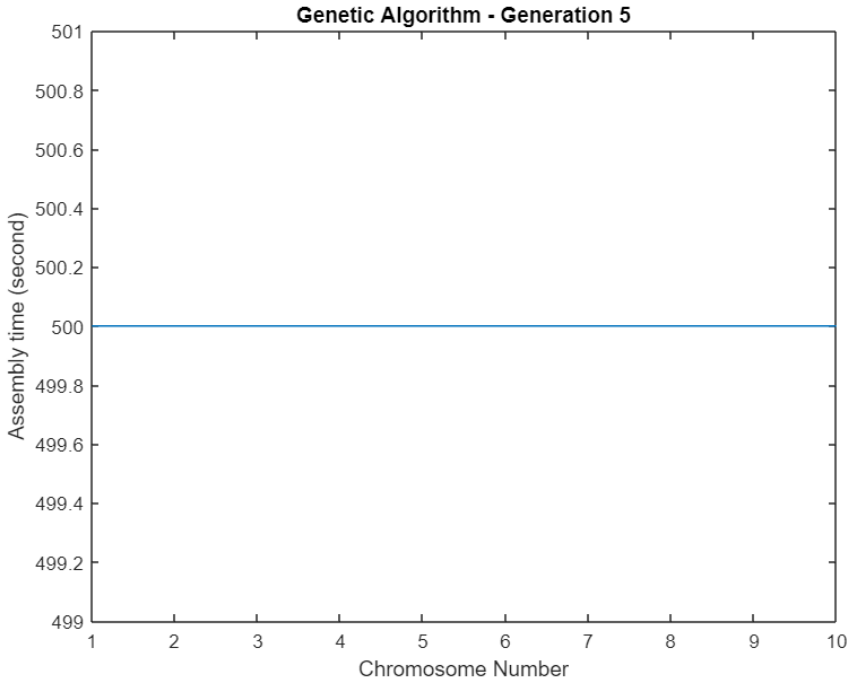


Figure 6.9e. Assembly time obtained using the GA in response to each of chromosomes in generation 5

Figure 6.10. shows the comparison in assembly time between the theoretical results and the computerised results obtained from the GA programming (using Java language) under the same conditions, which are associated with the generation number from 1 to 5, respectively.

There is a reason behind using the theoretical calculation in the research case studies and that because both products are between Simple product and Medium product (see section 6.2.), also the feasible assembly sequence for both products are defined whether by the product designer at the early stage or during the experiment. Based on the feasible assembly sequence and the assembly time for each component and the calculation formula that has been applied during this research, then it can be clear to obtain 2 types of results (Theoretical results and Computerised results) and compare between them to find the optimal solution. But it is really hard to apply the theoretical calculation for a large product (see section 6.2.) due to the number of components and hard to manage all feasible assembly sequence.

It can be seen that the assembly time obtained from generation 1 is 570 seconds, which is slightly higher than 567 seconds obtained from the theoretical result. For the result of generation 2, the assembly time obtained from both ways is approximately the same. After this generation, the difference of assembly time between theoretical results and computerised results is equal to 50 seconds. It is important to note that both in theoretical results and computerised results the minimum assembly time of the pump did not change. Therefore, it is possible to derive that the computerised algorithm is more effective in evolving and identifying new chromosome that can minimise the assembly time.

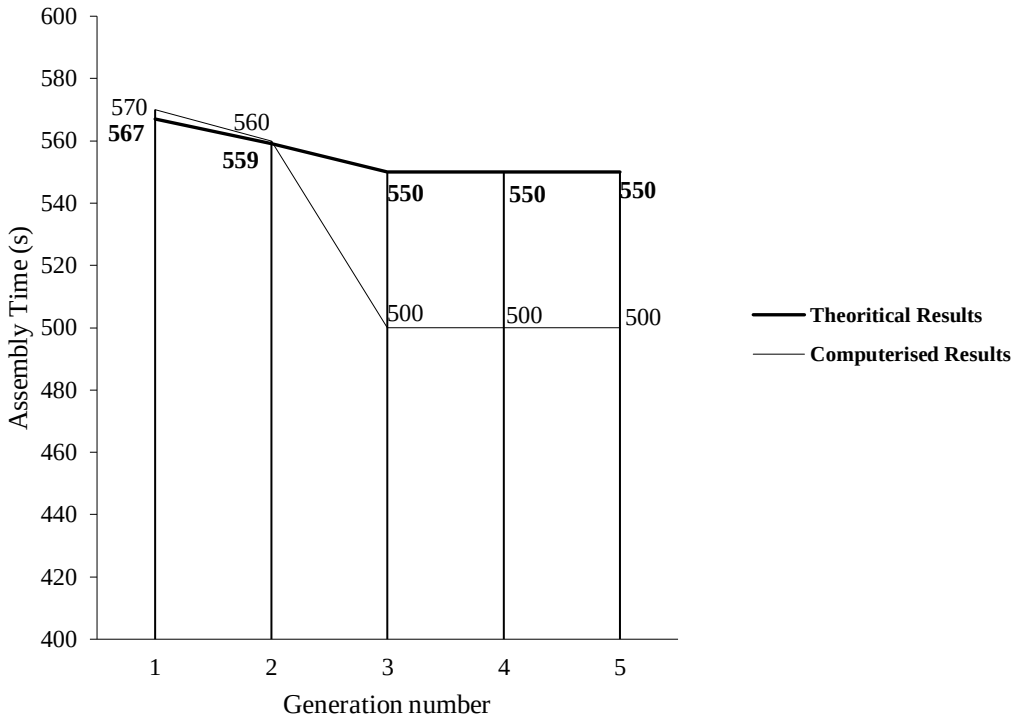


Figure 6.10. Comparison in assembly time between the theoretical result and computerised result using the GA in response to generation number

Output from the Glowswarm Algorithm Implementation for Pump

The Glowswarm Optimisation Algorithm was implemented and in Java and it is continuously iterated for 5 generations by creating new agents. In the end of each generation fitness values of the gents were generated to plot the graphs. The Java implementation of the GSOA algorithm is provided in appendix 2. From the analysis with GA algorithm the generation responses from the GSOA algorithm is slightly different. The figure 6.11a provides the generation 1 from the GSOA where the highest assembly time was 646s and the smallest assembly time was 520s. The figure 6.11b provides the generation 2 from the GSOA where the highest assembly time was still 649s and the smallest assembly time was 510s. This indicates that the assembly time from generation 1 to generation 2 was reduced by 10s. The figure 6.11c provides the generation 3 from the GSOA where the highest assembly time was 646s and the smallest assembly time was 500s. This indicates that the assembly time was further reduced in 3rd generation by 10s. The figure 6.11d provides the generation 4 from the GSOA where the highest assembly time was 530s and the smallest assembly time was 494s. The figure 6.11e provides the generation 5 from the GSOA where the highest assembly time was 500s and the smallest assembly time was 494s. Therefore, it is clear that the lowest assembly time taken in the GSOA was 494s which came in the 4th generation but got prevalence in 5th generation. Moreover, from the observation it is possible to state that the 3rd generation of GSOA had more fluctuation in the assembly times compared to other generations.

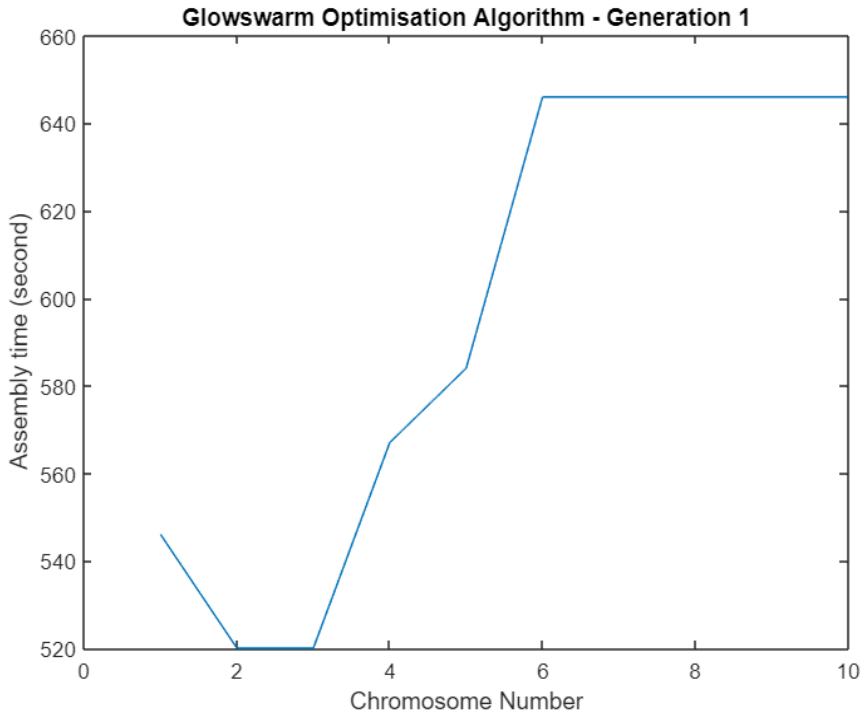


Figure 6.11a. Assembly time obtained using the GSOA in response to each of chromosomes in generation 1

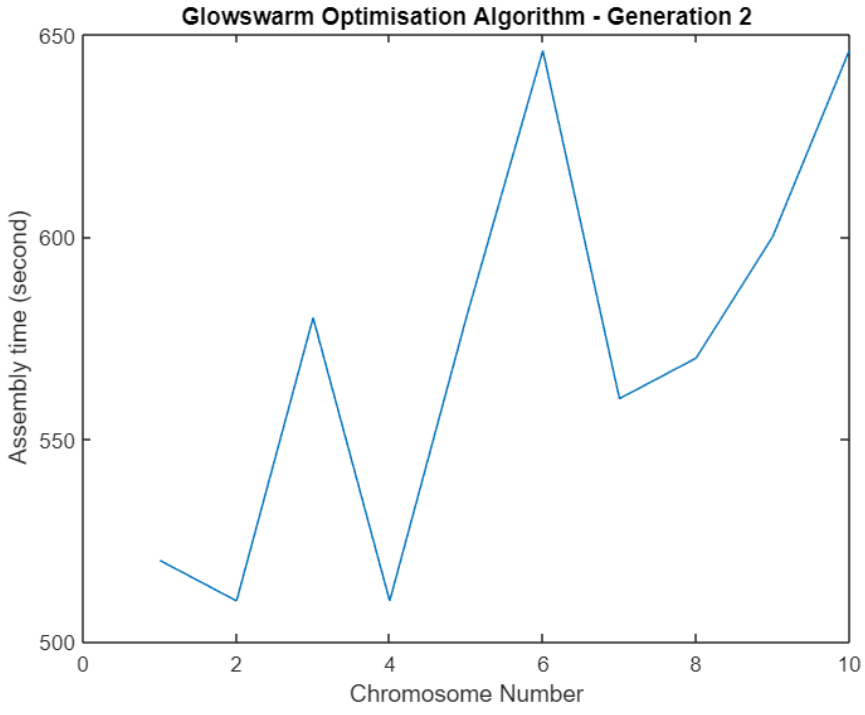


Figure 6.11b. Assembly time obtained using the GSOA in response to each of chromosomes in generation2

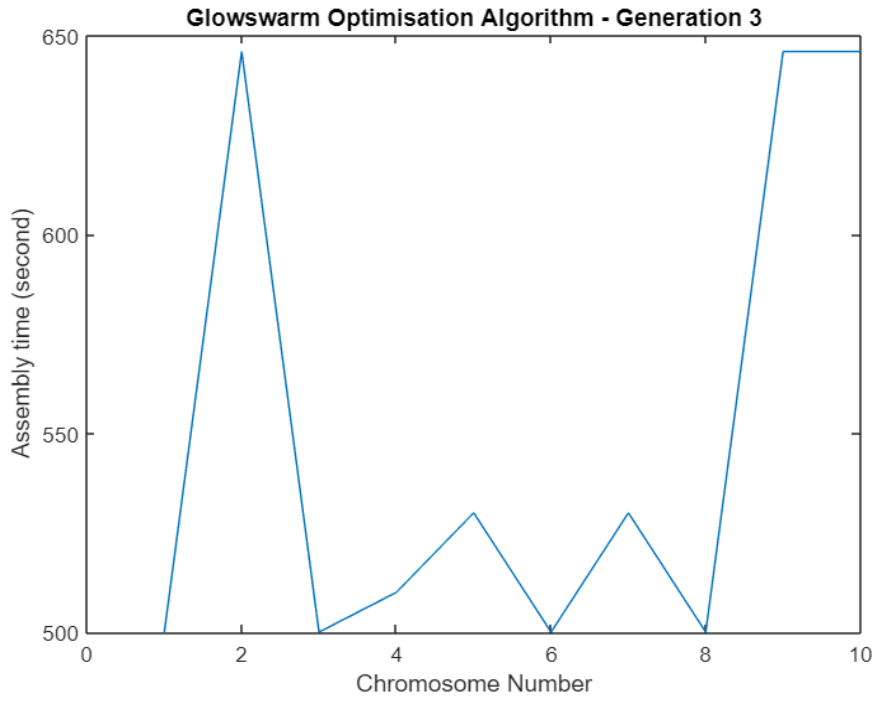


Figure 6.11c. Assembly time obtained using the GSOA in response to each of chromosomes in generation 3

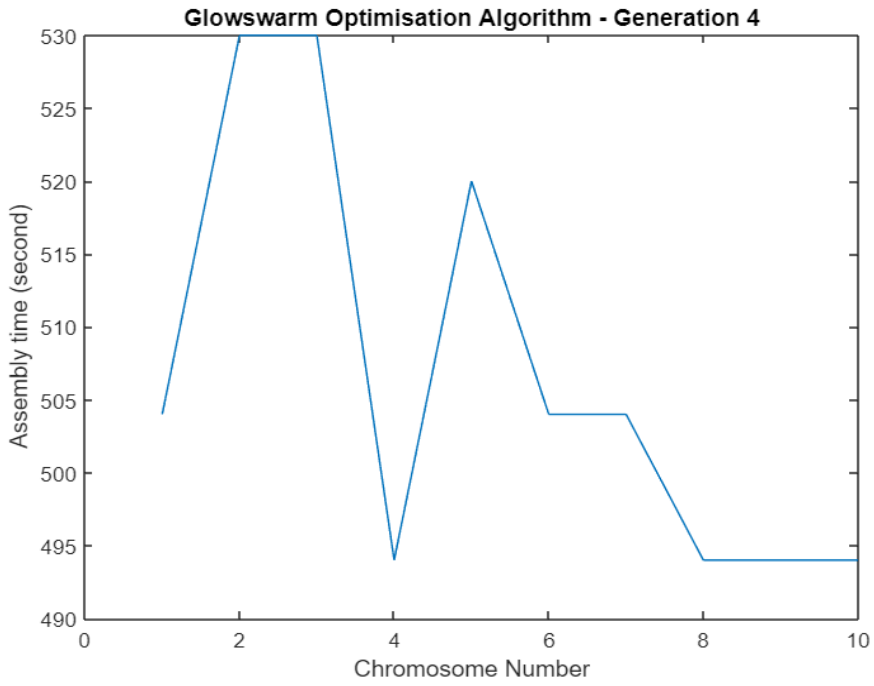


Figure 6.11d. Assembly time obtained using the GSOA in response to each of chromosomes in generation 4

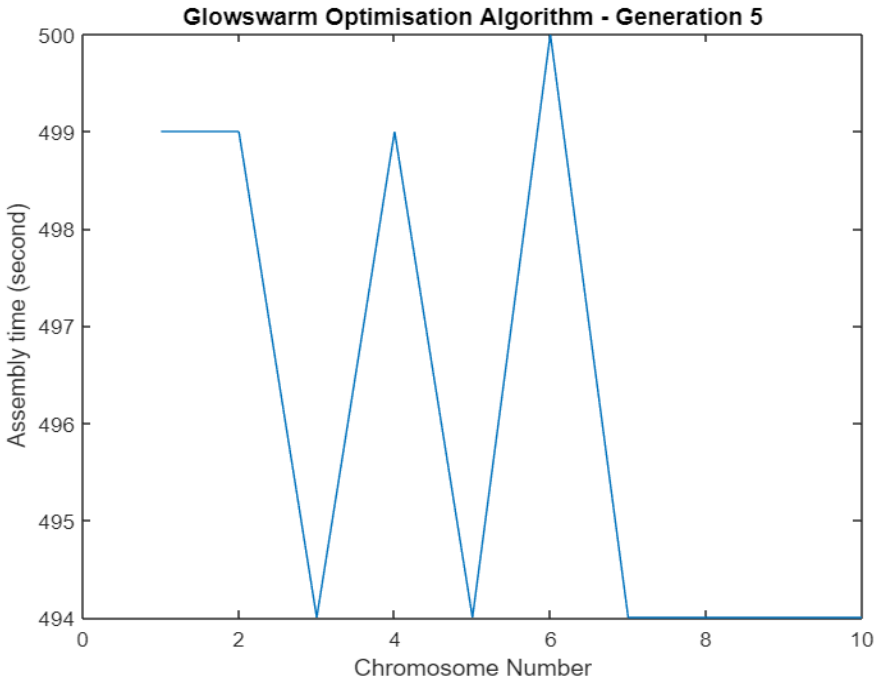


Figure 6.11e. Assembly time obtained using the GSOA in response to each of chromosomes in generation 5

Figure 6.12. shows the comparison in assembly time between the theoretical results and the computerised results obtained from the GSOA programming (using Java language) under the same conditions, which are associated with the generation number from 1 to 5, respectively. The graph indicates that that the computerised assembly time in 1st generation is higher than the theoretical results by 16s. However, the difference between them reduced in 2nd generation but still the theoretical results remained better than the computerised results. The theoretical results and computerised results of GSOA are same. However, from the 4th and 5th generations the computerised results are lower than theoretical results by 6s. However, it is identified in the theoretical results the lowest assembly time was identified in the 3rd generation where else, in the computerised results the lowest assembly time was identified only in the 4th generation. It is possible to derive that the computerised algorithm is more effective in evolving and identifying new chromosome that can minimise the assembly time.

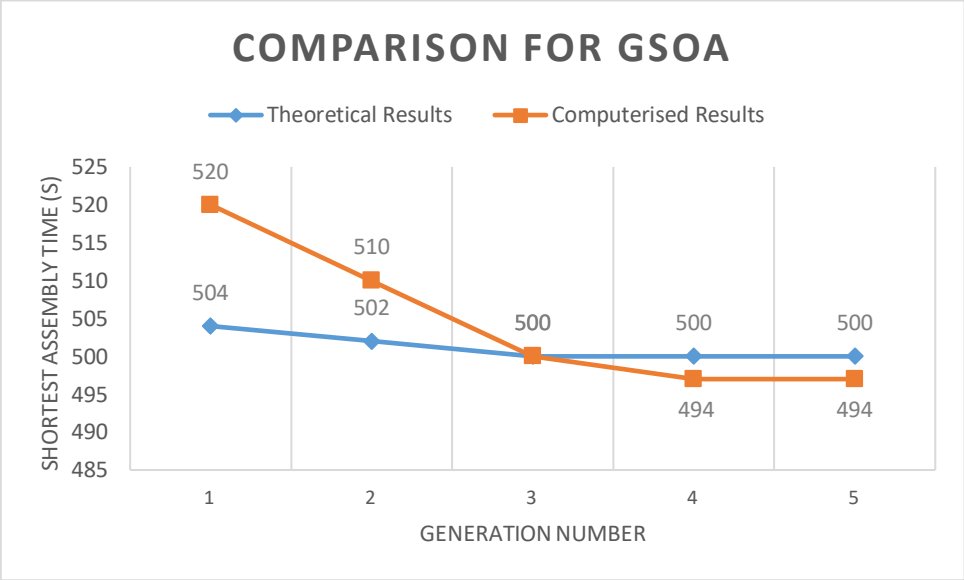


Figure 6.12 shows the comparison in assembly time between the theoretical result and computerised result of GSOA in response to the generation number

By comparing the results obtained using the GA and the GSOA, respectively, it can be seen in Figure 6.9e and 6.11e that both the computerised results have the lowest value of assembly time which is 500 seconds for GA and 494 seconds for GSOA. The comparative result also shows that the GSOA outperforms the GA as the minimal assembly time obtained using the GA is 500 seconds, compared to 494 seconds using the GSOA as illustrated in Figure 6.10 and Figure 6.12. As a result of this, there is an average of 6 seconds per unit in the reduction of assembly time of the engine pump valve assembly. However, it is also identified that in the 5^{th} generation all the iterations got the lowest assembly time for GA but only some of the iterations got the lowest assembly time for GSOA.

