On the automatic generation of product assembly sequences

C. K. CHOI†*, X. F. ZHA‡, T. L. NG§ and W. S. LAU¶

Automatic assembly planning is recognized as an important tool for achieving concurrent product and process development thus reducing manufacturing costs. It plays an important role in designing and planning the assembly systems. In order to recommend a good sequence of assembly operations, the process planner needs to generate all such feasible sequences. In this paper, the representation of the product assembly relationships and the assembly processes is described. Considering the assembly knowledge and the assembly constraints, a methodology is presented to generate all feasible assembly sequences by determining and decomposing the levelled feasible subassemblies. A prototype reasoning expert system is developed to generate all feasible assembly sequences automatically. The research findings are exemplified with a simple assembly to illustrate the method.

1. Introduction

The choice of the sequence in which components or subassemblies are put together in the mechanical assembly of a product can drastically affect the efficiency of the assembly process. For example, one feasible and reasonable sequence may require less fixturing, less changing of tools, and include simpler and more reliable operations than others. Therefore, assembly sequence planning plays an important role in designing and planning the product assembly process.

Traditionally, the product assembly sequence was planned by an experienced production engineer. However, the planning of assembly sequences is sometimes a trivial and error-prone task especially when there exists a large number of potential assembly sequences in a complex assembly. Therefore, there is a growing need to systematize and to computerize the generation of assembly sequences. So far, many research activities have focused on various aspects of assembly sequence planning such as assembly modelling, assembly sequence representation and assembly sequence generation algorithms.

In this paper, focuses are put on the approach and the system for automatic generation of assembly sequences, which are the key issues of computer aided assembly process planning (CAAPP). A new representation of the product assembly relationships and a methodology for generating the detailed feasible subassembly sets of assembly plans for a given product will be described. Based on the assembly knowledge or information and the method of determination and decomposition of feasible subassembly subsets, an automatic knowledge reasoning system has been
developed to generate the assembly sequences. The proposed approach is a combination of the ‘assembly’ approach and the ‘disassembly’ approach.

This paper is structured as follows: §§ 1 and 2 present the brief introduction and literature review about the assembly sequence generation and planning respectively. Section 3 deals with the modelling of assembly process that contains the representation of assembly relationships (liaison graph, liaison matrix, knowledge assembly liaison graph), predicate-based representation of assembly constraints, graph-based representation of the assembly or disassembly sequences. Sections 4 and 5 present a new approach to assembly sequence generation and the knowledge based automatic reasoning system; § 6 gives an example to illustrate the proposed approach; § 7 summarizes and gives some suggestions for the future research work.

2. Overview of previous work

2.1. Assembly sequence generation

Many researchers have made attempts to generate the assembly sequences of a product. Bourjault (1984) proposed a procedure to obtain all the precedence knowledge about the liaisons of an assembly by answering a set of structured questions based on this proposed liaison model of the assembly. Defazio and Whitney (1987) simplified Bourjault’s procedure and reduced the number of the asked questions to $2n$ against $2^n$ of Bourjault’s. These two methods study the assembly from the point of view of assembling the product. For the representation of precedence knowledge of an assembly, there have been several methodologies widely used in the past such as the set theory, binary matrix, directed graph, establishment condition, precedence relationships and function diagrams (Thaler 1989a,b, Homen de Mello and Sanderson 1990). However, these methods can only represent the partial assembly precedence knowledge. To represent the set of assembly sequences completely, Homen De Mello and Sanderson (1990, 1991a,b) proposed an AND/OR graph representation of all the possible configurations of the assembly and generate the assembly sequences of a product using a disassembly or decomposition method, based on the assumption that the disassembly sequence is the reverse of a feasible assembly sequence. Although this representation is complete, the resultant AND/OR graph is huge and the number of nodes grows exponentially as the part number in the assembly increases (Ben-Arieh and Kramer 1994). Based on a mathematical model of a product obtained through the definition of three matrices: the interference matrix, the contact matrix and the connection matrix, Dini and Santochi (1992) proposed a procedure for the detection and selection of the subassemblies and the assembly sequence of a product, which can be applied in a flexible assembly system. One of the main limitations of this model is that only partial disassemblies or assemblies are considered. In the component ordering method proposed by Lee and Ko (1987) a simple sequence was generated and interference checking ensures that the components to be assembled do not collide during assembly. A different method proposed by Lin and Chang (1991) uses a three layer strategy and a special tree structure to represent the assembly and to generate one feasible sequence. In order to obtain the detailed assembly plans, Ben-Arieh and Kramer (1994) presented a methodology and algorithms to consistently generate all feasible assembly sequences with consideration for the various combinations of subassembly operations. The algorithms are implemented using a LISP program. In addition, Ben Arieh (1994) applied a fuzzy set based method to evaluate the degree of difficulty of each assembly operation and then select the ‘best’ sequence of assembly opera-
tions. These methods and algorithms are less interactive and need much more space to store the representation of assembly sequences and processing time to process the assembly operations for the complex assemblies. Therefore, it is difficult to generate the detailed assembly plans automatically and to deal with the coordination and feasibility of various subassemblies efficiently.

