Automatic Generation of Mechanical Assembly Sequences

L.S. Homem de Mello and A.C. Sanderson*

CMU-RI-TR-88-19

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

December 1988

*Current Address: Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY 12180-3590.

Table of Contents

List of Figures

List of Tables

Abstract

This paper presents an algorithm for the generation of mechanical assembly sequences and a proof of its correctness and completeness. The algorithm employs a relational model of assemblies. In addition to the geometry of the assembly, this model includes a representation of the attachments that bind one part to another. The problem of generating the assembly sequences is transformed into the problem of generating *disassembly* sequences in which the *disassembly* tasks are the inverse of feasible assembly tasks. This transformation leads to a decomposition approach in which the problem of disassembling one assembly is decomposed into distinct subproblems, each being to *disassemble* one subassembly. It is assumed that exactly two parts or subassemblies are joined at each time, and that whenever parts are joined forming a subassembly all contacts between the parts in that subassembly are established. The algorithm returns the AND/OR graph representation of assembly sequences. The correctness of the algorithm is based on the assumption that it is always possible to decide correctly whether two subassemblies can be joined, based on geometrical and physical criteria. This paper presents an approach to compute this decision. An experimental implementation for the class of products made up of polyhedral and cylindrical parts having planar or cylindrical contacts among themselves is described. Bounds for the amount of computation involved are presented.

1. Introduction

The choice of the sequence in which parts or subassemblies are put together in the mechanical assembly of a product can drastically affect the efficiency of the assembly process. For example, one sequence may require less fixturing, less changing of tools, and include simpler and more reliable operations than others. The choice of the assembly sequence is usually made by a human expert. In the case of manufacturing, the choice is typically made by an industrial engineer. In the case of repair, the choice is made by the maintenance personnel. No clear systematic procedure seems to be followed in either case. Humans seem to use common sense and past experience blended in a fuzzy, sometimes inconsistent, and not well understood way.

There is a growing need to systematize and to computerize the generation of assembly sequences for several reasons:

- Although many experienced industrial engineers have a knack for devising efficient ways to assemble a given product, systematic procedures are needed to guarantee that no good assembly sequence has been overlooked. For complex products, the number of feasible assembly sequences may be so large that even skillful engineers may fail to notice many possibilities. The availability of a systematic procedure that is proven correct and complete will guarantee that all feasible sequences and only the feasible sequences will be generated.
- The planning and programming chores in manufacturing are time consuming and error-prone. For small batches of production, the cost of planning and programming can weigh heavily in the total production cost. Moreover, the time spent in planning and programming may excessively delay the actual production. The automation of these chores will expedite their execution, reduce their cost, and improve their quality. Systematic procedures are needed in order to facilitate the automation of planning and programming of assembly systems.
- In simultaneous engineering environments, the automation of sequence planning will help the designer to assess the assembly process requirements of different design solutions for a given product. For some products, small changes in the design can have a large impact on the assembly alternatives.
- Autonomous systems for applications such as space or deep sea exploration will need the ability to generate assembly or disassembly sequences that fit the particular situation they encounter. It is virtually impossible to preprogram all possible situations those systems might face, particularly if execution errors can occur and the systems are expected to recover autonomously.
- In less structured, more dynamic manufacturing systems or facilities there is a need to adapt the assembly process to different machines. The need to produce different products in the same shop may lead to the choice of an assembly sequence for a product that may not be the most efficient but uses the idle equipment in the shop. Knowledge of all assembly sequence options of each product is needed in order to optimize the overall use of machines and tools. Similarly, when the same product is assembled in different shops, the knowledge of all assembly sequences is needed in the selection of the sequence more suitable to the equipment available in each shop.

This paper presents an algorithm for the generation of mechanical assembly sequences and a proof of its correctness and completeness. The algorithm takes a description of the product and returns the corresponding AND/OR graph representation of assembly sequences [17]. It is assumed that exactly two parts or subassemblies are joined at each time, and that after parts have been put together they remain together. It is also assumed that whenever parts are joined forming a subassembly, all contacts between the parts in that subassembly are established. These assumptions are consistent with the trend towards product designs that are suitable for automatic assembly [2, 5].

The correctness of the algorithm is based on the assumption that it is always possible to decide correctly whether two subassemblies can be joined, based on geometrical and physical criteria. This paper presents an approach to compute this decision. An experimental implementation for the class of products made up of polyhedral and cylindrical parts having planar or cylindrical contacts among themselves is described.

The amount of computation involved in generating the AND/OR graph representation of assembly plans depends on the number of parts that make up the product, on how those parts are interconnected, and also on the resulting AND/OR graph. Bounds for the amount of computation involved are presented.

2. Background

The algorithm presented in this paper takes as input a representation of the product, and generates the set of all feasible assembly sequences which is represented as an AND/OR graph. This section reviews previous work on modeling assemblies, on representing assembly sequences, and on generating assembly sequences.

2.1. Modeling assemblies

The research on high level languages for robotic assembly has explored the use of assembly models. That research aimed at the automatic generation of the actions that a robot should perform in order to assemble a product. Typically the sequence in which parts should be put together was given.

One of the earliest works on robot programming was the RAPT [1] system in which bodies were described in terms of their features such as planar faces, shafts, and holes. The spatial relationships between parts were described by triples $\langle type\text{-}of\text{-}spatial\text{-}relation\ , feature.1\ , feature.2 \rangle$. For example, $\langle fits\ , S_i\ , H_j \rangle$ describes the spatial relationship between the shaft S_i and the hole H_j. The set of spatial relations between parts was input to an inference engine, and the relative positions of parts or their degrees of freedom were determined. Later extensions to RAPT [28, 29] allowed the user to describe assemblies not only by the spatial relationships between the parts but also by the actions required to bring those parts together.

Taylor [37] developed a representation of assemblies based on *attribute graphs*. The nodes in these graphs correspond to either objects, or features of objects. Entities that have volume such as assemblies and parts are objects, whereas entities that do not have volume such as surfaces and edges are features. Each link in the graph associates one node either to another node or to a link. For example, a subpart link may associate a part, which is an object node, to an assembly, which is another object node; and a nominal-transformation link may associate a feature node containing a 4×4 homogeneous coordinate transform matrix to a subpart link. The information describing the shape of an object is contained either in the node corresponding to that object, if the shape is simple, or in the nodes corresponding to its subparts, if the shape is complex. In the latter case, which is the case of assemblies, the composition of the subpart's shapes may be described either by homogeneous transform feature nodes associated to the subpart links, or by associations of features of subparts corresponding to spatial relationships between those features. Taylor allows redundancy of shape description and both types of descriptions for the compositions of shapes may coexist.

In the AUTOPASS system [39], the representation of assemblies was based on a graph structure in which each node represented a volumetric entity, either a part, or a sub-part, or an assembly, and the edges were directed and labeled to indicate four kinds of relationships: part-of, attachment, constraint, and assembly-component. The nodes had attributes which included the volumetric description and the location of the corresponding object. The part-of relationship induced a tree structure on the assembly model.

Unlike the work described above, which aimed at high level languages for robotic assembly, the work of Bourjault [6] aimed at modeling the assembly process. Towards that goal he used two types of graphs to represent products. The *graph of contacts* ("graphe de liaisons mécaniques" [6]) contains one node for each part in the assembly, and one edge for each contact between two parts. Since the same pair of parts may have more than one contact, the graph of contacts is not necessarily simple. From the graph of contacts, Bourjault defined the *graph of connections* ("graphe de liaisons fonctionelles" [6]) which has one node for each part in the assembly, and one edge for each pair of parts that have at least one contact. By definition, the graph of connections is always a simple graph.

The model of assemblies presented in section 3 is similar to the attributed graph used previously [37, 39] but extended to incorporate the attachment of contacts. This extension is needed to make possible the reasoning about the feasibility of assembly tasks.

2.2. Representation of assembly sequences

One assembly sequence can be represented by an ordered list of tasks; therefore it is possible to represent the set of all assembly sequences by a set of lists, each corresponding to a different assembly sequence. Since many assembly sequences share common subsequences, attempts have been made to create more compact representations.

One early attempt was the use of a set of tasks and a set of precedence constraints relating two tasks [15]. But as discussed elsewhere [17], there are products for which standard precedence constraints cannot encompass all sequences.

Directed graphs of assembly states can explicitly encompass the set of all assembly sequences. The nodes in these graphs may be either a partition of the set of parts [18], or a subset of connections of pairs of parts [6, 11]. Figure 1 shows a directed graph of assembly states for a three part product. The nodes in figure 1 are labeled by the partitions of the set of parts containing the subsets of parts of each subassembly already assembled at each state of the assembly process. Lower and upper bounds for the size of these graphs as a function of the number of parts in the product are presented elsewhere [18].

AND/OR [17] graphs of subassemblies can also encompass the set of all assembly sequences. The nodes in these AND/OR graphs correspond to subassemblies and the hyperarcs correspond to assembly tasks in which two subassemblies are joined to yield a larger more complex subassembly. The hyperarcs point from the node corresponding to the larger subassembly to the nodes corresponding to the smaller subassemblies. Figure 2 shows the AND/OR graph of subassemblies for a three-part product. The nodes in figure 2 are labeled by the set of parts that make up their corresponding subassemblies.

Although for three-part assemblies the AND/OR graph has more nodes than the directed graph of assembly states, for assemblies with large number of parts the AND/OR graph has substantially fewer nodes than the directed graph of assembly states. Moreover, the AND/OR graph of subassemblies shows explicitly the possibility of parallel execution of assembly tasks. Lower and upper bounds for the size of these AND/OR graphs as a function of the number of parts in the product are presented elsewhere [18].

Bourjault [6] showed that a set of logical expressions can be used to encode the directed graph of assembly states. For a product that has L connections between pairs of parts, Bourjault represented each state in the directed graph of assembly states by a binary vector $\underline{e} = [\lambda_1, \lambda_2, \cdots, \lambda_L]$ in which the i^{th} component is true or false respectively if the i^{th} connection is established in that state or not. Let S_i be the set of states from which the i^{th} connection can be established without precluding the completion of the assembly. Clearly, if S_i has K elements, each element satisfies

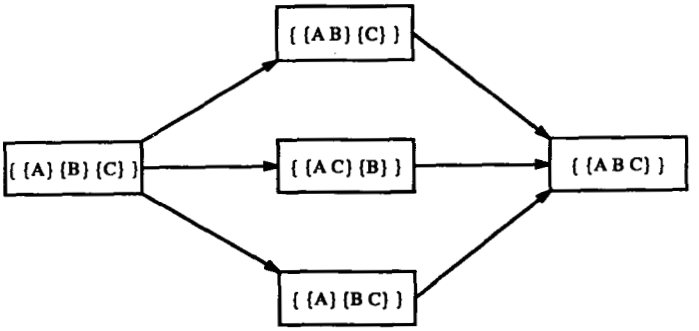


Figure 1: The directed graph of assembly states of a three-part assembly

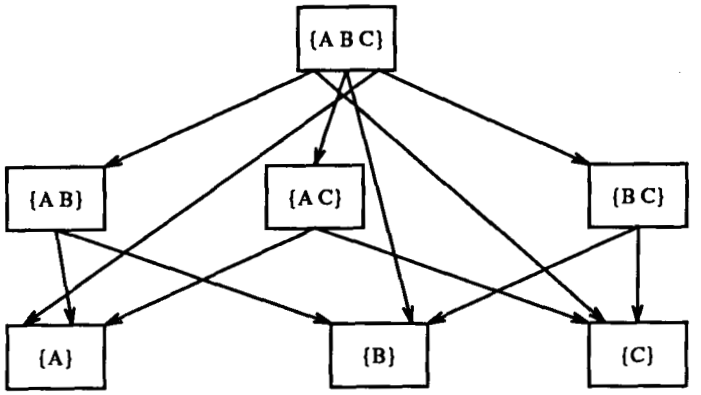


Figure 2: The AND/OR graph for a three-part assembly

$$\sum_{k=1}^{K} \prod_{l=1}^{L} \gamma_{kl} = \text{true}$$

where both the sum and the product are logical operations, and γ_{kl} is either the symbol λ_l if the k^{th} element in S_i corresponds to a state in which the l^{th} connection has been established, or the symbol $\overline{\lambda_l}$ if the k^{th} element in S_i corresponds to a state in which the l^{th} connection has not been established. Bourjault calls the left side of the equation above *the establishment condition for the i^{th} connection* ("conditions de réalisabilité" [6]) and he represents it by C_i. It is often possible to simplify the expression of C_i using the rules of boolean algebra. For some products, the simplified expressions are very short. By knowing the establishment conditions for all connections of a product, as well as the product's graph of connections, one can reconstruct the directed graph of assembly states.

2.3. Generation of assembly plans

Planning has been an important research topic in artificial intelligence, and the AI approach has dominated much of the research in robot task planning using domain-independent methods. The central idea of domain independent planning is to have one general purpose inference engine which can be used for any domain by describing the initial state, the goal, and the operators in a logic formalism. But domain-independent planners have serious limitations that preclude their use in generating assembly sequences based on a description of the product. Chapman [10] reviews the literature on domain-independent planning and discusses their main limitations.

Bourjault [6] has explored ways to obtain the establishment conditions C_i without enumerating all the states in the directed graph of assembly states. For example, he noticed that an affirmative answer to the question

- Is it true that the i^{th} connection cannot be established if the j^{th} connection has already been established?

means that no C will contain expressions including $\overline{\lambda_i}\,\lambda_j$, and that C_j will not contain expressions including $\overline{\lambda_i}\,\overline{\lambda_j}$, unless the i^{th} and the j^{th} connections can be established simultaneously. Bourjault's method uses a cleverly chosen sequence of questions that can in many cases expedite the obtainment of the establishment conditions for all connections.

De Fazio and Whitney [11] proposed a set of questions smaller than that used by Bourjault. The user of their method must first draw the graph of connections corresponding to the assembly, and then answer a pair of questions for each connection. For the i^{th} connection, the questions are:

1. what connections must be done prior to doing the i^{th} connection?
2. what connections must be left to be done after doing the i^{th} connection?

The answers to these questions should be expressed in the form of precedence relationships between connections or between logical combinations of connections. For example, the answers to the two questions for the i^{th} connection C_i could be:

1. $(C_j \text{ or } (C_k \text{ and } C_m)) \rightarrow C_i$
2. $C_i \rightarrow (C_s \text{ or } (C_t \text{ and } C_u))$

The symbol "$\rightarrow$" reads *must precede*, and C_j, C_k, C_m, C_s, C_t, and C_u are other connections between parts of the assembly. Once the precedence relationships have been generated, a computer program can generate the assembly sequences. Lui [25] describes a program that generates the assembly sequences based on the precedence relationships and on the graph of connections.

