DESIGN AND IMPLEMENTATION OF
AUTOMATED ASSEMBLY PLANNING METHODOLOGY

A Thesis in
Industrial Engineering
by
Phani Teja Nallamothu

© 2013 Phani Teja Nallamothu

Submitted in Partial Fulfillment
of the Requirements
for the Degree of
Master of Science

May 2013
The thesis of Phani Teja Nallamothu was reviewed and approved* by the following:

Soundar Kumara
Allen E. Pearce/Allen M. Pearce Professor of Industrial and Manufacturing Engineering
Thesis Co-Advisor

Daniel A Finke
Research Associate, Applied Research Laboratory
Thesis Co-Advisor

Christopher Saldana
Assistant Professor of Industrial and Manufacturing Engineering

Paul Griffin
Professor of Industrial and Manufacturing Engineering
Head of the Department of Industrial and Manufacturing Engineering

*Signatures are on file in the Graduate School.
ABSTRACT

Assembly sequencing is the most important phase of the broader problem of assembly planning. Assembly planning includes other areas such as resource allocation, line balancing, work cell layout etc. Assembly sequencing can be defined as the ordering of collision free operations which bring all the parts together when the geometric description of all the parts in the final assembly is given. Assembly costs account for more than 50% of the production costs. A good assembly plan can reduce the production costs by a huge amount. We consider assembly sequencing problem in this thesis. Automating the assembly planning process will help the designers make the decisions which will result in better assembly planning and hence reduced production time and costs.

In order to automate the assembly sequence generation, we need the CAD data of the final assembly. In this thesis, STEP files are taken as input to generate the assembly sequences for a given assembly, as it is the most commonly used CAD format. Assembly sequences are generated by representing the CAD information from the STEP files as Liaison graph and blocking graph. Liaison graph is the graph which represents the contact information between any two parts in an assembly. Blocking graph represents the blocking relations between all the parts in an assembly. Once the Liaison and blocking graphs are generated, feasible assembly sequences are generated from those two graphs.

As the number of components increases, time required to assemble the parts increases exponentially. In order to reduce the assembly/ disassembly time, it is necessary to group different parts into subassemblies. These subassemblies can be assembled/ disassembled in parallel which results in saving a lot of time. The information from the liaison graphs and fasteners is used in generating subassemblies from the assembly STEP file.
An assembly sequence is considered optimal when the sequence generated is feasible, i.e. it satisfies all the geometric constraints and the assembly cost associated with that sequence is minimal when compared to all other sequences. Assembly cost can be computed based on several criteria such as the number of direction changes, number of gripper changes, and the similarity in consequent assembly operations. Since the assembly sequence planning problem is highly constrained and large scale problem, genetic algorithms have been used to find the optimal assembly plan. Through sampling the feasible solutions by using relevant criteria, the time to compute optimal solution can be reduced considerably. Genetic algorithms are used to reduce the computation time drastically. As the traditional genetic algorithms will generate a lot of infeasible solutions during evolution, this will reduce the efficiency of algorithm. To address this issue, a new guided genetic algorithm is presented which deals only with feasible assembly sequences at every point of evolution. The criteria used to compute the assembly cost is the number of reorientations/ direction changes. Finally, all these algorithms are implemented on two test cases to illustrate their effectiveness.
# Table of Contents

List of Figures .................................................................................................................. vii
List of Tables ..................................................................................................................... ix

ACKNOWLEDGEMENTS ..................................................................................................... x

Chapter 1: Introduction ..................................................................................................... 1
  1.1 Assembly Planning ...................................................................................................... 2
  1.2 Motivation .................................................................................................................. 5
  1.3 Scope of the Thesis ..................................................................................................... 6
  1.4 Organization of the Thesis ......................................................................................... 7

Chapter 2: Literature Review ............................................................................................. 8
  2.1 Combinatorial Approach ............................................................................................ 9
  2.2 Geometric Approach ................................................................................................ 11
  2.3 Assembly Sequence Optimization Algorithms ........................................................ 12

Chapter 3: Liaison Graph Generation ................................................................................. 16
  3.1 Liaison Graph Definition .......................................................................................... 16
  3.2 Methodology for Liaison Graph Generation .............................................................. 18

Chapter 4: Assembly Sequence Generation ....................................................................... 22
  4.1 Blocking Graph Definition ....................................................................................... 22
  4.2 Methodology for Blocking Graph Generation ......................................................... 25
  4.3 Disassembly Sequence Generation ........................................................................... 30
  4.4 Methodology for Assembly Sequence Generation ................................................... 31

Chapter 5: Subassembly Identification ................................................................................. 33
  5.1 Methodology for Sub-Assembly Identification ......................................................... 34
  5.2 Sub-Assembly Identification Example ..................................................................... 36

Chapter 6: Optimal Assembly Sequence Generation ......................................................... 40
  6.1 Terminology for Genetic Algorithm ....................................................................... 41
  6.2 Guided Genetic Algorithm ....................................................................................... 43

Chapter 7: Results and Analysis ......................................................................................... 49
  7.1 Test Case 1: ............................................................................................................. 49
    7.1.1 Liaison Graph .................................................................................................... 49
    7.1.2 Blocking Graphs ................................................................................................. 50
    7.1.3 Assembly Sequences ......................................................................................... 51
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Different aspects of Assembly Planning</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>An Illustration of Assembly Sequence Generation Literature</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>5-part assembly</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Initial Liaison Graph</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Intermediate Liaison Graph</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>Final Liaison Graph</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>Liaison Graph generation algorithm</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>STEP file of a chassis</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>Triangular Meshed Part Example</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>Bounding Box for a part</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>Bounding Box for intersection of parts</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>Illustration of projection method to generate DBG</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>5-part assembly</td>
<td>23</td>
</tr>
<tr>
<td>14</td>
<td>Initial Blocking Graph</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>Intermediate Blocking Graph in +X direction</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>Final Blocking Graph of 5-part assembly in +X direction</td>
<td>25</td>
</tr>
<tr>
<td>17</td>
<td>Blocking graph generation algorithm</td>
<td>25</td>
</tr>
<tr>
<td>18</td>
<td>STEP file of a Chassis</td>
<td>26</td>
</tr>
<tr>
<td>19</td>
<td>Triangular Meshed Part Example</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>Bounding box of a part</td>
<td>27</td>
</tr>
<tr>
<td>21</td>
<td>Ray projection from Bounding Box-1 to Bounding Box-2</td>
<td>28</td>
</tr>
<tr>
<td>22</td>
<td>Ray projection from Bounding Box-2 to Bounding Box-1</td>
<td>28</td>
</tr>
<tr>
<td>23</td>
<td>Blocking Graph after 1st part is removed in +X direction</td>
<td>30</td>
</tr>
<tr>
<td>24</td>
<td>Blocking graph after removing 2 parts</td>
<td>30</td>
</tr>
<tr>
<td>25</td>
<td>Assembly Sequence generation algorithm</td>
<td>31</td>
</tr>
<tr>
<td>26</td>
<td>Sub-assembly Identification algorithm</td>
<td>34</td>
</tr>
<tr>
<td>27</td>
<td>9-part assembly [39]</td>
<td>36</td>
</tr>
<tr>
<td>28</td>
<td>Liaison Graph for 9-part assembly</td>
<td>36</td>
</tr>
<tr>
<td>29</td>
<td>Modified Liaison Graph for 9-part assembly</td>
<td>37</td>
</tr>
<tr>
<td>30</td>
<td>Blocking Graph of 9-part assembly in +Z direction</td>
<td>38</td>
</tr>
<tr>
<td>31</td>
<td>Modified Blocking Graph with 5, 6 as a single node-h</td>
<td>38</td>
</tr>
<tr>
<td>32</td>
<td>Modified Blocking graph with 1, 9 as a single node h</td>
<td>39</td>
</tr>
<tr>
<td>33</td>
<td>Modified Blocking Graph with 1, 8, and 9 as single node h</td>
<td>39</td>
</tr>
<tr>
<td>34</td>
<td>Flow chart for guided genetic algorithm</td>
<td>44</td>
</tr>
<tr>
<td>35</td>
<td>Flowchart for initial feasible population generation</td>
<td>45</td>
</tr>
<tr>
<td>36</td>
<td>5-part assembly</td>
<td>49</td>
</tr>
<tr>
<td>37</td>
<td>Liaison Graph for 5-part assembly</td>
<td>49</td>
</tr>
<tr>
<td>38</td>
<td>Blocking graph of 5-part assembly in +X direction</td>
<td>50</td>
</tr>
<tr>
<td>39</td>
<td>Blocking graph of 5-part assembly in +Y direction</td>
<td>50</td>
</tr>
<tr>
<td>40</td>
<td>Blocking graph of 5-part assembly in +Z direction</td>
<td>51</td>
</tr>
</tbody>
</table>
Figure 41: STEP view of a 23-part assembly ................................................................. 52
Figure 42: Liaison Graph for 23-part assembly .............................................................. 53
Figure 43: Blocking Graph for 23-part assembly in +X direction ..................................... 53
Figure 44: Blocking Graph of 23-part assembly in +Y direction ...................................... 54
Figure 45: Blocking Graph for 23-part assembly in +Z direction ...................................... 54
Figure 46: Engine Assembly ....................................................................................... 56
Figure 47: Piston Sub-assembly .................................................................................... 57
Figure 48: Muffler Sub-assembly .................................................................................. 57
Figure 49: Filter Sub-assembly ..................................................................................... 58
Figure 50: Exploded view of Engine Sub-assembly ......................................................... 58
Figure 51: 12-part assembly CAD view ....................................................................... 59
Figure 52: Fitness v/s Generations for 12 part assembly ................................................. 62
Figure 53: Execution time v/s Population size/No: of generations .................................. 62
Figure 54: Average Fitness v/s No: of generations for 4-part assembly ......................... 63
Figure 55: Average Fitness v/s No: of generations for 9-part assembly ......................... 63
Figure 56: Average Fitness v/s No: of generations for 11-part assembly ....................... 64
List of Tables