6.4. A BALL PEN CASE STUDY

The ball pen is defined as a small product due to the number of components. This product has been used by some researchers for different reasons (e.g. explaining the assembly system) (Fawaz and Qian 2017), (see section 3.5).

The current issues with a ball pen product are described below:

- Reducing assembly times (therefore costs) is vital particularly for small manufacturing companies to survive in an increasingly competitive market. Thus, the study provides and approach in obtaining an optimal or near-optimal assembly sequence of the product for a small-sized company.
- The material of components.

This second experiment will use GA and GSOA to define the optimal assembly sequence time of ball pen and comparing the results from each algorithm to find the optimal result.

Table 6.13. shows a list of components used for assembly of a ball pen. Figure 6.13. illustrates a sequential order of assembly components of a ball pen. Figure 6.15. shows the feaseible assembly sequences of the ball pen. Figure 6.15. illustrates the liaison graph of the ball pen assembly sequences. Table 6.14. shows the liaisons between two possible assembly components. Table 6.15. shows the average time taken for assembly between two possible components.

Table 6.13. Assembly components of the ball pen

Component Number	Component Name
1	Cap
2	Head
3	Tube
4	Ink (fluid)
5	Body
6	Button

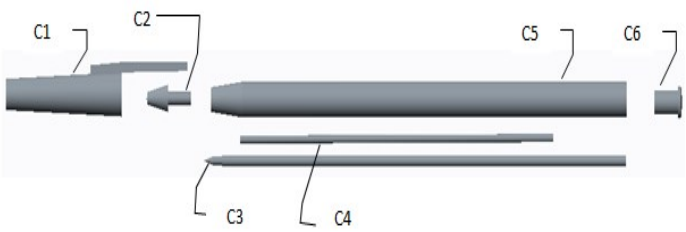


Figure 6.13. The ball pen assembly components

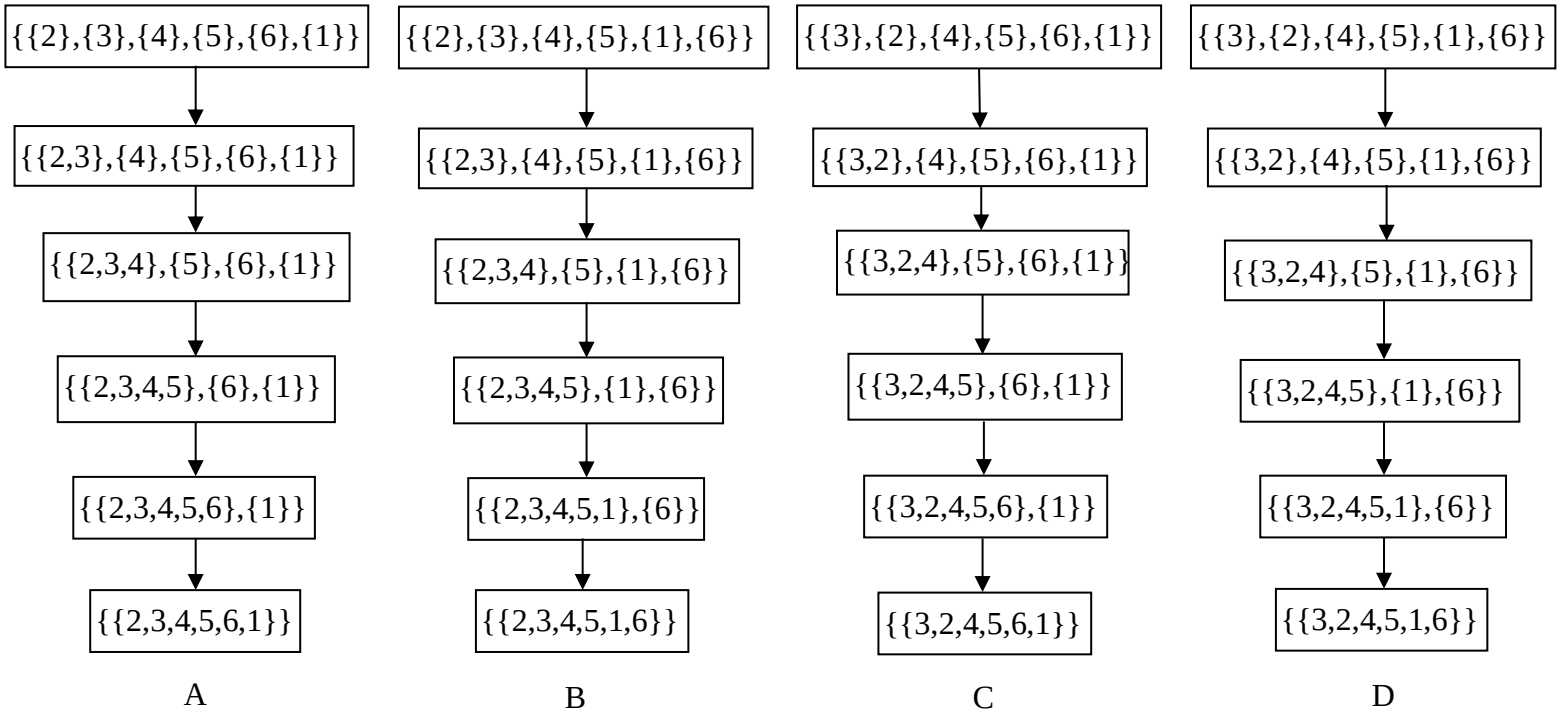


Figure 6.14. The feasible assembly sequences (A, B, C, D) of the ball pen

The components of the entire ball pen are assigned an assembly part numbers, ranging from 1-6, most importantly, all the possible sequences are equally shown in Figure 6.14. The four chosen possible sequences are taken as only reasonable paths and for the sake of time and cost management, as well as putting simplicity into consideration. The first part to start the assembly cannot be c4 or c1. Starting with c4 is obviously impossible, as the ink (liquid) has to be contained in something, in this case in c2 and c3. The assembly might start with c2, to which c3 is added, then the link c4 is squirted, the body c5 and the button c6 are inserted, then the cap concludes the assembly. This assembly sequence is: c2, c3, c4, c5, c6, c1, another feasible assembly sequence is: c2, c3, c4, c5, c1, c6.

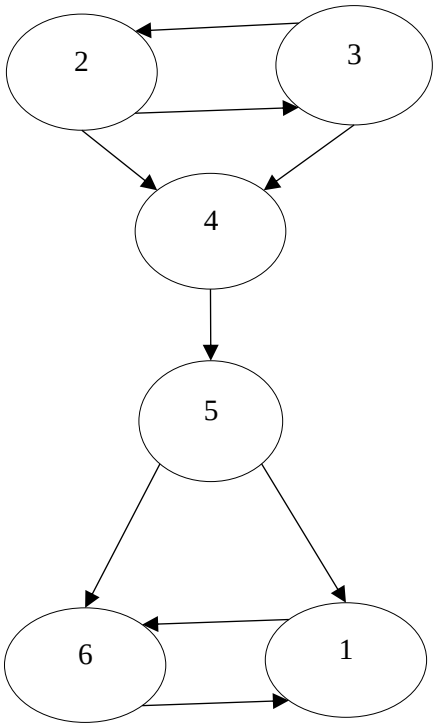


Figure 6.15. The liaison graph for the ball pen

Table 6.14. The liaisons between two possible assembly components of the ball pen

Component number	a1	a2	a3	a4	a5	a6
a1	0	1	0	0	0	0
a2	1	0	1	0	1	0
a3	0	1	0	1	0	0
a4	0	0	1	0	1	0
a5	0	1	0	1	0	1
a6	0	0	0	0	1	0

Table 6.15. Average assembly time between two possible components of the ball pen

Component number	Cap	Head	Tube	Ink	Body	Button
Cap	0	3	4	5	6	6
Head	3	0	4	5	6	8
Tube	4	5	0	6	7	7
Ink	5	6	7	0	8	8
Body	6	7	8	9	0	7
Button	3	4	6	7	4	0

Output from the Genetic Algorithm Implementation for Pen

The Genetic Algorithm was implemented and in Java and it is continuously iterated for 5 generations by creating new chromosome. Each result shows the assembly time in response to each of 10 chromosomes, of which each depicts a possible assembly sequence for the pen. In the end of each generation assembly time in seconds were computed of the chromosomes were generated to plot the graphs. The Java implementation of the GA algorithm is provided in appendix A. The figure 6.16a provides the generation 1 from the GA where the highest assembly time was 40s and the smallest assembly time was 30s. The figure 6.16b provides

the generation 2 from the GA where the highest assembly time was still 38s and the smallest assembly time was 30s. This indicates that the assembly time from generation 1 to generation 2 did not reduce. The figure 6.16c provides the generation 3 from the GA where the highest assembly time was 33s and the smallest assembly time was 26s. This indicates that the assembly time was reduced in 3rd generation by 4s. The figure 6.16d provides the generation 4 from the GA where the highest assembly time was 33s and the smallest assembly time was 26s. The figure 6.16e provides the generation 5 from the GA where there is no highest or smallest assembly time where all the iteration got the same results which is 26s. Therefore, it is clear that the lowest assembly time taken in the GA was 26s which came in the 3rd and 4th generation but got prevalence in 5th generation. Moreover, from the observation it is possible to state that the 2nd generation of GA had more fluctuation in the assembly times compared to other generations.

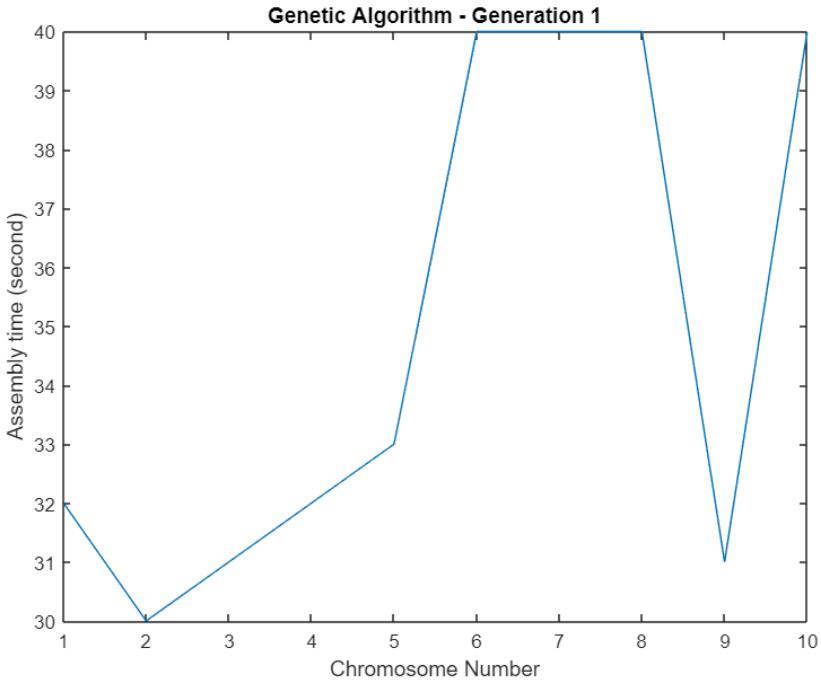


Figure 6.16a. Assembly time obtained using the GA in response to each of chromosomes in generation 1

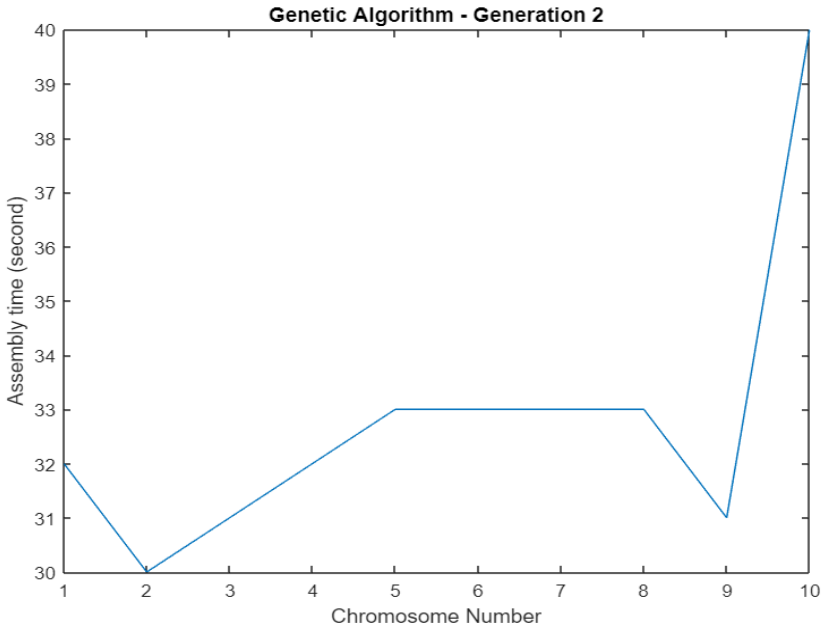


Figure 6.16b. Assembly time obtained using the GA in response to each of chromosomes in generation 2

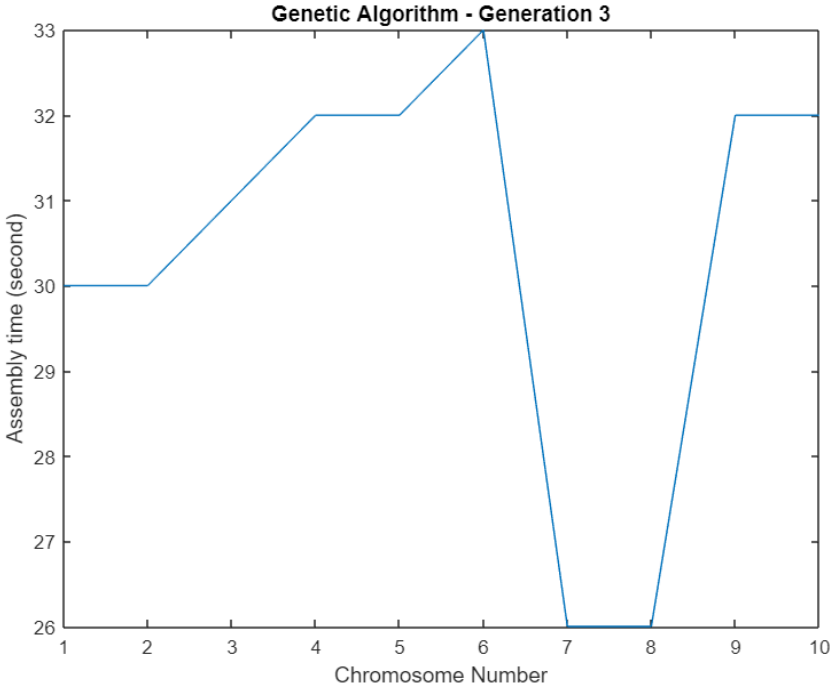


Figure 6.16c. Assembly time obtained using the GA in response to each of chromosomes in generation 3

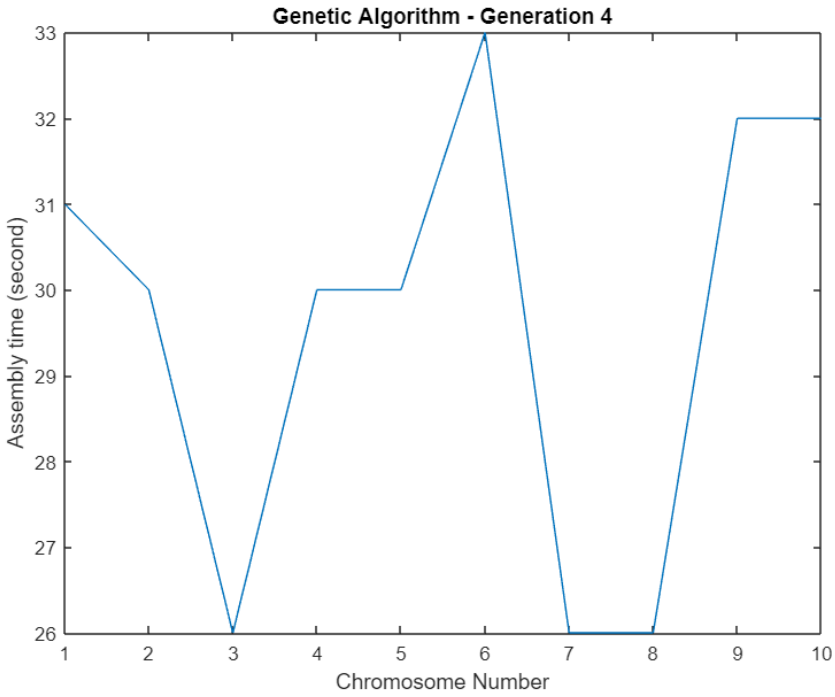


Figure 6.16d. Assembly time obtained using the GA in response to each of chromosomes in generation 4

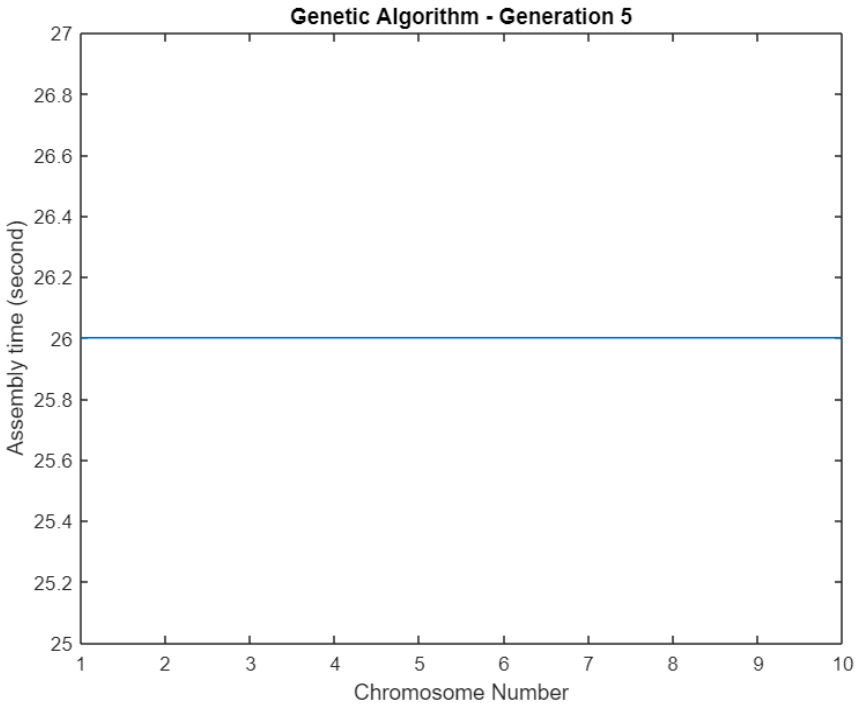


Figure 6.16e. Assembly time obtained using the GA in response to each of chromosomes in generation 5

Figure 6.17. shows the comparison in assembly time between the theoretical results and the computerised results obtained from the GA programming (using Java language) under the same conditions, which are associated with the generation number from 1 to 5, respectively. The graph indicates that that the computerised assembly time in 1st generation is higher than the theoretical results by 1s. However, the difference between them was same in 2nd generation. However, since 3rd generation the assembly time of computerised results is 2s lower than the theoretical results. It is important to note that both in theoretical results and computerised results the minimum assembly time of the pen did not change. Therefore, it is possible to derive that the computerised algorithm is more effective in evolving and identifying new chromosome that can minimise the assembly time.

