AUTOMATIC DISASSEMBLY AND TOTAL ORDERING IN THREE DIMENSIONS

Tony C. Woo

Department of Industrial & Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

Technical Report 87-9

March 1987

Automatic Disassembly and Total Ordering in Three Dimensions

Tony C. Woo

Department of Industrial and Operations Engineering

The University of Michigan

Ann Arbor, Michigan 48109-2117

March 1987

ABSTRACT

Generating a sequence of motions for removing components in a three-dimensional assembly, one at a time, is considered -- the robot motion being strictly translational. We map the boundary representation of an assembly to a tree structure called Disassembly Tree (DT). Traversing the DT in in-order yields a minimal sequence for disassembly. Traversing the DT in pre- or post-order yields a minimal sequence for assembly.

In this paper, an assembly is classified by the logical complexity of its DT (as ordered graphs whose nodes are components in the given assembly) and by the geometric complexity of the nodes in DT (in terms of the number of motions needed to remove a single component). Whether a component can be removed in one motion is next described as a predicate. Then, the predicate is used in an algorithm for constructing a DT. For a class of assemblies that exhibit total ordering, the algorithm decides whether all the components can be removed in a single motion by constructing a DT in O(N) time, on the average, where N is the total number of mating faces in the assembly.

1. INTRODUCTION

Automatic assembly may be studied from the point of view of disassembly. If we visualize the process of assembly as having a finite number of "states", each being captured by a single frame on film, then rolling the film backward corresponds to disassembly. The process of assembly/disassembly is reversable, if the components are rigid and there is no internal energy stored. (Such would not be the case with components of variable geometry such as a spring or a clip fastener.)

Given a collection of components, a goal in automatic assembly is to arrive at a logical sequence with preferably the shortest total travel between the component bins. Reversing the process, we are given a finished assembly and are to arrive at a minimal sequence for disassembly. A possible approach may be an "Onion Peeling" procedure -- one starts from the outside and works inward, by exploiting the given structure of an assembly.

While we are appealing to the intuition that automatic disassembly may be more feasible than automatic assembly, it is useful to examine the inherent structure of assemblies that is independent of the approach. We shall introduce three measures of complexity -- geometrical, logical, and dimensional.

The "Sofa Moving Problem" [5] is an instance of a <u>geometrically</u> complex assembly/disassembly problem in which an object (possibly non-convex) is to be navigated through corridors and obstacles without collision. If we associate a sofa to a single component being removed, then motion planning is concerned with the generation of a geometric solution (the shortest path, for instance) under constraints (that the obstacles are immobile and that collision is to be avoided, for example).

Assembly planning, on the other hand, is concerned with the generation of a logical sequence of steps, each of which may be a geometrically simple motion plan. The "Tower of Hanoi" is an instance -- its solution is well-known [1]. For less structured instances, such as stacking blocks given a non-unique initial configuration, more powerful reasoning mechanisms employing artificial intelligence may be needed[8]. Indeed, when the problem is logically and geometrically complex, it can be very difficult. The "Warehouseman's Problem", in which sofas are to be unstacked and removed, is known to be PSPACE-complete [2].

The degrees of freedom permitted in planning is another measure of complexity -- that of dimensionality. When the motion is restricted to translation only, and the domain is in the plane, the solution for the Sofa Moving Problem is reduced from a doubly exponential time algorithm [6] to one that is of polynomial time. If N is the number of walls, the motion for a circular disk can be planned in O(NlogN) time [3] and for two disks in $O(N^3)$ time [7].

While there are other parameters influencing the complexity of the assembly/disassembly problem, such as the number of jigs and fixtures, and the number of robots simultaneously performing the task, they will be incorporated into these three measures -- geometrical, logical and dimensional, in the following characterization. First, we identify the notion of "clearing" a component from a subassembly. The criterion may be:

C1. that the convex hull of the component in question does not intersect the convex hull of the subassembly; or

C2. that the component can be translated and rotated to the perimeter of a sphere of sufficiently large radius; or

C3. that the component can be translated to infinity.

For simplicity, we adopt C3 as the criterion in this paper.

By allowing three degrees of freedom for translation in three dimensions, we have specified the domain of our problem, as illustrated by Figure 1.1, along the axis of "Dimensionality".

<Insert Figure 1.1>

Next, we specify the "Logical" complexity of our problem. Let the solution to disassembly be represented by a tree, the nodes of which represent the components. In Figure 1.2 (a), the assembly A has four interlocking components. It can be disassembled not individually but only by removing a subassembly (a_1a_2) first.