Table 1: Adjacency Matrix in +X direction .......................................................................................... 60
Table 2: Adjacency Matrix in +Y direction .......................................................................................... 60
Table 3: Adjacency Matrix in +Z direction .......................................................................................... 61
Table 4: Execution time of guided genetic algorithm for different assemblies .............................. 64
ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Dr. Soundar Kumara for his valuable guidance throughout my studies at Penn State. The discussions with him gave me valuable insights on how to approach a solution for complex problems. Though he is a genius and vastly experienced, his down to earth nature gave me the freedom to express my ideas. I would also like to thank him for the continuous support and suggestions he gave in other aspects of life apart from education.

I would like to make a special mention of Dr. Daniel A Finke, for his suggestions throughout the course of this thesis. His valuable insights have helped me approach the problem from industrial point of view. I thank him for all the feedback and encouragement he gave during the weekly meetings.

I also acknowledge the support provided by DARPA under the grant HR0011-12-C-0075 iFAB Foundry for making this work feasible.

I would like to thank Dr. Christopher Saldana for his valuable time in reading this thesis. I would also like to thank my colleagues Supreet Mandala and Deepak Agrawal for their valuable ideas and constructive feedback to complete this thesis. I am also thankful to Dr. Paul Griffin for giving me an opportunity to pursue my Masters in Industrial Engineering at Penn State.

Finally, I would also like to thank my family for their continuous motivation, moral support and guidance throughout my career.
Chapter 1: Introduction

Traditionally, product design takes place in two stages. First, the design engineer will design the product based on the requirements. Then the manufacturing engineer will manufacture the product if he can. Else he will pass on the limitations to the design engineer to redesign the product. This process repeats continuously until a satisfactory product can be designed. This will increase the time to manufacture the product. So there is a need to reduce the new product development time. This creates a huge task for designers to consider factors such as design for assembly/disassembly, design for manufacturability, and design for end-of life etc., collectively known as DFX. The solution to this problem is Concurrent Product design. By considering these factors earlier in the design stage, the time and the cost involved in the development of a product can be reduced significantly. Amongst these factors, assembly planning is the most important factor in concurrent product design and it is very time consuming. According to Nof et al. in [1] assembly planning costs account for close to 50% of the production costs. Hence a good assembly planning system can reduce the production cost by a huge amount. Automating the assembly planning process will help the designers make the decisions which will result in better assembly planning and hence reduced production time and costs. The advantages of automated assembly planning are:

- Multiple feasible assembly sequences can be generated and the engineer can chose the sequence which he wants based on criteria such as machine availability, holding cost etc.
- Manufacturing constraints can be incorporated faster into the design process which will result in shorter design time.
- In batch manufacturing, the changeover times are very less. Automated assembly planning will help faster changeovers of people, machines, tools etc.
1.1 Assembly Planning

Assembly planning can be divided into three major areas as shown in Fig.1:

1. Assembly sequence planning.
2. Assembly line balancing.
3. Planning based on economic and environmental factors.

Of the many costs associated with assembly planning, assembly sequence generation is the major contributor. The sequence of mating operations carried out to assemble a group of components into a final stable assembly known as Assembly sequence planning. This is the process which consumes most of the time and effort in product development.

As the number of parts becomes larger the number of possible sequences increases exponentially and without prior knowledge of the best assembly sequences, it would be very difficult for the designers to choose an optimal assembly sequence.
Given a list of all the equipment and a set of design operations, assembly line balancing is the optimal allocation of tasks to each resource to minimize the production cost and time. Problem of resource allocation and work-cell layout are also studied in this field.

Assembly planning also considers end-of-life options of a product. To maximize the benefit in repair and recycling, a designer needs to choose assembly operations and materials involved such that final product is economical and environmentally efficient throughout its lifecycle.

While other fields have been widely explored in recent years, research in automating the assembly sequence generation is sparse, necessitating further research.

The problem of assembly sequence generation can be simplified if the number of parts in the assembly is less and components can be grouped together as subassemblies to generate lesser number of sequences. Subassemblies can be defined as a group of components which are stable and can be assembled/ disassembled together into the main assembly. The advantages of grouping components into subassemblies are

1. Assembly time will be reduced as the assemblies can be done in parallel.
2. Computation time will be reduced as there will be less number of parts to deal with.

Automated assembly planners will find the feasible assembly sequences by considering the geometric and physical constraints. Once the assembly sequence generation is automated, it is necessary to find out the optimal assembly sequence in order to save time and cost. The optimal assembly sequence is the sequence which has the minimum number of reorientations, minimum number of tool changes and gripper changes etc. Although, theoretically, the exhaustive search methods can guarantee an optimal solution, it is infeasible because of the
computational complexity. As the number of parts increases, the computational complexity increases exponentially. Researchers have studied different methods to solve this problem, the most efficient ones being the ones where heuristic rules are applied. The two contrasting objectives these new methods must solve are reducing the computational complexity and maintaining all the valid sequences.

Since the assembly sequence planning problem is a highly constrained, large scale problem, researchers have used genetic algorithms (GA) to find the optimal assembly plan. Genetic algorithms are used to drastically reduce the computation time. Different genetic algorithms like traditional genetic algorithms, variations in traditional genetic algorithms, enhanced genetic algorithms have been used to find the optimal assembly sequence from a product representation. Genetic algorithms were proven to be highly effective to solve NP-hard problems. The basic idea of using genetic algorithms in assembly planning is to represent assembly sequences as chromosomes, apply genetic operators on these chromosomes to find a better child chromosome based on a fitness function like cost or time. Experimental results by several researchers confirm that starting from an initial population of random sequences; genetic algorithms are bound to converge to an optimal solution.

But in case of heavily constrained assembly planning problems, GA’s will generate considerable number of infeasible solutions during evolution which will reduce the efficiency of the algorithm. To address these issues, a guided genetic algorithm has been presented which deals only with feasible assembly sequences during the entire evolution process. As a result of using guided GA’s, we can overcome the limitations of traditional GA’s.
1.2 Motivation

Assembly sequence plays a major role in determining some of the important characteristics such as the difficulty of assembly, in process testing and unit production cost. The role of assembly sequencing in the early phase of product design is fundamental for optimizing not only the manufacturability of the Assembly costs account for 10-30 % of the total cost of many industrial products. Assembly sequence planning can directly influence the productivity, quality and also the fixed costs which involve machinery and other equipment.