Both of these approaches [6, 11] lend themselves to interactive systems in which a computer program generates the questions, a human expert supplies the answers, and the program then generates the precedence relationships between connections or between logical combinations of connections. For simple cases, these approaches have the advantage that they exploit the engineer's intuitive understanding of parts relations and feasibility of operations. For complex cases, it may be very difficult for a human expert to answer the questions and to guarantee the correctness of the answers. And even assuming that the questions are answered correctly, proofs of correctness and completeness of the algorithms are needed to guarantee that the resulting precedence relations are satisfied by all the feasible assembly sequences and only by the feasible assembly sequences. Neither Bourjault nor De Fazio and Whitney have formally proven the correctness and completeness of their algorithms.

Furthermore, it seems very difficult to develop computer programs that will answer the questions in either method directly from a description of the assembly; any system based on these approaches will need the human expert to supply the answers. In the cases in which precedence relationships, together with the assembly's graph of connections provide a useful representation of assembly sequences, an alternative to have the questions answered by a human expert is to have them answered by a program that takes as input the set of assembly sequences generated by the algorithm presented in this paper.

3. A Relational Model for Assemblies

A mechanical assembly is a composition of parts interconnected forming a stable unit. Each part is a solid object. Parts are interconnected whenever they have one or more surfaces in contact. Surface contacts between parts reduce the degrees of freedom for relative motion. A cylindrical contact, for example, prevents any relative motion that is not a translation along the axis or a rotation around the axis. Attachments may act on surface contacts and eliminate

all degrees of freedom for relative motion. For example, if a cylindrical contact has a pressure-fit attachment, then no relative motion between the parts is possible.

The representations of products developed for high level robot programming languages emphasized the geometric aspects such as the shape of the parts and the contacts between parts. That emphasis is consistent with the goal of generating a sequence of robot actions that will join two subassemblies, given the sequence in which parts or subassemblies should be put together. However for the generation of the assembly sequences, a purely geometric description of the product is not sufficient. There are sequences that would be feasible from a geometric point of view, but are unfeasible in practice due to forces resulting from fasteners. Therefore, a model of assemblies to be used in generating assembly sequences must represent explicitly the fastenings that bind one part to another.

The representation of assemblies used by the algorithms described in sections 4 and 5 is a relational model that includes three types of entities: parts, contacts, and attachments. It also includes a set of relationships between entities. Both entities and relationships can have attributes. Formally, the relational model of an assembly is a 5-tuple $\langle P , C , A , R , a\text{–}functions \rangle$ in which

- P is a set of symbols, each of which corresponds to one part in the assembly. No two elements of P correspond to the same part.
- C is a set of symbols, each of which corresponds to a contact between surfaces of two parts of the assembly. No two elements of C correspond to the same contact. The two surfaces must be compatible. An example of a pair of compatible surfaces are a cylindrical shaft and a cylindrical hole. The same pair of parts may have more than one contact. And the same surface of one part may be in contact with surfaces of two or more other parts.
- A is a set of symbols, each of which corresponds to an attachment that acts on a set of contacts. No two elements of A correspond to the same attachment. An attachment always has an agent, which can be either the attached contact, or another contact, or a part. The access to an attachment may be blocked by one or more parts.
- R is a set of symbols, each of which corresponds to a relationship between pairs of elements of $P \cup C \cup A$. No two elements of R correspond to the same relationship.
- *a–functions* is a set of attribute functions[2] whose domains are subsets of $P \cup C \cup A \cup R$. These functions associate entities or relationships to their characteristics such as the type of attachment, the entities related by a relationship, and the shape of a part.

This definition of a relational model representation of assemblies is sufficiently general to encompass a large class of assemblies. The set of functions can be enlarged to include all the information that might be necessary to generate assembly sequences. In practice, it may be convenient to restrict the class of assemblies represented. Our current experimental implementation has the following restrictions:

- The contacts between parts involve one of the following pairs of compatible surfaces:
  - planar surface and another planar surface,
  - cylindrical shaft and cylindrical hole,
  - polyhedral shaft and polyhedral hole,
  - threaded cylindrical shaft and threaded cylindrical hole.

[2]A function is defined as a subset of the cartesian product of two sets (the domain and the range) that has no two pairs whose first elements are the same, and such that every element in the domain appears in one pair.

- The types of attachments are:
  - glue attachment,
  - pressure fit attachment,
  - clip attachment,
  - screw attachment.
- The attribute functions are the following:
  - The function that associates a part to a description of its shape:

    $shape : P \rightarrow S$

    where S is the set of all shape descriptions.
  - The function that associates a part to a description of its location:

    $location : P \rightarrow T$

    where T is the set of all 4×4 homogeneous transformation matrices. The matrix T_i associated to part p_i, corresponds to the position and orientation of a reference frame attached to part p_i with respect to a global frame of reference for the whole assembly. The choice of this global frame of reference is arbitrary, but the same global reference must be used for all parts.
  - The function that associates a contact to its type:

    $type\text{–}of\text{–}contact : C \rightarrow contact\text{–}types$

    where $contact\text{–}types = \{$ planar , cylindrical , slot , threaded-cylindrical $\}$.
  - The function that associates a planar contact to the coordinates, with respect to the assembly's global frame of reference, of a vector normal to the contact plane

    $normal : \{ c \mid [c \in C] \wedge [type\text{–}of\text{–}contact(c) = \text{planar}] \} \rightarrow \mathbf{R}^3$
  - The function that associates a planar contact to the part-relationship that relates the contact to the part that is back of the contact:

    $back : \{ c \mid [c \in C] \wedge [type\text{–}of\text{–}contact(c) = \text{planar}] \} \rightarrow R$

    This function must be consistent with the function *normal*.
  - The function that associates a planar contact to the part-relationship that relates the contact to the part that is forward of the contact:

    $forward : \{ c \mid [c \in C] \wedge [type\text{–}of\text{–}contact(c) = \text{planar}] \} \rightarrow R$

    This function must be consistent with the function *normal*.
  - The function that associates a cylindrical, slot, or threaded-cylindrical contact to the coordinates, with respect to the assembly's global frame of reference, of the line of the axis of both the hole and the shaft.

    $axis : \{ c \mid [c \in C] \wedge$
    $[type\text{–}of\text{–}contact(c) \in \{ \text{cylindrical}, \text{slot}, \text{threaded-cylindrical} \}] \} \rightarrow \mathbf{R}^3 \times \mathbf{R}^3$
  - The function that associates an attachment to its type:

    $type\text{–}of\text{–}attachment : A \rightarrow attachment\text{–}types$

    where $attachment\text{–}types = \{$ clip , pressure , screw , glue $\}$.
  - The function that associates a relationship to its type:

    $type\text{–}of\text{–}relationship : R \rightarrow relationship\text{–}types$

    where $relationship\text{–}types = \{$ part-contact, target-attachment, agent-attachment, blocking-part-attachment $\}$

- The function that associates a part or a contact to its part-contact relationships:

  $part\text{–}contact\text{–}relationships: P \cup C \rightarrow \Pi(R)$

  where $\Pi(R)$ is the set of all subsets of R.
- The function that associates a part-contact relationship to its part:

  $part: \{ r \mid [r \in R] \wedge [type\text{–}of\text{–}relationship(r) = \text{part-contact}] \} \rightarrow P$
- The function that associates a part-contact relationship to its contact:

  $contact: \{ r \mid [r \in R] \wedge [type\text{–}of\text{–}relationship(r) = \text{part-contact}] \} \rightarrow C$
- The function that associates an attachment or a contact to its target-attachment relationships:

  $target\text{–}attachment\text{–}relationships: C \cup A \rightarrow \Pi(R)$
- The function that associates an attachment, a contact or a part to its agent-attachment relationships:

  $agent\text{–}attachment\text{–}relationships: P \cup C \cup A \rightarrow \Pi(R)$
- The function that associates a target-attachment relationship to its contact

  $target: \{ r \mid [r \in R] \wedge [type\text{–}of\text{–}relationship(r) = \text{target-attachment}] \} \rightarrow C$
- The function that associates an agent-attachment relationship to its agent

  $agent: \{ r \mid [r \in R] \wedge [type\text{–}of\text{–}relationship(r) = \text{agent-attachment}] \} \rightarrow P \cup C$
- The function that associates a blocking-part-attachment relationship to its blocking-part

  $blocking\text{–}part: \{ r \mid [r \in R] \wedge$
  $\wedge [type\text{–}of\text{–}relationship(r) = \text{blocking-part-attachment}] \} \rightarrow P$
- The function that associates a target-attachment, a blocking-part-attachment, or an agent-attachment relationship to its attachment

  $attachment: \{ r \mid [r \in R] \wedge [type\text{–}of\text{–}relationship(r) \in B] \} \rightarrow A$

  with $B = \{$ target-attachment, blocking-part-attachment, agent-attachment $\}$.

The relational model of an assembly must be consistent. For example, if $part(r_1) = p_1$ and $contact(r_1) = c_1$ then $r_1 \in part\text{–}contact\text{–}relationships(p_1)$ and $r_1 \in part\text{–}contact\text{–}relationships(p_1)$ must hold. Furthermore, the relational model of an assembly must satisfy some syntactic constraints, the most important of which are:

- every contact must have exactly two part-contact relationships;
- every part must have at least one part-contact relationship, except in the case the assembly has only one part;
- every attachment must have at least one target-attachment relationship, and at least one agent-attachment relationship.

The relational model of an assembly can be represented by a graph plus the associated attribute functions. Figure 3 shows a simple product, and figure 4 shows its corresponding relational model graph.

The nodes in figure 4 correspond to the entities. Nodes corresponding to part entities are rectangles, nodes corresponding to contact entities are circles, and nodes corresponding to attachment entities are triangles. All nodes contain labels indicating their corresponding entities. The attribute functions associated with the contact entities are shown in Table 1.

The labeled lines connecting two nodes in figure 4 correspond to the relationships. Except for R5, R6, R13, and R14, all relationships are part-contact. Relationships R5 and R13 are target-attachment; they indicate that the contacts C2 and C5, respectively, are attached. Relationships R6 and R14 are agent-attachment; they indicate that the agents of the attachments are the target contacts themselves. Next section (see figures 9 and 10) shows an example

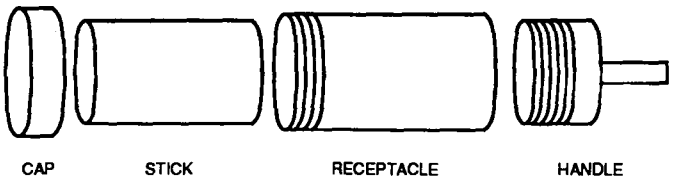


Figure 3: A simple product in exploded view

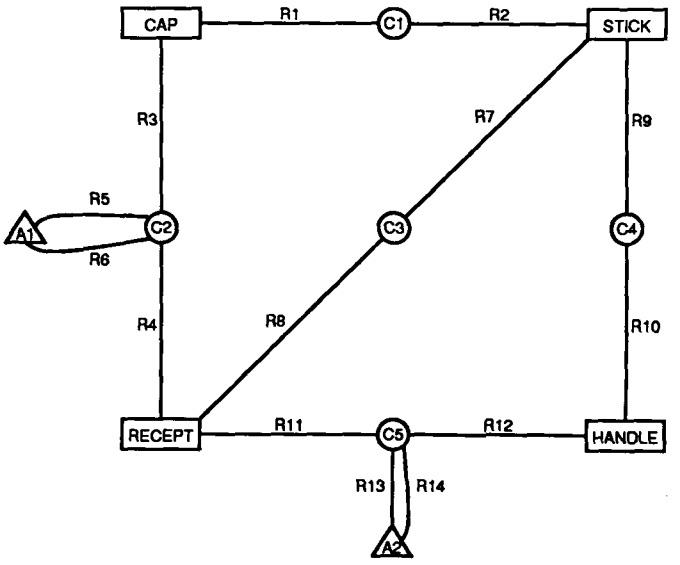


Figure 4: The relational model graph for the product show in figure 3

of an attachment whose agent is not the contact itself.

Given the relational model of a product $\langle P, C, A, R, a\text{–}functions \rangle$, a number of other useful representations may be generated. For example, the graph of connections of the assembly, as defined by Bourjault [6] (see section 2), is the simple graph $\langle V, E \rangle$ in which

$$V = P$$

$$E = \{ (p_i, p_j) \mid [p_i \in P] \wedge [p_j \in P] \wedge$$

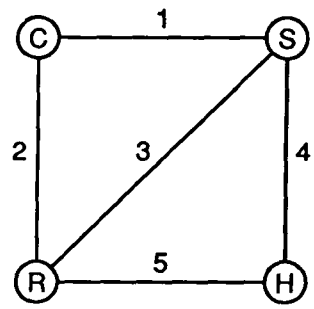
$$\wedge \exists c \, \exists r_1 \, \exists r_2 \, [[c \in C] \wedge [\{r_1, r_2\} = part\text{–}contact\text{–}relationships(c)] \wedge$$

$$\wedge [p_i = part(r_1)] \wedge [p_j = part(r_2)]] \}$$

Figure 5 shows the graph of connections for the simple product shown in figure 3.

Table 1: Attribute Functions for the Contact Entities in Figure 4

	C1	C2	C3	C4	C5
type-of-contact	planar	threaded-cylindrical	cylindrical	planar	threaded-cylindrical
normal	(0 1 0)	nil	nil	(0 1 0)	nil
back	cap	nil	nil	stick	nil
forward	stick	nil	nil	handle	nil
axis	nil	((0 0 0) (0 1 0))	((0 0 0) (0 1 0))	nil	((0 0 0) (0 1 0))
part-contact relationships	(R1 R2)	(R3 R4)	(R7 R8)	(R9 R10)	(R11 R12)
target-attachments relationships	nil	(R5)	nil	nil	(R13)
agent-attachment relationships	nil	(R6)	nil	nil	(R14)



C = cap S = stick R = receptacle H = handle

Figure 5: The graph of connections for the product shown in Figure 3

3.1. Subassemblies

A subassembly is a nonempty subset of parts that either has only one element (i.e. only one part), or is such that every part has at least one surface contact with another part in the subset. Although there are cases in which it is possible to join the same pair of parts in more than one way, a unique assembly geometry will be assumed for each pair of parts. This geometry corresponds to their relative location in the whole assembly. A subassembly is said to be stable if its parts maintain their relative position and do not break contact spontaneously. All one-part subassemblies are stable.