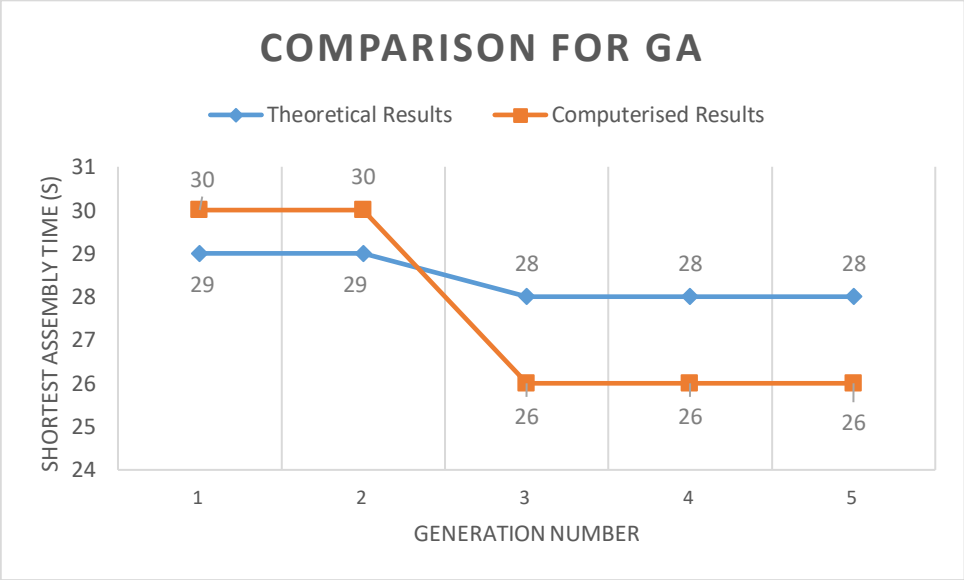


Figure 6.17 shows the comparison in assembly time between the theoretical result and computerised result of GA in response to the generation number

Output from the Glowswarm Algorithm Implementation for Pen

The Glowswarm Optimisation Algorithm was implemented and in Java and it is continuously iterated for 5 generations by creating new agents. In the end of each generation fitness values of the gents were generated to plot the graphs. The Java implementation of the GSOA algorithm is provided in appendix B. From the analysis with GA algorithm the generation responses from the GSOA algorithm is slightly different. The figure 6.18a provides the generation 1 from the GSOA where the highest assembly time was 38s and the smallest assembly time was 31s. The figure 6.18b provides the generation 2 from the GSOA where the highest assembly time was still 38s and the smallest assembly time was 31s. This indicates that the assembly time from generation 1 to generation 2 did not reduce. The figure 6.18c provides the generation 3 from the GSOA where the highest assembly time was 33s and the smallest assembly time was 24s. This indicates that the assembly time was further reduced in 3rd generation by 7s. The figure 6.18d provides the generation 4 from the GSOA where the highest assembly time was 32s and the smallest assembly time was 24s. The figure 6.18e provides the generation 5 from the GSOA there is no highest or smallest assembly time because all the iterations were 24s assembly time. Therefore, it is clear that the lowest assembly time taken in the GSOA was 24s which came in the 3^{rd} and 4^{th} generation but got prevalence in 5th generation. Moreover, from the observation it is possible to state that the 2nd generation of GSOA had more fluctuation in the assembly times compared to other generations.

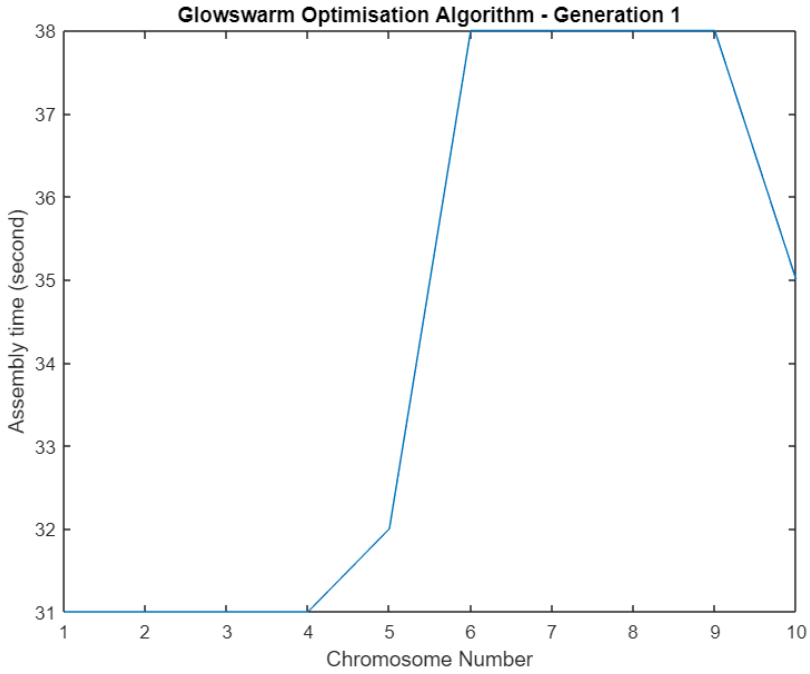


Figure 6.18a. Assembly time obtained using the GSOA in response to each of chromosomes in generation 1

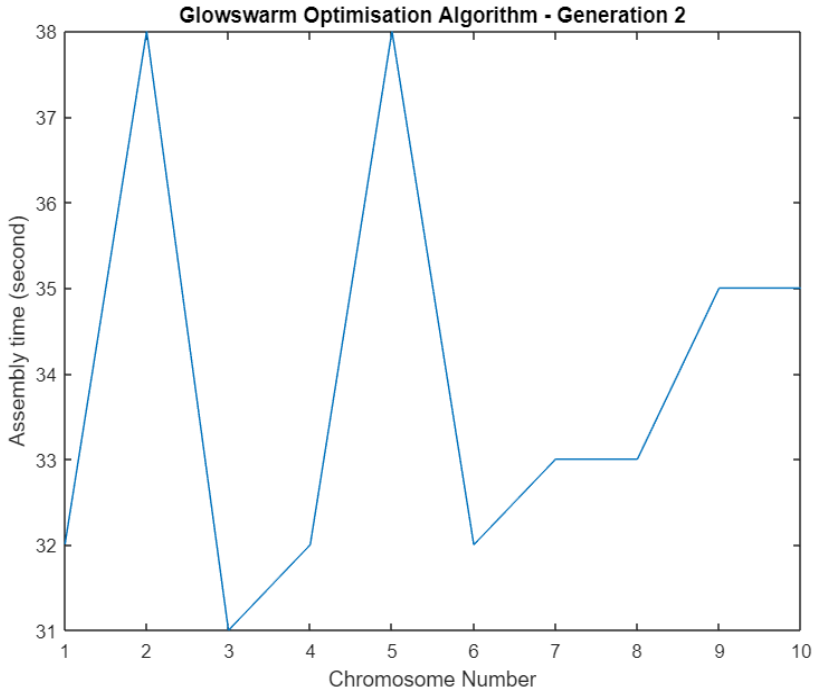


Figure 6.18b. Assembly time obtained using the GSOA in response to each of chromosomes in generation 2

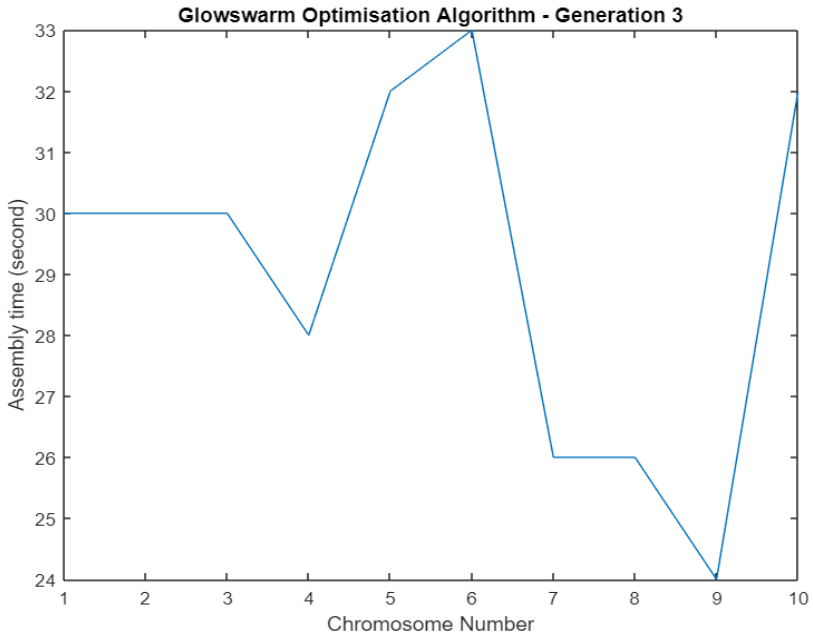


Figure 6.18c. Assembly time obtained using the GSOA in response to each of chromosomes in generation 3

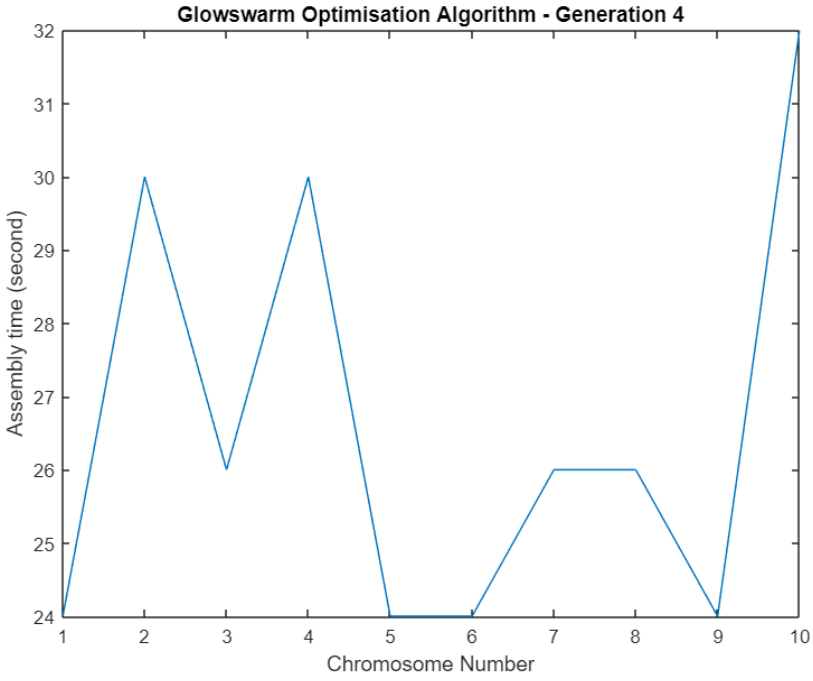


Figure 6.18d. Assembly time obtained using the GSOA in response to each of chromosomes in generation 4

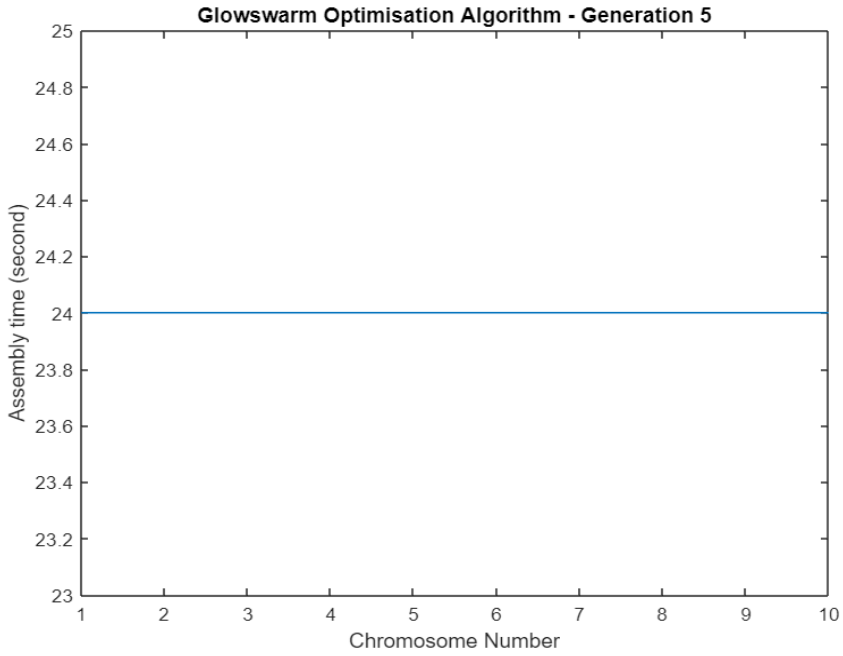


Figure 6.18e. Assembly time obtained using the GSOA in response to each of chromosomes in generation 5

Figure 6.19. shows the comparison in assembly time between the theoretical results and the computerised results obtained from the GSOA programming (using Java language) under the same conditions, which are associated with the generation number from 1 to 5, respectively. The graph indicates that that the computerised assembly time in 1st generation is higher than the theoretical results by 2s. However, the difference between them increased in 2nd generation by 4s. However, from the 3rd, 4th and 5th generations the computerised results are lower than theoretical results by 1s. However, it is identified in the lowest assembly time was identified in the 3rd generation. It is possible to derive that the computerised algorithm is more effective in evolving and identifying new chromosome that can minimise the assembly time.

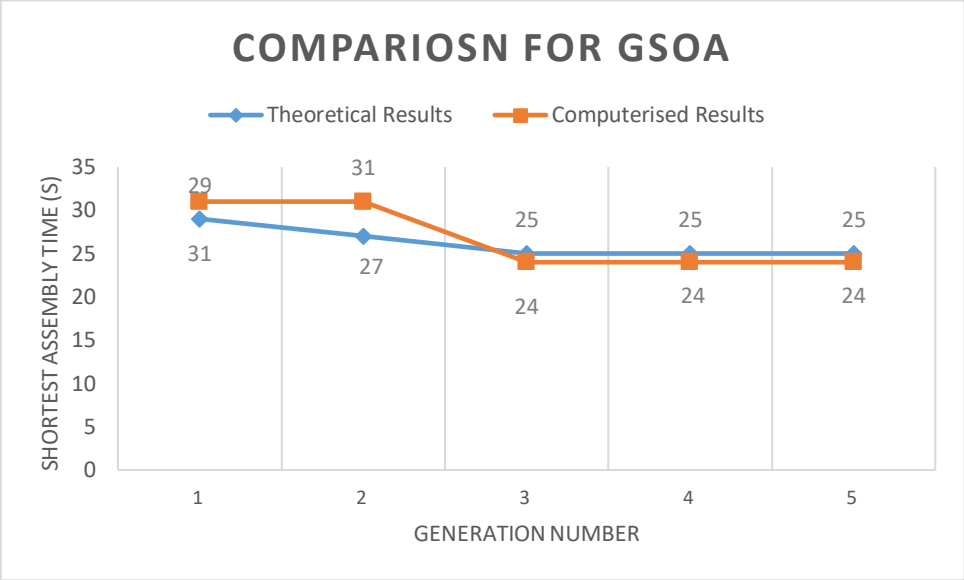


Figure 6.19 shows the comparison in assembly time between the theoretical result and computerised result of GSOA in response to the generation number

By comparing the results obtained using the GA and the GSOA, respectively, it can be seen in Figure 6.16e and 6.18e that both the computerised results have the lowest value of assembly time which is 26 seconds for GA and 24 seconds for GSOA. The comparative result also shows that the GSOA outperforms the GA as the minimal assembly time obtained using the GA is 26 seconds, compared to 24 seconds using the GSOA as illustrated in Figure 6.17 and Figure 6.19. As a result of this, there is an average of 2 seconds per unit in the reduction of assembly time of the ball pen assembly.

CHAPTER 7

DISCUSSION, CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

7.1. DISCUSSION

The thesis reports a study into investigation of GA and GSOA optimisation methods for solving assembly sequence optimisation problems with a development of a multi-objective optimisation model that can be used for quantifying energy consumptions and costs for assembly of a product. Assembly has an important share in manufacturing costs and lead time, thus, optimisation of assembly sequence of a product can have a significant positive impact. Chapter 1 introduced the ASP and its optimisation involves components and operations of these components in possible forms. The research was carried out in chapter 2 by examining the previous attempts to solve the ASP optimisation problem, and the authors identified a number of issues, so far that it is not adequately handled in the context of optimising dealing with a large scale, highly constrained, combinatorial optimisation problem, which are, precisely, the features of the ASP problem. It was pointed out that all previous developments in S/O of the ASP only concerned reduced-size problems. Due to the character of the problem and the lack of proper tools, it is the impossibility to tackle full-scale problems required large scale, artificial, reductions in its size and complexity. This has been done, previously, by using artificially simplifying hypothesis, Hence, the results may not be used directly in practice, if and when they were obtained.

Chapter 3, 4 reviewed,The study shows that GA can be used as an optimisation tool for solving combinatorial problems by representing assembly sequences as chromosomes. Their selection as an optimisation tool for ASP was justified, along with particulars of the special

GA to be used in this case.It was also pointed out that a successful S/O of the ASP has to start with a proper modelling activity. Thus, prior to attempting to S/O the ASP problem, a number of models had to be considered and assessed. A proper modelling method of the AS problem encoded as chromosomes is crucial as it directly influences the type and variety of assembly sequences and plans that can be generated/optimised. It also modelling the product for assembly purposes - encoding and storing constraints in assembly. The quality of this model directly affects the complexity of relations that appear between the components and assembly operations in a product and can be considered during assembly sequence generation and optimisation. The degree of realism can also be incorporated in the optimisation algorithms. In Chapter 5, the idea of GSOA was derived from the nature of glowworms who are able to modify the amplitude of their light emission and use the bioluminescence glow for different purposes. It is involved in a deployment of glowworms, luciferin-update, movement and local-decision domain. Through a literature review, it was found that the GSOA method was not reported as being used for solving the assembly sequence optimisation problem. The glowworm swarm optimisation is the latest and most advance method of swarm intelligence method. Also, this study shows that the GSOA can be an effective approach used for a simultaneous search in obtaining an optimal solution in terms of assemble sequence of a product of multiple optimal values usually based on different objective functions.

7.2. CONCLUSIONS

The study demonstrates the feasibility and applicability using the GA and the GSOA approaches for resolving the assembly sequence optimisation problem for the car engine pump valve and the ball pen products in terms of reduction of assembly time. The result indicates

that the GSOA outperforms the GA with a reduction of 3 seconds in assembly time per unit of the car engine pump valve. The study also demonstrated that this can be a useful decision-making tool in obtaining an optimal or near-optimal assembly sequence of for product designers. Moreover, it can be proofed that GSOA gives minimal assembly time than GA by looking at the results from the ball pen case study.

7.2.1. GA and GSOA

The study demonstrates the feasibility and applicability using the GA and the GSOA approaches for resolving the assembly sequence optimisation problem for the car engine pump valve. The aim of this study aimed to reduce assembly time using in terms of reduction of assembly time, both algorithms were implemented in Java. Both GA and GSOA programing approaches were described. It The result indicates that the GSOA outperforms is outperforming the GA with a reduction of 6 seconds in assembly time per unit of the car engine pump valve. A reduction of 6 seconds is not generic and not the same if GSOA applied to different product. Furthermore, the results of the second case study (ball pen) shows that GSOA has an optimal assembly time than GA with a reduction of 2 seconds. The study also demonstrated that this can be a useful decision-making tool in obtaining an optimal or near-optimal assembly sequence of for product designers.

7.2.2. NOVELTY

The previous studies show that GSOA has not been used to solve the ASP issued and the results of this research shows that GSOA gives optimal results specially for assembly sequence time.

7.3. FUTURE WORK

It is suggested that the further work in assembly planning and optimisation may consider the following issues:

- Development of a multi-objective GSOA model which can be used for making a trade-off decision based on a number of criteria specified by users.
- This model can also incorporate a number of parameters relating to walking-worker assembly of products in which assembly performance can be largely affected by human workers in a human-centred assembly system.
- Mathematical or analytical modelling techniques might not be sufficient if a detailed analysis is required for a complex assembly process as the objective function may not be expressible as an explicit function of the input parameters. Thus, an integrated simulation-based GSOA method incorporating these parameters based on a discrete even simulation model is recommended as part of this study.
- Development of the proposed GSOA to become a commercial product, linked, as a module, in CAD modelling packages.
- Simulation can manipulated by upgrading GSOA.
- Applying a hybrid GSOA for solving other assembly issues.

REFRENCES

Akagi F. and Osaki H. (1980). The method of analysis of assembly of assembly work based on the fastener method. Bulletin of the JSME 23(184): 1670-75.

Alçada-Almeida, L., Coutinho-Rodrigues, J. and Current, J. (2009). A multi-objective modeling approach to locating incinerators, Socio-Economic Planning Sciences, 43, 111–120.

Aljarah I. and Ludwig S. A. (2013). A New Clustering Approach based on Glowworm Swarm Optimisation, IEEE Congress on Evolutionary Computation, pp. 2642-2649.

Aljarah I. and Ludwig S.A. (2016). A Scalable MapReduceenabled Glowworm Swarm Optimisation Approach for High Dimensional Multimodal Functions, International Journal of Swarm Intelligence Research (IJSIR), vol. 7, no. 1, pp. 32-54.

Arthur C. Sanderson, Luiz S. Homem de Mello and Hui Zhang (1990). Assembly Sequence Planning, AI Magazine, vol. 11, no. 1, pp. 1-20.