Traditionally an experienced industrial engineer generates an assembly sequence for any given product. But there is no guarantee that the optimal assembly sequence hasn’t been overlooked. Also for complex products, there will be many possible assembly sequences. It will be a Herculean task even for an experienced engineer to generate all feasible assembly sequences as the number of sequences grows exponentially with the number of products. If we can automate the assembly sequence generation, we can guarantee that only the feasible sequences will be generated. The planning of assembly sequences is very time consuming. There will be many production delays because of the time taken in assembly planning. For a small batch production, assembly planning cost contributes heavily to the total production cost. In order to expedite the assembly sequence generation, reduce the cost and improve the quality, systematic procedures for automating the assembly sequence generation are required. A small design change can have a large impact on the assembly sequence alternatives. Automating the assembly sequence generation in these situations will help the designer assess the different assembly sequences for a given product. One can gain huge benefits if the assembly sequences are generated during the initial stages of the product design. The designer can produce a product for which the assembly sequence is already considered which will reduce the need for further Design
for Assembly analysis. Autonomous systems for space, military and deep sea explorations need the ability to generate assembly or disassembly situations for a particular situation which is not previously modeled. In modern times, there is a need to produce different products on the same shop floor. It will be very helpful if we have the knowledge of all the feasible assembly sequences of each product. The assembly sequence used for each product might not be optimal but will be very efficient if it uses the idle equipment in the shop floor. For most of the products, disassembly sequence is the reverse of the assembly sequence. The ease of disassembling a component is an important factor for reducing service time, repair costs and simplifying recycling procedures. Production system, production environment, product complexity and product variety are some of the factors which influence assembly sequence generation. Thus, automating the assembly sequence generation is very important to reduce manufacturing costs, improve production efficiency and expedite the product design.

All the aforementioned points suggest that as the mechanical design of an assembly becomes increasingly complex, the sequence in which those parts will be assembled will become extremely important.

1.3 Scope of the Thesis

Assembly planning covers a wide variety of areas and there are a lot of research questions that still are unanswered. The major problems which are addressed in this thesis are:

1. Automate the assembly sequence generation process using CAD data of the final assembly
2. Identify subassemblies which can be assembled in parallel from the main assembly.
3. Generate optimal assembly sequences based on several selection criteria such as number of reorientations, number of gripper changes etc. using a guided genetic algorithm.

1.4 Organization of the Thesis

Chapter 2 deals with the literature review on assembly planning. In Chapter 3, the definition of liaison graph and the algorithm to generate a liaison graph from CAD data is explained. Blocking graph definition, the algorithm to generate blocking graph are discussed in Chapter 4 along with assembly sequence generation algorithm. Chapter 5 covers the sub-assembly identification algorithm followed by a guided genetic algorithm to generate optimal assembly sequence in Chapter 6. The final Results and Conclusions are discussed in Chapter 7. References follow.
Chapter 2: Literature Review

Assembly planning includes topics such as resource allocation, work cell layout, tolerance related issues, assembly sequence generation. Of all the sub problems in assembly planning, assembly sequencing is the most important problem. The different levels of granularity of an assembly can be sub-assemblies, parts, features and boundary representations. All the research which has already been done in the area of assembly sequence generation can be broadly classified into two categories based on the knowledge of precedence relations of all the parts in the assembly. The two approaches are

1. Combinatorial approach and
2. Geometric approach

Figure 2 illustrates the literature review on assembly sequence generation.

![Figure 2: An Illustration of Assembly Sequence Generation Literature](image-url)
If we have the knowledge of the precedence relations, we use the combinatorial approach; else geometric approach will be followed. We discuss the background literature related to combinatorial and geometric approach in the next sections.

2.1 Combinatorial Approach

In the combinatorial approach, assembly sequences are represented in several ways. Once the assembly sequences are represented, optimization algorithms can be applied on them to find out the optimal assembly sequence. Assembly representations are mostly part and sub-assembly based. Parts can be represented by labels, nodes, vector elements etc. Sub-assemblies can be described as nodes, lists, vector graphs etc. De Mello and Sanderson in [2] suggested that assembly sequence representations can be divided into explicit and implicit ones. These representations require the liaison graph input whose nodes are the parts in the assembly and the edges are the contacts between the parts. Mantripragada and Whitney in [3] proposed an augmented version of the liaison graph which captures dimensional constraints among one or more degrees of freedom between the parts. De Mello and Sanderson in [4] also proposed another version of the liaison graph where they added the information about the type of contact (based on geometry) and attachment (glue, screw, pressure fit). Walter in [5] represented a single assembly sequence by a partial assembly tree which is a binary tree whose root node is the final assembly and the leaf nodes are the individual parts. Walter in [5] also added additional information such as ordering of operations, insertion information and fixturing information. Multiple assembly sequences can be represented by a directed graph whose nodes are a set of parts and the arcs are the feasible assembly operations. De Mello and Sanderson in [2], Romney in [6], Wilson and Rit in [7], Thomas et al. in [8], Lee and Saitou in [9] have all used AND/OR graphs to represent assembly sequences. In an AND/OR graph, nodes are the stable
subassemblies and the hyper arcs correspond to feasible assembly operations. Leaf nodes are the individual parts and the root node is the entire assembly. Explicit representations of an assembly can be used to determine precedence relations between connections and also to encode feasible assembly sequences. Bourjault in [10] systematically used the precedence relations between specific pairs of connections. Fazio and Whitney in [11] used precedence relations between a given connection and a set of connections. These precedence relations are found out by asking a series of questions to a human expert. De Mello and Sanderson in [4] proposed methods to generate precedence relations from liaison diagrams by using cut-set methods. Bonneville et al. in [12] encoded precedence relations as geometric constraints which show absence of collision free trajectories to allow the merging of two sub-assemblies. Walter in [5] constructed mating operations from all possible precedence relations-‘strictly precede’, ‘precede or accompany’, ‘same operation’, ‘and different operation’. Naphade et al. in [13] define assembly sequencing as a constraint satisfaction problem in terms of establishment conditions. Huang and Lee in [14] considered the effect of precedence constraints not only on assembly tasks but also the resources like fixtures and grasps. Goldwasser and Motwani in [15] considered some optimal criteria such as number of directions in which part can be removed, number of tools and tool changes, number of re-orientations, depth of assembly sequence etc.

The major issue which is not addressed in the above literature is that in all the cases, the precedence relations are determined by the user. This method of finding precedence relations makes it practically impossible to generate precedence relations for bigger assemblies because of the explosion problem. This issue can be addressed if the precedence relations are generated automatically without the user intervention.
2.2 Geometric Approach

Research in combinatorial approach assumes knowledge of the precedence relations. The second approach is the geometric approach where the precedence relations are learnt from the CAD assembly description.

Lee and Shin in [32], Wilson and Rit in [7] proposed a generate and test approach for precedence relations, but this method will be impractical for bigger assemblies because of the inherent combinatorial explosion.

Depending on how the information from the CAD models is used, the researchers have followed two different paths. The first path uses the spatial representations of all the parts to determine blocking relations between all parts in the assembly. The second path explores geometric information to do randomized path planning. Blocking graphs determine the directions in which all the parts can be separated from one another. The space partitions of separating directions are computed assuming that the parts are free flying objects.

Romney et al. in [6] computed local translational freedom cones on S2 and then used the blocking graphs to determine the feasible assembly sequences. Schweikard and Schwarzer in [33] expressed contact relations between the parts as linear constraints to determine feasible translational directions. But these local translational freedom cones are difficult to compute for parts which have complex geometry as these algorithms will work effectively only against polyhedral parts i.e., parts which have simple planar geometry.

Wilson and Latombe in [34] introduced the concept of Non Directional Blocking Graphs to determine assembly partitions. It is a subdivision of translations into finite cells where the blocking relations within each cell are fixed. This approach will be very complicated to
implement as it is sensitive to floating point approximations. Also, the complexity grows exponentially with the space dimensions of allowed motions.

Wilson in [35] developed the interference diagram to determine collision free path for a sub assembly. The C-Space of any part is the set of all possible values of the degrees of freedom. The C-obstacles is the subset of C-space where the part collides with other parts. Path Planning means determining a path for each part which is contained in the C-free(C-Space – C-obstacles). La Valle in [36], Choset et al. in [37] used local planners to find out the neighbor free configurations to build a roadmap and then applied graph search on this road map to find the solution path.

One of the major drawbacks in the methods discussed earlier for generating assembly sequences using geometric approach is the difficult of these algorithms in their implementation. Also most of the research work done in this field focuses on polyhedral objects. Practically, the objects are more complex rather than regular polyhedrons. So these algorithms might not work efficiently for all practical cases.