2.2. Integrated assembly planning

Apart from the above mentioned, another important development offers an integrated approach to assembly planning by linking aspects of product design, design for assembly (DFA), assembly planning and production layout within a manufacturing enterprise in one procedure. All of them require some interactive input, which currently seems to be the only way of generating practical results. However, the aims of these few systems are quite different. One of these systems is the standard assembly analysis tool (STAAT). Incorporated into a larger CAD tool, this system can generate and evaluate the geometric assembly sequences of complex products from 20 to 40 parts and 500 to 1500 faces, and thus provides immediate feedback to a team of product designers about the complexity of assembling the product being designed. The researchers of STAAT are working now on extending both these results and the underlying theory to more sophisticated cases (Romney et al. 1995). However, the most comprehensive one is the project of integrated design and assembly planning (IDAP) (Seidel and Bullinger 1991, Seidel and Swift 1989, Richter 1991), which is a very long project with large scale interaction and is still under development. In this system, a model of OPNET (operation network) is constructed to reflect the constraints intrinsic in the product itself. An OPNET model is structured in the form of a graph that represents the relationships among tasks and subtasks for DFA evaluation and assembly process planning. An interactive operation network editor has been developed to integrate the procedures for network generation, modification, queries and evaluation under a uniform graphical man-machine interface (Seidel 1989, Seidel and Swift 1989).

3. Assembly process modelling

3.1. Notations and assumptions

Notations:

- Components of an assembly are represented as \( p_1, p_2, \ldots, p_n \).
- A subassembly is represented by a list of components such as \( [p_1, p_2, p_3, p_4] \).
- A subassembly operation is represented by a combination of two sub-assembly operations such as \( [p_1] [p_2, p_3, p_4] [p_1, p_2] [p_3, p_4] \) etc. The list should be read from left to right, i.e. the subassembly operation direction or sequence is from left to right, for example, \( [p_1] \rightarrow [p_2, p_3, p_4] \) and \( [p_1, p_2] \rightarrow [p_3, p_4] \).
- An assembly sequence is represented implicitly by nested lists of components. For example \( [[[p_1, p_2], [p_3, [p_4]], [p_5, p_6]]] \) represents the following sequence of assembly operations: \( p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_4 \rightarrow p_6 \).

Assumptions:

- For a disassembleable product, the assembly sequence is the reverse of the disassembly sequence if there is no destructive operation in the disassembly.
Each component is a solid rigid object, that is, there is no changing of shape during assembly and disassembly. Components are interconnected whenever they have one or more compatible surface in contact.

- Assembly is carried out step by step and components or subassemblies are assembled one at a time.
- Once an assembly operation is completed the subassembly would remain unchanged at all assembly stages.