Atheer B. and Nordin M. J. (2017). Mutation and Memory mechanism for improving Glowworm Swarm Optimisation Algorithm, 7th Annual Computing and Communication Workshop and Conference (CCWC), IEEE.

Azzi, A., Battini, D., Faccio, M., and Persona, A. (2012). Mixed model assembly system with multiple secondary feeder lines: layout design and balancing procedure for ATO environment. International Journal of Production Research, 50(18), 5132-5151.

Barrera J. and Coello C.A.C. (2009). A review of particle swarm optimisation methods used for multimodal optimisation, in Innovations in swarm intelligence: Springer, pp. 9-37.

Ben-Arieh D. B. (1994). Computer-Aided Process Planning for Assembly l -Generation of Assembly Operation Sequence. Int. J. Prod. Res. 32(3): 643-656.

Ben-Arieh D. B. (1994). A Methodology for Analysis of Assembly Operation's difficulty. Int. J. Prod. Res 32(8): 1879-1895.

Biao Y., Chaoyong Z., Kunlei L. and Xinyu S. (2008). A crossover honey-bees mating optimisation algorithm for assembly sequence planning problem, 8th International Conference on Natural Computation, pp. 1-6.

Boothroyd G. and P. Dewhurst (1994). Product design for manufacture and assembly. New York,

Marcel Dekker.

Bourjault A. (1984). Contribution a une Approche Methodologique de l'Assemblage Automatise. Sciences Physiques, Universite de Franche Comte.

Brits R., Engelbrecht A.P. and Van Den-Bergh F. (2002). A niching particle swarm optimizer. Proc. of the 4th Asia-Pacific Conference on Simulated Evolution and Learning, Singapore, pp. 692–696.

Chang C.C., Tseng H.E. and Meng L.P. (2009). Artificial immune systems for assembly sequence planning exploration, Eng. App. of Artif. Intell, 22, pp. 1218-1232.

Cheng R. and Gen M. (1999). A Survey of Job-Shop Scheduling Based on Genetic Algorithms. EDA Conference '99, Vancouver.

Choi C. K. and Zha X. F. (1998). On the Automatic generation of product Assembly Sequences. Int. J. Prod. Res. 36: 617-633.

Choi Y.K., Lee D.M. and Cho Y.B. (2009). An approach to multi-criteria assembly sequence planning using genetic algorithms, Int. J. of Adv. Manuf. Tech. 42, pp. 180-188.

Davidor Y. (1991) Genetic Algorithms and Robotics: a Heuristic Strategy for Optimisation. Singapore, World Scientific.

De Fazio T. L. and D. E. Whitney (1987). Simplified Generation of All Mechanical Assembly Sequences. IEEE Journal of Robotics and Automation RA-3 no. 6: 640-658.

De Fazio T. L. and Rhee S. J. (1997). A Design-Specific Approach to Design-for- Assembly (DFA) for Complex Mechanical Assemblies. IEEE International Symposium on Assembly Task Planning, Marina del Rey, CA.

DeFloriani L. and G. Nagy (1991). Representation of Solid Objects by a Modular Boundary Model. Computer-Aided Mechanical Assembly Planning. Boston, Kluwer Academic: 41-80.

Delchambre A. (1992). Computer_aided Assembly Planning. London, Chapman & Hall.

Dini G. and Santochi M. (1992). Automated Sequencing and Subassembly Detection in Assembly Planning. Annals of the CIRP 41(1): 1-4.

Dini G. and Failli F. (1999). Generation of Optimised Assembly Sequences Using Genetic Algorithms. Annals of the CIRP 48(1): 17-20.

Donoso, Y. and Fabregat, R. (2007). Multi-Objective Optimisation In Computer Networks Using Metaheuristics. New York (US): Taylor & Francis Group.

Dorigo M. and Di Caro G. (1999) Ant colony optimization: a new meta-heuristic. IEEE, Congress on Evolutionary Computation-CEC99, Washington, DC, USA.

Ehrgott, M. (2005) Multicriteria Optimisation. 2nd ed., Springer, New York.

Falkenauer E. and Delchambre A. (1992). A Genetic Algorithm for Bin Packaging and Line Balancing. IEEE Intl. Conference on Robotics and Automation, Nice, France.

Fawaz H and Qian W. (2017). A Genetic Algorithm for Solving an Assembly Sequence Problem, Applied Mechanics and Materials, ISSN: 1662-7482, Vol. 872, pp 420-424, Trans Tech Publications, Switzerland.

Gairola A. (1986). Design Analysis for Automatic Assembly. International Journal for Production Research 24(4): 839-849.

Gao H., Xu W., Sun J. and Tang Y. (2010). Multilevel Thresholding for Image Componentation Through an Improved Quantum-Behaved Particle Swarm Algorithm, IEEE Transactions on Instrumentation and Measurement, vol.59, no.4, pp. 934,946.

Gen M. and Cheng R. (1997). Genetic Algorithms and Engineering Design, John Wiley & Sons, Inc., 605 Third Ave., New York, NY 10158, 411, pp. 89-95.

Gen M. and Cheng R. (1996). A Survey of Penalty Techniques in Genetic Algorithms, Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya, Japan, pp. 804-809.

Golabi S. I. (1996) Automatic generation of all geometrically feasible assembly sequences using solid modelling, University of South Australia.

Goldberg D. E. (1989). Genetic Algorithms in Search, Optimisation & Machine Learning, Addison Wesley Publishing Company, Inc.

Gorai A., and Ghosh A. (2011). Hue-preserving color image enhancement using particle swarm optimisation, Recent Advances in Intelligent Computational Systems (RAICS), IEEE, pp.563,568.

Gottipolu R. B. and K. Ghosh (1997). Representation and Selection of Assembly Sequences in Computer-Aided Assembly Process Planning. Int. J. Prod. Res. 35(12): 3447-3465.

Hart, W. E., Laird, C., Watson, J. P. and Woodruff, D. L. (2012). Pyomo–optimisation modeling in python, 67, Springer Science & Business Media.

Haupt R. L. and S. E. Haupt (1998). Feasible Genetic Algorithms. New York, John Wiley & Sons, Inc.

He L., Tong X. and Wang Q. (2013a). Glowworm Swarm Optimisation Algorithm Based on Hierarchical Multi-subgroups, Journal of Information & Computational Science 10: 4, pp.1245-1251, 2013.

He L., Tong X. and Wang Q. (2013b). Glowworm Swarm Optimisation Algorithm with improved movement pattern, Proc. of the IEEE 6th International Conference on Intelligent Networks and Intelligent Systems.

Holland J. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor, University of Michigan Press.

Homem de Mello L. S. and A. C. Sanderson (1989). A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. IEEE International Conference on Robotics and Automation.

Homem de Mello L. S. and A. C. Sanderson (1990). AND/OR Graph Representation of Assembly Plans. IEEE Transactions on Robotics and Automation 6, no. 2(April 1990): 188-199.

Homem de Mello L. S. and A. C. Sanderson (1991a). Representation of Mechanical Assembly Sequences." IEEE Transactions on Robotics and Automation 7, no. 2: 211-227.

Homem de Mello L. S. and A. C. Sanderson (1991b). Two Criteria for the Selection of Assembly Plans: Maximising the Flexibility of Sequencing the Assembly Tasks and Minimising the Assembly Time through Parallel Execution of Assembly Tasks. IEEE Transactions on Robotics and Automation 7, no. 5.

Homem de Mello L. S. and S. Lee, Eds. (1991c). Computer-Aided Mechanical Assembly Planning. Boston/Dordecht/London, Kluver Academic Publishers.

Homem de Mello, L. S. (1989). Task Sequence Planning for Robotic Assembly. Electrical and Computer Engineering. Pittsburg, Pennsylvania, Carnegie Mellon: 212.

Hong D.S. and Cho H.S. (1999). A genetic-algorithm based approach to the generation of robotic

assembly sequence, Control Engineering Practice, 7, pp. 151-159.

Hongbo S. and shuxia L. (2008). The Comparison Between Genetic Simulated Annealing Algorithm and Ant Colony Optimisation Algorithm for ASP, Proceedings of the IEEE International Conference on Automation and Logistics, pp. 1-6.

Hongbo S., Shuxia L., Degang G. and Peng L. (2006). Genetic Simulated Annealing Algorithm-Based Assembly Sequence Planning, International Technology and Innovation Conference, pp. 1573-1579.

Huang K. and Zhou Y. Q. (2012). A Novel Chaos Glowworm Swarm Optimisation Algorithm for Optimisation Functions, Proc. of Springer, Bio-Inspired Computing and Applications, Volume 6840 of the series Lecture Notes in Computer Science pp. 426-434.

Huang Y. F. and C. S. G. Lee (1988). Precedence Knowledge in Feature Mating Operation Assembly Planning. West Lafayette, Indiana, Engineering Research Center for Intelligent Manufacturing Systems, School of Engineering, Purdue University.

Huang Y. F. and C. S. G. Lee (1991). A Framework of Knowledge-based Assembly Planning. IEEE International Conference on Robotics and Automation, Sacramento, California.

Jayakumar D. N. and Venkatesh P. (2014). Glowworm swarm optimisation algorithm with topsis for solving multiple objective environmental economic dispatch problem, Applied Soft Computing, vol. 23, pp. 375-386.

Jayakumar, D. N. and Venkatesh, P. (2014). Glowworm swarm optimisation algorithm with topsis for solving multiple objective environmental economic dispatch problem, Applied Soft Computing, vol. 23, pp. 375-386.

Jones R. E. and R. H. Wilson (1996). A Survey of Constraints in Automated Assembly Planning. IEEE International Conference on Robotics and Automation.

Jones R. E., Wilson R.H., and Calton T. L. (1997). Constraint-Based Interactive Assembly planning. IEEE International Conference on Robotics and Automation.

Jones R. E., Wilson R. H., and Calton T. L. (1998). On constraints in assembly planning. IEEE Transactions on Robotics and Automation, 16(6): 849–863.

Karr C. L. and Freeman L. M. (1999). Industrial Applications of Genetic Algorithms. International

Series on Computational Intelligence. New York.

Kavraki L. and Latombe J. C. (1993). On the Complexity of Assembly Partitioning. Information Processing Letters 48(5): 229-235.

Kazerooni M. (1997). An Integrated Methodology for Cellular Manufacturing System Design. IT, Engineering and Environment. Adelaide, South Australia.

Kennedy J. and Eberhart R. (1995). Particle Swarm Optimisation. Proc. IEEE International Conference on Neural Networks, pp. IV: 1942–1948.

Kowalczyk R. (1997). Constrained Genetic Operators preserving Feasibility of Solutions in Genetic Algorithms. IEE Genetic Algorithms in Engineering Systems: Innovations and Applications.

Krause J. and Ruxton G. D. (2002). Living in Groups. Oxford: Oxford University Press.

Krcmar M. and A. P. Dhawan (1994). Application of Genetic Algorithms in Graph Matching. IEEE World Congress on computational Intelligence.

Krishnanand, K. N. and Ghose, D. (2005). Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. In Proceedings of IEEE swarm intelligence symposium, pp. 84-91.

Krishnanand K. N. and Ghose D. (2006 a). Glowworm swarm based optimisation algorithm for multimodal functions with collective robotics applications, Multiagent and Grid Systems, vol. 2, no. 3, pp. 209-222.

Krishnanand K. N. and Ghose D. (2006 b). Theoretical foundations for multiple rendezvous of glowworm-inspired mobile agents with variable local decision domains. In Proceedings of American control conference. Piscataway: IEEE Press, pp. 3588-3593.

Krishnanand K. N. and Ghose D. (2008). Theoretical foundations for rendezvous of glowworm-inspired agent swarms at multiple locations, Robotics and Autonomous Systems, vol. 56, no. 7, pp.549-569.

Krishnanand K. N. and Ghose D. (2009a). Glowworm swarm optimisation: a new method for optimizing multimodal functions, Int. J. Computational Intellingence Studies, vol.1, no.1, pp. 93-119.

Krishnanand K. N. and Ghose D. (2009b). Glowworm swarm optimisation for simultaneous

capture of multiple local optima of multimodal functions, Swarm Intelligenece, vol. 3, no. 2, pp. 87–124.

Lazzerini B. and G. Dini (1999). Assembly Planning Based on Genetic Algorithms. 18th International Conference of the North American Fuzzy information.

Lee S. (1992a). Backward Assembly Planning with Assembly Cost Analysis. IEEE International Conference on Robotics and Automation, Nice, France.

Lee S. (1992b). Backward Assembly Planning. Artificial intelligence applications in manufacturing. D. S. N. S. H. K. A. (Fazel) Famili. Menlo Park, CA, AAAI Press/MIT Press: 61-101.

Lee S. (1994). Subassembly Identification and Evaluation for Assembly Planning. IEEE Transactions on Systems, Man and Cybernetics 24(3): 493-503.

Lee S. and Y. G. Shin (1991). Assembly Coplanner: Cooperative Assembly Planner Based on Subassembly Extraction. Computer-Aided Mechanical Assembly Planning. S.

Lee S. and Kim G. J. (1993). Combining Assembly Planning with Redesign: An Approach for More Effective DFA. IEEE International Conference on Robotics and Automation.

Lei, D. and Guo, X. (2015). A parallel neighbourhood search for order acceptance and scheduling in flow shop environment. International Journal of Production Economics, 165, 12-18.

Liao W. H., Kao Y. and Li Y. S. (2011). A sensor deployment approach using glowworm swarm optimisation algorithm in wireless sensor networks, Expert Systems with Applications, vol. 38, no. 10, pp. 12180-12188.

Marian R., Luong L.H.S. and Abhary K. (2006). A genetic algorithm for the optimisation of assembly sequences, Computers & Industrial Engineering, 50, pp. 503–527.

Marian R., Abhary K. and Luong L. (2000a). On the Definition of Fitness Function for the Optimisation of Assembly Sequences using GA. ICME - The Eight International Conference on Manufacturing Engineering, Sydney.

Marian R., Abhary K. and Luong L. (2001). Definition and Representation of Precedence Relations in Assembly Optimisation. 5-th Intl &9-th Annual Mechanical Engineering Conference, Rasht - Iran.

Marian R., Luong L. and Abhary K. (1999). Applications of Genetic Algorithms in Design for Assembly. EDA Conference, Vancouver.

Marian R., Luong L. and Abhary K. (1999). Chromosome Generation for Assembly Planning Using a Guided Search. The Third Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems, Canberra, Australia.

Marian R., Luong L. and Abhary K. (1999). Optimisation of Assembly Sequences Using Genetic Algorithms. 10-th International DAAAM Symposium, Viena, Austria.

Marian R., Luong L. and Abhary K. (2000b). A new crossover technique for Assembly Sequence Planning Using GA. Computer Integrated Manufacturing CIM 2000, Singapore.

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

Marinaki M. and Marinakis Y. (2016). A Glowworm Swarm Optimisation algorithm for the Vehicle Routing Problem with Stochastic Demands, Proc. of ELSEVIER Expert Systems With Applications 46, 145–163.

Messac, A. (2015). Optimisation in Practice with MATLAB: For Engineering Students and Professionals, 1st Edition, Cambridge University Press.

Michalewicz Z. (1992). Genetic algorithms + Data Structures = Evolution Programs. Berlin, Springer-Verlag.

Michalewicz Z. (1994). Genetic algorithms + Data Structures = Evolution Programs. Berlin, Springer-Verlag.

Michalewicz Z. (1996). Genetic algorithms + Data Structures = Evolution Programs. Berlin, Springer-Verlag.

Milner J. M. and Graves S. C. (1994). Using Simulated Annealing to Select Least-Cost Assembly Sequences. IEEE International Conference on Robotics and Automation.

Min L., Xiaohu Y., Yongxing C., Qian P. and Hailong Z. (2013). A Method for the Oil Chromatographic On-line Data Reconciliation Based on GSO and SVM, TENCON Spring Conference, pp. 322-326.

Mo X., Li X. and Zhang Q. (2016). The variation step adaptive Glowworm swarm optimisation algorithm in optimum log interpretation for reservoir with complicated lithology. In Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNCFSKD), 12th International Conference, IEEE, pp. 1044-1050.

Molloy, O., S. Tilley (1998). Design for manufacturing and assembly - concepts, arhitectures and implementation. London, Chapman & Hall.

Nilsson N., J. (1980). Principles of Artificial Intelligence. Palo Alto, CA, Tioga Publishing Co.

Nof S. Y., Wilbert W. and Warnecke H. (1997). Industrial Assembly, Chapman & Hall.

Ou L.M. and Xu X. (2013). Relationship matrix based automatic assembly sequence generation from a CAD model, Comp.-Aided Design, 45, pp. 1053-1067.

Pal S. K. and P. P. Wang, Eds. (1996). Genetic algorithms for pattern recognition. Boca Raton, FL, CRC Press.

Pan G. and Xu Y. (2016). Chaotic glowworm swarm optimisation algorithm based on Gauss mutation. In Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2016 12th International Conference, IEEE, pp. 205-210).

Pandu, R. G. (2009). Multi-Objective Optimisation: Techniques And Applications In Chemical Engineering (Advances in Process Systems Engineering). Singapore: World Scientific Publishing.

Pareto, V. (1906). Manual of political economy. New York: A.M. Kelley.

Park J. H. and H. Asada (1994). Sequence Optimisation for High Speed Robotic Assembly Using Simulated Annealing. EEE International Conference on Robotics and Automation.

Partridge B. L. (1982). The structure and function of fish schools. Science American, 245, pp. 90-99.

Partridge B. L. and Pitcher T. J. (1980). The sensory basis of fish schools: relative role of lateral line and vision. Journal of Comparative Physiology, 135, pp. 315-325.

Pengzhen D., Zhenmin T. and Yan S. (2014). A Quantum Glowworm Swarm Optimisation Algorithm based on Chaotic Sequence, International Journal of Control and Automation Vol.7, No.9, pp.165-178.

Rashid M., Hutabarat W. and Tiwari A. (2011). A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches, Int. J. of Adv. Manuf. Tech.59, pp. 335-349.

Romney B. and Goddard C. (1995). An Efficient System for Geometric Assembly Sequence Generation and Evaluation. ASME International Computers in Engineering Conference, Boston, Massachussets.

Ruzica, S., and Wiecek, M. M. (2003). A Survey of Approximation Methods In Multi-objective Programming. Research Report, Fachbereich Mathematik, Universit¨at, Kaiserslautern, Gottlieb-Daimler-Straße, 676(63), Kaiserslautern, Germany.

Santochi M. and G. Dini (1992). Computer Aided Planning of Assembly Operations: The Selection of Assembly Sequences. Robotics and Computer Integrated Manufacturing 9(6): 439-446.

Sebaaly M. F. and Fujimoto H. (1996a). A Genetic Planner for Assembly Automation. 5th International Conference on Concurrent Engineering Research and Applications, Tokyo, Japan.

Sebaaly M. F. and Fujimoto H. (1996b). A genetic planner for assembly automation. In Proceedings of international conference on evolutionary computation (pp. 401–406). Nagoya: IEEE.

Sebaaly M. F. and Fujimoto H. (1996c). Linear and Non-Linear Assembly Planning: Fuzzy Graph Representation and GA Search.

Senin N., Groppetti R. and Wallace D.R. (2000). Concurrent assembly planning with genetic algorithms, Robotics and Computer Integrated Manufacturing, 16, pp. 65-72.

Shin C. K. and Hong D. S. (1995). Disassemblability Analysis for Generating Robotic Assembly Sequences. IEEE International Conference on Robotics and Automation.

Shpitalni M. and Elber G. (1989). Automatic assembly of three-dimensional structures via connectivity graphs. annals of the CIRP 38(11): 21-28.

Tang Z., Zhou Y., and Chen X. (2013). An improved glowworm swarm optimisation algorithm based on parallel crossover mutation, in International Conference on Intelligent Computing, pp. 198-206: Springer.

Tseng H.E. (2006). Guided genetic algorithms for solving a larger constraint assembly problem, International Journal of Production Research, Vol. 44, No. 3, pp. 601–625.

Tseng Y.J., Kao H.T. and Huang F.Y. (2010a). Integrated assembly and disassembly sequence planning using a GA approach, International Journal of Production Research, Vol. 48, No. 20, pp. 5991–6013.

Tseng Y.J., Chen J.Y., Huang F.Y (2010b). A multi-plant assembly sequence planning model with integrated assembly sequence planning and plant assignment using GA, Int J Adv Manuf Technol, 48, pp. 333–345.

Thiruvenkadam K. and Perumal N. (2017). A Review on Glowworm Swarm Optimization, International Journal of Information Technology (IJIT) – Vol. 3 Issue 2, pp. 49-56.