2.3 Assembly Sequence Optimization Algorithms

Once the feasible assembly sequences are generated, optimization algorithms can be applied on them to generate the optimal assembly sequence. Nilsson in [16] used A* algorithm which guarantees to find a minimum cost path i.e. assembly sequence. Martelli and Montanari in [17] used AO* algorithm to find the best assembly sequence. Caselli and Zanichelli in [18], Cao and Sanderson in [19] built assembly Petri Nets from AND/OR graphs to derive efficient computations of assembly sequences. Zha et al. in [20] used knowledge based petri nets which can automatically adjust the theoretical and real assembly parameter deviations which guarantee
the best plan for assembly sequence. Ben–Arie et al. in [21] used a hybrid of colored petri net and knowledge based system to calculate the degree of difficulty in each assembly sequence which helps in choosing the optimal sequence. Motavalli and Islam in [22] used a simulated annealing technique where they defined an energy function in terms of the number of reorientations and the total assembly time. They encoded the precedence constraints in a precedence diagram and tested for each assembly sequence. Chen in [23] encoded the precedence knowledge by using neural networks and expressed AND/OR graph precedence constraints between liaisons in the connection strengths between neurons of a Hopfield net. Cao and Xiao in [30] used immune optimization approach where assembly sequence problems are represented as antigens and the assembly sequences are represented as antibodies. Wang et al. in [31] applied ant colony optimization approach to find out the optimal assembly sequences. Holland in [24] initially used genetic algorithms to find high quality solutions to complex engineering problems. He proved that better solutions can be produced by applying genetic operators like crossover, mutation, inversion to a known set of population. He also proved that by applying genetic operators continuously, over time, continually better solutions can be produced. Bonneville in [25] is among the first ones to use genetic algorithms to optimize the assembly sequences generated. He used mutation and crossover to compute optimal assembly sequence. Here, the initial population is supplied by the user. Later genetic operators are applied to the best plans from any given generation and the new plans are generated. As a result, population size will grow tremendously. The algorithm will terminate after searching the entire search space or after generating several good feasible assembly sequences. Fujimoto in [26] used genetic operators like mutation, crossover and reproduction to find the optimal assembly sequence. Here, random initial population is generated. Parent assembly sequences are selected
probabilistically and then the operators are applied to generate the offspring. They classify the solution space into families where each family contains single valid assembly sequence. When the genetic operators are applied to the assembly sequences selected in random, invalid offspring are replaced by their corresponding family valid assembly sequence through a transformation function. Chen in [27] used five operators (reproduction, crossover, mutation, cut and paste, break join) to find optimal assembly sequences for two example products. Here, initial population can be supplied by the user or it can be selected randomly. The population size is also fixed. The next generation is selected from the best offspring and the best parents. The algorithm terminates when the total fitness doesn’t improve in several generations. Hong and Cho in [28] used crossover and mutation to generate assembly sequences for two mechanical products. Here initial population is generated randomly. Fitness function is based on the total assembly cost, and sequences are selected probabilistically based on the fitness function values. The algorithm terminates after a specified number of generations or if the fitness doesn’t improve over a period of time. Senin in [29] used crossover and mutation to generate assembly plans for several test structures. He showed that the genetic algorithms, using any selection methods will find high quality assembly plans when compared to enumeration techniques. For complex test cases, genetic algorithms will give low quality plans when compared to enumeration techniques. However the execution times are very less in genetic algorithms when compared to enumeration techniques.

From the above literature we can see that only small amount of research has been done to generate assembly sequences using CAD data. Most of the research which has been done till now is based on the premise that the precedence relations are known beforehand which is not the case for complex assemblies. Though there is some research work done using geometric approach,
these algorithms will be difficult to implement in practice and also they work for polyhedral assemblies and won’t be efficient if they are applied for complex assemblies. This thesis addresses the problems of generating liaison graphs and blocking graphs without any prior knowledge of precedence relationships even for complex assemblies. This thesis also tries to address the problem of detecting subassemblies from bigger assemblies which will help to minimize the assembly time as the subassemblies can be assembled in parallel. Also, there is a major drawback in all the genetic algorithms discussed above to generate the optimal assembly sequence. Though the above algorithms generate an optimal assembly sequence, they also generate number of infeasible solutions during evolution. This will increase the computational time exponentially. In order to address this issue, we used a guided genetic algorithm in [38] which deals with only feasible sequences at all instances of time during the evolution process.
Chapter 3: Liaison Graph Generation

3.1 Liaison Graph Definition

Liaison graph is the representation of contact information between any two parts in a given assembly. In a liaison graph, each part in the assembly is represented as a node. There will be an edge between any two nodes in the liaison graph if the parts corresponding to the nodes are touching each other in the assembly. Liaison graph is illustrated for the assembly in Fig.3

The first step is to represent each part in the assembly as a node in the liaison graph as in Fig.4
From Fig.3, we can see that Lid is connected to Screw1, Screw2 and Box. So there will be an edge between Lid node and the other three nodes. Fig.5 shows the intermediate liaison graph.

![Intermediate Liaison Graph](image1)

Figure 5: Intermediate Liaison Graph

Similarly other connections between the parts can be seen from the assembly figure and are represented similarly. Fig.6 shows the final liaison graph for the 5-part assembly.

![Final Liaison Graph](image2)

Figure 6: Final Liaison Graph
3.2 Methodology for Liaison Graph Generation

The algorithm for generating the liaison graph is explained below.

1. Get the STEP file of the assembly. STEP stands for Standard for Exchange of Product Data. STEP file represents 3D objects in Computer Aided Design (CAD) and related information. A sample STEP file can be seen in the Fig.8.
2. Perform triangular meshing for the STEP file. We used Open Cascade to generate the triangular mesh. Open Cascade divides each STEP file into several faces. Each face is further divided into multiple triangles. We will have the coordinates of all the triangles once we perform triangular meshing. An example of a triangular meshed part is shown in Fig. 9.

3. Once the triangular meshing is done using Open Cascade, we construct bounding boxes for all the parts. A bounding box is a cuboid with minimum volume which encloses the entire part. It is constructed by knowing the coordinates of the lower and the upper most points of a part from the STEP file. Bounding box can be constructed for both parts and
faces of the part if we know the upper and lower most points. Fig.10 shows the bounding box constructed for a small part.

Figure 10: Bounding Box for a part

4. Once the bounding boxes for all the parts are generated, we have to check whether the bounding box of a part is intersecting with the bounding box of any other part. If the bounding boxes of the two parts are not intersecting, we can say that there is no connection between the two parts. However, if the bounding boxes of any two parts are intersecting, we have to dig deeper to know whether there is any connection between the two parts or not. There might be cases where the bounding boxes of the parts are intersecting whereas the actual parts are not touching each other. An example of such a case is shown below in Fig.11.

Figure 11: Bounding Box for intersection of parts
5. Once we know that the bounding boxes of the two parts are touching each other, we have to check whether the bounding boxes of any face of the first part touches the bounding box of any face of the second part. If the bounding boxes of the faces of both the parts don’t touch each other, we can conclude that there is no connection between the parts. However, if the bounding box on any of the face of first part touches the bounding box of any other face in the second part, we have to check for actual intersection of the parts.

6. After knowing that the bounding boxes of two faces of the two parts are touching each other, we check the actual intersection of parts by checking whether the triangles on the face of first part are intersecting with any of the triangles on the face of the second part of the intersecting faces.

7. If the 3-D triangles on the two parts intersect, we can conclude that the two parts are touching each other. In the liaison graph, we represent all the parts in the assembly as nodes. There will be an edge between these two parts.

8. This process is repeated until all the parts are checked for connections. The final graph will be a liaison graph for the given assembly.

A detailed liaison graph is generated and shown in case studies 1 and 2 in the Results section.
Chapter 4: Assembly Sequence Generation

Assembly sequence plays a major role in determining some of the important characteristics like the difficulty of assembly, in process testing and unit production cost. The role of assembly sequencing in the early phase of product design is fundamental for optimizing not only the manufacturability of the product but also expediting the design process itself. Assembly Sequence generation is one of the most expensive and time consuming process in a production planning process. It is a herculean task for the traditional shop floor engineer to generate an assembly sequence for an assembly with thousands of parts. The process of automatic assembly sequence generation is to create a sequence of parts from the CAD files in which parts can be assembled into a final assembly. In order to generate assembly sequences for a particular assembly, there is a need to know the blocking relations between individual parts so that the sequences generated will be feasible. The blocking relations between any two parts are represented by blocking graphs. We have automated the assembly sequence generation by using the blocking graphs among the six principal directions ±x, ±y, ±z. First we will generate the directional blocking graphs and then use these directional blocking graphs to find out the feasible assembly sequences.