3.2. Representation of assembly relationships

3.2.1. Liaison graph and its matrix representation

The liaison relationships between two components can be implemented by the total relative constraints which can be extracted from the CAD drawing and classified as fit contact, plane contact and meshing contact. The assembly liaison relationships of a product can be described by the liaison graph (LG), which is defined as:

\[ \text{Lg} = G(V, E) \]

where \( V \) is the set of vertices and \( E \) is the set of connecting arcs. Each vertex represents a component being assembled and each arc represents the liaison between two components (De Fazio and Whitney 1987, Lee 1989, Shin and Ho 1994). The assembly liaison relationships of a product can also be represented by liaison matrix \( \mathbf{R} \) (Dini and Santochi 1992), which is defined as:

\[
\mathbf{R} = \{r_{ij}\} \tag{1}
\]

where,

\[
r_{ij} = \begin{cases} 
1 & \text{with liaison relationship between } p_i \text{ and } p_j; \\
0 & \text{without liaison relationship between } p_i \text{ and } p_j.
\end{cases}
\]

The row of matrix \( \mathbf{R} \) represents the liaison relationships between a component and other components. The column of matrix \( \mathbf{R} \) represents the components connected by liaison relationships. The sub-matrices of \( \mathbf{R} \) represent the local liaison relationships in subassemblies.

3.2.2. Knowledge assembly liaison group (KALG)

The knowledge assembly liaison group (KALG) is an enhancement of the assembly liaison graph. It includes much more assembly information and constraints. In constructing the KALG, the following rules would be used:

1. Each component assembled (and not to be disassembled anymore) is a vertex of KALG.
2. Connecting components with the same type and functions should be regarded as one vertex.
3. If components \( p_i \) and \( p_j \) are adjacent to each other, there exists an arc connecting \( p_i \) and \( p_j \). If \( p_i \) is in contact with \( p_j \), the arc is expressed by a solid line. If constraints exist between \( p_i \) and \( p_j \), the arc is expressed by a dotted line.
4. The direction of an arc connecting two components or vertices (such as \( p_i \) and \( p_j \)) would be determined according to the following:
   - If \( p_i \) is a connecting component, the direction is from \( p_i \) to \( p_j \);
   - If \( p_i \) is inside \( p_j \), or inserted into \( p_j \), the direction is from \( p_i \) to \( p_j \);
   - If \( p_j \) is underneath \( p_i \), the direction is from \( p_i \) to \( p_j \);
If \( p_i \) is a base component and \( p_i \) is connected with \( p_j \) on the side, the direction is from \( p_i \) to \( p_j \);

In the KALG, this relationship can be described by a directed solid line \( p_i \rightarrow p_j \) or an oriented dotted line \( (p_i \rightarrow\rightarrow p_j) \). The assembly liaison relationships and KALG of a product can be generated from the geometric data available in a CAD system or from inquiries directed to the user.

### 3.3. Representation of assembly constraints

In an assembly procedure, there exist assembly constraints among different components. The constraints consist of hard constraints and soft constraints. The hard constraints are the geometrical and mechanical constraints, and soft constraints are made by assembly planners. In this paper, the following types of constraints are discussed:

#### 3.3.1. Topological constraints

An assembly can be represented by the topological structure of its components and the liaison relationships between components. It means that two components are topologically interconnected or at least one component connects with a component in the subassembly directly. The existence of topological constraints of a subassembly means the coherence of its local liaison graph, which is defined as there being at least one path composed by connecting arcs from an arbitrary component in a subassembly to any other components in the subassembly (Zha 1994). Figure 1 shows the local liaison graphs of an assembly composed of four components and their coherence.

Figure 1 (a), (b) and (c) are coherent, but Fig. 1 (d), (e) and (f) are not. Therefore this type of constraint can be easily identified from KALG and can also be explicitly described by the liaison sub-matrix. As shown in Table 1, the predicates such as \textit{connect}, \textit{connect1}, \textit{not_connect}, \textit{not_connect1} have been used to describe the topological constraint knowledge.