Thomas J. P. and Nissanke N. (1996). A Hyerarchical Petri Net Framework for the Representation and Analysis of Assembly. IEEE Transactions on Robotics and Automation 12, no. 2: 268-279.

Tichem M. and Storm T. (1999). How to Achieve a Breakthrough in Industrialisation of Flexible Assembly Automation. 9-th International Flexible Automation and Intelligent Manufacturing (FAIM) Conference, Tilburn, The Netherlands.

Van Den-Bergh F. (2002). An analysis of Particle Swarm Optimizers. PhD Thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa.

Van Den-Bergh F. and Engelbrecht A. P. (2002). A new locally convergenet particle swarm optimizer. Proc. of IEEE International Conference on Systems, Man, and Cybernetics, pp. 96–101.

Wang, R., & Fang, H. (2001). Aggregate production planning with multiple objectives in a fuzzy environment. European Journal of Operational Research, 133, 521–536.

Whitley D. (2014). An executable model of a simple genetic algorithm. Foundations of genetic algorithms, 2 (15-19), pp. 45-62.

Wilson R. J. and J. J. Watkins (1990). Graphs - An Introductory approach. New York, John Wiley and Sons.

Wolter J. and P. Chandrasekaran (1991). A Concept for a Constraint-Based Representation of Functional and Geometric Design Knowledge. ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, AUstin, Texas, USA.

Wolter J. D. (1989). On the automatic generation of assembly plans. IEEE International Conference

on Robotics and Assembly Planning, Scottsdale, Arizona.

Wolter J. D. (1990). A Constraint Based Approach to Planning with Subassemblies. IEEE International Conference on Systems Engineering.

Wolter J. D. (1990). Representing Subassembly Trees by Deepest Common Ancestor Relations, Texas A&M University.

Wolter J. D. (1991). A Combinatorial Analysis of Enumerative Data Structures for Assembly Planning. IEEE International Conference on Robotics and Automation, Sacramento, California, USA.

Wolter J. D. (1988). On the Automatic Generation of Plans for Mechanical Assembly. Computer, Information and Control Engineering. Ann Arbor, Michigan: 127.

Wu B., Qian C., Ni W. and Fan S. (2012). The improvement of glowworm swarm optimisation for continuous optimisation problems, Expert Systems with Applications, vol. 39, no. 7, pp. 6335-6342.

Xing Y. and Wang Y. (2012). Assembly sequence planning based on a crossover particle swarm optimisation and genetic algorithm, Int. J. of Prod. Res, 24, pp. 7303-7312.

Yandra (1999). Optimisation of AGV-Based Flexible Manufacturing Systems Design Using Genetic Algorithms and Expert Systems. IT, Engineering and Environment. Adelaide, South Australia.

Yang J. and X. Li (2013). Map reduce based method for big data semantic clustering, in Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC'13. Washington, DC, USA: IEEE Computer Society, pp. 2814-2819.

Yang Y., Zhou Y. and Gong Q. (2010). Crossover Artificial Glowworm Swarm Optimisation Algorithm for Solving System of Nonlinear Equations, Journal of Computational Information Systems 6:10, pp. 3431-3438.

Yao X. (1991). Simmulated Annealing with Extended Neighbourhood. International Journal of Computer Mathematics 40: 169-189.

Yasin A., Puteh N., Daud R., Omar M. and Abdullah S.L.S. (2010). Product assembly sequence optimisation based on genetic algorithm, Int. J. on Comp. Sci. and Eng. 29, pp. 3065-3070.

Yu J. and Wang C. (2013). A max–min ant colony system for assembly sequence planning, Int J Adv Manuf Technol, 67, pp. 2819–2835.

Yu Z. and Jine Z. (2012). Glowworm Swarm Optimisation and Heuristic Algorithm for Rectangle Packing Problem, International Conference on Information Science and Technology, IEEE, pp. 136-140.

Yu Z. and Yang X. (2013). Full Glowworm Swarm Optimisation Algorithm for Whole-Set Orders Scheduling in Single Machine, The ScientificWorld Journal, Volume, Article ID 652061, 6 pages.

Zeng C., Gu T., Chang L. and F. Li (2013). A Novel Multi-agent Evolutionary Algorithm for Assembly Sequence Planning, Journal of Software, Vol. 8, No. 6, pp. 1518-1525.

Zhan Z. H., Zhang J., Li Y. and Chung H. S. (2009). Adaptive particle swarm optimisation, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 39, no. 6, pp. 1362-1381

Zhang Y., MA X. and MIAO Y. (2011). Localization of Multiple Odor Sources Using Modified Glowworm Swarm Optimisation with Collective Robots, Proceedings of the 30th Chinese Control, pp. 1899-1904.

Zhang Y., Ogura H., Ma X., Kuroiwa J. and Odaaka T. (2014). A Genetic Algorithm Using Infeasible Solutions for Constrained Optimisation Problems, The Open Cybernetics & Systemics Journal, 8, 904-912.

Zhou W., Yan J., Li Y., Xia C. and Zheng J. (2013). Imperialist competitive algorithm for assembly sequence planning, Int. J. of Adv. Manuf Tech. 67, pp. 2207–2216.

Zhou Y., Zhe O., Jiakun L. and Gaoli S. (2012). A Novel K-means Image Clustering Algorithm Based on Glowworm Swarm Optimisation, przegląd elektrotechniczny, pp. 266-270.

Zhou, C. C., Yin, G. F. and Hu, X. B. (2009). Multi-objective optimisation of material selection for sustainable products: artificial neural networks and genetic algorithm approach. Materials & Design, 30(4), 1209-1215.

APPENDIX 1

Types of assembly plans:

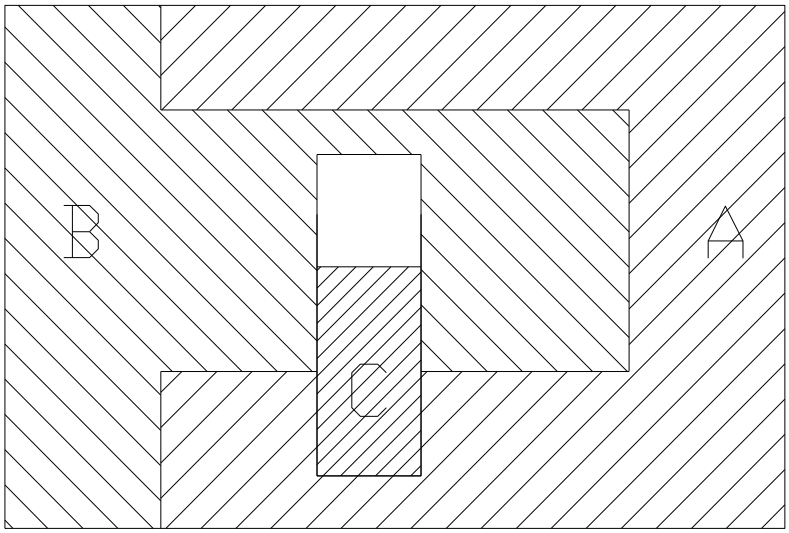


Figure A1.1. A product that can be assembled with a C-S-L-NM assembly

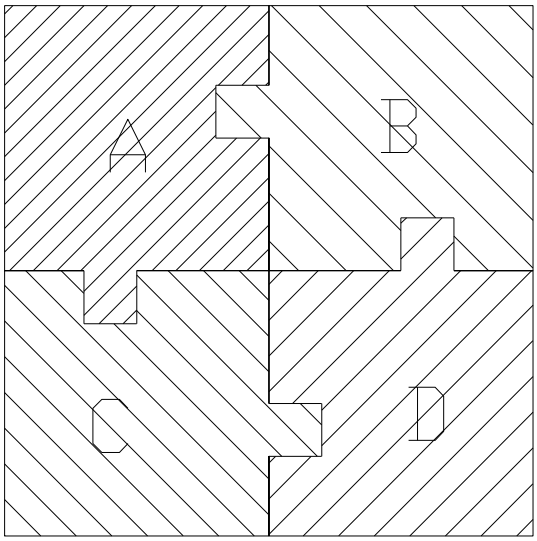


Figure A1.2. A product that can be assembled with a C-S-NL-M assembly

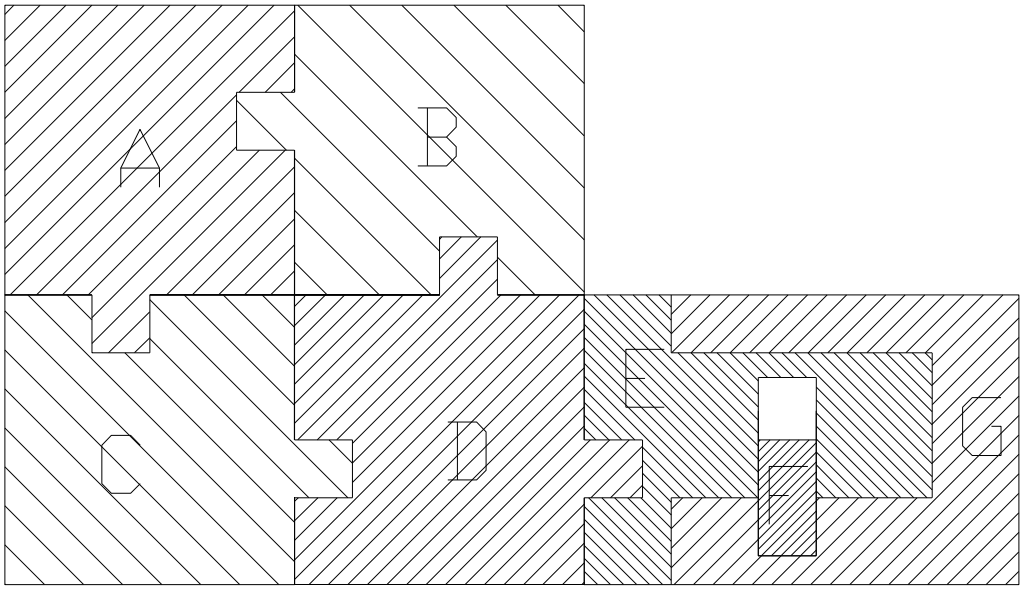


Figure A1.3. A product that can be assembled with a C-S-NL-NM assembly

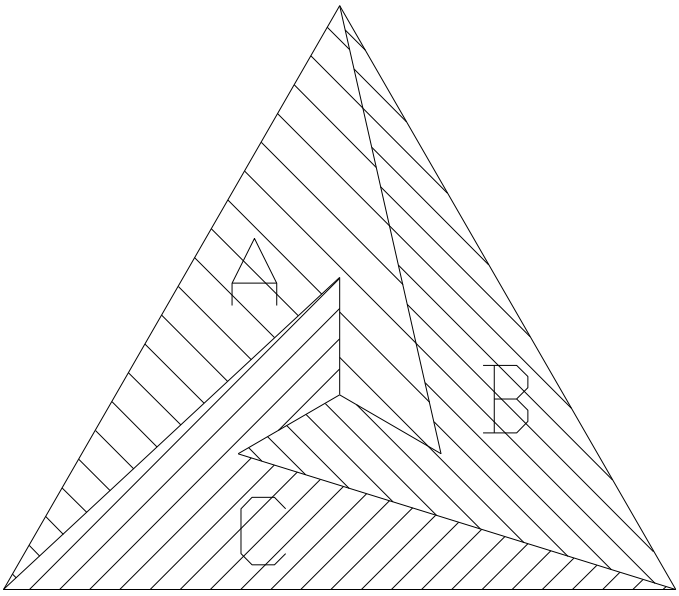


Figure A1.4. A product that can be assembled with a C-NS-L-M assembly

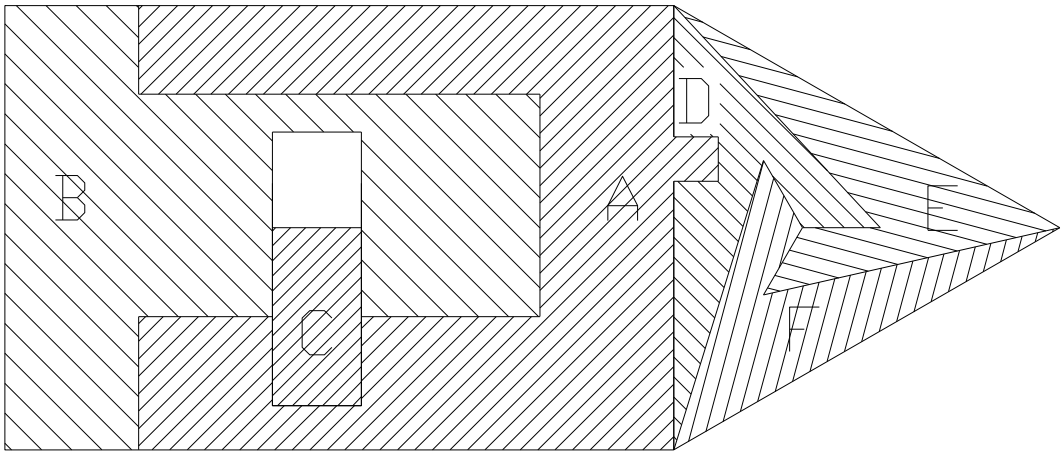


Figure A1.5. A product that can be assembled with a C-NS-L-NM assembly

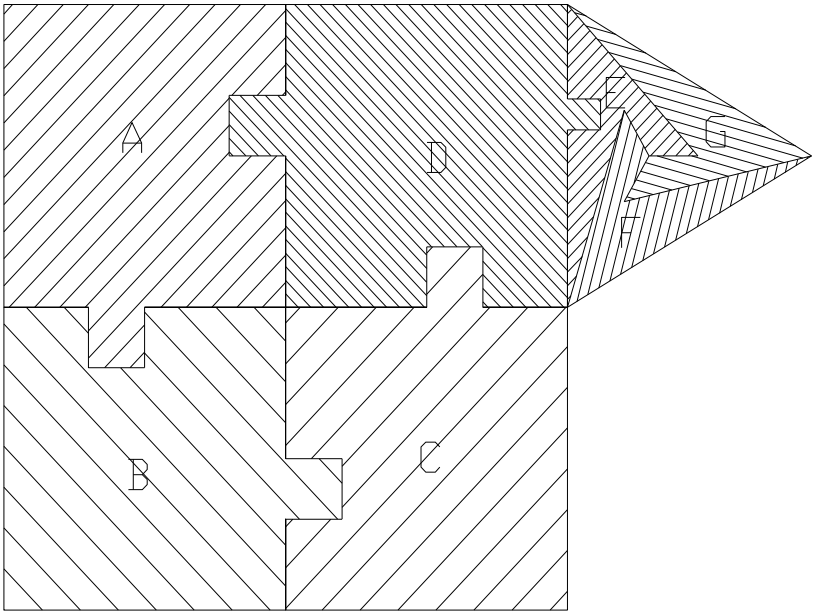


Figure A1.6. A product that can be assembled with a C-NS-NL-M assembly

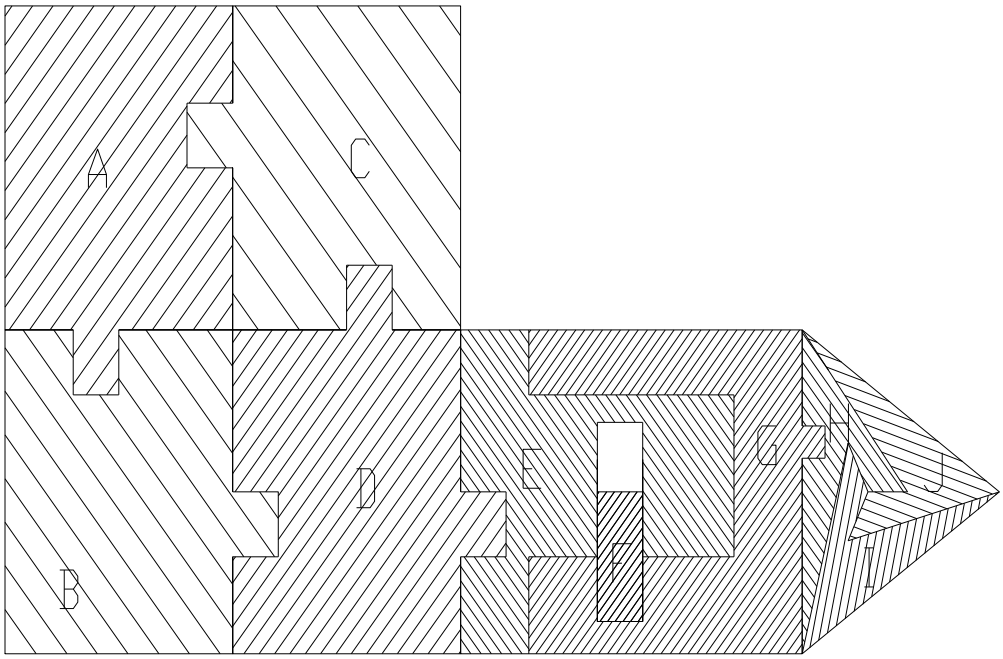


Figure A1.7. A product that can be assembled with a C-NS-NL-NM assembly

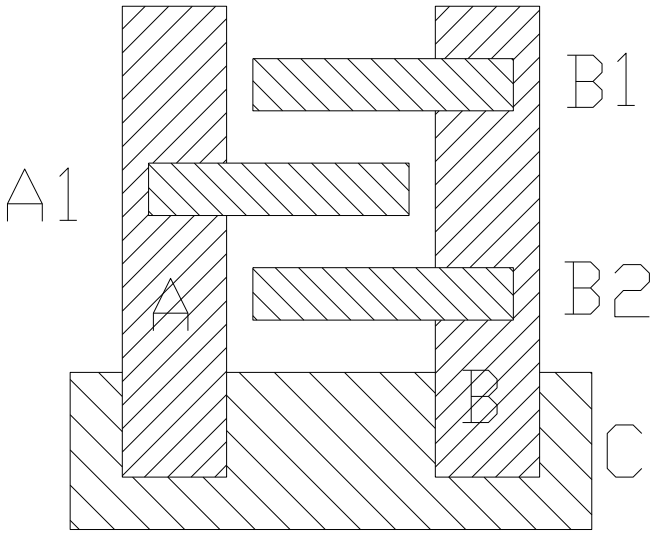


Figure A1.8. A product that can be assembled with a NC-S-L-M assembly

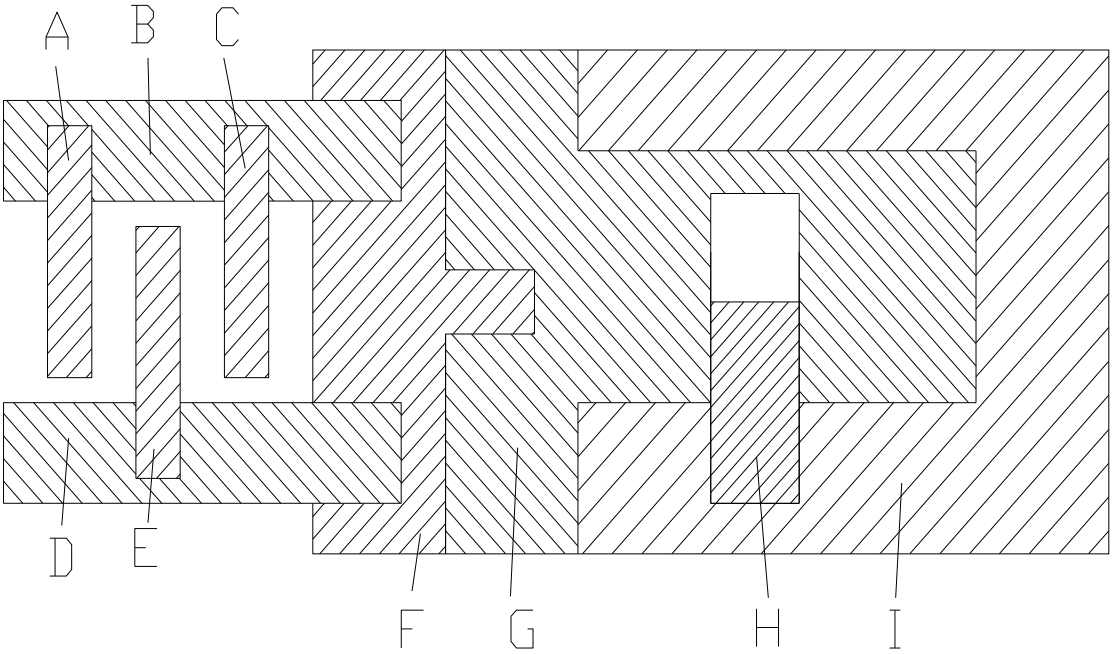


Figure A1.9. A product that can be assembled with a NC-S-L-NM assembly

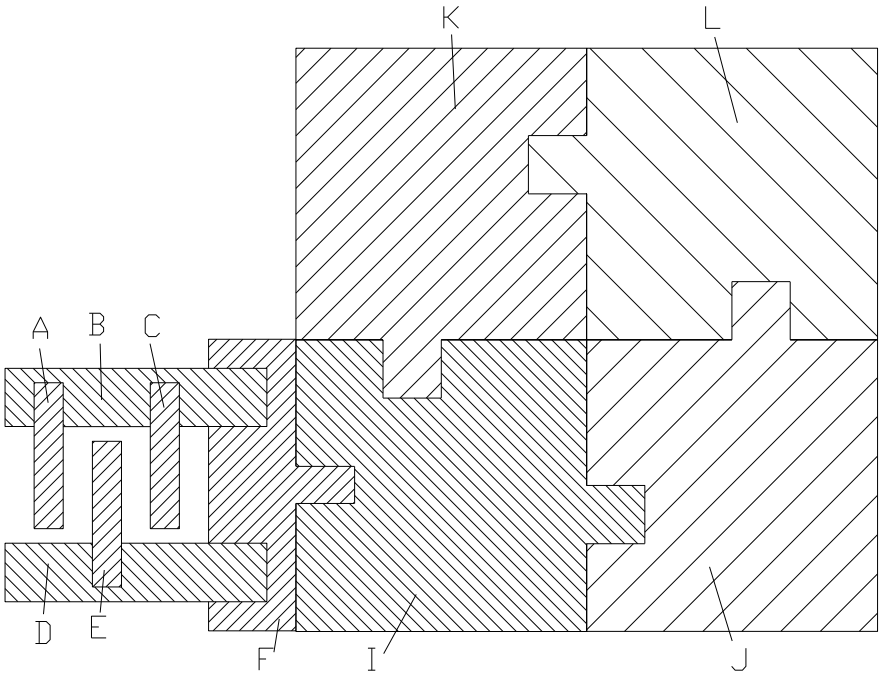


Figure A1.10. A product that can be assembled with a NC-S-NL-M assembly

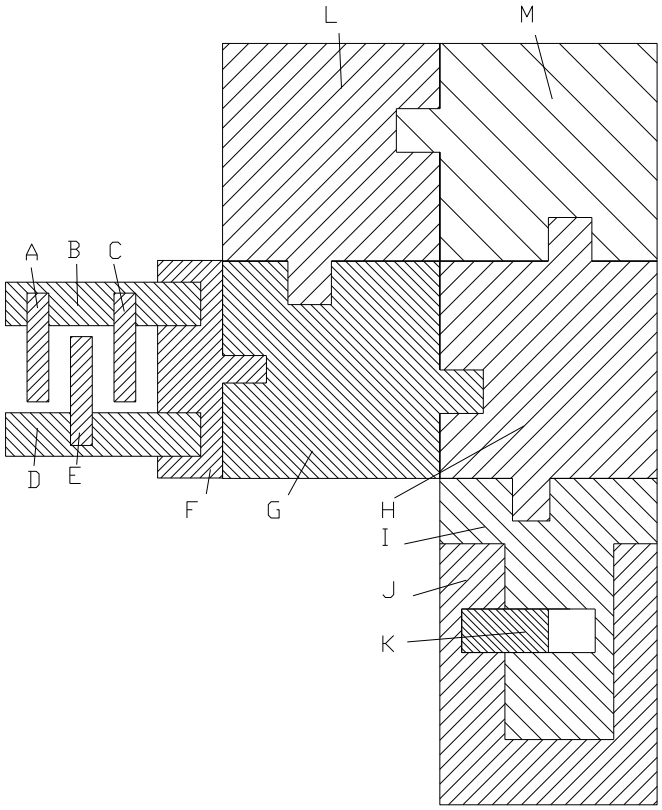


Figure A1.11. A product that can be assembled with a NC-S-NL-NM assembly

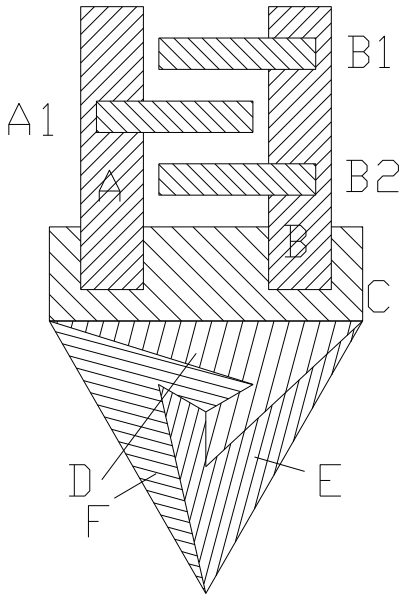


Figure A1.12. A product that can be assembled with a NC-NS-L-M assembly

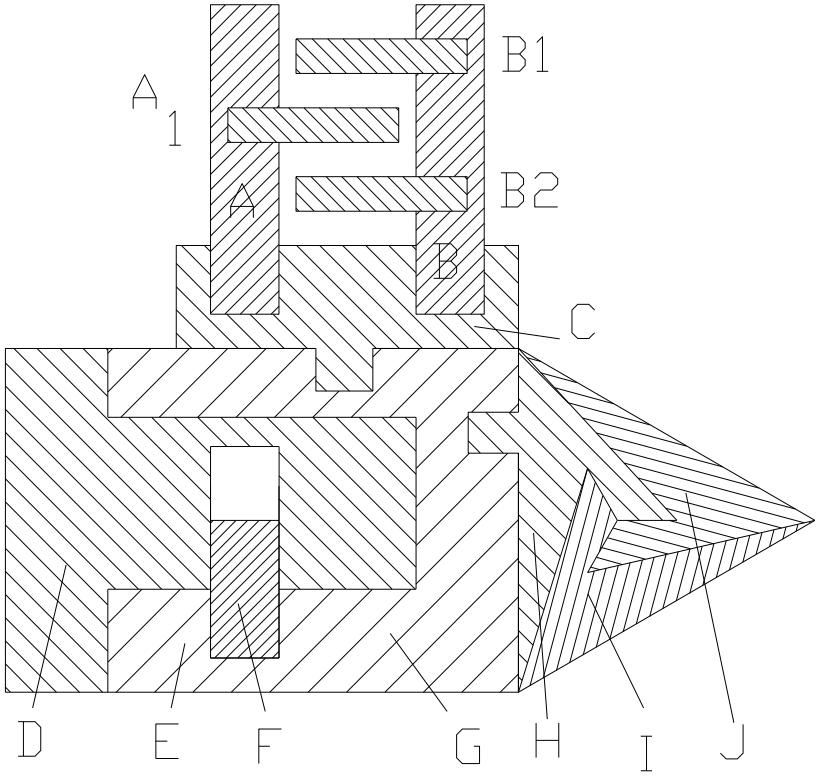


Figure A1.13. A product that can be assembled with a NC-NS-L-NM assembly

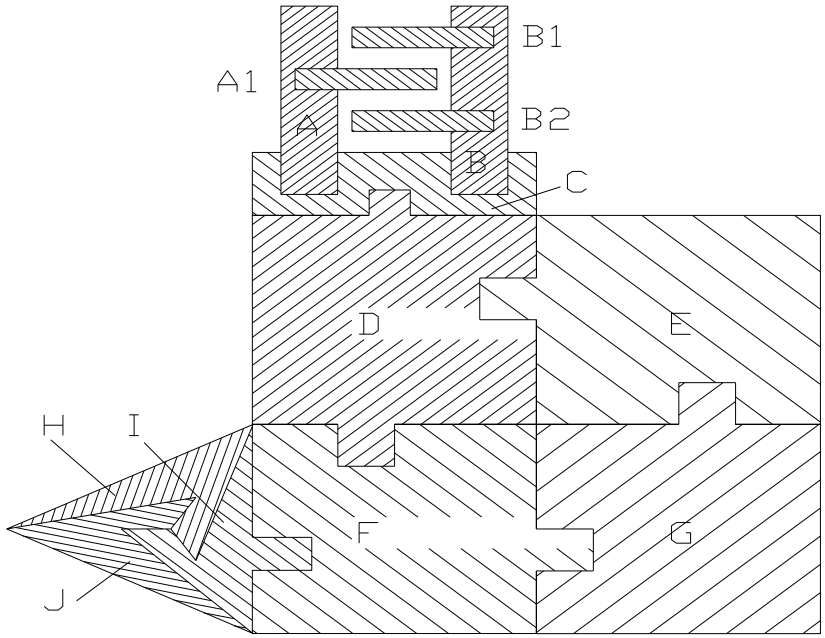


Figure A1.14. A product that can be assembled with a NC-NS-NL-M assembly

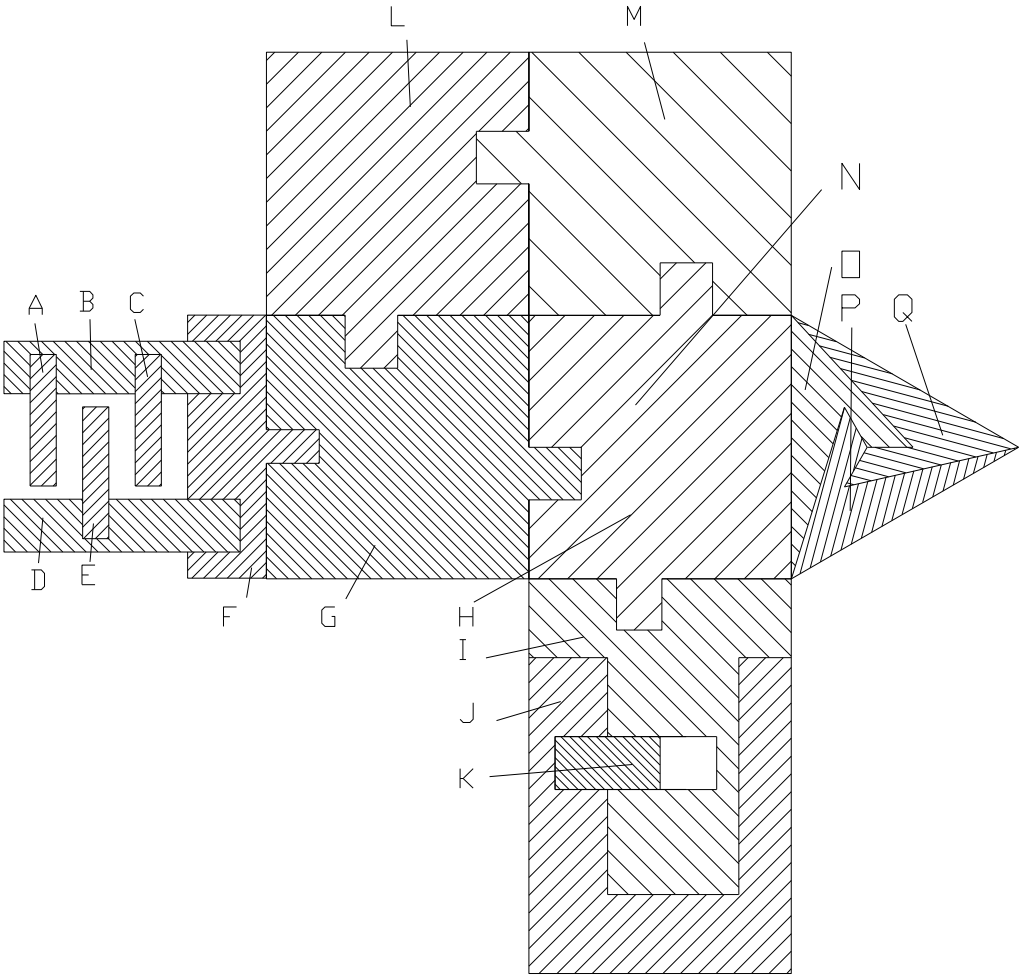
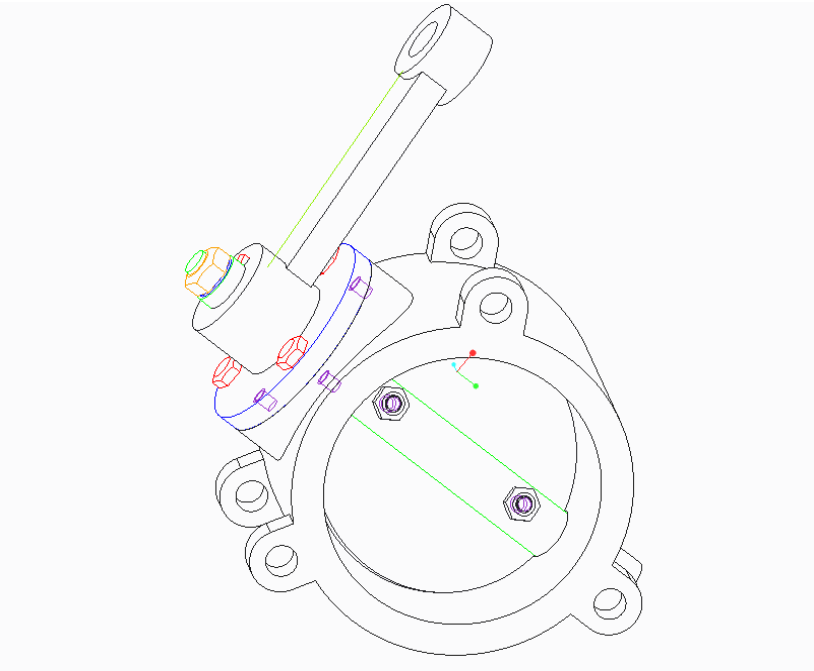


Figure A1.15. A product that can be assembled with a NC-NS-NL-NM assembly

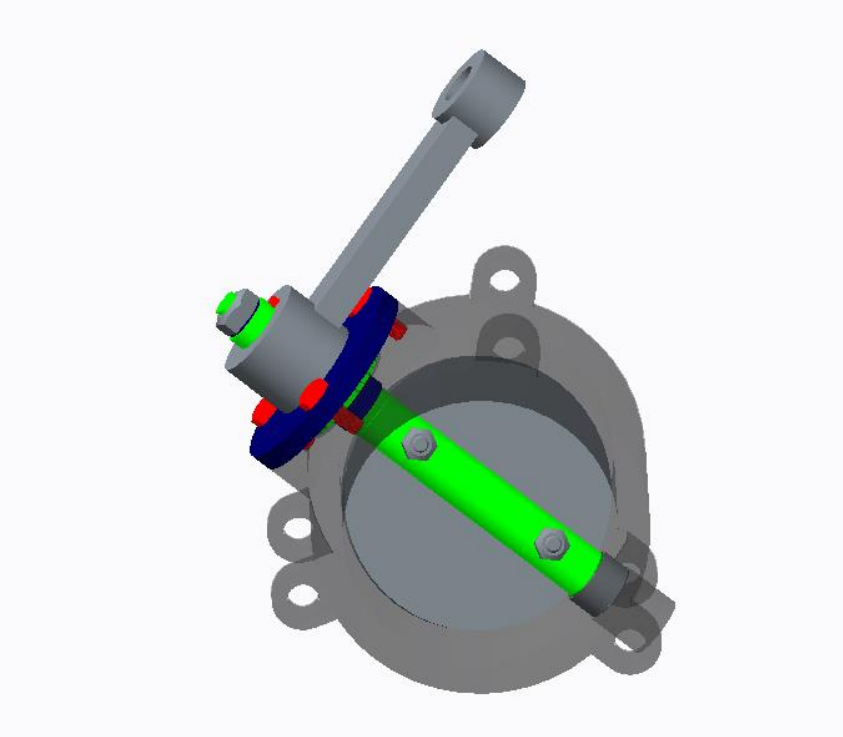
APPENDIX 2

CASE STUDY 1:

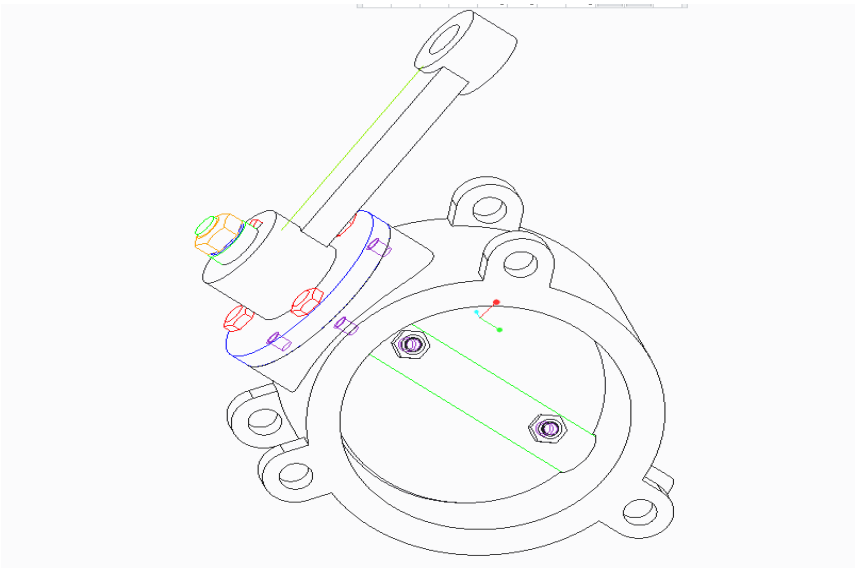
Another drawing (3D and 2D) of assembling the car engine pump valve by using Creo:



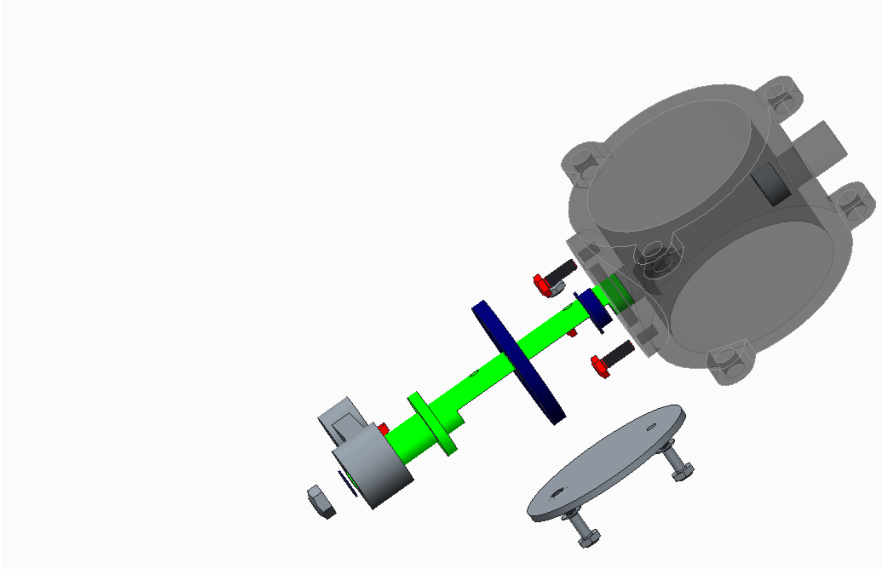
(a)



(b)



(c)



(d)

Figure A2.1. (a, b, c and d) Assembling the car engine pump valve components (3D)