4.1 Blocking Graph Definition

Directional Blocking Graph (DBG) is the representation of precedence of removal of parts in a given direction. Each component in the assembly is represented as a node and there will be a directed edge from one part (A) to another (B) if part B is blocking part A when part A is translated in that direction. Here we have considered six principal directions ±x, ±y, and ±z, because assembly sequencing problem is a NP-hard problem and it would be computationally expensive and infeasible to solve for infinite directions.
DBG can be calculated by a projection method where part (A), which has to be removed, is projected in a given direction with respect to other parts (B) and if the projections overlap and if the part B is in front of part A, then Part B is blocking Part A. An illustration of this is shown in Fig.12.

Figure 12: Illustration of projection method to generate DBG

The following assembly in Fig.13 illustrates the directional blocking graph in + x direction.

Figure 13: 5-part assembly
The first step is to represent each part in the assembly as a node in the directional blocking graph as in Fig.14

![Figure 14: Initial Blocking Graph](image)

From the 5-part assembly, we can see that Box is preventing the removal of all other parts in +X direction. So there will be directed edges from the nodes of all other 4 parts to the Box node. The intermediate directional blocking graph will look like the graph in Fig.15

![Figure 15: Intermediate Blocking Graph in +X direction](image)

Similarly other blocking relations can be seen from the assembly diagram and are represented in the blocking graph. The final blocking graph in +x direction is shown in Fig.16
4.2 Methodology for Blocking Graph Generation

![Diagram showing the methodology for blocking graph generation]

Figure 17: Blocking graph generation algorithm
The algorithm for generating the directional blocking graph is explained below.

1. Get the STEP file of the assembly. STEP file represents 3D objects in Computer Aided Design (CAD) and related information. A sample STEP file can be seen in the Fig.18

![Figure 18: STEP file of a Chassis](image1)

2. Perform triangular meshing for the STEP file. We used Open Cascade to generate the triangular mesh. Open Cascade divides each STEP file into several faces. Each face is further divided into multiple triangles. We will have the coordinates of all the triangles once we perform triangular meshing. An example of a triangular meshed part is shown in Fig.19

![Figure 19: Triangular Meshed Part Example](image2)
3. Once the triangular meshing is done using Open Cascade, we construct bounding boxes for all the parts. A bounding box is a cuboid with minimum volume which encloses the entire part. It is constructed by knowing the coordinates of the lower and the upper most points of a part from the STEP file. Bounding box can be constructed for both parts and faces of the part if we know the upper and lower most points. Fig. 20 shows the bounding box constructed for a small part.

![Figure 20: Bounding box of a part](image)

4. Once the bounding boxes for all the parts in the assembly are generated, we have to choose the direction in which the blocking graph will be generated. In order to find the blocking relations between any two parts in a given direction, we have to check whether the bounding boxes of any two parts intersect when the bounding box of one part is projected onto the other part in the given direction. To check the intersection of bounding boxes, we project rays from the vertices of first bounding box in a given direction and see whether these rays intersect the second bounding box as shown in Fig. 21
We also have to project rays from the second bounding box onto the first bounding box in the negative direction like in Fig. 22 as there might be cases where the size of second bounding box is much smaller than the first bounding box like in Fig. 21, the rays will not intersect the box even though the bounding boxes intersect when part 1 is translated in +X direction.

If the rays don’t touch the bounding boxes in both the directions, we can conclude that the two parts won’t block each other in the given direction. There will not be any directed edge from the first part to the second part in that particular direction. However, if the rays touch the bounding boxes, we have to check for the bounding box intersection of the faces of both the parts.

5. Once we know that the bounding boxes of the two parts are touching each other, we have to check whether the bounding boxes of any face of the first part touches the bounding
box of any face of the second part in a given direction. If the rays from the bounding boxes of the faces of one part doesn’t touch the bounding box of the other part we can conclude that there is no blocking relation between the two parts. However, if the rays from the bounding box on any of the face of first part touches the bounding box of any other face in the second part; we have to check for actual intersection of the parts.

6. After knowing that at least one of the bounding boxes of the faces of the two parts collide with each other when one part is translated in a given direction, we check the actual intersection of parts by checking whether the triangles on the face of first part are intersecting with any of the triangles on the face of the second part of the intersecting faces when one triangle is projected on to the plane of the other triangle in the given direction.

7. If the 3-D triangles on the faces of the two parts collide when the parts are translated, we can conclude that the two parts are touching each other. In the blocking graph, we represent all the parts in the assembly as nodes. There will be a directed edge from the first part to the second part for the blocking graph in the given direction.

8. This process is repeated until all the parts are checked for blocking relations. The final graph will be a blocking graph for the assembly in the given direction called as the directional blocking graph.

This procedure can be applied to find the blocking graph in any given direction. In this thesis, the major focus is on the principal directions, ±X, ±Y, ±Z. Once the blocking graphs are generated, we can find the feasible assembly sequence by identifying the disassembly sequence and then inverting the disassembly sequence to get the assembly sequence.
4.3 Disassembly Sequence Generation

Once the blocking graphs in the six principal directions are generated, we have to get a disassembly sequence in order to generate assembly sequence. To generate a disassembly sequence, we have to search for nodes in the blocking graphs which don’t have any outgoing edges. For the 5-part assembly, we can see that the Box node doesn’t have any outgoing edge in the +X blocking graph. So Box has to be disassembled first. The remaining blocking graph is

![Blocking Graph after 1st part is removed in +X direction](image1)

Figure 23: Blocking Graph after 1st part is removed in +X direction

From the above graph in Fig.23, we can see that Cargo doesn’t have any outgoing edges. So it has to be removed next. The blocking graph will become like the graph shown in Fig.24

![Blocking graph after removing 2 parts](image2)

Figure 24: Blocking graph after removing 2 parts
Similarly, Lid can be removed next as it doesn’t have any outgoing edges followed by Screw1 and Screw2. So the final disassembly sequence will be

\[ \text{Box (}+X\text{)} \rightarrow \text{Cargo (}+X\text{)} \rightarrow \text{Lid (}+X\text{)} \rightarrow \text{Screw1 (}+X\text{)} \rightarrow \text{Screw2 (}+X\text{)} \]

The assembly sequence will be the inverse of disassembly sequence. It will be

\[ \text{Screw2 (}−X\text{)} \rightarrow \text{Screw1 (}−X\text{)} \rightarrow \text{Lid (}−X\text{)} \rightarrow \text{Cargo (}−X\text{)} \rightarrow \text{Box (}−X\text{)} \]

The above example illustrates how an assembly sequence is generated from the blocking graphs. The assembly sequence generation algorithm is discussed in detail in the next section.

### 4.4 Methodology for Assembly Sequence Generation

![Figure 25: Assembly Sequence generation algorithm](image-url)

The assembly sequence generation algorithm is discussed in detail in the next section.
The assembly sequence generation algorithm is as follows:

1. From the directional blocking graph generation algorithm described earlier, generate the blocking graphs in all the six principal directions.

2. From the six blocking graphs, we will make a list of nodes/list of set of nodes which don’t have any outgoing edges. This means that these parts are not being blocked by any other parts when they are translated in that direction and are free to be disassembled in that particular direction.

3. Select a node/set of nodes randomly from the above list and store the information of the node and its disassembly direction.

4. Update the blocking graphs by removing the node/set of nodes selected above from all the six directional blocking graphs.

5. Move to step 1 and continue doing this until the entire disassembly sequence has been generated.

6. Invert the disassembly sequence to get the assembly sequence.

Using the above assembly sequence generation algorithm, multiple feasible assembly sequences can be generated.
Chapter 5: Subassembly Identification

As the number of parts in the assembly increases, the number of assembly/disassembly sequences increases exponentially. The complexity of this problem can be reduced by dealing with only feasible assembly sequences and discarding the infeasible sequences in the early stage of assembly/disassembly sequence generation process. The computation time can be further reduced by grouping components into sub-assemblies and treating these sub-assemblies as individual components. As a result of this, the number of parts involved in the assembly sequence generation process will be less and thereby having a significant reduction in the computation time. This chapter presents a methodology to extract sub-assemblies automatically from the assembly file taking the fastener data as input. The algorithm presented in this section is an implementation of the work done by N.S.Ong et al. [39]. The liaison graph and the blocking graphs used in this algorithm are generated by the algorithms discussed in the earlier chapters. This algorithm takes the fasteners information as input and gives out stable sub-assemblies as output. The basic idea of this algorithm is to group components that are fastened together as possible sub-assemblies and determine which of these possible sub-assemblies can form stable sub-assemblies. A stable sub-assembly can be define as a group of components that will not fall off in any direction when the sub-assembly is oriented during assembly/disassembly process. The algorithm for automatic detection of sub-assemblies from assembly CAD files is described below.
5.1 Methodology for Sub-Assembly Identification

Figure 26: Sub-assembly Identification algorithm

The above algorithm is explained in detail below.