#### 3.3.2. Geometric constraints

In general, geometric constraints are referred to as the relative or allowable position and orientation relations between two components or subassemblies. They can be represented by predicates such as \textit{interference}, \textit{position}, and \textit{orientation}, as shown in Table 1. Sometimes the geometric constraints lead to assembly precedence (see § 3.3.3 below). The geometric constraints can be generated from the geometric data available in CAD system and KALG or from inquiries directed to the user.

#### 3.3.3. Partial precedence constraints

The precedence constraints specify that a component or a subassembly should be assembled in a desired direction or precedence. These constraints can be obtained from geometric and non-geometric information provided by the user or available in CAD database and KALG. Here, they are represented by predicates such as \textit{precede} (\textit{inside}, \textit{left}, \textit{right}, \textit{base}, \textit{beneath}, etc) as shown in Table 1. Therefore the partial precedence relationships have the properties:

- Chain properties:
  \[
  \text{precede}(p_1,p_2) \wedge \text{precede}(p_2,p_3) \wedge \text{precede}(p_3,p_4) \Rightarrow \text{precede}(p_1,p_4)
  \]
Commutative properties:
\[ \text{precede}(p_1, p_2) = \text{precede}(p_2, p_1); \]

Distributive properties:
\[ \text{precede}(p_1 \land p_2, p_3) = \text{precede}(p_1, p_3) \land \text{precede}(p_2, p_3); \]
\[ \text{precede}(p_3, p_1 \land p_2) = \text{precede}(p_3, p_1) \land \text{precede}(p_3, p_2); \]
\[ \text{precede}(p_1 \lor p_2, p_3) = \text{precede}(p_1, p_3) \lor \text{precede}(p_2, p_3); \]
\[ \text{precede}(p_3, p_1 \lor p_2) = \text{precede}(p_3, p_1) \lor \text{precede}(p_3, p_2); \]
where, \( \land \) = AND; \( \lor \) = OR.

3.3.4. Stability and security constraints

A subassembly is said to be stable if its components maintain their relative position and do not break contact spontaneously. The stability of subassembly contains gravity stability and anti-disturbance stability under the action of gravity, assembly stability under the action of assembly force and plastic stability under the action of inner spring force. The security of subassembly is defined as zero degrees of freedom for relative motion of all components in the subassembly.
The subassembly is said to be stable if it satisfies one of the following:

- The subassembly has fastening constraints;
- The components have tight or overfit mating;
- The centre of gravity of the subassembly falls within the supporting surface;
- Each component in subassembly has stability;

The subassembly has security if it satisfies one of the following:

- The subassembly is fastened;
- The components have no direct liaison with fasteners and have zero degrees of freedom of relative motion;
- The component has direct liaison with fasteners and has motion degree of freedom of fastening constraints only.
A subassembly without stability or security constraints means that it is unstable and changeable. So the stability and security constraints would be represented by predicates such as unstable and changeable as shown in Table 1.

3.3.5. Cost constraints

The cost of an assembly operation can be described by many indexes such as assembly time and assembly efficiency. So the cost constraint of an assembly operation can be described by predicates such as time-consuming and low_efficiency as shown in Table 1.

In general, a subassembly or a subassembly operation is considered to be unfeasible if the following conditions exist, such as without topological constraints, without stability or security constraint or with interference constraints. The precedence constraint is the key to determination of feasibility of the subassembly and the subassembly operation and the generation of component based and subassembly based sequence. However, the subassembly or the subassembly operation with cost constraints should also be considered in detail.

3.4. Graph-based representation of assembly

There are three main approaches to the assembly representation, namely language-based representation, graph-based representation, and advanced data structure representation with three different underlying goals (Ben-Arieh and Kramer 1994). The graph-based approach uses an information source such as a CAD database, or information supplied by the user. Its forms are numerous, which include directed graph, AND/OR graph (Homen de Mello and Sanderson 1990), connectivity graph (Shpitalni et al. 1989), Petri nets (Zhang 1989), hierarchical partial order graphs (Lee and Shin 1988), precedence diagram (De Fazio and Whitney 1987), liaison diagram (Bullinger and Ammer 1984), assembly constraint graph (Wolter 1989) and interference graph (De Florian and Nagy 1989).