Given the relational model of a product $\langle P , C , A , R , \textit{a-functions} \rangle$, the relational model of a subassembly of that product is a relational model $\langle P_S , C_S , A_S , R_S , \textit{a-functions}_S \rangle$ in which $P_S \subseteq P$, $C_S \subseteq C$, $A_S \subseteq A$, $R_S \subseteq R$, and every function in $\textit{a-functions}_S$, is a subset of the corresponding function in *a-functions*. In addition to the syntactic constraints mentioned above that every relational model of an assembly must satisfy, the relational model $\langle P_S , C_S , A_S , R_S , \textit{a-functions}_S \rangle$ of a subassembly of $\langle P , C , A , R , \textit{a-functions} \rangle$ must also satisfy the constraint:

$$\forall c \, \forall r_1 \, \forall r_2 \, [\, [c \in C] \wedge [\{r_1, r_2\} = \textit{part-contact-relationships}(c)] \wedge$$

$$\wedge \, [\textit{part}(r_1) \in P_S] \wedge [\textit{part}(r_2) \in P_S] \,] \rightarrow [c \in C_S]$$

This constraint corresponds to the assumption that whenever parts are joined forming a subassembly all contacts between the parts in that subassembly are established. It requires that those contacts in the model of the assembly whose two part-contact relationships involve parts in the subassembly must also be in the model of the subassembly. For example, for the product shown in figure 3, there is no subassembly relational model in which $P_S = \{$ CAP, RECEPTACLE, STICK $\}$, and $C_S = \{$ C2, C3 $\}$. If both the cap and the stick are in P_S, then contact C1 must also be in C_S. This constraint allow the characterization of any subassembly $\langle P_S , C_S , A_S , R_S , \textit{a-functions}_S \rangle$ of a product $\langle P , C , A , R , \textit{a-functions} \rangle$ by its set of parts P_S only. This feature will be used in the algorithm for the generation of mechanical assembly sequences described in the subsequent sections. In that algorithm, the intermediate subassemblies will be characterized by their sets of parts. Given a subset of parts P_S, there is a corresponding subgraph $\langle V_S, E_S \rangle$ of the assembly's graph of connections $\langle V, E \rangle$. In this subgraph, the set of nodes V_S includes all the elements of V that correspond to the parts in V_S. And the set of edges E_S includes all the elements of E that have both end points in V_S. A subset of parts P_S characterize a subassembly if and only if the corresponding subgraph $\langle V_S, E_S \rangle$ is connected (i.e. has only one component). A predicate that is satisfied only by the subsets of parts that correspond to subassemblies can be defined as follows:

Definition 1: The **subassembly** predicate associated to subassemblies of assembly $\Psi = \langle P , C , A , R , \textit{a-functions} \rangle$ is the predicate

$sa_\Psi : \Pi(P) \rightarrow \{\text{true}, \text{false}\}$

with $sa_\Psi(\theta) = \text{true}$ if the subgraph $\langle V_S, E_S \rangle$ in which

$$V_S = P_S$$

$$E = \{ (p_i, p_j) \mid [p_i \in P_S] \wedge [p_j \in P_S] \wedge$$

$$\wedge \exists c \, \exists r_1 \, \exists r_2 \, [\, [c \in C] \wedge [\{r_1, r_2\} = \textit{part-contact-relationships}(c)] \wedge$$

$$\wedge \, [p_i = \textit{part}(r_1)] \wedge [p_j = \textit{part}(r_2)] \,] \,\}$$

is connected.

4. Decompositions of a Relational Model of an Assembly

The problem of generating the assembly sequences for a product can be transformed into the problem of generating the *disassembly* sequences for the same product. Since assembly tasks are not necessarily reversible, the equivalence of the two problems will hold only if each task used in *disassembly* is the reverse of a feasible assembly task, regardless of whether this reverse task itself is feasible or not. The expression *disassembly task*, therefore, refers to the reverse of a feasible assembly task.

As mentioned in the introduction, it was assumed that exactly two parts or subassemblies are joined at each time. It was also assumed that whenever parts are joined forming a subassembly, all contacts between the parts in that subassembly are established. In the *disassembly problem*, each task splits one subassembly into two smaller subassemblies, maintaining all contacts between the parts in either of the smaller subassemblies.

A decomposition approach can be used to solve the *disassembly problem*. In this approach the problem of *disassembling* one assembly is decomposed into two distinct subproblems, each being to *disassemble* one subassembly. Every decomposition must correspond to a *disassembly task*. If solutions for both subproblems that result from the decompositions are found, a solution for the original problem can then be obtained by combining the

solutions to the two subproblems and the task corresponding to the decomposition. For subassemblies that contain one part only, a trivial solution containing no assembly task always exists. This decomposition approach lends itself to an AND/OR graph representation of assembly sequences [17]. The correspondence between the AND/OR graph and the directed graph representations of assembly sequences is discussed elsewhere [18].

From now on, references to products, to assemblies, or to subassemblies are references to their relational models, which are always assumed to be consistent and to satisfy the syntactic constraints of a relational model of an assembly. A real product will be referred to as a *physical product*, a real assembly as a *physical assembly*, and a real subassembly as a *physical subassembly*.

A decomposition of an assembly $\langle P , C , A , R , a\text{-}functions \rangle$ is a pair of its subassemblies $\langle P_{S1} , C_{S1} , A_{S1} , R_{S1} , a\text{-}functions_{S1} \rangle$ and $\langle P_{S2} , C_{S2} , A_{S2} , R_{S2} , a\text{-}functions_{S2} \rangle$ such that $P_{S1} \cup P_{S2} = P$ and $P_{S1} \cap P_{S2} = \emptyset$. The set $C_{S1-S2} = C - (C_{S1} \cup C_{S2})$ is referred to as the *contacts of the decomposition*; they are the contacts that belong to C and do not belong to either C_{S1} or C_{S2}. The contacts of a decomposition of an assembly define a cut-set in that assembly's graph of connections. Conversely, a cut-set in the graph of connections of an assembly define a decomposition of that assembly.

A decomposition of an assembly is said to be feasible if it satisfies three predicates: *GEOMETRIC-FEASIBILITY*, *MECHANICAL-FEASIBILITY*, and *STABILITY*. These predicates reflect the feasibility of joining the physical subassemblies to produce the physical assembly.

The *GEOMETRIC-FEASIBILITY* predicate is true if there is a collision-free path to bring the two subassemblies into contact from a situation in which they are sufficiently far apart. For the assembly shown in figure 3, for example, there is no collision-free path that will bring the stick into contact with the subassembly made up of the cap, the receptacle, and the handle. Joining the stick to the subassembly made up of the three other parts is said to be *geometrically unfeasible*. Joining the stick to the subassembly made up of the cap and the receptacle, however, is *geometrically feasible* since there is a collision-free path to bring the two subassemblies into contact.

The *MECHANICAL-FEASIBILITY* predicate is true if it is feasible to establish the attachments that act on the contacts of the decomposition. Figure 6 shows a three-part assembly in which the part in the center (part B) is attached to the part in the right (part C) through two built-in bolts. Although it is geometrically feasible to join the part in the right (part C) to the subassembly made up of the two other parts, it is impossible to establish the attachments because the access to the bolts is blocked by the part in the left (part A). Joining the part in the right (part C) to the subassembly made up of the two other parts is said to be *mechanically unfeasible*.

The *STABILITY* predicate is true if the parts in either physical subassembly maintain their relative position and do not break contact spontaneously. For the assembly shown in figure 7, the subassembly made up of the parts B and C is not stable since the two parts will break contact spontaneously due to gravity, regardless of their orientation in space[3].

In practice, the feasibility of joining two subassemblies depends on the availability of adequate resources such as machines, tools, and fixtures. For the general analysis presented here, it is assumed that all such resources are available.

As discussed in section 3, the subassemblies of a given assembly $\Psi = \langle P , C , A , R , a\text{-}functions \rangle$ can be

[3]Although contacts like that between parts B and C in figure 7 are not handled by our experimental implementation, they illustrate very clearly the stability or unstability of subassemblies.

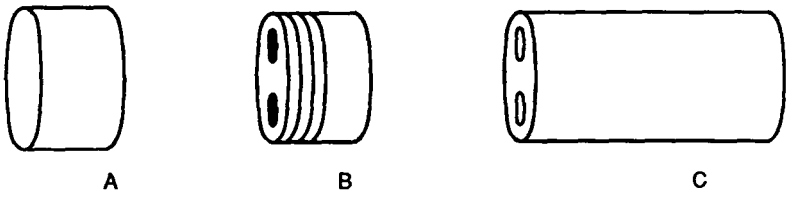


Figure 6: An assembly that illustrates the mechanical feasibility predicate

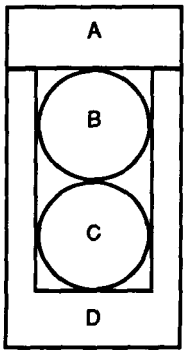


Figure 7: An assembly that illustrates the stability predicate

characterized by their sets of parts. Therefore, the three predicates described above can be defined as follows:

Definition 2: The **geometric-feasibility** predicate associated to subassemblies of assembly $\Psi=\langle P, C, A, R, a\text{–}functions\rangle$, in which $P=\{p_1, p_2, \cdots, p_N\}$, is the predicate

$$gf_\Psi: \Pi(P) \times \Pi(P) \rightarrow \{\text{true}, \text{false}\}$$

with $gf_\Psi(\theta_1, \theta_2)=\text{true}$ if and only if $\theta_1 \cap \theta_2 = \varnothing$ and there is a collision-free path to bring the two physical subassemblies of Ψ characterized by θ_1 and θ_2 into contact from a situation in which they are sufficiently far apart.

Definition 3: The **mechanical-feasibility** predicate associated to subassemblies of assembly $\Psi=\langle P, C, A, R, a\text{–}functions\rangle$, in which $P=\{p_1, p_2, \cdots, p_N\}$, is the predicate

$$mf_\Psi: \Pi(P) \times \Pi(P) \rightarrow \{\text{true}, \text{false}\}$$

with $gf_\Psi(\theta_1, \theta_2)=\text{true}$ if and only if $\theta_1 \cap \theta_2 = \varnothing$, and it is feasible to establish the attachments that act on the set of contacts between parts in θ_1 and parts in θ_2.

Definition 4: The **stability** predicate associated to subassemblies of assembly $\Psi=\langle P, C, A, R, a\text{–}functions\rangle$, in which $P=\{p_1, p_2, \cdots, p_N\}$, is the predicate

$$st_\Psi: \Pi(P) \rightarrow \{\text{true}, \text{false}\}$$

with $st_\Psi(\theta)=\text{true}$ if and only if the parts in θ maintain their relative position and do not break contact spontaneously.

The *GEOMETRIC-FEASIBILITY* predicate can be computed using path planning algorithms [13, 20, 38] to generate a collision-free path to bring the two subassemblies into contact, or, equivalently, a collision-free path to separate the two subassemblies. These algorithms typically involve large amounts of computation and more efficient approaches to general path feasibility tests are needed. For many industrial assemblies, the computation of geometric feasibility can be significantly reduced by performing a simple local analysis which can indicate that a

collision-free path does not exist. For a given decomposition, this local analysis looks at the assembled assembly and checks whether there exists an incremental translation of one of the two subassemblies that is not blocked by any of the contacts between one of its parts and one of the parts in the other subassembly.

For many types of contacts there are very few feasible motions between the parts. For example, the only direction along which a pin in a hole can translate is the direction of the axis. Whenever the part or subassembly has such a constraining contact, the local analysis can be performed by checking the compatibility of the most restrictive contact with all other contacts. In the case of the pin in the hole, the local analysis consists of checking whether a translation of the pin along its axis is not blocked by any of the other contacts between the pin and the other part or subassembly.

The local analysis is more difficult when the part or subassembly to be *disassembled* is constrained by planar contacts only. Each planar contact leaves an infinite number of unconstrained directions along which translation is possible. All these directions have positive projection over the normal to the surface of the blocking part, pointing towards the outside of the blocking part.

In order to decide whether a set of planar contacts does not completely constrain one part or subassembly, one must find whether there is a nonzero solution to the system of linear inequalities

$$\sum_{j=1}^{3} n_{ij}\ x_j \geq 0 \qquad i=1,2,\cdots,N$$

where $\underline{n}_i=[n_{i1}\ \ n_{i2}\ \ n_{i3}]$ is the normal to the surface of the i^{th} contact. This system of linear inequalities defines a polyhedral convex cone. It has been shown [16] that such a polyhedral convex cone can be *built up* from its (unique) d-dimensional face and its $(d+1)$-dimensional faces (if any), where $d=3-\text{rank}(M)$, and M is the matrix of the coefficients n_{ij}. If d is greater than zero, then the polyhedral convex cone has a face of dimension greater than zero and therefore the system of inequalities has a nonzero solution. If d is equal to zero, then the system of inequalities has a nonzero solution only if the polyhedral convex cone has at least one one-dimensional face. The existence of a one-dimensional face can be determined by checking the $N\cdot(N-1)$ pairwise intersections of the planes corresponding to the inequalities. Each intersection of two distinct planes is a line. If one of the two unity vectors, $\underline{t}$ and $-\underline{t}$ along the intersection line of two planes has positive projection over all the normal vectors $n_1, n_2, \cdots, n_N$, then the half-line defined by that vector ($\underline{t}$ or $-\underline{t}$) is a one-dimensional face of the polyhedral convex cone.

If there is a nonzero solution to the system of inequalities, then the part or subassembly is not completely constrained. Otherwise the subassembly is completely constrained, and there is no need to look for a collision free path.

Our current implementation is not limited to only checking the existence of a nonzero solution to the system of linear inequalities, but includes the computation of the polyhedral convex cone of all solutions. Appendix I addresses the reasoning about the feasibility of local translations for robotic assembly of a part constrained by planar contacts.

The usefulness of the local analysis is further enhanced by the use of *virtual* contacts to describe blocking relationships equivalent to contacts. In the product shown in figure 3, for example, if the stick did not touch the handle, the local analysis as described above would indicate that the stick can translate (incrementally) along its axis. In a case like this, a *virtual* planar contact, analogous to C4 on figure 4, would be added to the relational model indicating the blocking of the stick by the handle.

The *MECHANICAL-FEASIBILITY* predicate can be computed by inspection of the relational model of the

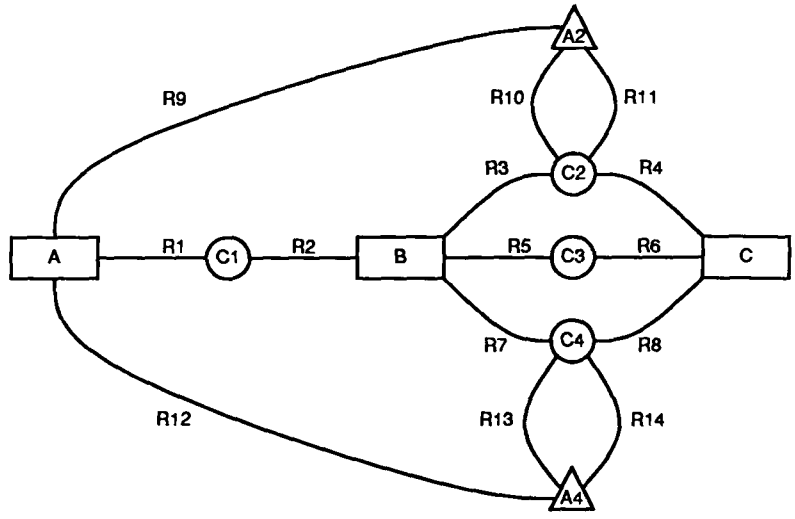


Figure 8: The relational model of the assembly shown in figure 6

assembly. In our current implementation, a procedure *MECHANICAL-FEASIBILITY* checks whether the attachments acting on the contacts of the decomposition are not blocked in the resulting assembly, and are not present in either one of the subassemblies. Two examples will illustrate this computation.