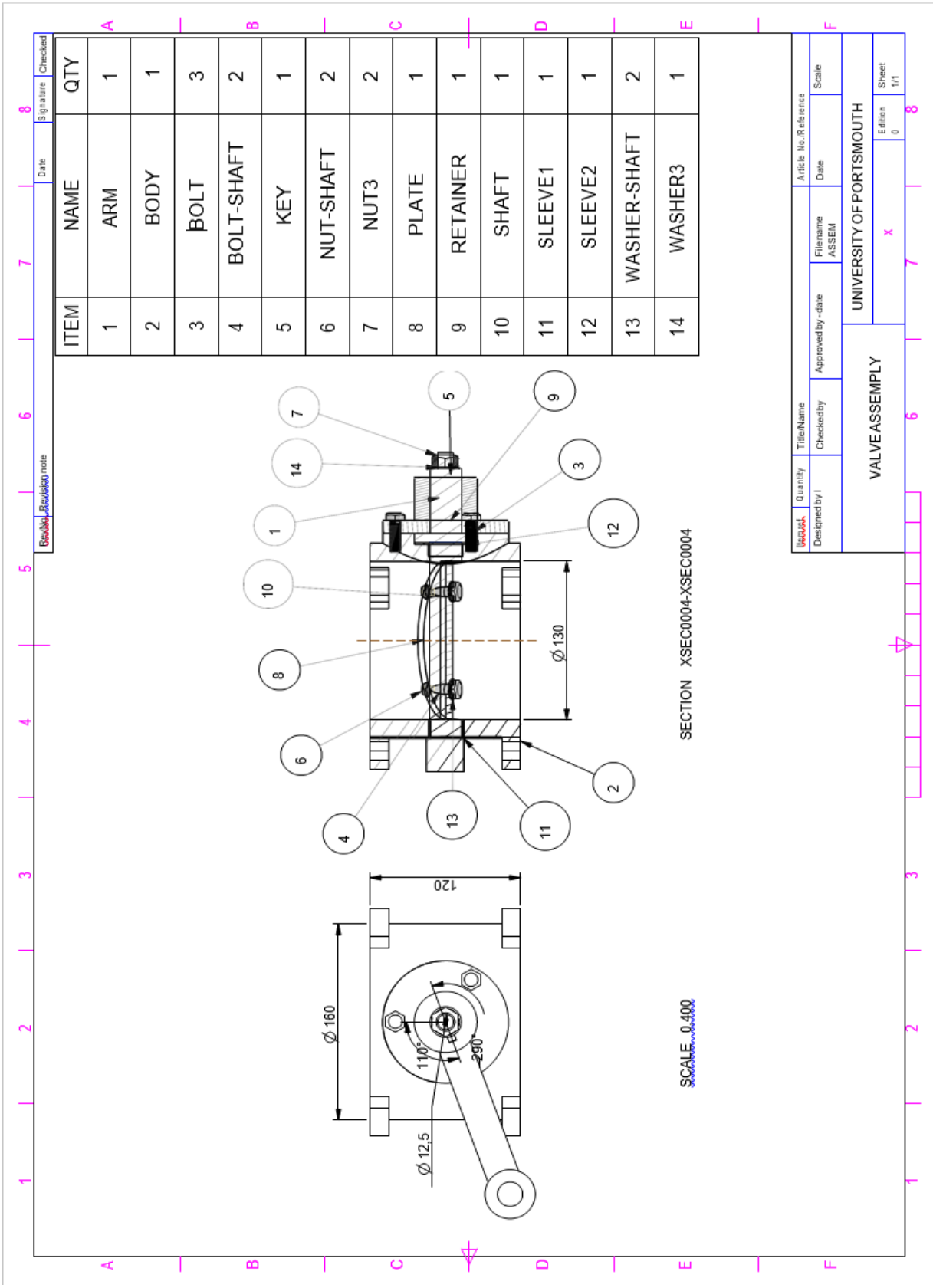


Figure A2.2. Assembling the car engine pump valve components (2D)

Table A.2.1. Selections from feasible assembly sequence of the car engine pump valve obtained by running number of generations, as detailed in Chapter 6

Sleeve1 (11)	Plate (8)	Nut-Shaft (6)	Bolt-Shaft(4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Washer-Shaft (13)	Nut-Shaft (6)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Shaft (10)	Washer-Shaft (13)	Nut-Shaft (6)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Shaft (10)	Nut-Shaft (6)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Nut-Shaft (6)	Bolt-Shaft(4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Nut-Shaft (6)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)

Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Washer-Shaft (13)	Nut-Shaft (6)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Shaft (10)	Washer-Shaft (13)	Nut-Shaft (6)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Shaft (10)	Nut-Shaft (6)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Sleeve2 (12)	Retainer (9)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Nut-Shaft (6)	Bolt-Shaft(4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Washer-Shaft (13)	Nut-Shaft (6)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Shaft (10)	Washer-Shaft (13)	Nut-Shaft (6)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)

Sleeve1 (11)	Plate (8)	Shaft (10)	Nut-Shaft (6)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Nut-Shaft (6)	Bolt-Shaft(4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Washer-Shaft (13)	Nut-Shaft (6)	Shaft (10)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Shaft (10)	Washer-Shaft (13)	Nut-Shaft (6)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Shaft (10)	Nut-Shaft (6)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Nut-Shaft (6)	Bolt-Shaft(4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)

Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Washer-Shaft (13)	Nut-Shaft (6)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Shaft (10)	Washer-Shaft (13)	Nut-Shaft (6)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Shaft (10)	Nut-Shaft (6)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Nut-Shaft (6)	Bolt-Shaft(4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Washer-Shaft (13)	Nut-Shaft (6)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Shaft (10)	Washer-Shaft (13)	Nut-Shaft (6)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)

Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Shaft (10)	Nut-Shaft (6)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Nut-Shaft (6)	Bolt-Shaft(4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Nut-Shaft (6)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Washer-Shaft (13)	Nut-Shaft (6)	Shaft (10)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Shaft (10)	Washer-Shaft (13)	Nut-Shaft (6)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Shaft (10)	Nut-Shaft (6)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Sleeve2 (12)	Bolt(3)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)

Plate (8)	Sleeve1 (11)	Nut-Shaft (6)	Bolt-Shaft(4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Sleeve1 (11)	Plate (8)	Bolt-Shaft(4)	Nut-Shaft (6)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Washer-Shaft (13)	Nut-Shaft (6)	Shaft (10)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Shaft (10)	Washer-Shaft (13)	Nut-Shaft (6)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Shaft (10)	Nut-Shaft (6)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Retainer (9)	Sleeve2 (12)	Bolt(3)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Nut-Shaft (6)	Bolt-Shaft(4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Nut-Shaft (6)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Washer-Shaft (13)	Nut-Shaft (6)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)

Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Shaft (10)	Washer-Shaft (13)	Nut-Shaft (6)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Shaft (10)	Nut-Shaft (6)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Bolt(3)	Retainer (9)	Sleeve2 (12)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Nut-Shaft (6)	Bolt-Shaft(4)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Nut-Shaft (6)	Washer-Shaft (13)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Washer-Shaft (13)	Nut-Shaft (6)	Shaft (10)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Shaft (10)	Washer-Shaft (13)	Nut-Shaft (6)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Bolt-Shaft(4)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)
Plate (8)	Sleeve1 (11)	Shaft (10)	Nut-Shaft (6)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)

Plate (8)	Sleeve1 (11)	Nut-Shaft (6)	Shaft (10)	Washer-Shaft (13)	Bolt-Shaft(4)	Body (2)	Arm(1)	Bolt(3)	Sleeve2 (12)	Retainer (9)	Key (5)	Washer3(14)	Nut3(7)

Table A.2.2. Selections from the results obtained by running the GA for the car engine pump valve, as detailed in Chapter 7

		Evaluation	Fitness Function	Probability	Cumulative Probability	Random generation
Generation 1						
Chromosome1	11,8,6,4,13,10,2,1,12,3,9,5,14,7	593	0.001684	0.100630	0.100630	0.897400
Chromosome2	11,8,6,4,13,10,2,1,12,3,9,5,14,7	593	0.001684	0.100630	0.201261	0.332350
Chromosome3	11,8,6,4,13,10,2,1,12,3,9,5,14,7	593	0.001684	0.100630	0.301891	0.212770
Chromosome4	8,11,10,6,13,4,2,1,12,9,3,5,14,7	567	0.001761	0.105237	0.407128	0.473301
Chromosome5	8,11,10,6,13,4,2,1,12,9,3,5,14,7	567	0.001761	0.105237	0.512365	0.202089
Chromosome6	11,8,6,4,13,10,2,1,3,9,12,5,14,7	646	0.001546	0.092387	0.604752	0.876247
Chromosome7	11,8,6,4,13,10,2,1,3,9,12,5,14,7	690	0.001546	0.092387	0.697139	0.316298
Chromosome8	8,11,10,6,13,4,2,1,12,9,3,5,14,7	567	0.001761	0.105237	0.802376	0.058517
Chromosome9	11,8,6,4,13,10,2,1,3,9,12,5,14,7	646	0.001546	0.092387	0.894763	0.066287
Chromosome10	8,11,10,6,13,4,2,1,12,9,3,5,14,7	567	0.001761	0.105237	1.000000	0.886208
Generation 2						
Chromosome1	8,11,10,6,13,4,2,1,12,9,3,5,14,7	567	0.001761	0.101771	0.101771	0.764770
Chromosome2	8,11,6,10,13,4,2,1,12,9,3,5,14,7	559	0.001786	0.103224	0.204995	0.257894
Chromosome3	11,8,6,4,13,10,2,1,12,3,9,5,14,7	593	0.001684	0.097316	0.302311	0.038804
Chromosome4	8,11,6,10,13,4,2,1,12,9,3,5,14,7	559	0.001786	0.103224	0.405536	0.278019
Chromosome5	8,11,10,6,13,4,2,1,12,9,3,5,14,7	567	0.001761	0.101771	0.507306	0.625510
Chromosome6	11,8,6,4,13,10,2,1,12,3,9,5,14,7	593	0.001684	0.097316	0.604622	0.665656
Chromosome7	11,8,6,4,13,10,2,1,9,12,3,5,14,7	567	0.001761	0.101771	0.706393	0.086999
Chromosome8	8,11,10,6,13,4,2,1,12,9,3,5,14,7	563	0.001773	0.102492	0.808885	0.060807
Chromosome9	11,8,6,4,13,10,2,1,3,9,12,5,14,7	690	0.001546	0.089344	0.898229	0.744691
Chromosome10	8,11,10,6,13,4,2,1,12,9,3,5,14,7	567	0.001761	0.101771	1.000000	0.479808

Generation 3						
Chromosome1	8,11,10,6,13,4,2,1,12,9,3,5,14,7	567	0.001761	0.098508	0.098508	0.098182
Chromosome2	11,8,6,4,13,10,2,1,12,3,9,5,14,7	593	0.001684	0.094196	0.192703	0.412804
Chromosome3	8,11,10,6,13,4,2,1,12,9,3,5,14,7	563	0.001773	0.099206	0.291909	0.369934
Chromosome4	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.101547	0.393456	0.891797
Chromosome5	8,11,4,10,13,6,2,1,12,9,3,5,14,7	520	0.001812	0.101363	0.494819	0.596652
Chromosome6	8,11,4,6,10,13,2,1,12,9,3,5,14,7	520	0.001812	0.101363	0.596182	0.492588
Chromosome7	8,11,4,6,10,13,2,1,12,9,3,5,14,7	531	0.001802	0.100815	0.696997	0.363546
Chromosome8	11,8,4,6,13,10,2,1,12,3,9,5,14,7	538	0.001789	0.100094	0.797090	0.276685
Chromosome9	8,11,6,4,13,10,2,1,12,9,3,5,14,7	520	0.001815	0.101547	0.898637	0.660970
Chromosome10	8,11,4,6,10,13,2,1,12,9,3,5,14,7	520	0.001812	0.101363	1.000000	0.772859
Generation 4						
Chromosome1	8,11,4,10,13,6,2,1,12,9,3,5,14,7	507	0.001812	0.100089	0.100089	0.127972
Chromosome2	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100270	0.200359	0.443244
Chromosome3	11,8,4,6,13,10,2,1,12,3,9,5,14,7	528	0.001789	0.098835	0.299194	0.655711
Chromosome4	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100270	0.399464	0.389733
Chromosome5	8,11,4,6,10,13,2,1,12,9,3,5,14,7	519	0.001802	0.099548	0.499012	0.645236
Chromosome6	8,11,4,10,13,6,2,1,12,9,3,5,14,7	504	0.001812	0.100089	0.599101	0.392541
Chromosome7	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100270	0.699371	0.227274
Chromosome8	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100270	0.799641	0.683385
Chromosome9	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100270	0.899911	0.733207
Chromosome10	8,11,4,6,10,13,2,1,12,9,3,5,14,7	500	0.001812	0.100089	1.000000	0.438438
Generation 5						
Chromosome1	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100018	0.100018	0.133368
Chromosome2	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100018	0.200036	0.409262
Chromosome3	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100018	0.300054	0.127552
Chromosome4	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100018	0.400072	0.563917
Chromosome5	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100018	0.500091	0.706026

Chromosome6	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100018	0.600109	0.493229
Chromosome7	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100018	0.700127	0.389490
Chromosome8	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100018	0.800145	0.358732
Chromosome9	8,11,6,4,13,10,2,1,12,9,3,5,14,7	500	0.001815	0.100018	0.900163	0.024617
Chromosome10	8,11,4,6,10,13,2,1,12,9,3,5,14,7	500	0.001812	0.099837	1.000000	0.656380

Table A.2.3. Selections from the results obtained by running the GSOA for the car engine pump valve, as detailed in Chapter 6

Swarms Population	Fitness value
11,8,6,4,13,10,2,1,12,9,3,5,14,7	510
11,8,4,6,13,10,2,1,12,9,3,5,14,7	507
11,8,4,13,6,10,2,1,12,9,3,5,14,7	520
11,8,4,13,6,10,2,1,12,3,9,5,14,7	504
8,11,6,4,13,10,2,1,12,9,3,5,14,7	550
8,11,4,6,10,13,2,1,12,9,3,5,14,7	540
8,11,4,10,13,6,2,1,12,9,3,5,14,7	551
11,8,6,4,13,10,2,1,12,3,9,5,14,7	593
11,8,4,6,13,10,2,1,12,3,9,5,14,7	502
1,5,3,4,2,6,7,8,9,10,11,12,13,14	563
1,6,4,3,2,5,7,9,8,10,13,12,11,14	560
1,9,4,2,3,5,7,6,8,11,10,14,13,12	640
13,10,11,9,12,14,8,5,7,6,3,4,2,1	500
5,2,3,1,13,11,10,12,14,6,7,9,8,4	515
5,7,6,8,9,4,3,1,2,10,14,13,12,11	516
5,7,6,8,9,4,3,1,2,10,14,13,11,12	613
5,7,6,8,9,4,3,1,2,10,14,12,11,13	494
5,7,6,8,9,4,3,1,2,14,13,12,11,10	509
5,7,6,8,9,4,3,1,10,14,13,12,11,2	514
13,10,9,12,14,8,5,7,6,3,4,2,1,11	562

1,3,2,5,7,9,8,10,13,12,11,14,6,4	511
1,5,3,4,2,6,8,9,10,11,12,13,14,7	494
1,9,4,3,5,7,6,8,11,10,14,13,12,2	532
13,10,11,9,12,14,5,7,6,3,4,2,1,8	540
5,7,6,8,9,4,3,2,14,13,12,11,10,1	525
1,9,4,2,3,5,7,6,8,10,14,13,12,11	494
13,10,11,12,14,8,5,7,6,3,4,2,1,9	520
13,11,9,12,14,8,5,7,6,3,4,2,1,10	570
11,8,4,6,13,2,1,12,9,3,5,14,7,10	551
8,11,4,6,10,2,1,12,9,3,5,14,7,13	564
13,11,9,12,14,8,7,6,3,4,2,1,10,5	578
11,8,4,6,13,2,1,12,9,3,14,7,10,5	494
8,11,4,6,10,2,1,12,9,3,14,7,13,5	502
7,6,8,9,4,3,2,14,13,12,11,10,1,5	509
1,9,4,3,7,6,8,11,10,14,13,12,2,5	502
1,2,3,4,6,7,8,9,10,11,12,13,14,5	576
1,3,2,4,5,6,7,8,9,10,11,12,13,14	494
3,2,1,4,6,5,7,9,8,11,10,13,12,14	545
6,3,1,2,4,5,8,9,10,11,14,13,12,7	587
1,3,2,4,7,8,9,10,11,12,13,14,6,5	613
3,2,1,4,6,5,7,9,11,10,13,12,14,8	600
6,3,1,2,4,5,8,9,11,14,13,12,7,10	601
8,11,6,4,13,10,2,1,12,9,3,5,14,7	494
1,3,2,4,7,,8,9,10,11,12,13,14,6,5	560
7,6,8,9,4,3,2,14,13,12,11,10,1,5	504
13,11,9,12,14,8,5,7,6,3,4,2,1,10	575
5,7,6,8,9,4,3,2,14,13,12,11,10,1	525
13,11,9,12,14,8,7,6,3,4,2,1,10,5	570

APPENDIX 3

The software codes used during this research:

Code 1:

```
%initialize population


N=6; % number of parts
M = 100;% population size
preMatrix = [];
cr=0.20; %set cross over rate
mr=0.1; %set mutation rate
cp; %cross point
G; %total genes length
mr; %mutation rate
gn; %number of genes in assembly chromosome
f = [0,0,0,0,0,0,0]; %initialize fitness sum for each assembly
chromosome
F=0; % total fitness sum for all assembly chromosome
R; % roulette wheel probability for each assembly chromosome
r; % random number generated for each assembly chromosome
P; % cumulative probability of each assembly chromosome
parentChromosome; %denotes selected parents chromosome that will
mate
genearation =1;
% generate initial random chromosomes

global init= randi([4,10],M,N);disp(R); % generate random
chromosome values 1-10 minutes



%validity check using precedence matrix
function a = checkValidity(randNum,prec)
    % Check random Matrix against precedence Matrix
    a = 0; % validity checker variable
    randCount =1;% used to hold randNum count
    len=length(prec);
    sumRow =0;
    if(randCount=1) % check for first element to see if its a
valid starting sequence
       disp(randNum(1));
       S = sum(prec(randNum(1)),2);
```

```
        disp(S);

        % start

    else
        return 0;
    end


end

% generate valid initial random chromosomes
%val=randi([4,10],M,N);
while 1
    if genCount>=100
        break;
    end
    randMatrix = randperm(4)
    checkValidity(randMatrix,precMatrix);
    genCount=genCount+1;

end


%main loop runs for 100 generation

while generation < 100 do
    %Evaluation & Selection of assembly chromosome

        for countOuter = 1:C % for each chromosome
            %Evaluate each chromosome
            for countInner = 1 :i

f[countOuter]+=1/chromosome[countOuter][countInner];

            end %end inner for loop
            %compute total fitness sum for all chromosome
                 F += f [countOuter];
            end %end outer for loop
```

```
    %chromosomes probability computation using Roulette wheel
    %obtaining probability using Roulette Wheel
                for countR =1 To countR <=C  % for each
chromosome
                                    %compute probability of each
chromosome
                                    R[countR] = f[countR]/F;
                end %end for loop
                     %cumulative probability
                     holdCount; % hold current chromosome count
                      for countC =1 To countC <=C % for each
chromosome
                                     holdCount = countC;
                                    %compute probability of each
chromosome
                                while (holdCount>=1):
                                     P[countC] += R[holdCount];
                                     --holdCount;
                                end %end while loop
                      end %end for loop



                     %chromosomes actual selection
                     %generate random number for each
chromosomes
                     for countN =1 To countN <=C  % for each
chromosome
                                r[countN] = rand(0,1) % generate
random number between 0 and 1
                     end %


%choose which chromosome to retain, if the random number
generated is less than the
%the cumulative probability of the any of the chromosome, the
chromosome at the
%first instance is replaced e.g. If random number r [1] is
greater than P [1] and %smaller than P [2] then select
Chromosome [2] as a chromosome in the new %population for next
generation:
     for countR =1 To countR <=C  % for each chromosome random
number
```

```
          for countCP=1 to CountCP<C  % for each chromosome
cumulative probability
                              if(r[countR] <  P[countCP])
                                  chromosome[countCP] = chromosome
[countR]
                                  end %end if statement
                              end%end inner for loop
          end %end outer for loop

    generation +=generation;


%CrossOver (using one-cut point)

    % select parents assembly chromosomes to mate
    countP=1; %parent chromosome count
    for countC=1:C  % for each chromosome
                                      cp[countC] = rand(0,1)
%generate random number between 0 & 1
                                      if(r[countC] < cr)
                                          parentChromosome[countP]
= chromosome[countC]
                                      end %end if statement
    end %end for loop
    % mate parent chromosomes
    for countC=1:CountP  % for each parent chromosome
        R[countC] = rand(1, CountP) %generate random number
between 1 & parent chromosome count

                                      if(r[countC] != countP)
                                          %replace chromosome at
countC with chromosomes at countC++ %from randomly generated
cross point cp

swapFromRandomCrossPoint(parentChromoses[countC],
parentChromosomes[++countC], cp)
                                      else
                                           %replace chromosome at
countP with chromosomes at 1 %from randomly generated cross
point cp

swapFromRandomCrossPoint(parentChromoses[countP],
parentChromosomes[1], cp)
```

```
                                        end %end if statement
    end %end for loop

% Mutation
%compute total length of chromosomes
        G=gn * i
% number of mutations
         M = r*G
% Carry out mutation and replace mutated chromosomes with random
number from (1-6mins)
  count =0;% to track which gene is referred to by random number
        for countM=1:M  % for each mutation M
               r[countM] = rand (1, G) %generate random number
between 1 and G
            for countOuter =1:C % for each chromosome
                %Evaluate each chromosome
                    for countInner= 1:i
                                    count++;
                                      if(count==r[countM])
                                             if(r[countM]<mr)

chromosomes[countOuter][countInner]= random(1,6);
                                             end % end inner if
statement
                                      end % end outer if statement
                    end %end inner for loop
            end %end outer for loop
        end %end outer outer for loop


    %compute fitness again
    for countOuter =1 :C % for each chromosome
        %Evaluate each chromosome
        for countInner= 1 :i

f[countOuter]+=1/chromosome[countOuter][countInner];
        end %end inner for loop
    %compute total fitness sum for all chromosome
         F += f [countOuter]
    end %end outer for loop


end % end main while loop
```