In this paper, the AND/OR graph is used to represent the assembly sequence. When the AND/OR graph is applied to describe the product assembly or disassembly sequence, the vertex in the highest level \( n - 1 \) represents the product. The vertices in lower levels \( (n - 2, n - 3, \ldots, 1, 0) \) represent the feasible subassemblies. The edges represent the feasible assembly operations. Assembly sequence AND/OR graph represents the feasible subassemblies and the paths of disassembly or assembly. It can be described by a mathematical expression as:

\[
\{S_{f\text{op}}, S_{f} \}
\]

where \( S_{f\text{op}} \) represents the set of feasible subassembly operations, and \( S_{f} \) represents the set of feasible subassemblies.

If \( P \) represents the set of components, \( S_{sp}(P) \) and \( S_{fsp}(P) \) represents the set of components of subassembly and feasible subassembly respectively, and \( S(P) \) represents the total subsets of components, then the AND vertices can be described by:

\[
A_{\land} = \left( P_{k}, \{ P_{i} \setminus P_{j} \} \right), \quad P_{k} = P_{i} \lor P_{j}
\]

A feasible subassembly \( S_{f} \) can be represented by:

\[
S_{f} = \{ P \in S(P), S_{sp}(P), S_{fsp}(P) \}
\]
If a feasible subassembly composed of two kinds of component sets $P_i$ and $P_j$ is represented by $FS(P_i, P_j)$ and $PS(P_i, P_j) = P_i \cup P_j$ is the juxtaposed set of $P_i$ and $P_j$, then the set of feasible subassembly operations $S_{fsop}$ can be written as:

$$S_{fsop} = \left[ \left( P_k, \{P_i, P_j\} \right) \left( P_i, P_j, P_k \subseteq S(P) \right) \right] \land PS(P_i, P_j) \land FS(P_i, P_j) \quad (4)$$

An example of an assembly AND/OR graph is shown in Fig. 2.

4. Assembly sequence generation
4.1. Determination of levelled feasible subassemblies
4.1.1. Theoretical levelled subassembly configurations

From the assumption of assembling one component at a time, we can classify the whole assembly procedure of $n$-component product into $n-1$ levels of assemblies. Level 0 assembly is a primary subassembly each consists of a primitive component. Level $n-1$ assembly is the last subassembly which forms the product. Level $i-1$ ($i = 2, ..., n-1$) assembly is a subassembly consisting of at least two up to $n-1$ components. The number of theoretical subassembly configurations in level $i-1$ assembly is $C_n^i (i = 1, 2, ..., n)$ which is determined by compound principles in mathematics. All theoretical levelled subassembly configurations are shown in Tables 2 and 3.

4.1.2. Removal of levelled unfeasible subassemblies

In spite of the large number of theoretical sequences, only a few of them are feasible. The above mentioned types of constraints limit the number of feasible sequences. The key to determination of infeasibility of a subassembly is to identify all the constraint conditions and the corresponding predicates in assembly knowledge base (AKB) for all subassemblies. This can be implemented automatically by the method of the backtracking mechanism in artificial intelligence (AI).

4.1.3. Identification of levelled feasible subassemblies

Based on the constraint knowledge in AKB, all unfeasible subassemblies in level $i-1$ assembly can be eliminated. The feasible subassembly subsets of this level can then be formed. The number of feasible subassemblies in level $i-1$ can be expressed

![Figure 2. Example assembly AND/OR graph.](image-url)
Table 2. Theoretical no. of subassemblies.