Figure 8 shows the relational model of the assembly shown in figure 6. Relationships R10 and R13 are agent-attachment, relationships R11 and R14 are target-attachment, and relationships R9 and R12 are blocking-part-attachment; all other relationships are part-contact. The relationships R9 and R12 indicate that part A blocks the access to attachment A2 and to attachment A4. One of the disassembly tasks whose mechanical feasibility must be computed is the separation of part C from the subassembly made up of parts A and B. The mechanical unfeasibility of this task can be detected by inspection of the relational model which indicates that the attachments acting on the contacts of the decomposition are blocked by part A. After part A is removed, those attachments will no longer be blocked and part C can be separated from part B.

Figure 9 shows an assembly that has three parts: a box, a cover, and a clip that attaches the cover to the box. Figure 10 shows this assembly's relational model. Relationships R7, R8, and R9 in figure 10 are target-attachment; they indicate that the three contacts C1, C2, and C3 are attached by attachment A1. Relationship R10 is agent-attachment; it shows that the agent of attachment A1 is the clip. One of disassembly tasks whose mechanical feasibility must be computed is the separation of the cover from the subassembly made up of the box and the clip. The mechanical unfeasibility of this task can be detected by inspection of the relational model which shows that the contacts cannot be detached while the agent of the attachment is present. The separation of the clip from the subassembly made up of the box and the cover, however, is feasible because the agent of the attachments is being separated.

The computation of the *STABILITY* predicate will depend on additional assumptions about the assembly process. For example, it may be assumed that all subassemblies can be made stable through the use of jigs and fixtures. In our current implementation we made this assumption and we do not compute the *STABILITY* predicate. In previous work aimed at selecting an assembly sequence [17], we assessed the stability of a subassembly by the degrees of freedom for relative motion between parts. Similarly, one can establish a threshold on the degrees of freedom for relative motion above which a subassembly would be considered unstable.

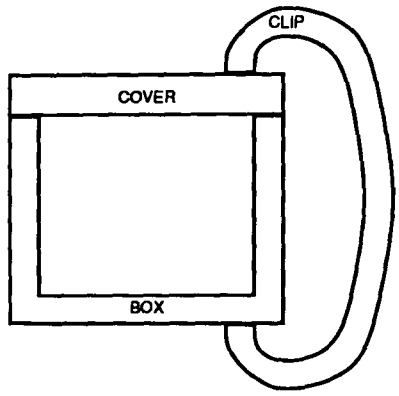


Figure 9: Assembly example

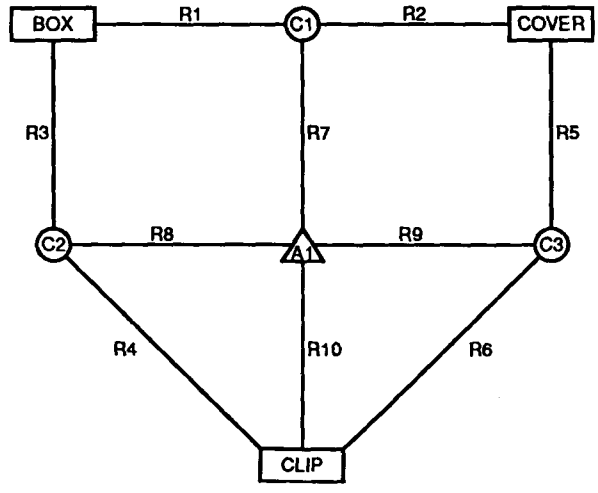


Figure 10: Relational model for the assembly example show in figure 9

An alternative approach for the computation of the *STABILITY* predicate is to check whether there is an orientation of the subassembly such that there is no relative motion between parts due to gravity. As a first approximation, friction can be ignored since it typically helps the stability. Boneschanscher et al. [4] have taken this approach with the additional assumption that the subassembly sits on a table. They used a convex hull algorithm to find candidate orientations in which the subassembly can sit on a table, and for these orientations they checked the static stability. Their analysis takes friction into account.

For the discussion in the next section, which presents the algorithm for generating the assembly sequences, it is assumed that there exist correct algorithms for computing the three predicates discussed above, and that they are combined into the procedure *FEASIBILITY-TEST* shown in figure 11.

procedure *FEASIBILITY-TEST(decomposition, assembly)*

return *AND* (*GEOMETRIC-FEASIBILITY(decomposition, assembly)*,
STABILITY(decomposition),
MECHANICAL-FEASIBILITY(decomposition, assembly))

end procedure

Figure 11: Procedure *FEASIBILITY-TEST*

5. The Algorithm for Generating All Assembly Sequences

As discussed in the previous section, this research takes a decomposition approach to the problem of generating assembly sequences. The basic idea underlying the approach is to enumerate the decompositions of the assembly and to select those decompositions that are feasible. The decompositions are enumerated by enumerating the cut-sets of the assembly's graph of connections. Knowledge of the feasible decompositions allows the construction of the AND/OR graph representation of assembly plans. Each feasible decomposition corresponds to a hyperarc in the AND/OR graph connecting the node corresponding to the assembly to the two nodes corresponding to the two subassemblies. The same process is repeated for the subassemblies and subsubassemblies until only single parts are left.

It has been shown [12, 21] that the set of all cut-sets of a graph $\langle V, E \rangle$ is a subspace of of the vector space over the Galois field modulo 2 associated with the graph. The vectors in this vector space are the elements of $\Pi(E)$, the set of all subsets of E. It has also been shown that the fundamental system of cut-sets relative to a spanning tree is a basis of the cut-set subspace. Therefore, the cut-sets of a graph can be enumerated by constructing a spanning tree of the graph, finding the fundamental system of cut-sets relative to that spanning tree, and computing all the combinations of fundamental cut-sets. In our current implementation, the cut-sets are enumerated using a more efficient approach. We first look at all connected subgraphs having the cardinality of their set of nodes smaller than or equal to half of the cardinality of the set of nodes in the whole graph. For each of these subgraphs, the set of edges of the whole graph that have only one end in the subgraph defines a cut-set if their removal leaves the whole graph with exactly two components.

Figure 12 shows the procedure *GET-FEASIBLE-DECOMPOSITIONS* which takes as input the relational model of an assembly and returns all feasible decompositions of that assembly. The procedure first generates the graph of connections for the input assembly and computes the cut-sets of this graph. Each cut-set corresponds to a decomposition. The procedure *GET-DECOMPOSITIONS* is used to find the decomposition that corresponds to a cut-set, and the procedure *FEASIBILITY-TEST* discussed in the previous section is used to check whether that decomposition is feasible or not. The feasible decompositions are stored in the list *feasible-decompositions* which was empty at the beginning. After all cut-sets have been processed, the procedure returns the list *feasible-decompositions*.

An example will illustrate the computation of the feasible decompositions of an assembly. When passed the relational model of the assembly in figure 3, procedure *GET-FEASIBLE-DECOMPOSITIONS* will compute the graph of connections shown in figure 5, and all its cut-sets, which are indicated in figure 13. The analysis of those cut-sets will indicate the feasible decompositions. The first cut-set yields a feasible decomposition since it is feasible to join the cap and the subassembly made up of the three other parts. The second cut-set also yields a feasible decomposition because it is feasible to join the subassembly consisting of the cap plus the receptacle, and the subassembly consisting of the stick plus the handle. The third cut-set, however, does not yield a feasible

```
procedure GET-FEASIBLE-DECOMPOSITIONS(assembly)
feasible-decompositions ← NIL
graph ← GET-GRAPH-OF-CONNECTIONS(assembly)
cut-sets ← GET-CUT-SETS(graph)
while cut-sets is not empty do
    begin loop1
    next-cut-set ← FIRST(cut-sets)
    cut-sets ← TAIL(cut-sets)
    next-decomposition ← GET-DECOMPOSITION(next-cut-set)
    if FEASIBILITY-TEST(next-decomposition)
        then feasible-decompositions ← UNION(feasible-decompositions, LIST(next-decomposition))
    end loop1
return feasible-decompositions
end procedure
```

Figure 12: Procedure *GET-FEASIBLE-DECOMPOSITIONS*

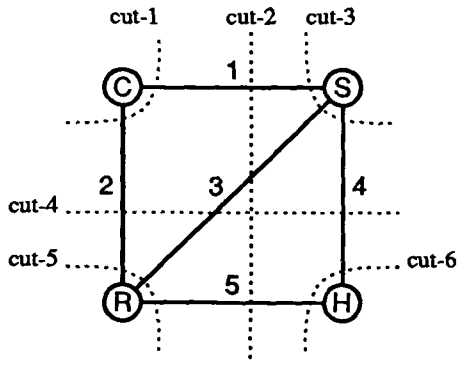


Figure 13: The cut-sets of the graph of connections for the assembly shown in Figure 3

decomposition, since it is not possible to join the stick and the subassembly made up of the three other parts. Similarly, the fourth and the sixth cut-sets yield feasible decompositions while the fifth cut-set does not. Therefore, procedure *GET-FEASIBLE-DECOMPOSITIONS* will return a list containing the four decompositions that correspond to the first, second, fourth, and sixth cut-sets.

Figure 14 shows the procedure *GENERATE-AND-OR-GRAPH* which takes the relational model of an assembly, and returns the AND/OR graph representation of all assembly sequences for that assembly. The nodes in the AND/OR graph returned are pointers to relational models of assemblies.

Procedure *GENERATE-AND-OR-GRAPH* uses the lists *closed* and *open* to store the pointers to the relational models of the subassemblies whose decompositions into smaller subassemblies respectively have and have not been

```
procedure GENERATE-AND-OR-GRAPH(assembly)
open ← LIST(GET-POINTERS(LIST(assembly)))
closed ← NIL
hyperarcs ← NIL
while open is not empty do
    begin loop1
    next-subassembly ← FIRST(open)
    open ← TAIL(open)
    closed ← UNION(closed, LIST(next-subassembly) )
    decompositions-of-next-subassembly ← GET-FEASIBLE-DECOMPOSITIONS(next-subassembly)
    while decompositions-of-next-subassembly is not empty do
        begin loop2
        next-decomposition ← FIRST(decompositions-of-next-subassembly)
        decompositions-of-next-subassembly ← TAIL(decompositions-of-next-subassembly)
        subassemblies ← GET-POINTERS(next-decomposition)
        hyperarcs ← UNION(hyperarcs, LIST(LIST(next-subassembly, subassemblies)))
        while subassemblies is not empty do
            begin loop3
            next-subassembly ← FIRST(subassemblies)
            subassemblies ← TAIL(subassemblies)
            if next-subassembly is not in open or in closed, add it to open; otherwise ignore it
            end loop3
        end loop2
    end loop1
return LIST(closed, hyperarcs)
end procedure
```

Figure 14: Procedure *GENERATE-AND-OR-GRAPH*

generated.

The procedure takes one element of *open* at a time, moves it to *closed*, and uses procedure *GET-FEASIBLE-DECOMPOSITIONS* to generate all decompositions of the relational model pointed by that element. For each decomposition, procedure *GENERATE-AND-OR-GRAPH* uses the procedure *GET-POINTERS* to get the pointers to the relational models of the subassemblies. Procedure *GET-POINTERS* checks whether each resulting subassembly has appeared before or not. If the subassembly has appeared before, its pointer is used, otherwise a new pointer is created. The new pointers are inserted into *open*. Each decomposition yields one hyperarc of the AND/OR graph.

Figure 15 shows the resulting AND/OR graph for the product shown in figure 3.

A more efficient implementation of the method for the generation of assembly sequences presented above will include additional tests aimed at avoiding unnecessary computation[4]. One such test is to check whether the feasibility of a decomposition follows from the feasibility of other decompositions. For example, the feasibility of the decomposition corresponding to hyperarc 10 in figure 15 follows from the feasibility of the decompositions corresponding to hyperarcs 4 and 5. If it was geometrically and mechanically feasible to *disassemble* the handle from the whole assembly (hyperarc 4), then it is geometrically and mechanically feasible to *disassemble* the handle from a subassembly. And since the subassembly made up of the stick and the receptacle is stable (hyperarc 5), it follows that the decomposition corresponding to hyperarc 10 is feasible. This test indicates that if the decompositions corresponding to hyperarcs 4 and 5 have already been analysed and found to be feasible, then it is not necessary to perform the computation corresponding to procedure *FEASIBILITY-TEST* in the analysis of the decomposition that corresponds to hyperarc 10. Similarly, another additional test would check whether the unfeasibility of a decomposition follows from the unfeasibility of other decompositions already analysed.

6. Analysis of the Algorithm

This section presents an analysis of the algorithm for the generation of all assembly sequences. First, a proof of the correctness and completeness of the algorithm *GET-FEASIBLE-DECOMPOSITIONS* is presented. These results are then used to prove the correctness and completeness of the algorithm *GENERATE-AND-OR-GRAPH*. At the end, an assessment of the computation involved in executing *GENERATE-AND-OR-GRAPH* is presented.

6.1. The correctness of algorithm *GET-FEASIBLE-DECOMPOSITIONS*

The partial correctness of the algorithm *GET-FEASIBLE-DECOMPOSITIONS* is immediate. The list *cuts* is initially empty. Only feasible decompositions are added to the list *cuts*. Therefore, the list returned by *GET-FEASIBLE-DECOMPOSITIONS* does not contain any element that is not a feasible decomposition of the assembly input.

The total correctness follows from the fact that there is only a finite number of cut-sets in a graph. The list *cut-sets* contains initially all cut-sets of the graph of functional connection corresponding to the assembly input. At each execution of *loop1*, one element is removed from the list *cut-sets*. Therefore, after a finite number of executions of *loop1* the list *cut-sets* becomes empty, and the algorithm terminates.

This proof assumes that the algorithm for generating the cut-sets of a graph is correct and complete. As discussed

[4]Our current implementation consists of the basic algorithms presented in the text and does not yet include these additional tests.