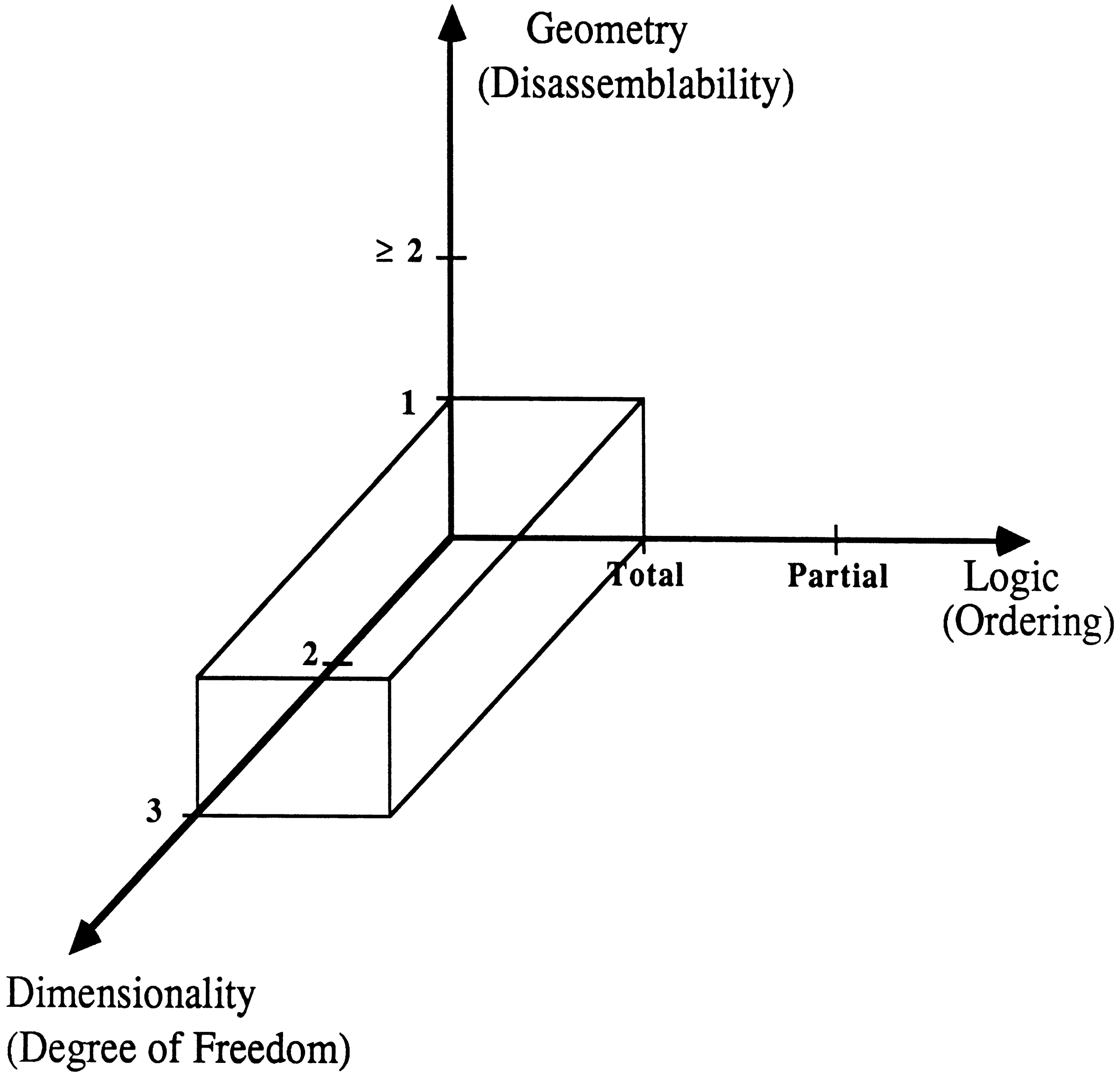


Figure 1.1 Complexity of Assemblies

Definition 1.1 An assembly is k-parallel if the disassembly of a component requires the immediately prior clearing of k components in parallel.