<table>
<thead>
<tr>
<th>No. of components</th>
<th>Assembly levels</th>
<th>Total no. of subassemblies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0, 1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0, 1, 2</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>0, 1, 2, 3</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>0, 1, 2, 3, 4</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>0, 1, 2, 3, 4, 5</td>
<td>63</td>
</tr>
<tr>
<td>7</td>
<td>0, 1, 2, 3, 4, 5, 6</td>
<td>127</td>
</tr>
<tr>
<td>8</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
<td>255</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>0, 1, ..., n - 1</td>
<td>C_n + C_{n}^2 + ... + C_{n}^{n-1} = 2^n - 1</td>
</tr>
</tbody>
</table>

Table 3. Theoretical subassembly configurations of $n$-component product.

as

$$N_{ifs}^i = C_n^i - N_{infs}^i, \quad i = 1, 2, 3, \ldots, n$$

where $N_{ifs}^i$ is the number of feasible subassemblies in level $i - 1$;

$C_n^i$ is the number of theoretical assembly configurations in level $i - 1$; $N_{infs}^i$ is the number of unfeasible subassemblies in level $i - 1$.

The set of feasible subassemblies in level $i - 1$ can be determined by the following formula

$$\{ S_{ifs}^i \} = \{ S_{ts}^i \} - \{ S_{infs}^i \}, \quad i = 1, 2, 3, \ldots, n$$

where $\{ S_{ifs}^i \}$ is a set of feasible subassemblies in level $i - 1$, $S_{ts}^i$ is a set of theoretical subassembly configurations in level $i - 1$, and $\{ S_{infs}^i \}$ is a set of total unfeasible subassemblies in level $i - 1$.

4.1.4. Decomposition of feasible subassemblies

A feasible subassembly in level $i - 1$ assembly is likely to be decomposed into many corresponding combinations of the feasible subassemblies in lower level $j$ ($j = 0, 1, 2, \ldots, i - 2$), which are defined as subassembly operations here. For
example, the levelled feasible subassemblies and the subassembly operations of a four component product are shown in Table 4.

### 4.2. Generation of assembly sequences

After the feasible subassemblies and their subassembly operations are determined, the assembly sequences can be easily generated by AND/OR logical relations and their graph representation. As the stability and security constraints and cost constraints are introduced in the formation of feasible subassembly subsets, the evaluation and choice of assembly sequences can be accomplished by these constraints.

### 5. Automatic reasoning

In §4, all assembly or disassembly sequences of a product can be generated by determining the levelled feasible subassemblies and their decompositions into corresponding subassembly operations. However, this method needs much more reasoning experience or intervention and the user has to verify many feasible sequences (n! maximum for an n-component product). In order to generate assembly or disassembly sequences automatically, an automatic knowledge reasoning system would be required to solve this problem based on principles of artificial intelligence (AI). A prototype assembly planning expert system (APES) has been developed for determining and decomposing the feasible subassemblies and generating the assembly sequence and the assembly sequence graph. The system is written in artificial intelligence language Visual Prolog 4.0. The procedure flow chart of the system is shown in Fig. 3.

<table>
<thead>
<tr>
<th>Level</th>
<th>Subassembly</th>
<th>Subassembly operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$p_1, p_2, p_3, p_4$</td>
<td>$p_4$, $p_1, p_2, p_3$, or $p_1, p_2, p_3, p_4$, or $p_1, p_2, p_3, p_4$</td>
</tr>
<tr>
<td>2</td>
<td>$p_1, p_2, p_3$</td>
<td>$p_3$, $p_1, p_2$, or $p_4$, $p_1, p_2, p_3$, or $p_1, p_2, p_3, p_4$, or $p_1, p_2, p_3, p_4, p_1$</td>
</tr>
<tr>
<td>1</td>
<td>$p_1, p_2$</td>
<td>$p_2$, $p_1$, or $p_2, p_1$, $p_1, p_2, p_3$, or $p_2, p_1, p_2, p_3$, or $p_1, p_2, p_3, p_4$, or $p_2, p_1, p_2, p_3, p_4$</td>
</tr>
<tr>
<td>0</td>
<td>$p_1, p_2, p_3, p_4$</td>
<td>$p_1, p_2, p_3, p_4$</td>
</tr>
</tbody>
</table>