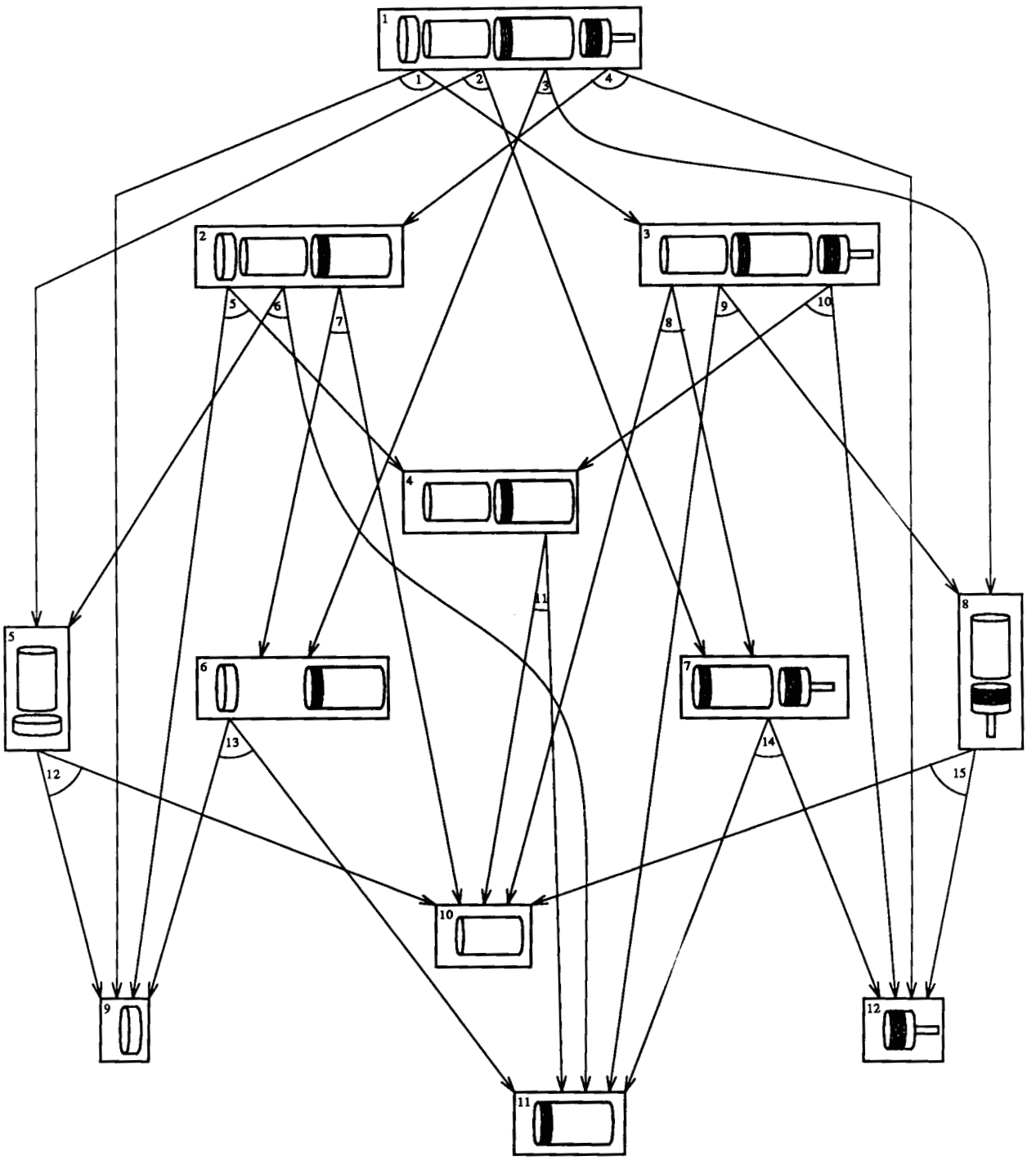


Figure 15: The AND/OR graph for the assembly shown in figure 3

in the previous section, the enumeration of the cut-sets of a graph is studied in graph theory; for example, Deo [12] and Liu [21] discuss that problem.

This proof also assumes that it is possible to decide correctly whether a decomposition is feasible or not, based on geometrical and physical criteria, as discussed in the section 4.

6.2. The completeness of algorithm *GET-FEASIBLE-DECOMPOSITIONS*

There is a one-to-one correspondence between cut-sets in the graph of connections of an assembly, and the decompositions of that assembly. Therefore, since algorithm *GET-FEASIBLE-DECOMPOSITIONS* goes over all cut-sets of the graph of connections, all feasible decompositions will be generated.

As in the proof of correctness above, this proof of completeness assumes the use of a correct and complete algorithm for the generation of all cut-sets of a graph, and a correct algorithm for deciding the feasibility of a decomposition.

6.3. The correctness of algorithm *GENERATE-AND-OR-GRAPH*

List *closed* is updated at only one point, and it only gets elements that were previously in the *open* list. The *open* list contains initially a pointer to the relational model of the assembly input, which is a node of the AND/OR graph. List *open* is updated inside *loop3* where it gets pointers to the relational models of the subassemblies that are part of a feasible decomposition, and therefore, are nodes of the AND/OR graph. Therefore, the elements in the *open* list, and consequently the elements in the *closed* list, are always pointers to relational models either of the original assembly, or of subassemblies that take part of a feasible decomposition.

The *hyperarcs* list is initially empty. It is updated only inside *loop2* where it gets the hyperarc corresponding to a feasible decomposition. Therefore, algorithm *GET-FEASIBLE-DECOMPOSITIONS* can only return a set of nodes and a set of hyperarcs of the AND/OR graph. This establishes the partial correctness of the algorithm.

List *open* gets only subassemblies and no subassembly is inserted more than once. Since there is a finite number of subassemblies, the algorithm terminates. This establishes the total correctness of the algorithm.

6.4. The completeness of algorithm *GENERATE-AND-OR-GRAPH*

Since algorithm *GET-FEASIBLE-DECOMPOSITIONS* is complete, all possible decompositions of all subassemblies that are inserted into the list *open* yield a hyperarc. Furthermore, all subassemblies that result from a decomposition are inserted into list *open*, and later are moved to list *closed*. Therefore, the first list returned contains all subassemblies that resulted from some decomposition, and the second list returned contains one hyperarc for each decomposition of each subassembly.

6.5. Complexity

The amount of computation involved in the generation of the AND/OR graph for a given assembly depends on the number N of parts that make up the assembly, on how interconnected those parts are, and also on the resulting AND/OR graph.

The number of prospective decompositions (i.e. cut-sets of the graphs of functional connections) that must be analysed will be used in this section as a measure of the amount of computation involved in the generation of all

assembly sequences[5]. Two models for how the parts in the assembly are interconnected are considered in order to provide bounds in the estimate of computational complexity:

1. a *strongly connected* assembly in which every part is connected to every other part; and
2. a *weakly connected* assembly in which there are $N-1$ connections between the N parts, with the i^{th} connection being between part the i^{th}, and the $(i+1)^{th}$ parts.

And three possibilities for the resulting AND/OR graph are considered:

1. a *balanced tree* AND/OR graph in which there is at most one hyperarc leaving each node and this hyperarc points to two nodes whose corresponding subassemblies either have the same number of parts, or their number of parts differ by one;
2. *one-part-at-a-time tree* AND-OR graph in which there is at most one hyperarc leaving each node, and this hyperarc points to two nodes one of which corresponds to a one-part subassembly; and
3. a *network* AND/OR graph in which there are as many hypearcs leaving each node as there are cut-sets in the graph of functional connections of the node's corresponding subassembly.

The resulting total number D of decompositions that must be analysed as a function of the number N of parts that make up the assembly for each possible combination of how the parts are interconnected and the type of the resulting AND/OR graph is:

1. Weakly connected assemblies:
   a. Balanced tree AND/OR graph: the number of prospective decompositions that must be analysed is $N-1$ for the initial assembly, $N-2$ for all subassemblies, $N-4$ for all subsubassemblies, and so on. Therefore[6],

$$D=(N-1)+(N-2)+(N-4)+\cdots+(N-2^{\text{int}(\log_2 N)}) =$$

$$= \sum_{i=0}^{\text{int}(\log_2 N)} (N-2^i) = N\cdot[\text{int}(\log_2 N)+1] - 2^{[\text{int}(\log_2 N)+1]} + 1$$

   b. One-part-at-a-time tree AND/OR graph: the number of prospective decompositions that must be analysed is $N-1$ for the initial assembly, $N-2$ for the $(N-2)$ part subassembly, $N-3$ for the $(N-2)$-part subassembly, and so on. Therefore,

$$D=(N-1)+(N-2)+(N-3)+\cdots+2+1 = \sum_{i=1}^{N-1}(N-i) = \frac{N\cdot(N-1)}{2}$$

   c. Network AND/OR graph: the number of prospective decompositions that must be analysed is $N-1$ for the N-part subassembly, $N-2$ for each of the two $(N-1)$-part subassemblies, $N-3$ for each of the three $(N-2)$-part subassemblies, and so on. Therefore,

$$D = 1\cdot(N-1)+2\cdot(N-2)+3\cdot(N-3)+\cdots+(N-1)\cdot 1 =$$

$$= \sum_{i=1}^{N-1} i\cdot(N-i) = \frac{(N+1)\cdot N\cdot(N-1)}{6}$$

---

[5]The overall complexity of algorithm *GENERATE-AND-OR-GRAPH* should take into account the computation involved in generating the cut-sets of the graph of functional connections.

[6]We use the notation int (x) to represent the largest integer that is less than or equal to x. For example, int (3) = 3 and int (3.5) = 3.

2. Strongly connected assemblies:

a. Balanced tree AND/OR graph: the number of prospective decompositions that must be analysed is $(2^{N-1}-1)$ for the initial assembly, $(2^{\text{int}(\frac{N-1}{2})}-1) + (2^{\text{int}(\frac{N-2}{2})}-1)$ for all subassemblies, $(2^{\text{int}(\frac{N-1}{4})}-1) + (2^{\text{int}(\frac{N-2}{4})}-1) + (2^{\text{int}(\frac{N-3}{4})}-1) + (2^{\text{int}(\frac{N-4}{4})}-1)$ for all subsubassemblies, and so on. Therefore,

$$D = \sum_{i=0}^{\text{int}(\log_2 N)} \sum_{j=0}^{2^i-1} [\,2^{\text{int}(\frac{N-j-1}{2^i})} - 1\,]$$

b. One-part-at-a-time tree AND/OR graph: the number of prospective decompositions that must be analysed is $(2^{N-1}-1)$ for the N-part subassembly, $(2^{N-2}-1)$ for the $(N-1)$-part subassembly, $(2^{N-3}-1)$ for the $(N-2)$-part subassembly, and so on. Therefore,

$$D=(2^{N-1}-1)+(2^{N-2}-1)+(2^{N-3}-1)+\cdots+(2-1) = 2^N-N-1$$

c. Network AND/OR graph: the number of prospective decompositions that must be analysed is $(2^{N-1}-1)$ for the N-part subassembly, $(2^{N-2}-1)$ for each of the $\binom{N}{N-1}$ $(N-1)$-part subassemblies, $(2^{N-3}-1)$ for each of the $\binom{N}{N-2}$ $(N-2)$-part subassemblies, and so on. Therefore,

$$D = \binom{N}{N}\cdot(2^{N-1}-1) + \binom{N}{N-1}\cdot(2^{N-2}-1) + \cdots + \binom{N}{2}\cdot(2-1) =$$

$$= \sum_{i=2}^{N}\binom{N}{i}\cdot(2^{i-1}-1) = \frac{3^N+1}{2} - 2^N$$

For each of the three possibilities of the resulting AND/OR graph, table 2 shows the number of decompositions that must be analysed for weakly connected assemblies and table 3 shows the number of decompositions that must be analysed for strongly connected assemblies, as a function of the number of parts that make up the product. The figures in table 3 are given as a reference since it is very unlikely that there would be a twenty-part assembly in which every part is connected to every other part.

The results above take into account the fact that the type of the resulting AND/OR graph is not known a priori. For example, for the weakly connected assembly whose AND/OR graph is a balanced tree, all the $N-1$ cut-sets of the whole assembly were included in the number of decompositions that are tested, although there is only one cut-set that yields two subassemblies that have the same number of parts.

As discussed in the end of section 5, a more efficient implementation of the method for the generation of assembly sequences presented in this paper will include additional tests aimed at avoiding unnecessary computation. One such test is to check whether the feasibility of a decomposition follows from the feasibility of other decompositions. In the case of strongly connected assemblies in which all decompositions of all subassemblies are feasible, the computation can be significantly reduced if this test is performed before analysing each decomposition. Since all decompositions of the whole assembly are feasible, all decompositions of all subassemblies should also be

Table 2: The number of decompositions that must be analysed for each type of resulting AND/OR graph, as a function of the number of parts, for weakly connected assemblies.

Number of Parts N	Balanced-tree AND/OR graph	One-part-at-a-time AND/OR graph	network AND/OR graph
2	1	1	1
3	3	3	4
4	5	6	10
5	8	10	20
6	11	15	35
7	14	21	56
8	17	28	84
9	21	36	120
10	25	45	165
15	45	105	560
20	69	190	1,330
25	94	300	2,600
30	119	435	4,495

Table 3: The number of decompositions that must be analysed for each type of resulting AND/OR graph, as a function of the number of parts, for strongly connected assemblies.

Number of Parts N	Balanced-tree AND/OR graph	One-part-at-a-time AND/OR graph	network AND/OR graph
2	1	1	1
3	4	4	6
4	9	11	25
5	20	26	90
6	39	57	301
7	76	120	966
8	145	247	3,025
9	284	502	9,330
10	551	1,013	28,501
15	16,604	32,752	7,141,686
20	525,389	1,048,555	1,742,343,625
25	16,783,550	33,554,406	423,610,750,290
30	536,904,119	1,073,741,793	102,944,492,305,501

feasible. Therefore, with a simple additional test, the total number of decompositions that must be analysed is reduced from $\frac{3^N+1}{2} - 2^N$ to $(2^{N-1}-1)$.

7. Conclusion

A correct and complete algorithm for the generation of all mechanical assembly sequences was presented. To the authors' knowledge, no previous algorithm for the generation of all mechanical assembly sequences has been proven correct and complete.

The problem of generating assembly sequences was transformed into the equivalent problem of generating *disassembly* sequences. The algorithm operation consists in looking at all the decompositions of the assembly, i.e. all the ways the assembly can be *split* into two subassemblies. This is done by generating all cut-sets of the assembly's graph of connections, and checking which cut-sets correspond to feasible decompositions. A decomposition is feasible if it possible to obtain the assembly by joining the two subassemblies. The same process is repeated for the subassemblies, for the subsubassemblies, and so on, until only single parts are left. At the end, the AND/OR graph representation of assembly sequences is returned.

The algorithm also lends itself to an interactive implementation in which a computer program generates questions that are answered by a human expert. Each question addresses the feasibility of a decomposition. But unlike previous methods [6, 11], it is possible to have a computer program, instead of a human, to answer the questions directly from a description of the assembly. Our current implementation, which has the restrictions on the types of assemblies discussed in section 3, includes programs that answer the questions.