1. Take the STEP files of assembly for which we want to identify the sub-assemblies as input.

2. Generate the liaison graph and blocking graphs from the STEP file using the algorithms discussed in earlier chapters.
3. Also take the fasteners information as another input to generate the sub-assemblies. From this information, we can tell the type of connection (whether it is a fastened connection or normal connection) between any two components in the assembly.

4. Generate the modified liaison graph by changing the weights of the edges connecting the fastened components. We increase the weight of the edge between any two components by 1 when the type of connection between the components is a fastened connection.

5. Identify the component/node having the highest degree. It is called the base component. Base component is the one where all other parts come and get assembled to it to form the main assembly.

6. Group the components into possible sub-assemblies. Possible sub-assemblies are the group of components that are fastened together.

7. Modify the blocking graphs by grouping all components in the possible sub-assemblies into a single node in the blocking graph.

8. Check whether there are any loops present in the modified blocking graphs. A presence of a loop indicates that those two components are blocking each other and it will be difficult to dis-assemble each of the components separately. They have to be dis-assembled together.

9. If there are loops present, add the component forming the loop to the components of possible sub-assembly. Represent all these components as a single node in the blocking graphs. Check for loops again and repeat the process until no more loops are present in the blocking graphs.

10. If there are no loops present in the blocking graphs, we have to check whether the components of possible sub-assemblies contain the base component. This check is done
to ensure that all other components will come and get assembled to the base component. If the base component is present in the possible sub-assembly, discard the sub-assembly and look for other possible sub-assemblies. If the base component is not present, the possible sub-assembly will be the stable sub-assembly.

5.2 Sub-Assembly Identification Example

The 9-part assembly example shown in Fig.27 illustrates the sub-assembly identification algorithm.

![Figure 27: 9-part assembly](39)

The liaison graph for the above 9-part assembly generated using the algorithm discussed in earlier chapters is shown in Fig.28.

![Figure 28: Liaison Graph for 9-part assembly](39)
From the assembly, we can see that there are fastened connections between 1-9, 2-4, 3-4 and 5-6.

Now we have to change the weights of these edges as they are fastened connections. Fig.29 shows the new liaison graph.

![Figure 29: Modified Liaison Graph for 9-part assembly](image)

In the above graph, the dotted edge indicates that there is a fastened connection between the two components. Now the node with highest degree is computed by taking the weights of normal line as 1 and the weights of dotted lines as 2. Component 4 will be the base component as it has the highest degree of 8 (2+2+1+1+1+1). The possible sub-assemblies will be (2, 3, 4), (5, 6) and (1, 9) as they have the fastened connections among them. Since (2, 3, 4) has the base component 4 in it, it will be eliminated from the possible sub-assemblies as we don’t want the base component in any of the possible sub-assemblies. The remaining possible sub-assemblies are (5, 6) and (1, 9). The blocking graph for the 9-part assembly in +Z direction is shown in the Fig.30.
Out of the two possible sub-assemblies, let us start with (5, 6). These two nodes will be represented as a single node “h” and the new blocking graph will be generated. From the figure below, we can see that “h” node is forming loops with node 4 and node 7. So “h” cannot be a sub-assembly unless 4 and 7 are also included in it. So the new sub-assembly will be 5,6,4,7. But this cannot be a sub-assembly as it includes the base component 4. This violates our assumption that the base component should not be included in any of the sub-assemblies. Fig.31 illustrates this.

Now we will represent the other possible sub-assembly (1, 9) as a single node “h” and the new blocking graph will be generated as in Fig.32. From Fig.32 we can see that “h” is forming loops...
with node 8. This implies that if we want to remove component 8, node “h” will be blocking it and if we want to remove node “h”, node 8 will be blocking it. The only way to dis-assemble them is to group all these components as a sub-assembly and dis-assemble them together. So 1, 8, 9 will form a sub-assembly. We can also see from Fig.33 that there won’t be any other loops when components 1, 8, 9 are grouped together as a single node.

![Figure 32](image1.png)

Figure 32: Modified Blocking graph with 1, 9 as a single node h

![Figure 33](image2.png)

Figure 33: Modified Blocking Graph with 1, 8, and 9 as single node h

There are no more loops in the blocking graph. The stable sub-assembly will be 1, 8, and 9.
Chapter 6: Optimal Assembly Sequence Generation

Automated assembly planners will find the feasible assembly sequences by considering the geometric and physical constraints. Once the assembly sequence generation is automated, it is necessary to find out the optimal assembly sequence in order to save time and cost. The optimal assembly sequence is the sequence which has the minimum number of reorientations, minimum number of tool changes and gripper changes etc. Although, theoretically, the exhaustive search methods can guarantee optimal solutions; it is infeasible because of the computational complexity. Researchers have studied different methods to solve this problem, the most efficient ones being the ones where heuristic rules are applied. The two contrasting objective these new methods must solve are reducing the computational complexity and also maintaining all the valid sequences.

Since the assembly sequence planning problem is a highly constrained problem researchers have used genetic algorithms (GA) to find the optimal assembly plan. Genetic algorithms are used to drastically reduce the computation time. Genetic algorithms were proven to be highly effective to solve NP-hard problems. The basic idea of using genetic algorithms in assembly planning is to represent assembly sequence as chromosomes, apply genetic operators on these chromosomes to find a better child chromosome (assembly sequence) based on a fitness function like cost or time. Experimental results by several researchers confirm that starting from an initial population of random sequences; genetic algorithms are bound to converge to an optimal solution.

But in case of large constraint assembly planning problems, GA’s will generate a lot of infeasible solutions during evolution which will reduce the efficiency of the algorithm. To
address these issues, a guided genetic algorithm has been presented which deals only with feasible assembly sequences at any point of time during the entire evolution process. As a result of using guided GA’s, we can overcome the limitations of traditional GA’s.

The following section defines the terminology used in genetic algorithms followed by the detailed description of guided genetic algorithm.

**6.1 Terminology for Genetic Algorithm**

Some of the concepts which have been used throughout the algorithm are described below

a. *Adjacency Matrix*

Let \( A = \)

\[
\begin{array}{cccccc}
  a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\
  a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\
  a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\
  a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\
  a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\
\end{array}
\]

be a 5 x 5 matrix which represents the blocking relations in a 5 part assembly in + X direction. Any element in the matrix \( a_{ij} = 1 \) if part \( i \) is getting blocked by part \( j \) when part \( i \) is translated in the + X direction. If part \( j \) is not blocking part \( i \) when part \( i \) is translated in the + X direction, then \( a_{ij} = 0 \). Similarly we have adjacency matrices in + Y, +Z, -X, -Y, -Z the six principal directions.
b. Chromosome Encoding

For an assembly with \( n \) parts, each chromosome is represented as a \( nx2 \) matrix. The first row indicates the sequence in which the parts should be removed and the second row indicates the directions in which the corresponding parts have to be removed. The directions are represented as 0, 1, 2, 3, 4, 5 for +X, +Y, +Z, -X, -Y, -Z respectively. For example, the chromosome

\[
\begin{array}{ccccccc}
5 & 2 & 3 & 4 & 1 & 6 & 7 \\
2 & 1 & 3 & 2 & 2 & 1 & 1 \\
\end{array}
\]

represents a disassembly sequence 5-2-3-4-1-6-7 where part 5 have to be removed in + Z direction, part 2 in +Y direction and so on.

c. Re-orientation

We will say that there is a reorientation when there is a change in direction of disassembly from one part to the next part.

\[
\begin{array}{ccccccc}
5 & 2 & 3 & 4 & 1 & 6 & 7 \\
2 & 1 & 3 & 2 & 2 & 1 & 1 \\
\end{array}
\]

The number of re-orientations in this case will be 4. (2 to 1, 1 to 3, 3 to 2, and then 2 to 1)

d. Fitness Function

Let the chromosome be

\[
\begin{array}{ccccccc}
5 & 2 & 3 & 4 & 1 & 6 & 7 \\
2 & 1 & 3 & 2 & 2 & 1 & 1 \\
\end{array}
\]
The fitness function will be
\[ F = \text{Number of parts} - \text{Number of reorientations} = 7 - 4 = 3 \]
The fitter individuals will always have a high probability of getting selected as parents for the next generation.