Table 4. Example of subassembly and subassembly operation.
Figure 3. Program flow chart.
The key features of the knowledge based reasoning system are a knowledge base composed of assembly knowledge and a reasoning mechanism formed by the inferring rules, which decide how to utilize the knowledge base. The assembly knowledge base (AKB) can be constructed by many predicate logical clauses. To construct inference rules, the predicates such as test\_constraint, exist and not\_exist are used to check whether the subassemblies and subassembly operations are feasible. In addition, there are rules adopted in the procedure of automatic inference, where, ‘\(\iff\)’ means ‘equivalent’ and ‘\(\Rightarrow\)’ means ‘implies’, as follows:

- Equivalent rule: A subassembly is a compound of at least one component. As a result, it is not associated with the location of components in the subassembly representation. For example,

\[
[p_1, p_2, p_3, p_4] \iff [p_1, p_2, p_4, p_3] \iff [p_1, p_4, p_3, p_2]
\]

- Topological constraint existence rule: Based on the above mentioned, the local liaison graphs for a subassembly with topological constraints must be coherent.

- Subassembly operation rule: If a subassembly is unfeasible then all its subassembly operations decomposed are not unfeasible.

- Superset rule: If there is no mating between two components or subassemblies due to interference in the approach path, then adding a component to either a component or a subassembly, which is not associated with the mating liaisons, will not change this situation. For example, not\_connect\(\ (p_1, p_2) \land \) interference\(\ (p_1, p_2) \land \) not\_connect\(\ (p_3, p_2) \Rightarrow \) not\_connect\(\ [p_1, p_3][p_2].\)

- Subset rule: If a mating can occur between two subassemblies, then removing a component from either a subassembly, which is not associated with the mating liaison(s), will not change this situation. For example, connect\(\ [p_1, p_2][p_3, p_4] \land \) connect\(\ (p_2, p_3) \land \) not\_connect\(\ (p_1, p_3) \Rightarrow \) connect\(\ [p_2][p_3, p_4].\)

6. Example

In this paper, a four-component assembly, which was described by Homen de Mello and Sanderson (1990) is taken as an example to illustrate the feasibility and efficiency of the proposed approach and the reasoning system. Figure 4 (a) describes the example model product composed of four components namely cap (1), stick (2), receptacle (3) and handle (4) respectively. Its KALG is shown in Fig. 4 (b). Based on the above mentioned approach and automatic reasoning system, the assembly sequences can be generated as follows:

(1) Theoretical subassembly configurations:

- level 0: \([\begin{array}{c}1 \\ 2 \\ 3 \\ 4 \end{array}])
- level 1: \([\begin{array}{c}1,2 \\ 1,3 \\ 1,4 \\ 2,3 \\ 2,4 \\ 3,4 \end{array}])
- level 2: \([\begin{array}{c}1,2,3 \\ 1,2,4 \\ 2,3,4 \end{array}])
- level 3: \([\begin{array}{c}1,2,3,4 \end{array}])

(2) Subassemblies with constraints:

(a) Topological constraints

not\_connect\(\ (\begin{array}{c}1 \\ 4 \end{array}) \) or not\_connect\(\ (\begin{array}{c}4 \\ 1 \end{array}) \Rightarrow \) not\_exist\(\ (\begin{array}{c}1,4 \end{array})\)

(b) interference\(\ (\begin{array}{c}1 \\ 4 \end{array}) \) or interference\(\ (\begin{array}{c}3 \\ 4 \end{array}) \Rightarrow \) not\_exist\(\ (\begin{array}{c}1,4 \end{array})\)
Figure 4(a). Example product modelling.

Figure 4(b). Example product KALG.