An approach to compute the answer to the question of whether it is feasible to obtain a given assembly by joining two subassemblies was presented. This approach is based on the use of a relational model description of the assembly. The model includes three types of entities: parts, contacts, and attachments; it also includes a set of relationships between entities. Both entities and relationships can have attributes. To decide whether a given decomposition is feasible, three predicates must be computed, using the data in the relational model:

- The *GEOMETRIC-FEASIBILITY* predicate which is true if there exists a collision-free path to bring the two physical subassemblies into contact from a situation in which they are sufficiently far apart.
- The *MECHANICAL-FEASIBILITY* predicate which is true if it is feasible to establish the attachments that act on the contacts of the decomposition.
- The *STABILITY* predicate which is true if the parts in each subassembly maintain their relative position and do not break contact spontaneously.

The key assumption in proving the correctness of the algorithm was that it is always possible to decide correctly, based on geometrical and physical criteria (i.e. using the three predicates above), whether it is feasible to obtain a given assembly by joining two subassemblies.

The amount of computation involved in generating all mechanical assembly sequences was assessed by determining the number of decompositions that must be analysed. That amount depends not only on the number of parts and on how they are interconnected, but on the solution AND/OR graph as well. The least amount of computation occurs for weakly connected assemblies in which each subassembly has only one feasible decomposition and that decomposition yields two subassemblies whose number of parts are either equal or differ by one. The maximum amount of computation occurs for strongly connected assemblies in which all decompositions of all subassemblies are feasible. This worst case, however, is very unlikely to occur in practice. Furthermore,

additional simple tests discussed in section 5 can reduce the amount of computation.

In practice, an evaluation of the alternative assembly sequences generated by the algorithm presented in this paper is needed in order to choose the sequence that will be actually used in the assembly process. Different evaluation funtions have been explored including a function based on parts entropy [32, 33], and a function based on the complexity of assembly tasks and the stability of subassemblies [17].

It is also possible to implement an interactive system in which a computer program generates the alternative sequences, as described in this paper, and a human expert then selects the best one. Still another possibility would be to use an evaluation function for a preselection of "good" alternative sequences and then have a human expert to make the final choice.

Whenever the amount of computation exceeds the available computational resources, at least two strategies may be followed:

1. The number of parts can be artificially reduced by treating subassemblies as single parts. An analysis of the graph of connections may indicate the clusterings of parts that yield bigger reductions in the amount of computation.
2. The algorithm generates fewer, hopefully the best, sequences using some heuristics to guide the generation of assembly sequence. Such heuristics should be compatible with the evaluation function used to choose among the alternative assembly sequences.

In both strategies, the computation will be reduced at the expense of the completeness, since not all possible sequences will be generated. The devolpment of a procedure to cluster parts into subassemblies to obtain a hierarchical model of the assembly, and the development of good heuristics to guide the generation of assembly sequences are issues for future research.

Acknowledgements

Randy Brost read part of this paper and gave us several constructive comments. We thank him for that. The responsibility for the paper, of course, remains with the authors.

This research was supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (Brazil), the Jet Propulsion Laboratory of the California Institute of Technology, and The Robotics Institute of Carnegie Mellon University.

Appendix

I. Reasoning about the Feasibility of Local Translations for Robotic Assembly of a Part Constrained by Planar Contacts

I.1. Introduction

The high level of planning the assembly of a product can be viewed as a path search in the state-space of possible configurations of the set of parts that comprises the product [17]. The initial state corresponds to the configuration in which all parts are disconnected from each other. The goal state corresponds to the configuration in which the parts are properly joined. The moves correspond to the assembly operations, since they change one state into another.

A complete description of the product is available for planning purposes. This description includes the shape of the parts, their relative positions, and the spatial and mechanical relations between parts.

The search can be conducted backwards from the goal state to the initial state. The moves in the backward search correspond to *disassembly tasks* which are defined to be the reverse of feasible assembly tasks. The preconditions for a *disassembly task* [34] include:

1. release of attachments.
2. stability of subassemblies.
3. separability of subassemblies:
    a. local analysis - test incremental motion;
    b. global analysis - find global trajectory.

The local analysis consists of checking whether there exists an incremental motion of one part or subassembly that is not blocked by any one of its contacts with other parts. For many types of contacts there are very few feasible motions between the parts. For example, a cylindrical pin in a hole can either translate in the direction of the axis, or rotate around the axis. Whenever the part or subassembly under consideration has such a constraining contact, the local analysis can be performed by checking whether at least one of the few motions that are compatible with the most restrictive contact is also compatible with all other contacts.

The local analysis is more difficult when the part (or subassembly) to be disassembled is constrained by planar contacts only. Each planar contact leaves an infinite number of unconstrained directions along which translation is possible. In this case, the intersection of the sets of translations that are not blocked by each contact cannot be found by discrete search over a finite set of directions.

This appendix presents an efficient procedure to obtain explicitly the set of directions along which an object that is constrained by several planar contacts can translate. In addition to answering the question of whether there is a direction along which translation is feasible, the procedure also produces a representation of the set of all those directions.

I.2. Background

Shoham [35] and Mani [26] analyzed the freedom of two dimensional objects when in contact with other objects. Although both studies include translational and rotational freedom, they are restricted to two dimensional objects, and neither one indicates how the analysis can be scaled up to three dimensions.

Jain and Donath [19] analyzed the translational and rotational freedom of parts in an assembly but with the constraint that the existing contacts between parts cannot be broken. They do not show how their approach can be extended to deal with the case of breaking of contacts.

Ejiri *et al.* [14] proposed the use of *restraint vectors* to decide whether a part constrained by planar contacts could translate. The restraint vector of a part P was defined to be

$$u_P=(a_{+x}, a_{-x}, a_{+y}, a_{-y}, a_{+z}, a_{-z})$$

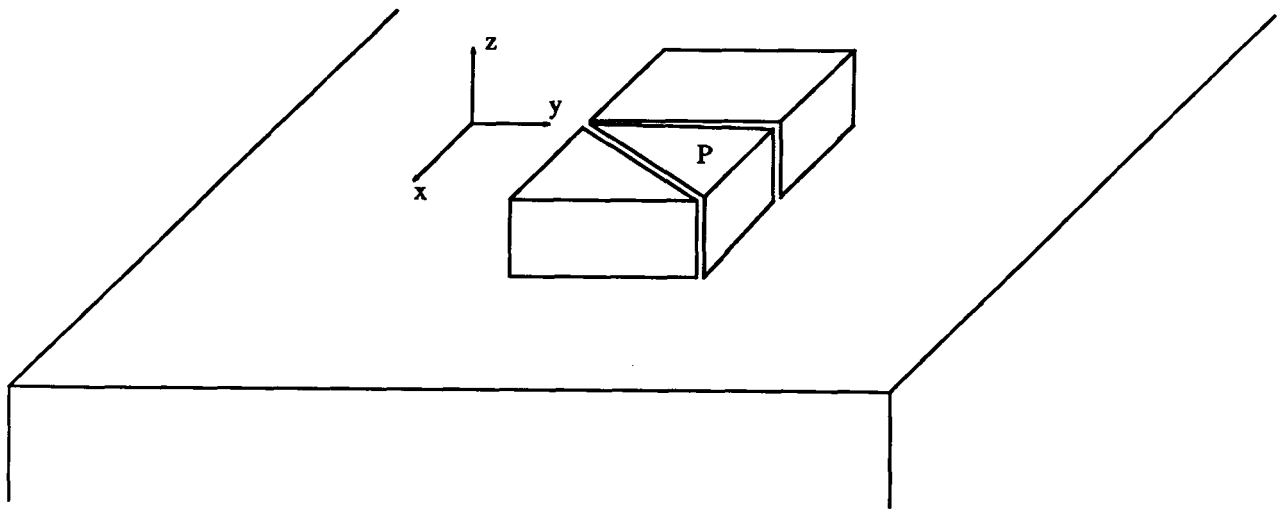
where

$$a_r=\begin{cases}1 & \text{if } P \text{ is restrained along direction } r\\ 0 & \text{if } P \text{ is not restrained along direction } r\end{cases}$$

and the logical possibility of disassembling part P was decided by the logical formula

$$\overline{a_{+x}a_{-x}}\cdot\overline{a_{+y}a_{-y}}\cdot\overline{a_{+z}}=\begin{cases}1, & \text{when possible}\\ 0, & \text{when impossible.}\end{cases} \quad (1)$$

That formula corresponds to the requirement that the part be free in the upper (positive z) direction and both x and y are free in either the positive or the negative direction. Although the above logical formula is a sufficient condition, it is not necessary. It is also not difficult to think of a situation in which disassembling a part is feasible and the logical formula yields 0. Figure 16 shows one such situation.



$$u_P=(1,1,0,1,0,1)$$

Figure 16: Part P can move but the logical formula (1) yields 0

Within the work in kinematics, Asada and By [3] introduced the concept of *Automatically Reconfigured Fixturing* which is a fixturing system that can be adapted to hold different workparts. It consists of a number of fixture elements that can be placed on a flat horizontal table to conform to the geometry of the workpart to be fixtured. The table has magnetic chucking capability which can be activated to secure the fixturing elements in place. When completely fixtured, the workpart will be in contact with a number of fixturing elements that will constrain its movements completely. The process of fixturing one workpart starts with a positioning phase in which the workpart is brought into contact with a subset of the fixturing elements which, in this paper, will be referred to as guiding elements. Once positioning is achieved, additional elements are placed on the table to constrain the workpart completely.

Asada and By carried out a kinematic analysis aimed at answering the following questions:

1. Is the location of the workpart that achieves contact with all guiding elements unique?
2. Is this location that achieves contact with the guiding elements accessible/detachable?
3. Do the additional elements (together with the guiding elements) constrain the workpart completely?

The last two questions are similar to the problem addressed in this paper. In the second question, they are interested in guaranteeing that there are feasible local motions for the workpart, so it can be brought into contact with all guiding elements. In the third question, they are interested in guaranteeing that there are no feasible local motions (i.e. that the workpart is constrained completely).

Asada and By modeled the contacts between the workpart and the fixturing elements as point contacts, and derived conditions for the feasibility of local motions, including both translations and rotations. Those constraints were used to check whether the configuration of the fixturing elements would constrain the workpart completely. They did not address how to determine the set of incremental motions that satisfy the derived conditions.

More recent research on robotic planning [22, 23] has aimed at enabling robots to execute tasks specified in *task-level* commands such as

move *<part-id>* **to** *<location-specification>*

in which the second term within angle brackets specifies a configuration (a position and an orientation) either as a homogeneous transform matrix or as a set of spatial relationships among objects. The translation of a *task-level* command into *robot-level* commands involves selecting fixtures, grasping points, gross motions, fine motions, etc.

It is clear that procedures that are able to construct a path for a part from an initial configuration to a final configuration can also be used to answer whether there exists a direction in which local translation of that part in the initial configuration is feasible. If there is a path, there is a direction in which local translation is feasible. Lozano-Pérez [24] shows one procedure to construct a path that avoids obstacles and lists the most significant literature on that subject. The procedures to construct a path, however, involve extensive computation, and therefore their use in the high level of planning will weaken the planner efficiency. One of the major advantages of hierarchical planning [31] is the possibility of abstracting the details at the high level.

I.3. Representation of Local Constraints

In most cases of two parts or subassemblies in contact, a pure rotation of one with respect to the other will not separate them. In these cases the motion must include a nonzero translational component in order to separate parts in contact. Therefore, to decide whether two parts in contact can separate from each other, the local analysis can

focus on translational motions only[7].

Let P and Q be two parts that have one planar contact. Let $\underline{n}$ be a vector perpendicular to the contact plane, and pointing towards part P. Part Q blocks translations of part P that have negative projections over $\underline{n}$. Therefore, to decide whether part P can translate by a vector $\underline{t}$, it is necessary to check whether $\underline{t} \bullet \underline{n}$ (the scalar product of $\underline{t}$ and $\underline{n}$) is greater than or equal to zero.

In general, a part P has N planar contacts with other parts: let $\underline{n}_i$ be a vector perpendicular to the plane of the i^{th} planar contact, pointing towards part P. Then, $\underline{t}$ must satisfy

$$\underline{t} \bullet \underline{n}_i \geq 0 \qquad i=1, 2, \cdots, N \tag{2}$$

in order to be a feasible translation for part P.

The set of inequalities (2) is a necessary but not sufficient condition for the *global* translation $\underline{t}$ of part P, since other parts that are not directly in contact with P may also constrain its movements. For *local* analysis, distant objects do not interfere, and system (2) becomes a sufficient condition. Moreover, if $\underline{t}_o$ satisfies system (2), so do all vectors $\gamma \underline{t}_o$ for any scalar γ greater than zero; and it is always possible to pick γ sufficiently small to guarantee that a translation by $\gamma \underline{t}_o$ of part P is feasible. Therefore, to answer whether part P can translate locally, it is sufficient to answer whether system (2) has a nonzero solution.

Each inequality in system (2) divides the space $\mathbf{R}^3$ into two halfspaces. The set of vectors satisfying the system of inequalities (2), which is the intersection of finitely many halfspaces, is a *polyhedral convex cone*. Polyhedral convex cones may have several different shapes and enumeration of these shapes will be useful for the search procedure. Figure 17 shows one example of a polyhedral convex cone which is the intersection of five halfspaces, each one defined by a plane that goes through the origin and that is perpendicular to a vector $\underline{n}_i$. Therefore, the cone can be characterized as the set of vectors that have greater than or equal to zero projection over vectors $\underline{n}_1, \underline{n}_2, \underline{n}_3, \underline{n}_4, \underline{n}_5$, which are perpendicular to the five faces of the cone, and have the appropriate (i.e. towards the inside) orientations. Alternatively, the same polyhedral convex cone can be characterized as the set of positive linear combinations of vectors $\underline{e}_1, \underline{e}_2, \underline{e}_3, \underline{e}_4, \underline{e}_5$, which have directions along the five edges of the cone, and the appropriate orientations [16].

These alternative representations of the polyhedral convex cone and their properties may be defined as follows:

Definition 5: Given a polyhedral convex cone C, any set of vectors $V = \{\underline{v}_1, \underline{v}_2, \cdots, \underline{v}_J\}$ with the property that any vector $\underline{x} \in C$ has positive projection over all vectors $\underline{v}_i \in V$ (i.e. $\underline{x} \bullet \underline{v}_i \geq 0$ for $i=1,2, \cdots, J$), is called a **tangential representation** of C.

Definition 6: Given a polyhedral convex cone C, any set of vectors $E = \{\underline{e}_1, \underline{e}_2, \cdots, \underline{e}_J\}$ such that any positive linear combination of the $\underline{e}_1, \underline{e}_2, \cdots, \underline{e}_J$ yields a vector in C and, conversely, any vector $\underline{x}$ in C can be expressed as a positive linear combination of $\underline{e}_1, \underline{e}_2, \cdots, \underline{e}_J$ (i.e. $\underline{x} = \sum_{i=1}^{J} \alpha_i \underline{e}_i$ with $\alpha_i \geq 0$ $i=1,2, \cdots, J$, if and only if $\underline{x} \in C$) is called a **point representation** of C.