### 6.2 Guided Genetic Algorithm

The user input for this algorithm will be the mutation probability, crossover probability, number of generations and population size. All the steps in the algorithm can be seen in the flowchart in Fig.34. The inputs for the guided genetic algorithm are the adjacency matrices which are generated by any of the methods described in the earlier chapters

**a. Initial Population**

Once the adjacency matrices are generated, we generate the initial feasible population taking the adjacency matrices in the six principal directions as input. The algorithm for generating initial feasible population is explained in Fig.35.

**b. Selection**

Once the initial feasible population is generated, we calculate the fitness of each chromosome using the fitness function described earlier. Two chromosomes are selected based on their fitness values using roulette wheel selection method.

**c. Guided Crossover**

This part of the algorithm starts by generating a random number between 0 and 1. If the random number generated is less than the crossover probability, we do the guided crossover. Else we copy the two parents as the two child chromosomes. The input for the guided
crossover will be the feasible chromosomes selected from the initial population. Let the two parent chromosomes be:

![Flow chart for guided genetic algorithm](image)

Figure 34: Flow chart for guided genetic algorithm
Figure 35: Flowchart for initial feasible population generation

Parent 1:

<table>
<thead>
<tr>
<th>5</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>1</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Parent 2:

<table>
<thead>
<tr>
<th>2</th>
<th>5</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Now randomly generate a cut point in parent 1. Let the randomly generated cut point be 3. Now copy the parent 1 into child 1 up to the cut point. Now the child 1 is

\[
\begin{array}{cccc}
5 & 2 & 3 & \\
2 & 1 & 3 & \\
\end{array}
\]

Update the adjacency matrices by removing the rows and columns 5, 2 and 3. Now generate all the feasible parts that can be removed by looking for rows which don’t have 1. Check whether the fourth position is same in both the parents and whether that element is present in the feasible parts list. If both conditions are true, copy that element into the child 1. If any of the above condition is not true, Out of all the feasible parts, randomly select one part and put in the fourth position in the child chromosome with its corresponding direction. We proceed like this until the entire chromosome is filled. In the above case, since the fourth position is same and it is not present in the child chromosome, we copy that element into child chromosome along with the corresponding direction. So the child 1 will be

\[
\begin{array}{cccc}
5 & 2 & 3 & 4 \\
2 & 1 & 3 & 2 \\
\end{array}
\]

At the fifth position, the genes in parent 1 and parent 2 are not equal. So we update the adjacency matrices and generate the feasible parts that can be removed. Let the feasible parts list be

\[
\begin{array}{cc}
1 & 7 \\
2 & 1 \\
\end{array}
\]
This indicates that part 1 can be removed in +Z direction and part 7 can be removed in +Y direction. Now we randomly chose 1 part from this list. Let the randomly chosen part be part 7.

So the child 1 will be

\[
\begin{array}{cccccc}
5 & 2 & 3 & 4 & 7 & \\
2 & 1 & 3 & 2 & 1 & \\
\end{array}
\]

At the sixth position, since the genes in parent 1 and 2 are not equal, we will generate the feasible parts that can be removed. Let the feasible parts list be

\[
\begin{array}{cc}
1 & 6 \\
2 & 2 \\
\end{array}
\]

Now we randomly select one part. Let the part be 6. So the child 1 will be

\[
\begin{array}{cccccccc}
5 & 2 & 3 & 4 & 7 & 6 & \\
2 & 1 & 3 & 2 & 1 & 2 & \\
\end{array}
\]

The last part can be removed in any direction. So the final child 1 will be

Child 1:

\[
\begin{array}{cccccccc}
5 & 2 & 3 & 4 & 7 & 6 & 1 \\
2 & 1 & 3 & 2 & 1 & 2 & 2 \\
\end{array}
\]

Similarly if we exchange the positions of parent 1 and parent 2 and repeat the whole process, we will get child 2.
So by doing this guided crossover, we will get two feasible child chromosomes from two feasible parent chromosomes.

\(\text{d. Guided Mutation}\)

Guided mutation is slightly different from the mutation operators in other genetic algorithms. Here we choose the integer value nearest to \((\text{mutation probability} \times \text{population size})\). Let the integer value be \(k\). In guided mutation, at the end of each generation, we will replace \(k\) random chromosomes from the current population with \(k\) random chromosomes from the initial feasible population.

Once the mutation operation is done, we will have the new generation of the population. This algorithm is again repeated to find the next generation of chromosomes.

From the above algorithm, we can see that during the entire evolution process, only feasible assembly sequences are produced. This is the major advantage of this algorithm. Because of this, the computational time will be reduced drastically (as will be shown in the results section). This algorithm has been implanted on several assemblies and the results are shown in the results section.
Chapter 7: Results and Analysis

The algorithms discussed in the earlier chapters have been applied to several test cases.

The results of two test cases are shown below.

7.1 Test Case 1:

The figure shown below in Fig.36 is a STEP view of a 5- part assembly.

![Figure 36: 5-part assembly](image)

The liaison graph generated is given in Fig. 37.

7.1.1 Liaison Graph

![Figure 37: Liaison Graph for 5-part assembly](image)
From the liaison graph in Fig.37, we can see the contact relations of all the parts in the 5-part assembly. The blocking graphs in +X, +Y, +Z directions are shown in figures 38, 39 and 40.

### 7.1.2 Blocking Graphs

**+X:**

![Blocking graph of 5-part assembly in +X direction](image)

Figure 38: Blocking graph of 5-part assembly in +X direction

**+Y:**

![Blocking graph of 5-part assembly in +Y direction](image)

Figure 39: Blocking graph of 5-part assembly in +Y direction

The blocking graphs in –X, -Y, -Z directions can be obtained by reversing the direction of arrows in the blocking graphs of +X, +Y, +Z directions respectively.
7.1.3 Assembly Sequences

The assembly sequence generated using the assembly sequence generation algorithm discussed in Section 4.4 is:

Step 1:  5 1 4 (-1, 0, 0)

Step 2:  2 (-1, 0, 0)

Step 3:  3 (-1, 0, 0)

7.1.4 Interpretation

The above assembly sequence can be interpreted as follows.

1. Assemble parts 5, 1 and 4 together as a sub-assembly and consider it as the first part in the main assembly.

2. Now assemble part 2 to sub-assembly (5-1-4) in –X direction.

3. Assemble part 3 to the remaining parts in –X direction.

There might be many other feasible assembly sequences but only one assembly sequence is listed above.
7.2 Test Case 2:

The figure shown in Fig.41 is a STEP view of a 23-part assembly.

![STEP view of a 23-part assembly](image)

The liaison graph and the blocking graphs generated for the 23-part assembly are:

### 7.2.1 Liaison Graph

From the STEP view, we can see that part 86425 is in contact with part 85408 and part 85408_2 which is represented correctly by the liaison graph in Fig.42.

Similarly the blocking graph generation algorithm is applied on the STEP file of 23-part assembly in the six principal directions +X, +Y, +Z, -X, -Y, -Z directions. The results are shown in Figures 43, 44 and 45.
7.2.2 Blocking Graphs

+X:

Figure 42: Liaison Graph for 23-part assembly

Figure 43: Blocking Graph for 23-part assembly in +X direction
+Y:

Figure 44: Blocking Graph of 23-part assembly in +Y direction

+Z:

Figure 45: Blocking Graph for 23-part assembly in +Z direction
7.2.3 Assembly Sequences

Step 1: 22 21 (-1, 0, 0)

Step 2: 8 1 23 20 14 7 (0, 1, 0)

Step 3: 9 (0, -1, 0)

Step 4: 3 (0, 0, 1)

Step 5: 10 (-1, 0, 0)

Step 6: 2 (0, 0, 1)

Step 7: 19 (0, 0, -1)

Step 8: 5 (1, 0, 0)

Step 9: 6 (0, 0, -1)

Step 10: 4 (0, 0, -1)

Step 11: 13 (0, 0, -1)

Step 12: 18 (0, 0, -1)

Step 13: 17 (0, -1, 0)

Step 14: 16 (-1, 0, 0)

Step 15: 15 (-1, 0, 0)

Step 16: 12 (-1, 0, 0)

Step 17: 11 (-1, 0, 0)
7.2.4 Interpretation

Parts 22 and 21 are assembled separately and brought together as a single sub-assembly. This will be the first step in our assembly sequencing. Now parts 8, 1, 23, 20, 14, 7 are assembled separately and brought together as a single sub-assembly. The directions information is also shown after the parts information. This will be the second step in this assembly sequencing. Similarly the other steps in the assembly sequence generation can be interpreted.

7.3 Test Case 3:

Fig.46 shows an engine assembly consisting of more than 100 parts.