Code 2:

```
import java.util.Scanner;
import java.util.stream.IntStream;
import java.util.Random;
public class GSOAlgorithm {
public static void main (String[] args)
{
 int noComponent =0; //variable to save the no of chromosome for simulation
 int noGeneration =0;
 int noIteration =0;
 int logIteration =0;
 Scanner s = new Scanner(System.in);
 System.out.println("Please send the number of components in a chromosome");
 noComponent =s.nextInt();
 s.nextLine(); // throw away the new line
 int[][] priorityMatrix = new int[noComponent][noComponent]; // initialise the priority matrix with the number of components
 int[][] setUpTimeMatrix = new int[noComponent][noComponent];

 //Scan the priority matrix values
 for (int i=0;i<noComponent;i++)
 {
  System.out.println("Enter Priority Matrix Row "+(i+1));
  for (int j =0;j<noComponent;j++ )
  {
   priorityMatrix[i][j] = s.nextInt(); // scan the integer values from the user
  }
```

```
}
//Scan the setup time matrix values
 for (int i=0;i<noComponent;i++)
 {
  System.out.println("Enter Setup Time Matrix Row "+(i+1));
  for (int j =0;j<noComponent;j++ )
  {
   setUpTimeMatrix[i][j] = s.nextInt(); // scan the integer values from the user
  }
 }
//display the priority matrix entered
System.out.println("Your priority matrix is given below:");
for (int i=0;i<noComponent;i++)
{
 for (int j =0;j<noComponent;j++ )
 {
  System.out.print(priorityMatrix[i][j]);
 }
 System.out.println();
}
System.out.println("Your SetUp time matrix is given below:");
for (int i=0;i<noComponent;i++)
{
 for (int j =0;j<noComponent;j++ )
 {
  System.out.print(setUpTimeMatrix[i][j]);
 }
 System.out.println();
```

```
}
System.out.println("Please insert the number of generation");
noGeneration =s.nextInt();
s.nextLine();
System.out.println("Please insert the number of iterations");
noIteration =s.nextInt();
s.nextLine();
int[] minF_Obj = new int[noIteration];

while (logIteration != noIteration)
{
System.out.println("Chromosome initialisation");
int[][] Chromosome = new int[noGeneration][noComponent];
for(int i=0;i<noGeneration;i++)
{
 Chromosome[i] = chromosomeInitialisation(priorityMatrix);
}
for(int i=0;i<noGeneration;i++)
{
 for (int j=0;j<noComponent;j++)
 {
  System.out.print(Chromosome[i][j]);
 }
 System.out.println();
}
System.out.println("Chromosome Evaluation");
int[] F_Obj = new int[noGeneration];
F_Obj = chromosomeEvaluation(Chromosome,setUpTimeMatrix);
```

```
for (int i=0;i<noGeneration;i++)
 System.out.println(F_Obj[i]);
System.out.println("Chromosome Fitness");
double[] Fitness = new double[noGeneration];
Fitness = chromosomeFitness(F_Obj);
for (int i=0;i<noGeneration;i++)
 System.out.println(Fitness[i]);
System.out.println("Chromosome Probability");
double[] Probability = new double[noGeneration];
Probability = chromosomeProbability(Fitness);
for (int i=0;i<noGeneration;i++)
 System.out.println(Probability[i]);
System.out.println("Chromosome Cumulative");
double[] Cumulative = new double[noGeneration];
Cumulative = chromosomeCumulative(Probability);
for (int i=0;i<noGeneration;i++)
 System.out.println(Cumulative[i]);
System.out.println("Chromosome RandomNumber");
double[] randomNumber = new double[noGeneration];
randomNumber = chromosomeRandomNumber(Chromosome);
for (int i=0;i<noGeneration;i++)
 System.out.println(randomNumber[i]);
System.out.println("Chromosome New Chromosome Generation");
int[][] newChromosome = new int[noGeneration][noComponent];
newChromosome = NewCromosomeGeneration(Chromosome,Probability, randomNumber);
for (int i=0;i<noGeneration;i++)
{
 for (int j=0;j<noComponent;j++)
```

```
 {
  System.out.print(newChromosome[i][j]);
 }
  System.out.println();
}
System.out.println("Chromosome crossover");
int cP= (int)(noGeneration *((double)10/(double)100));
int[][] crossover = new int[cP][noComponent];
crossover = chromosomeCrossover(Chromosome,10);
for (int i=0;i<cP;i++)
{
 for (int j=0;j<noComponent;j++)
 {
  System.out.print(crossover[i][j]);
 }
  System.out.println();
}
System.out.println("Chromosome mutation");
int[][] mutation = new int[2][noComponent];
mutation = chromosomeMutation(Chromosome,10,priorityMatrix);
for (int i=0;i<2;i++)
{
 for (int j=0;j<noComponent;j++)
 {
  System.out.print(mutation[i][j]);
 }
  System.out.println();
}
```

```
System.out.println("Chromosome New Generation");
int[][] newGeneration = new int[noGeneration][noComponent];
newGeneration = chromosomeNewGeneration(Chromosome,mutation,crossover);
for (int i=0;i<noGeneration;i++)
{
 for (int j=0;j<noComponent;j++)
 {
  System.out.print(newGeneration[i][j]);
 }
  System.out.println();
}
//new LineChart_GA("Evaluation of Chromosomes","Evaluation",F_Obj);
Chromosome =newGeneration;
int min =F_Obj[0];

for(int k=0;k<F_Obj.length;k++)
{
 if (min >F_Obj[k])
  min= F_Obj[k];
}
minF_Obj[logIteration]=min;
System.out.println(min);
logIteration++;
 }

 new LineChart_GA("Evaluation of Chromosomes","Evaluation",minF_Obj);
}
```

```
public static int[] chromosomeInitialisation (int[][] priorityMatrix)
{
 int noComponents = priorityMatrix.length;
 int[] chromosome = new int[noComponents];
 int[] firstComponent = new int[noComponents];
 int[] otherComponent = new int[noComponents];
 int sum =0;
 int counter = 0;
 int chromosomeCounter=0;
 boolean componentExist=false;
 for (int i=0;i < noComponents; i++) //for each components
 {
  if (i==0)//for the first component
  {
   for (int j=0;j<noComponents;j++)
   {
    sum = IntStream.of(priorityMatrix[j]).sum(); // Sum each row of priorityMatrix
    if (sum == 0)
    {
     counter=0;
     for (int k = 0; k < firstComponent.length; k++)
     {
         if (firstComponent[k] != 0)
             counter ++;
     }
     firstComponent[counter]= j+1;
    }
   }
```

```
//Check whether initial component for chromosome available (priority should be 0)
if(counter == 0)
 System.out.println("Given priority Matrix is not valid");
else
{
 for (int k = 0; k < chromosome.length; k++)
 {
    if (chromosome[k] != 0)
     chromosomeCounter ++;
 }
 //generate random number and pick one component
 Random rn = new Random();
 counter = 0;
 for (int k = 0; k < firstComponent.length; k++)
 {
    if (firstComponent[k] != 0)
        counter ++;
 }
 chromosome[chromosomeCounter]=firstComponent[rn.nextInt(counter)];
 }
}
else
{
 for (int j=0;j<noComponents;j++)
 {
  componentExist = false;
  sum = IntStream.of(priorityMatrix[j]).sum(); // Sum each row of priorityMatrix
  if (sum < i)
```

```
{
 for (int k = 0; k < chromosome.length; k++)
 {
  if (chromosome[k] == j+1)
   componentExist=true;
 }
 if (componentExist == false)
 {
  counter = 0;
  for (int k = 0; k < otherComponent.length; k++)
  {
      if (otherComponent[k] != 0)
          counter ++;
  }
  otherComponent[counter]= j+1;
 }
}
}
 chromosomeCounter=0;
 for (int k = 0; k < chromosome.length; k++)
 {
    if (chromosome[k] != 0)
     chromosomeCounter ++;
 }
 //generate random number and pick one component
 Random rn = new Random();
 counter = 0;
 for (int k = 0; k < otherComponent.length; k++)
```

```
    {
        if (otherComponent[k] != 0)
            counter ++;
    }
    chromosome[chromosomeCounter]=otherComponent[rn.nextInt(counter)];
  }
  otherComponent = new int[noComponents]; // empty an array
 }
 return chromosome; // return the chromosome generated
}
public static int[] chromosomeEvaluation (int[][] chromosomeMatrix, int[][] setupTime)
{
 int noGenerations = chromosomeMatrix.length;
 int noComponents = chromosomeMatrix[0].length;
 int[] F_Obj = new int [noGenerations];
 int component;
 int time=0;
 int assemblyTime =0;
 System.out.println(noComponents);
 System.out.println(noGenerations);
 for (int i=0; i<noGenerations;i++)
 {
  for (int j=0;j<noComponents;j++)
  {
   if (j==0)
   {
    component =chromosomeMatrix[i][j];
    assemblyTime = setupTime[component-1][component-1];
```

```
   }
   else
   {
    component = chromosomeMatrix[i][j];
    for(int k=0;k<=j;k++)
    {
     int tempComponent = chromosomeMatrix[i][k];
     time=time+setupTime[component-1][tempComponent-1];
    }
    assemblyTime =assemblyTime+time;
    time=0;
   }
  }
  F_Obj[i]= assemblyTime;
 }
 return F_Obj;
}
public static double[] chromosomeFitness (int[] F_Obj)
{
 int noGenerations = F_Obj.length;
 double Fitness[] = new double[noGenerations];
 for(int i=0;i<noGenerations;i++)
 {
  Fitness[i]=(1/(1+(double) F_Obj[i]));
 }
 return Fitness;
}
public static double[] chromosomeProbability (double[] Fitness)
```

```
{
 int noGenerations = Fitness.length;
 double Probability[] = new double[noGenerations];
 double sum =0;
 for(int i=0;i<noGenerations;i++)
  sum =sum+Fitness[i];
 for(int i=0;i<noGenerations;i++)
 {
  Probability[i]=Fitness[i]/sum;
 }
 return Probability;
}
public static double[] chromosomeCumulative (double[] Probability)
{
 int noGenerations = Probability.length;
 double Cumulative[] = new double[noGenerations];
 double sum =0;
 for(int i=0;i<noGenerations;i++)
 {
  for (int j=0;j<=i;j++)
  {
   sum =sum+Probability[j];
  }
   Cumulative[i]=sum;
 }
 return Cumulative;
}
public static double[] chromosomeRandomNumber (int[][] chromosome)
```

```
{
 int noGenerations = chromosome.length;
 double randomNumber[] = new double[noGenerations];
 for(int i=0;i<noGenerations;i++)
 {
  randomNumber[i]= Math.random(); //Generate random number between 0 and 1
 }
 return randomNumber;
}


public static int[][] NewCromosomeGeneration(int[][] chromosome,double[] Probability,
double[] randomNumber)
{
 int logs=0;
 int noGenerations = chromosome.length;
 int noComponents = chromosome[0].length;
 int[] tempMatrix = new int[noComponents];
 int[][] newChromosome = new int[noGenerations][noComponents];
 for (int i=0;i<noGenerations;i++)
 {
  for(int j=0;j<noGenerations;j++)
  {
   if (j==0 && logs ==0)
   {
    if (Probability[j] > randomNumber[i])
    {
     logs=1;
     for(int k=0;k<noComponents;k++)
```

```
    tempMatrix[k]=chromosome[j][k];
   }
  }
  else
  {
   if(j<noGenerations-1 && logs==0 && Probability[j] < randomNumber[i] &&
Probability[j+1] > randomNumber[i])
   {
    logs=1;
    for(int k=0;k<noComponents;k++)
     tempMatrix[k]=chromosome[j+1][k];
   }
   if(j<noGenerations-1 && logs==0 &&  Probability[j] > randomNumber[i])
   {
    logs=1;
    for(int k=0;k<noComponents;k++)
     tempMatrix[k]=chromosome[j][k];

   }
   if (logs==0 && j>noGenerations && Probability[j] > randomNumber[i] )
   {
    logs=1;
    for(int k=0;k<noComponents;k++)
     tempMatrix[k]=chromosome[j][k];
   }
  }
  if (j==noGenerations)
  {
```

```
   if (logs==0)
   {
    Random rn = new Random();
    tempMatrix=chromosome[rn.nextInt(noGenerations)];
   }
  }
 }
 newChromosome[i]=tempMatrix;
}
return newChromosome;
}

public static int[][] chromosomeCrossover(int[][] chromosome,int percentage)
{
 int noGenerations = chromosome.length;
 int noComponents = chromosome[0].length;

 int[] firstChromosome =new int[noComponents];
 int[] secondChromosome =new int[noComponents];
 int[] tempChromosome =new int[noComponents];
 int counter=0,logs=0,exist =0;
 int cP=(int)(noGenerations *((double)percentage/(double)100));
 int[][] crossover =new int[cP][noComponents];
 for (int i=0;i<cP;i++)
 {
  Random rn = new Random();
  firstChromosome=chromosome[rn.nextInt(noGenerations)];
  secondChromosome=chromosome[rn.nextInt(noGenerations)];
```

```
for(int j=0;j<noComponents;j++)
{
 if (j==0)
 {
  if (rn.nextInt(2)==1)
  {
   tempChromosome[0]= firstChromosome[j];
  }
  else
  {
   tempChromosome[0]= secondChromosome[j];
  }
 }
 else
 {
  if (rn.nextInt(2)==1)
  {
   for(int k=0;k<noComponents;k++)
   {
    exist =0;
    logs=0;
    if(logs==0)
    {
     for(int x=0; x<tempChromosome.length; x++)
     {
      if (tempChromosome[x]== firstChromosome[k])
       exist=1;
```

```
  }
  if (exist != 1)
  {
   counter=0;
   for (int y=0; y<tempChromosome.length; y++)
   {
       if (tempChromosome[y] != 0)
       {
        counter ++;
       }
   }
   tempChromosome[counter]= firstChromosome[k];
   logs=1;
  }

 }
}
}
else
{
 for(int k=0;k<noComponents;k++)
 {
  exist=0;
  logs=0;
  if(logs==0)
  {
   for(int x=0; x<tempChromosome.length; x++)
   {
```

```
     if (tempChromosome[x] == secondChromosome[k])
      exist=1;
    }
    if (exist != 1)
    {
     counter=0;
     for (int y=0; y<tempChromosome.length; y++)
     {
        if (tempChromosome[y] != 0)
        {
         counter =counter+1;
        }
     }
     tempChromosome[counter]= secondChromosome[k];
     logs=1;
    }
   }
  }
 }
 logs=0;
}
crossover[i]=tempChromosome;
tempChromosome =new int[noComponents];
firstChromosome = new int[noComponents];
secondChromosome = new int[noComponents];
exist=0;
logs=0;
```

```
 counter=0;
 }
 return crossover;
}

public static int[][] chromosomeMutation(int[][] chromosome,int percentage,int[][]
priorityMatrix)
{
 int noGenerations = chromosome.length;
 int noComponents = chromosome[0].length;

 int[] firstChromosome =new int[noComponents];
 int[] secondChromosome =new int[noComponents];
 int[][] tempChromosome =new int[noGenerations][noComponents];
 int sum1=0,log1=0,log2=0,counter=0, element1=0,element2=0,sum=0,value1=0,value2=0;
 int cP=(int)(noGenerations *((double)percentage/(double)100));
 int[][] mutation =new int[cP][noComponents];
 for (int i=0;i<cP;i++)
 {
  Random rn = new Random();
  firstChromosome=chromosome[rn.nextInt(noGenerations)];
  secondChromosome=chromosome[rn.nextInt(noGenerations)];
  while(log1==0)
  {
   value1=rn.nextInt(noComponents);
   element1=firstChromosome[value1];
   for(int j=0;j<noComponents;j++)
   {
```

```
 sum=sum+priorityMatrix[element1-1][j];
 }
 if (sum==0)
  log1=1;
}
sum=0;
while(log2==0)
{
 value2=rn.nextInt(noComponents);
 element2=secondChromosome[value2];
 for(int j=0;j<noComponents;j++)
 {
  sum=sum+priorityMatrix[element2-1][j];
 }
 if (sum==0)
  log2=1;
}
firstChromosome[value1]=element2;
secondChromosome[value2]=element1;
for(int j=0;j<mutation.length;j++)
{
 for(int k=0;k<mutation[0].length;k++)
 {
  sum1=sum1 +mutation[j][k];
 }
 if (sum1!=0)
  counter++;
}
```

```
  tempChromosome[counter]=firstChromosome;
  tempChromosome[counter+1]=secondChromosome;
  counter=0;
  sum=0;
 }
 mutation=tempChromosome;
 return mutation;
}

public static int[][] chromosomeNewGeneration(int[][] chromosome,int[][] mutation, int[][]
crossover)
{
 int noGenerations = chromosome.length;
 int noComponents = chromosome[0].length;
 int[][] newGenerationChromosome =new int[noGenerations][noComponents];
 int count=0,count1=0,sum1=0,sum2=0;
 for(int i=0;i<mutation.length;i++)
 {
  for(int k=0;k<mutation[0].length;k++)
  {
   sum1=sum1 +mutation[i][k];
  }
  if (sum1!=0)
   count++;
 }
 for (int i=0;i<count;i++)
 {
  newGenerationChromosome[i]=mutation[i];
 }
```

```
for(int k=0;k<(noGenerations);k++)
 {
  Random rn = new Random();
  newGenerationChromosome[k]=chromosome[rn.nextInt(noGenerations)];
 }
 return newGenerationChromosome;
}
}
```

FORM UPR16

Research Ethics Review Checklist

Please include this completed form as an appendix to your thesis (see the Research Degrees Operational Handbook for more information

Postgraduate Research Student (PGRS) Information				Student ID:	467507
PGRS Name:	Fawaz Saad T. Alharbi				
Department:	School of Mechanical and Design Engineering	First Supervisor:	Dr. Qian Wang		
Start Date: (or progression date for Prof Doc students)	01-02-2014				
Study Mode and Route:	Part-time ☐	MPhil ☐	MD ☐		
	Full-time ☒	PhD ☒	Professional Doctorate ☐		

Title of Thesis:	Investigation into GA and GSOA Optimisation Approaches for Solving Assembly Sequence Problems
Thesis Word Count: (excluding ancillary data)	37084

If you are unsure about any of the following, please contact the local representative on your Faculty Ethics Committee for advice. Please note that it is your responsibility to follow the University's Ethics Policy and any relevant University, academic or professional guidelines in the conduct of your study

Although the Ethics Committee may have given your study a favourable opinion, the final responsibility for the ethical conduct of this work lies with the researcher(s).

UKRIO Finished Research Checklist:
(If you would like to know more about the checklist, please see your Faculty or Departmental Ethics Committee rep or see the online version of the full checklist at: http://www.ukrio.org/what-we-do/code-of-practice-for-research/)

a)	Have all of your research and findings been reported accurately, honestly and within a reasonable time frame?	YES ☒ NO ☐
b)	Have all contributions to knowledge been acknowledged?	YES ☒ NO ☐
c)	Have you complied with all agreements relating to intellectual property, publication and authorship?	YES ☒ NO ☐
d)	Has your research data been retained in a secure and accessible form and will it remain so for the required duration?	YES ☒ NO ☐
e)	Does your research comply with all legal, ethical, and contractual requirements?	YES ☒ NO ☐

Candidate Statement:		
I have considered the ethical dimensions of the above named research project, and have successfully obtained the necessary ethical approval(s)		
Ethical review number(s) from Faculty Ethics Committee (or from NRES/SCREC):		2334-C814-964A-58BC-76A0-7645-A453-595F
If you have *not* submitted your work for ethical review, and/or you have answered 'No' to one or more of questions a) to e), please explain below why this is so:		
Signed *(PGRS)*:	FAWAZSAAD	Date: 03-07-2020

UPR16 – April 2018