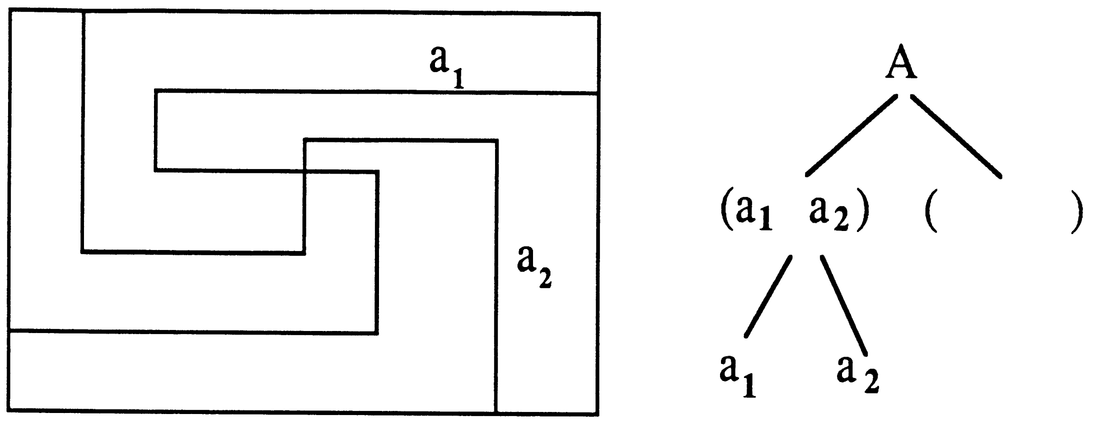
A k-parallel assembly is one that requires k robots working simultaneously. Alternateively, it may be viewed as one requiring a special jig that "joins" the k components under consideration, while using one robot. (The trade-off between an active processor, such as a robot, and a passive one, such as a fixture, while interesting, is not within the scope of this paper.)

<Insert Figure 1.2>

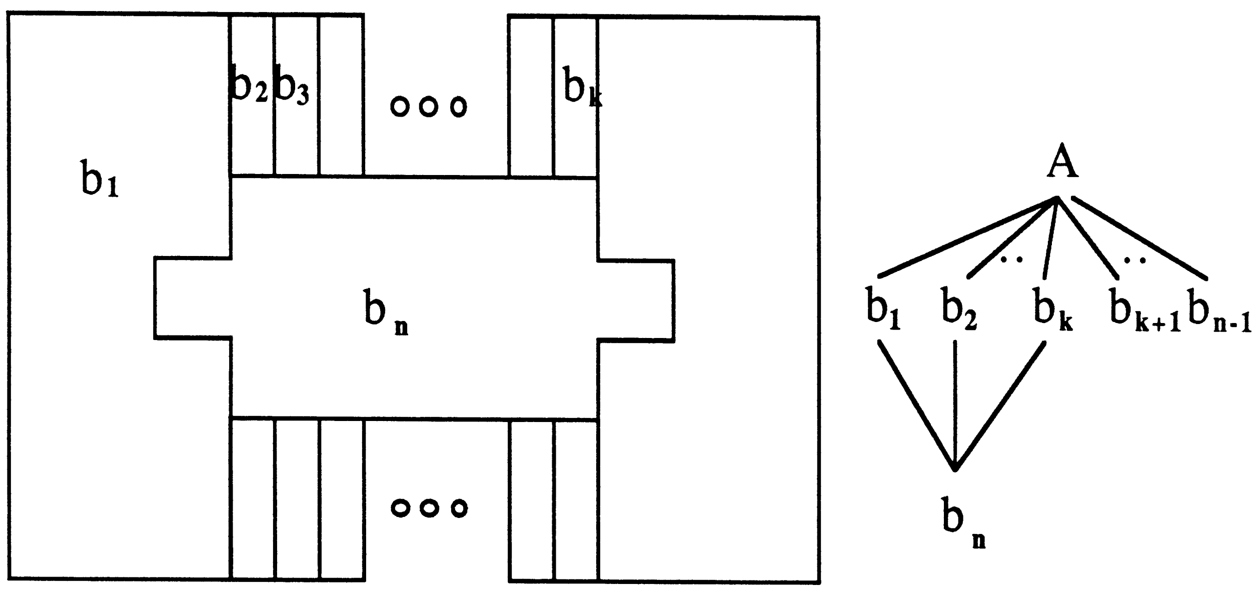
Now consider the component b_n in Figure 1.2(b). The tree in Figure 1.2 (b) shows that disassembly of b_n is possible after k of its neighbors have been cleared, not necessarily in parallel.

Definition 1.2 An assembly is k-sequential if the disassembly of a component requires the prior disassembly of k other adjacent components.

(It may be noted that parallelism is a necessity while sequentialism is sufficiency. In other words, a k-sequential assembly, though requiring only one robot k times, can be disassembled by using k robots operating in parallel, once. However, a k-parallel assembly cannot be disassembled using one robot k times.)



(a) Parallel Assembly



(b) Sequential Assembly

Figure 1.2 Partially Ordered Assemblies

Thus, observing the number of "parents" of a node in a disassembly tree, we have a characterization of assemblies by "Logical" ordering.

Definition 1.3 An assembly is partially ordered if the disassembly of a component requires an immediately prior disassembly of k components. If $k < 2$, the assembly is totally ordered.