(c) Precedence constraints

\[ \text{not_exist}([1,2,4]) \rightarrow \text{not_exist}([1][2,4]) \text{ and } \text{not_exist}([2][1,4]) \text{ and} \]

\[ \text{not_exist}([3]) \rightarrow \text{not_exist}([1][3,4]) \text{ and } \text{not_exist}([3][1,4]) \text{ and} \]

\[ \text{before}([2][1,3,4]) \text{ and } \text{inside}([2][5]) \rightarrow \text{not_exist}([1,3,4]) \]
(d) Stability and security constraints

\[
\text{unstable}(1, 4) \rightarrow \text{not exist}(1, 4) \quad \text{and} \\
\text{unstable}(1, 2, 4) \rightarrow \text{not exist}(1, 2, 4)
\]

(e) Cost constraints

\[
\text{time consuming}([3, 1, 2, 4]) \quad \text{and} \\
\text{time consuming}([3, 1, 2, 4])
\]

(3) Unfeasible subassemblies:

\[
[1, 4] [1, 2, 4] [1, 3, 4]
\]

(4) Feasible subassemblies and their decompositions:

The reasoning system first searches and outputs the feasible subsets and then decomposes them in different level assemblies as follows:

Level 0: \[
[1] [2] [3] [4]
\]

Level 1: \[
[1, 2] \Rightarrow [1, 2] \text{ or } [2, 1] \\
[1, 3] \Rightarrow [1, 3] \text{ or } [3, 1] \\
[2, 3] \Rightarrow [2, 3] \text{ or } [3, 2] \\
[2, 4] \Rightarrow [2, 4] \text{ or } [4, 2] \\
[3, 4] \Rightarrow [3, 4] \text{ or } [4, 3]
\]

Level 2: \[
[1, 2, 3] \Rightarrow [1, 2, 3] \text{ and } [2, 1, 3] \text{ or } [3, 1, 2] \\
[2, 1, 3] \Rightarrow [2, 1, 3] \text{ or } [3, 2, 1] \\
[1, 3, 2] \Rightarrow [1, 3, 2] \text{ or } [2, 3, 1] \\
[2, 3, 4] \Rightarrow [2, 3, 4] \text{ or } [4, 2, 3] \text{ and } [3, 2, 4] \text{ or } [4, 3, 2]
\]

Level 3: \[
[1, 2, 3, 4] \Rightarrow [1, 2, 3, 4] \text{ or } [2, 3, 4, 1] \text{ and } [1, 2, 3] \text{ or } [1, 2, 3]
\]

(5) AND/OR graph and feasible assembly sequences: The system outputs the following 12 feasible assembly sequences:

1-2-3-4, 1-3-2-4, 3-4-2-1, 2-1-3-4, 3-1-2-4, 4-3-2-1
2-3-1-4, 2-3-4-1, 2-4-3-1, 3-2-1-4, 3-2-4-1, 4-2-3-1

If cap (4) or handle (2) is considered as the base part of the assembly, the following precedence constraints must be added:

\[
\text{base}(1, [2, 3, 4]) \quad \text{or} \quad \text{base}(4, [1, 2, 3])
\]

So, four assembly sequences are remaining as follows:

1-2-3-4, 1-3-2-4, 4-3-2-1, 4-2-3-1

If cost constraints are described as above then in the end the two assembly sequences are:

1-3-2-4, 4-3-2-1

The assembly sequence AND/OR graph is shown in Fig. 4.
7. Conclusions

The application results show that the representation for product assembly process presented in this paper is efficient and the combination method for determination of feasible subassembly based on the principles of compound and classification and decomposition of feasible subassembly in assembly knowledge reasoning is useful in assembly sequence planning. A prototype knowledge reasoning system (APES) is developed in the Visual prolog 4.0 programming language to generate the assembly sequences automatically. This system is convenient to use and makes the speed of reasoning fast and the assembly sequences to be selected less subjective. It will find wide applications in knowledge based assembly planning systems. Further research work is required to enhance its interactivity and the integration with DFA and CAD systems, especially those about assembly representation (e.g. assembly constraints) that are based on assembly features and predicates. Also, the approach and system for concurrent design and planning for assembly in an integrated environment is expected to be developed. The authors are working now on these problems under the support of a project entitled concurrent integrated design and planning for flexible assembly.

Acknowledgments

The second author would like to thank the National Natural Science Foundation of China and ‘863’ High-tech. R&D Programme of China for their support for this research under grant No. 59485005.

References