Definition 7: Two distinct (point or tangential) representations of the same polyhedral convex cone are said to be **equivalent representations.**

Definition 8: A tangential representation of a polyhedral convex cone is said to be a **minimal tangential representation** if it has no equivalent tangential representation with fewer vectors.

---

[7]There are cases where two parts in contact can only separate from each other if one undergoes a pure rotation followed by a translation. These cases correspond to more complex contacts, and therefore require the use of more complex models. In products designed for assemby [2, 5], however, these more complex contacts are rare.

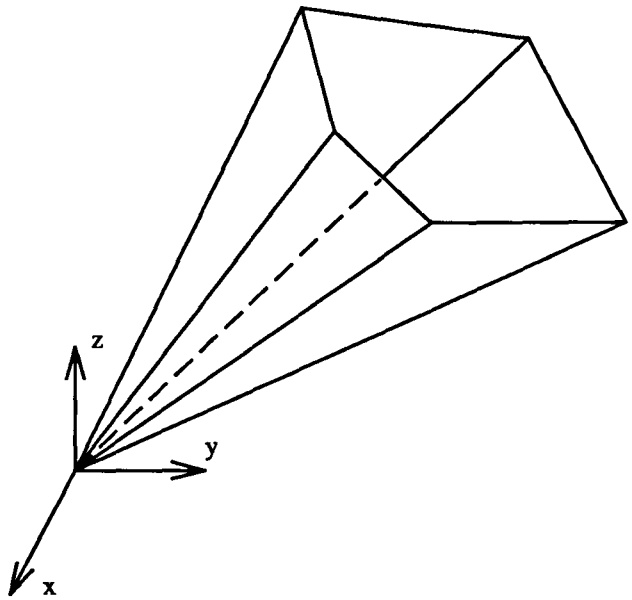


Figure 17: A polyhedral convex cone which is the intersection of five halfspaces

Definition 9: A point representation of a polyhedral convex cone is said to be a **minimal point representation** if it has no equivalent point representation with fewer vectors.

The minimal point representation of polyhedral convex cones is used in the local analysis of feasible directions of motion because one can readily check if the set of feasible nonzero translations is empty by checking whether the point representation has a nonzero vector. It is also useful as a basis for the global analysis because it allows the enumeration of the feasible translations.

In the next section, a syntax for a computer representation of polyhedral convex cones in $\mathbf{R}^3$ is developed based on the formulation described above. A procedure is defined which finds the computer representation from a tangential representation. The procedure provides the basis for testing the feasibility of *disassembly* operations.

I.4. Search Procedure for Feasible Local Translations

The solution procedure is based on enumeration of the 10 possibilities for the shape of a polyhedral convex cone in the three dimensional space $\mathbf{R}^3$. These are listed in table 4. A syntax for a computer representation of polyhedral convex cones, which incorporates explicitly their shapes is shown in figure 18. This representation is compact, yet captures all the information needed in the procedure to find the solutions of the system of inequalities (2).

Figure 19 shows procedure *SOLVE* which takes as input a tangential representation of a polyhedral convex cone, $T=\{\underline{n}_1, \underline{n}_2, \cdots, \underline{n}_N\}$ which need not be minimal, and returns its computer representation, in the syntax shown in figure 18.

Table 4: The possible shapes of polyhedral convex cones in three dimensional space

shape	definition	comments
SPACE	All points in $\mathbf{R}^3$.	This is a degenerate case in which N in the system of inequalities (2) (i.e. the number of planes) is zero.
HALFSPACE	All the points on one side of a plane and the points on the plane.	Typically, this is the case in which N in system (2) is one, but it is also the case in which all vectors $\underline{n}_1, \underline{n}_2, \cdots, \underline{n}_N$ are parallel and have the same orientation.
QUADRANT	The points within the intersection of two halfspaces whose defining planes are not parallel.	Typically, this is the case in which N is two, and the two vectors $\underline{n}_1$ and $\underline{n}_2$ are not parallel.
POLYGONAL	The points within the intersection of three or more halfspaces when no plane exists that contains all points in the cone.	This is the shape of the cone in figure 17.
PLANE	All the points that lie on a plane that goes through the origin.	This is the two dimensional correspondent of SPACE. Typically, this is the case in which N is two, and the two vectors are parallel and have opposite orientations.
HALFPLANE	All the points that lie on a plane that goes through the origin, and to one side of a line on that plane that also goes through the origin.	Typically this is the case in which N is three, two of the vectors are parallel and have opposite orientations, and the third vector is not parallel to the other two.
SECTOR	All the points that lie on a plane that goes through the origin, and also to one side of two lines on that plane that also go through the origin.	Typically, this is the case in which N is four, two of the vectors are parallel and have opposite orientations, and both the other two vectors are not parallel to any other vector.
LINE	All the points on one straight line.	One example of this is the case in which N is three and the three vectors lie on a plane, with no two vectors parallel, and no one of the three vectors can be expressed as a positive linear combination of the other two.
HALFLINE	All the points on one straight line, to one side of the origin.	Typically, this is the case in which N is five, four vectors define a line, and a fifth vector is not parallel to any one of the other four.
POINT	The origin.	This is also a degenerate case in which the only solution to the system of inequalities is the zero vector.

```
<cone> = (SPACE) | (HALFSPACE <vector>) |
         (QUADRANT (<vector> <vector>) ) |
         (POLYGONAL (<min-tangent-repr> <min-point-repr>) ) |
         (PLANE <vector>) | (HALFPLANE (<vector> <vector>) ) |
         (SECTOR <min-point-repr>) | (LINE <vector>) |
         (HALFLINE <vector>) | (POINT)

<min-tangent-repr> = (<vector-sequence>)

<min-point-repr> = (<vector-sequence>)

<vector-sequence> = <vector> | <vector> <vector-sequence>
```

Figure 18: The computer representations of cones

```
procedure SOLVE(tanrep)
solution ← (SPACE)
while FIRST(solution) ≠ POINT and tanrep is not empty do

    begin
    n ← FIRST(tanrep)
    tanrep ← TAIL(tanrep)
    solution ← INTER(solution n )
    end

return solution
end SOLVE
```

Figure 19: The procedure *SOLVE*

The solution procedure consists of finding, successively, a computer representation for the sequence of cones $C_0, C_1, C_2, \cdots, C_N$. The polyhedral convex cone C_0 is the whole space $\mathbf{R}^3$, and the polyhedral convex cones $C_1, C_2, \cdots, C_N$ have the sets $T_1, T_2, \cdots, T_N$, respectively, as (not necessarily minimal) tangential representations, where $T_i = \{\underline{n}_1, \underline{n}_2, \cdots \underline{n}_i\}$.

The computer representation of C_{i+1} is generated by procedure *INTER* using the fact that

$$C_{i+1} = C_i \cap \{\underline{x} \mid \underline{x} \bullet \underline{n}_{i+1} \geq 0\} \tag{3}$$

i.e., cone C_{i+1} contains the vectors that are in both cone C_i and in the halfspace defined by the plane perpendicular

to vector $\underline{n}_{i+1}$ and going through the origin. Procedure *SOLVE* terminates as soon as a cone of shape POINT is found in the sequence $C_0, C_1, C_2, \cdots, C_M$, since all the remaining cones in the sequence would also be of shape POINT.

Incorporating the shapes into the representations of the cones simplifies the reasoning needed to compute their intersection with a halfspace because for each shape of the cone, there are only a few possibilities for the shape of the intersection. Figure 20 shows a state diagram in which the nodes (i.e the states) correspond to the shapes of polyhedral convex cones in $\mathbf{R}^3$, and the arcs (i.e. the transitions) correspond to the possibilities for the shape of their intersection with a halfspace. The iterations performed by procedure *SOLVE* can be seen as a sequence of transitions in that state diagram. The initial state is the shape SPACE which is the shape of cone C_0. Each iteration in procedure *SOLVE* causes a transition in the state diagram, and the final state is the shape of the solution. The actual transition is computed by procedure *INTER* which also computes the necessary parameters to completely characterize the cone in the syntax of figure 18.

The inputs to procedure *INTER* are the computer representation of a cone C, and a vector $\underline{v}$. The output of *INTER* is the computer representation of cone $C' = C \cap \{\underline{x} \mid \underline{x} \bullet \underline{v} \geq 0\}$. The computation performed by *INTER* depends on the shape of the cone input. Figure 21 shows the cases in which the shape of the cone input is SPACE or HALFSPACE. The other cases, although more extensive, are not difficult to infer.

I.5. Example of the Computation of the Directions of Feasible Translations

Figure 22 shows an assembly that has two parts with seven planar contacts between them. The vectors perpendicular to the contacts and pointing towards the upper part are:

$\underline{n}_1 = (1\ 0\ 0)$ $\underline{n}_2 = (0\ 0\ 1)$ $\underline{n}_3 = (0\ 2\ 1)$ $\underline{n}_4 = (0\ {-2}\ 1)$ $\underline{n}_5 = (0\ 1\ 1)$ $\underline{n}_6 = (0\ {-1}\ 1)$ $\underline{n}_7 = (0\ 0\ 1)$

For this example, procedure *SOLVE* does seven iterations to find out the set of directions along which the upper part can translate. The first iteration produces the intersection of the whole space with the halfspace defined by the plane perpendicular to $\underline{n}_1$; the intersection is the halfspace itself, whose representation is (HALFSPACE (1 0 0)).

The second iteration produces the intersection of the halfspace obtained in the first iteration with the halfspace defined by the plane perpendicular to $\underline{n}_2$; because $\underline{n}_1$ and $\underline{n}_2$ are not parallel, the intersection has shape quadrant, and its representation is (QUADRANT (1 0 0) (0 0 1)).

The third iteration produces the intersection of this quadrant-shape cone with the halfspace defined by the plane perpendicular to $\underline{n}_3$; because $\underline{n}_1, \underline{n}_2$, and $\underline{n}_3$ are linearly independent, the intersection is a polygonal (triangular) cone whose representation is (POLYGONAL ((1 0 0) (0 0 1) (0 2 1)) ((0 1 0) (1 0 0) (0 -1 2))).

The fourth iteration produces the intersection of the polygonal-shape cone obtained in the third iteration with the halfspace defined by the plane perpendicular to $\underline{n}_4$; because the projections over $\underline{n}_4$ are less than zero for the first edge, zero for the second edge, and greater than zero for the third edge, the representation of the intersection is (POLYGONAL ((1 0 0) (0 -2 1) (0 2 1)) ((0 1 2) (1 0 0) (0 -1 2))).

The fifth, the sixth, and the seventh iterations do not change the polyhedral convex cone produced in the fourth iteration, which happens to lie entirely within the halfspace defined by the plane perpendicular to $\underline{n}_5$, the halfspace defined by the plane perpendicular to $\underline{n}_6$, and the halfspace defined by the plane perpendicular to $\underline{n}_7$. This conclusion can be made by observing that the three edges (0 1 2), (1 0 0), and (0 -1 2) have greater than or equal to zero projection over $\underline{n}_5$, $\underline{n}_6$, and $\underline{n}_7$.

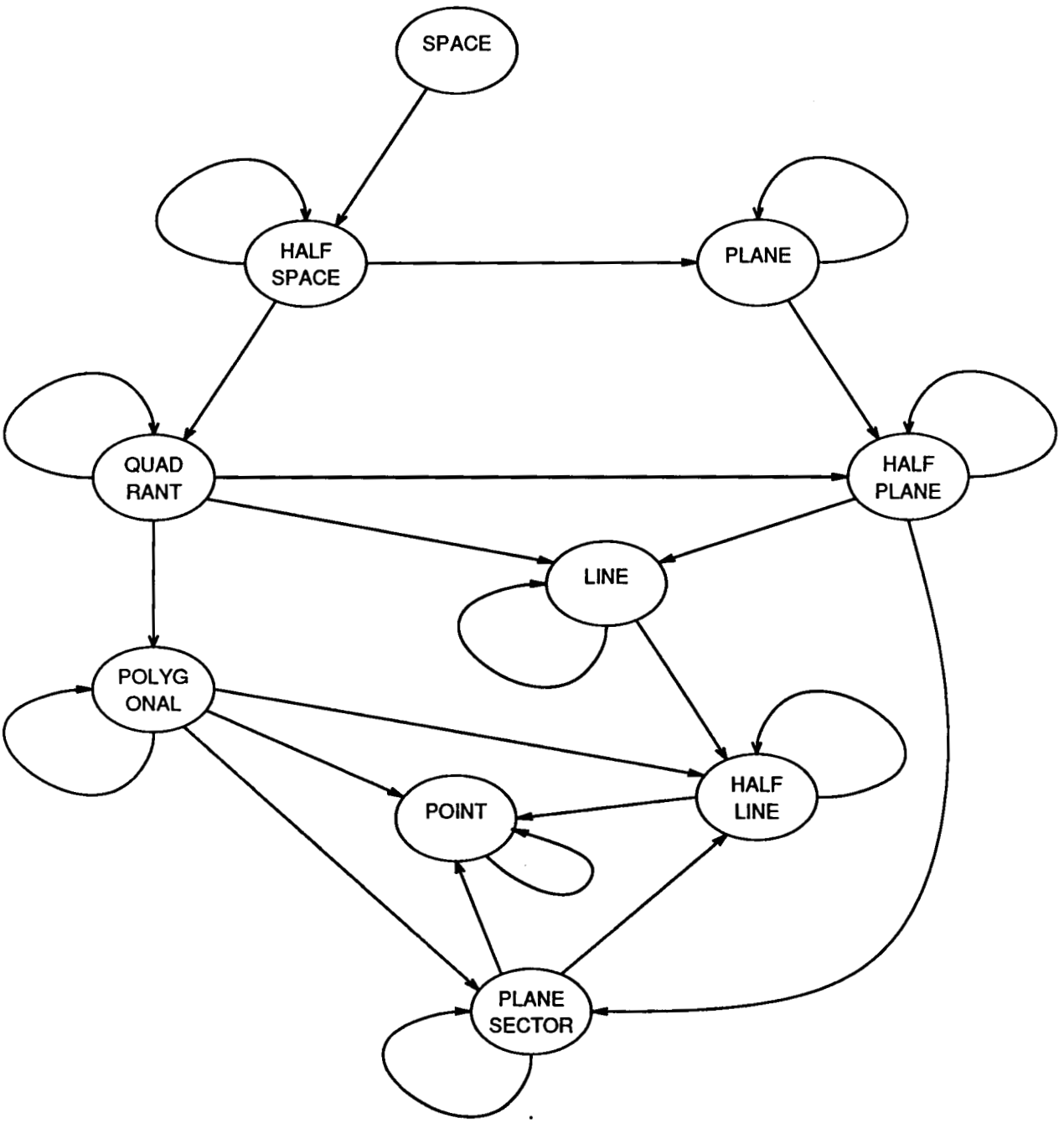


Figure 20: State diagram for procedure *SOLVE*

```
procedure INTER(cone v)
case

    (1) FIRST(cone) = SPACE return (HALFSPACE v)
    (2) FIRST(cone) = HALFSPACE do

        begin
        n ← SECOND(cone)
        if n and v are parallel do

            begin
            if n and v have the same orientation return (HALFSPACE v)
            return (PLANE v)
            end

        return (QUADRANT (v n))
        end
```