Figure 46: Engine Assembly

We have taken the fastener information as input and applied the sub-assembly identification algorithm.
7.3.1 Subassemblies
The following sub-assemblies have been identified using the sub-assembly algorithm.

1. Piston Sub-assembly

   ![Figure 47: Piston Sub-assembly](image)

2. Muffler Sub-assembly

   ![Figure 48: Muffler Sub-assembly](image)
3. Filter Sub-assembly

![Figure 49: Filter Sub-assembly](image)

The overall engine assembly and its exploded view can be seen in the Fig.50

![Figure 50: Exploded view of Engine Sub-assembly](image)

The base component identified is the Engine Block where all other parts get assembled to it. There are many other components which can be grouped as sub-assemblies. But they haven’t been identified as we are dealing only with components which are connected by fasteners. Other
sub-assemblies can be identified if we can have information regarding the other connections between different components like welded joints etc.

### 7.4 Test Case 4:

The guided genetic algorithm has been implemented on a Windows 7, i3 Processor 2.4 GHz, 4GB RAM and 64-bit Operating System. The programming language used in Java. The genetic algorithm is applied to several assemblies with different number of parts. The results are illustrated below.

#### 7.4.1 Optimal Assembly Sequence Generation

The guided genetic algorithm has been applied on the 12 part assembly shown in Fig.51 to find the optimal assembly sequence.

![12-part assembly CAD view](image)

First the blocking graphs (adjacency matrices) in all the six principal directions (+X, +Y, +Z, -X, -Y,-Z) are generated using the algorithms discussed in the earlier chapters. The blocking graphs for the above assembly are shown in Tables 1, 2, 3 below.
Similarly the blocking graphs in the –X, -Y, -Z directions have been generated by transposing the blocking graphs in +X, +Y, +Z directions. Once the blocking graphs are generated, they are used as input to the Guided genetic algorithm. The other inputs are Population size: 100 and No: of Generations: 150, mutation probability: 0.02, crossover probability: 0.80

Table 1: Adjacency Matrix in +X direction

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Adjacency Matrix in +Y direction

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Table 3: Adjacency Matrix in +Z direction

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The optimal disassembly sequence generated using the guided genetic algorithm for the 12 part assembly is:

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>1</th>
<th>9</th>
<th>10</th>
<th>5</th>
<th>2</th>
<th>12</th>
<th>11</th>
<th>8</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The assembly sequence is the reverse of disassembly sequence. The number of reorientations in the above example is zero. The plot in Fig.52 illustrates the evolution of genetic algorithm. It finally converges to the optimal sequence. From the plot, we can see that the average fitness of the population increases as the number of generations increases. The time taken for the execution of this algorithm using the computer mentioned earlier is 2186 milliseconds.
Fig. 52 shows the variation in execution time with the population size and also the variation in execution time with the number of generations.

From Fig. 53, we can see that the execution time increases with the increase in population size and also the number of generations. But population size has more impact on the execution time than the number of generations.
Similarly, the algorithm was tested on several assemblies with different number of parts. The following plots illustrate the increase in average fitness with the number of generations.

Figure 54: Average Fitness v/s No: of generations for 4-part assembly

Figure 55: Average Fitness v/s No: of generations for 9-part assembly
Figure 56: Average Fitness v/s No: of generations for 11-part assembly

From the above 3 plots, we can see that the average fitness increases with the number of generations and the genetic algorithm is always converging to an optimal/near optimal solution.

Table 4 illustrates the execution time with different assemblies.

<table>
<thead>
<tr>
<th>No. of Parts</th>
<th>Population Size</th>
<th>No. of Generations</th>
<th>Pc</th>
<th>Pm</th>
<th>Re-orientations</th>
<th>Time (milli-sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>100</td>
<td>150</td>
<td>0.8</td>
<td>0.02</td>
<td>1</td>
<td>921</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>150</td>
<td>0.8</td>
<td>0.02</td>
<td>0</td>
<td>1407</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>150</td>
<td>0.8</td>
<td>0.02</td>
<td>0</td>
<td>1967</td>
</tr>
<tr>
<td>12</td>
<td>100</td>
<td>150</td>
<td>0.8</td>
<td>0.02</td>
<td>0</td>
<td>2250</td>
</tr>
</tbody>
</table>

In Table 4, Pc and Pm represent the mutation and crossover probabilities. The advantages of the current guided genetic algorithm will be even more evident as the number of parts in the assembly increases.
Chapter 8: Conclusions and Future Work

Assembly planning accounts for more than 50% of the total costs. Assembly sequence generation is one of the major components in assembly planning. Traditionally, experienced shop floor engineers will generate the assembly sequences for an assembly based on their previous experience. Automating the assembly sequence generation will result in saving considerable time and money for the designer. In order to automate the assembly sequence process, we need the contact information and the blocking information between different parts in the assembly. CAD data of the assemblies are taken as input to find the spatial locations of parts with respect to each other. Liaison graph represents the contact relations between different parts in the assembly. Blocking graph represents the blocking relations between the parts in different directions. Based on the information from liaison graphs and blocking graphs, assembly sequences can be generated for any given assembly. Algorithms have been presented to generate liaison graphs, blocking graphs and to automate the assembly sequence generation process. The algorithms have been tested on several assemblies and the results are shown in the results section.

Currently, assembly sequences are generated only in the principal directions +X, +Y, +Z, -X, -Y, -Z. There might be cases where some components can be removed/ added only along their principal axes. If we can incorporate these directions in the blocking graphs, the sequences generated will be more effective. Also, the speed of the algorithms can be increased if some practical rules can be applied before generating the assembly sequences. By applying practical rules, the search space for generating assembly sequences will be drastically reduced. The current algorithms only work for components which can be assembled/ dis-assembled in a single translation. For some components, there is a possibility that they can’t be assembled/ dis-assembled in a single translation. They have to assembled/ dis-assembled in multiple steps. This
will be a very good research area as this will increase the efficiency of the sequences generated. There will be many assemblies which include wire harnesses in them. The assembly sequence generation algorithms discussed in this thesis will not work for those assemblies. Further research can be done to include wire harnesses etc. to generate assembly sequences for flexible parts i.e., flexible assembly planning. Further work can consider tooling, bill of materials knowledge etc. in automating the assembly sequence generation process.

Once the assembly sequences are generated, there is a need to reduce the assembly time. It can be reduced if different sub-assemblies are assembled in parallel and brought together for final assembly. We used the fasteners information, liaison graphs and blocking graphs of original assemblies and grouped several components as sub-assemblies. We have applied the sub-assembly identification algorithm on the engine assembly and identified 3 sub-assemblies i.e., piston assembly, muffler assembly and filter assembly.

This sub-assembly identification algorithm works only for fastened connections between components. There might be different connections like welding, press fit etc. between different parts. This algorithm can be improved and different other sub-assemblies can be identified if we can incorporate practical rules for other types of connections between parts. Future research might be focused on identifying sub-assemblies without the use of connections information.

Many feasible assembly sequences might be possible for an assembly. The optimal assembly sequence will be the one which takes less time, less cost, less number of re-orientations, gripper changes etc. In this thesis, a guided genetic algorithm is presented to generate an optimal assembly sequence taking the blocking relations between all the parts in an assembly as input. This novel guided genetic algorithm deals only with feasible assembly
sequences at all points of time. The optimization criterion used is to minimize the number of reorientations in the final assembly sequence. The proposed guided genetic algorithm has the structure of a traditional genetic algorithm except that the mutation and crossover operations are modified in such a way that only feasible assembly sequences will be generated even when crossover and mutation operations are performed. The algorithm has been implemented in Java. The initial population of feasible assembly sequences has been generated using the adjacency matrices (blocking graphs) in all the 6 principal directions. Taking mutation and crossover probabilities as user input, the algorithm has been implemented to several assemblies. This algorithm performs very fast when compared to other algorithms discussed in the literature review section as it deals with only the feasible assembly sequences during the entire evolution process.

One major limitation in all the literature on genetic algorithms for optimal assembly sequence generation and also in all other optimization algorithms is that these algorithms work only for assemblies in which all the parts can be removed one at a time. But this might not be the case in some assemblies. There might be cases where one has to remove two or three parts at a time before they could remove the next part. This can be done by identifying strongly connected components in the adjacency matrices and finding out the feasible assembly sequences. This will be a good research problem as no body addressed this issue till now. Also other optimization criteria such as number of tool changes and the number of gripper changes can be easily added into this algorithm. This guided genetic algorithm can have applications in other areas like job shop scheduling, assembly line balancing etc.
References