Figure 21: Part of procedure *INTER*

The final solution returned by the algorithm is the polyhedral convex cone whose computer representation is (POLYGONAL ((1 0 0) (0 -2 1) (0 2 1)) ((0 1 2) (1 0 0) (0 -1 2))); this means that any positive linear combination of the vectors (0 1 2), (1 0 0), and (0 -1 2) is a feasible translation for the upper part in figure 22. For this example, the result can be verified by inspection. The set of all directions $\underline{d}$ along which translation is feasible can be scanned systematically by letting

$$\underline{d} = a \cdot (0\ 1\ 2) + b \cdot (1\ 0\ 0) + c \cdot (0\ -1\ 2)$$

$$0 \le a \le 1$$

$$0 \le b \le \sqrt{1-a^2}$$

$$c = \sqrt{1-a^2-b^2}$$

I.6. Relations to Other Work

Within the research in robotic planning, the work of Brooks [7, 8] has some relation with the results presented in this paper. Brooks formalizes the process of checking and modifying robot plans to ensure that they will work in spite of inaccuracies of mechanical devices and the inaccuracies in the information the robot has about the position and orientation of parts within the workstation. That formalization leads to a system of (not necessarily linear) inequalities and Brooks uses a constraint manipulation system to decide whether the system has a solution and to find bounds for some functions of the variables in the plan. That constraint manipulation system, however, does not construct the set of solutions to the system of inequalities. Moreover, the conclusions drawn from that system tend

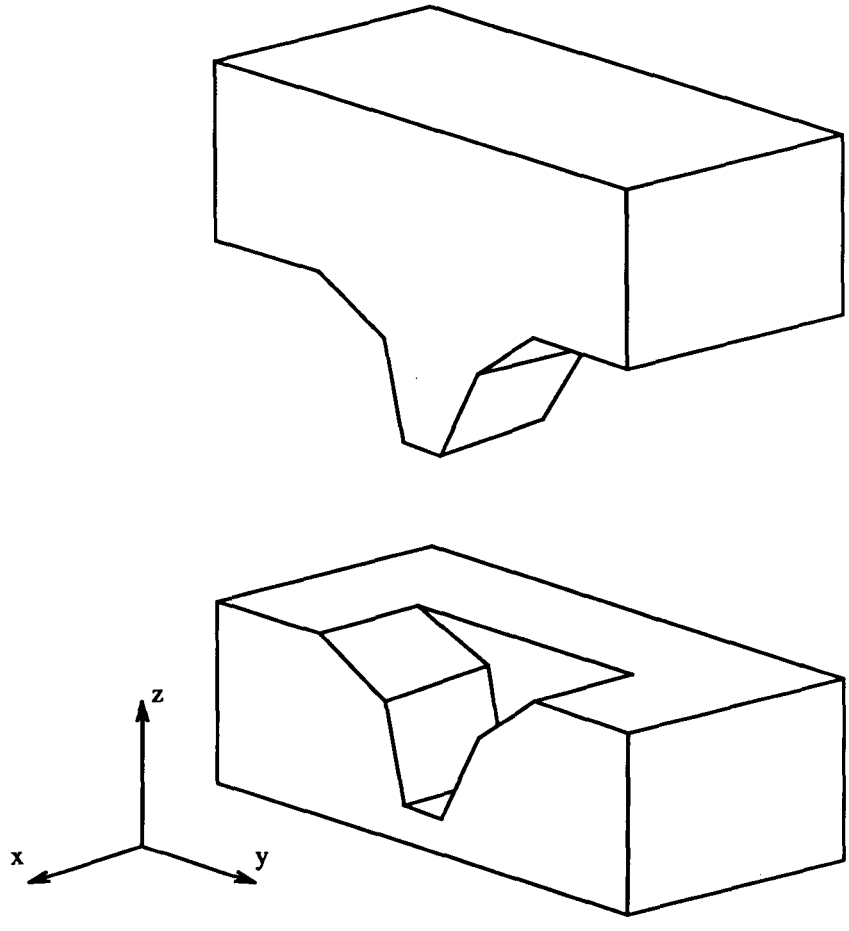


Figure 22: Two parts that have seven planar contacts

to be conservative; they are safe to be used in robot planning but they may lead to the elimination of plans that are reliable.

Systems of linear inequalities have been studied within linear programmming [36], where the extremal points of linear functions are sought in multidimensional spaces. Goldman and Tucker [16] present important theoretical results that have been used as basis for the formulation presented in this paper. Those results alone have been used by Ohwovoriole and Roth [27], in the context of mechanical assembly, to solve a system of inequalities in a five dimensional space. By restricting the dimension of the space to three, as we have done, more efficient procedures could be constructed.

In addition to being less efficient (although more general), the linear programming approach to solving systems of linear inequalities has problems in degenerate cases which are common in assembly planning. One degeneracy is the fact that the set of solutions is unbounded in all cases, except when no solution exists. Another degeneracy occurs whenever the feasible solutions lie on a plane (i.e. the set of solutions has volume zero); and this happens whenever parts have parallel faces which are in contact with other parts.

The intersection of halfspaces has been studied within computational geometry [30]. Brown [9] showed how the problem of finding the intersection of N halfspaces can be converted to the problem of finding the convex hull of N points; that leads to an algorithm that takes $O(N \log N)$ time.

Although more general, the algorithms in computational geometry have been designed for the case in which the solution set is bounded. Like the linear programming approach, these algorithms have problems in degenerate cases which are very common in assembly planning. The procedure presented in this paper, has been designed for the assembly planning problem. It is less general than those in computational geometry but is more efficient since it finds the solution in at most N steps, and it can handle the degenerate cases.

As mentioned in section I.2, the work of Asada and By [3] has some relation with the results presented in this paper. Although Asada and By modeled the contacts between the workpart and the fixturing elements as point contacts, for local translations, point contacts and planar contacts yield the same constraints. The conditions that Asada and By derive for local translations are the same conditions as equation 2 in this paper. (The reader is warned that there is an error in equation 20 of Asada and By paper; it should read $G_f \cdot \Delta q \geq 0$.) But Asada and By do not address how to determine the set of solutions to equation 2, which is their equation 20[8].

I.7. Conclusion

The problem of finding the directions of feasible local translations for a part constrained by planar contacts has been formulated mathematically as that of finding the set of solutions to a system of inequalities. The system of inequalities is represented by a polyhedral convex cone, and the solution procedure exploits the fact that in the three dimensional space $\mathbf{R}^3$ there are only 10 possibilities for the shape of a polyhedral convex cone.

A syntax for the computer representation of polyhedral convex cones in $\mathbf{R}^3$, which incorporates explicitly their shapes, is presented along with an implemented algorithm that uses that representation to produce the set of solutions. The algorithm can handle all possible cases and produces the solution in at most N (the number of planar contacts) steps. It may take less than N steps when the only solution to the system of inequalities is the zero vector.

In addition to providing the basis for testing the feasibility of assembly operations, the computer representation generated by the procedure is useful later in the assembly planning process to guide the search for a path, since it allows a systematic scan of all directions along which local translation is feasible.

[8]Since the formulation presented by Asada and By includes rotations, the resulting system of equations involves six variables. Therefore, solving that system would be more complex than solving a system of three variables as that of equation 2 in this paper.

References

[1] A. P. Ambler and R. J. Popplestone.
Inferring the Positions of Bodies from Specified Spatial Relationships.
Artificial Intelligence 6:157-174, 1975.

[2] M. M. Andreasen et al.
Design for Assembly.
Springer Verlag, 1983.

[3] H. Asada and A. B. By.
Kinematic Analysis of Workpart Fixturing for Flexible Assembly with Automatically Reconfigurable Fixtures.
IEEE Journal of Robotics and Automation RA-1(2):86-94, June, 1985.

[4] N. Boneschanscher et al.
Subassembly Stability.
In *Proceedings of AAAI-88*, pages 780-785. American Association for Artificial Intelligence, Morgan Kaufman, August, 1988.

[5] G. Boothroyd et al.
Automatic Assembly.
Marcel Dekker, Inc., New York, 1982.

[6] A. Bourjault.
Contribution a une Approche Méthodologique de L'Assemblage Automatisé: Elaboration Automatique des Séquences Opératoires.
Thèse d'État, Université de Besançon Franche-Comté, November, 1984.

[7] R. A. Brooks.
Symbolic Reasoning Among 3-D Models and 2-D Images.
Artificial Intelligence 17:285-348, 1981.

[8] R. A. Brooks.
Symbolic Error Analysis and Robot Planning.
International Journal of Robotics Research 1(4):29-68, 1982.

[9] K. Q. Brown.
Fast Intersection of Half Spaces.
Technical Report CMU-CS-78-129, Department of Computer Science - Carnegie Mellon University, June, 1978.

[10] D. Chapman.
Planning for Conjunctive Goals.
Artificial Intelligence 32(3):333-377, July, 1987.

[11] T. L. De Fazio and D. E. Whitney.
Simplified Generation of All Mechanical Assembly Sequences.
IEEE Journal of Robotics and Automation RA-3(6):640-658, December, 1987.

[12] N. Deo.
Graph Theory with Applications to Engineering and Computer Science.
Prentice-Hall, 1974.

[13] B. R. Donald.
A Search Algorithm for Motion Planning with Six Degrees of Freedom.
Artificial Intelligence 31:295-353, 1987.

[14] M. Ejiri et al.
A Prototype Intelligent Robot that Assembles Objects from Plan Drawings.
IEEE Transactions on Computers C-21(2):161-170, February, 1972.

[15] B. R. Fox and K. G. Kempf.
Opportunistic Scheduling for Robotics Assembly.
In *1985 IEEE International Conference on Robotics and Automation*, pages 880-889. IEEE Computer Society, 1985.

[16] A. J. Goldman and A. W. Tucker.
Polyhedral Convex Cones.
In Kuhn, H.W. and Tucker, A.W. (editors), *Linear Inequalities and Related Systems*, pages 19-40. Princeton University Press, 1956.

[17] L. S. Homem de Mello and A. C. Sanderson.
AND/OR Graph Representation of Assembly Plans.
In *AAAI-86 Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1113-1119. American Association for Artificial Intelligence, Morgan Kaufmann Publishers, 1986.

[18] L. S. Homem de Mello and A. C. Sanderson.
Task Sequence Planning for Assembly.
In *IMACS World Congress '88 on Scientific Computation*. Paris, July, 1988.

[19] A. Jain and M. Donath.
Knowledge Representation for Robot Based Assembly Planning.
In *Proceedings of the 1987 American Control Conference*, pages 181-187. American Automatic Control Council, IEEE, Piscataway NJ 08854, June, 1987.

[20] A. Koutsou.
Planning Motion in Contact to Achieve Parts Mating.
PhD thesis, University of Edinburgh, 1986.

[21] C. L. Liu.
Introduction to Combinatorial Mathematics.
McGraw-Hill, 1968.

[22] T. Lozano-Pérez.
Task Planning.
In M. Brady et al. (editors), *Robot Motion: Planning and Control*, pages 473-498. The Massachusetts Institute of Technology, 1982.

[23] T. Lozano-Pérez.
Robot Programming.
Proceedings of the IEEE 71(7):821-841, July, 1983.

[24] T. Lozano-Pérez.
A Simple Motion Planning Algorithm for General Robot Manipulators.
In *AAAI-86 Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 626-631. American Association for Artificial Intelligence, Morgan Kaufmann Publishers, August, 1986.

[25] M. M. Lui.
Generation and Evaluation of Mechanical Assembly Sequences Using the Liaison-Sequence Method.
Master's thesis, Massachusetts Institute of Technology, May, 1988.
Also published as Report CSDL-T-990, The Charles Stark Draper Laboratory, Inc.

[26] M. Mani.
Kinematic Synthesis of Workholding Fixtures.
Report 860901, Artificial Intelligence & Factory Automation Group, Gould Research Center, and Northwestern University, October, 1986.

[27] M. S. Ohwovoriole and B. Roth.
An Extension of Screw Theory.
Transactions of the ASME, Journal of Mechanical Design 103(4):725-735, October, 1981.

[28] R. J. Popplestone et al.
RAPT: A Language for Describing Assemblies.
The Industrial Robot 5(3):131-137, September, 1978.

[29] R. J. Popplestone et al.
An Interpreter for a Language for Describing Assemblies.
Artificial Intelligence 14:79-107, 1980.

[30] F. P. Preparata and M. I. Shamos.
Computational Geometry.
Springer-Verlag, 1985.

[31] E. D. Sacerdoti.
A Structure for Plans and Behavior.
Elsevier North-Holland, 1977.

[32] A. C. Sanderson.
Parts Entropy Methods for Robotic Assembly System Design.
In *IEEE 1984 International Conference on Robotics and Automation*, pages 600-608. IEEE Computer Society Press, 1984.

[33] A. C. Sanderson and L. S. Homem de Mello.
Planning Assembly/Disassembly Operations for Space Telerobotics.
In W. C. Chiou (editor), *Space Station Automation III*, pages 109-115. SPIE - The International Society for Optical Engineering, November, 1987.
Proceedings of SPIE - Volume 851.

[34] A. C. Sanderson and L. S. Homem de Mello.
Task Planning and Control Synthesis for Flexible Assembly Systems.
In Wong, Andrew K. C. and Pugh, Alan (editors), *Machine Intelligence and Knowledge Engineering for Robotic Applications*, pages 331-353. Springer-Verlag Berlin Heidelberg, 1987.
NATO Advanced Science Institutes Series, Vol. F33.

[35] Y. Shoham.
Naive Kinematics: one aspect of shape.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 436-442. International Joint Conferences on Artificial Intelligence, Inc., August, 1985.

[36] M. Simonnard.
Linear Programming.
Prentice Hall, 1966.

[37] R. H. Taylor.
A Synthesis of Manipulator Control Programs From Task-Level Specification.
Memo AIM 282, Stanford Artificial Intelligence Laboratory, July, 1976.

[38] J. M. Valade.
Geometric Reasoning and Automatic Synthesis of Assembly Trajectory.
In *Proceedings of the 85 International Conference on Advanced Robotics*, pages 43-50. Japan Industrial Robot Association, IFS Publications Limited and North Holland Elsevier Scientific Publishers, 1985.

[39] M. A. Wesley et al.
A Geometric Modeling System for Automated Mechanical Assembly.
IBM J. Res. Develop. 24(1):64-74, January, 1980.