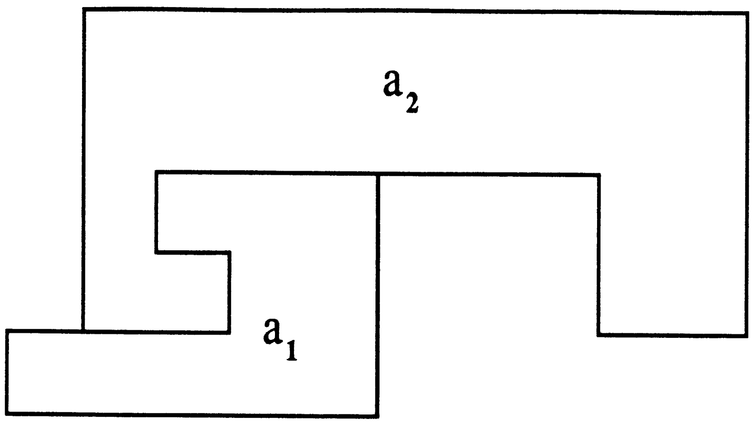
In this paper, we consider assemblies that are totally ordered. Corresponding to each component, a node in the disassembly tree has exactly one parent.

Finally, we examine the Geometric complexity of assemblies as prescribed by Figure 1.1. Consider the assembly shown in Figure 1.3 (a). The clearing of component a_1 requires two translations. Recalling the general "Sofa Moving Problem", in which multiple motions are required, we have a characterization for the nodes in a disassembly tree (whose structure is characterized by partial or total ordering).

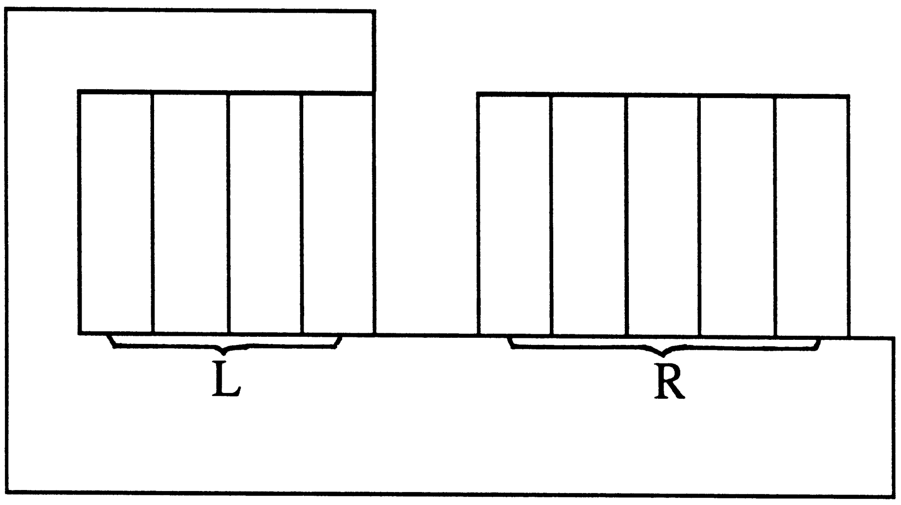
<Insert Figure 1.3>

Definition 1.4 A component is m-disassemblable if m motions are needed to clear it.

While a k-parallel assembly differs from a k-sequential one in the number of processors, an m-disassemble component differs from an n-disassemblable one by the amount of storage. By Definition 1.4, an m-disassemblable component requires (m-1) temporary



(a) 2-Disassemblable



(b) 1-Disassemblable

Figure 1.3 Totally Ordered Assemblies

locations before it reaches its destination. Viewed as a program, a motion plan for an m-disassemblably component requires (m-1) assignments or temporary variables. (While the trade off between the number of processors and the amount of storage is an issue, we shal return our focus to characterization.) In this paper, we consider assemblies that are totally ordered, for which each component is 1-disassemblable with three degrees of freedom.

Constructing a disassembly tree for three dimensional, totally ordered, 1-disassemblable components is not entirely trivial. One of the difficulties comes from the observation that improper ordering of the components affects the m-disassemblability of each component. Consider Figure 1.3 (b) in which there are L components on the left and R components on the right. If removed first, each of the L components are 2-disassemblable. However, after all R components are first removed, the L components become 1-disassemblable. To determine which one of the k = (L + R) components is 1-disassemblable suggest a decision procedure that could take k + (k-1) + (k-2) + ... steps leading to an $O(k^2)$ time algorithm. In this paper we give a linear time algorithm.

The remaining sections are organized as follows. Section 2 discusses the determination of 1-disassemblability. Section 3 presents an algorithm for ordering the components in the form of a disassembly tree.

2. COMPONENT DISASSEMBLABILITY

In this paper, we assume that components in an assembly "touch" each other, i.e. they are not disjoint nor do they intersect (occupying the same space). For simplicity, we shall first consider assemblies with only two components -- one being the component under consideration for removal and the other representing the subassembly (the rest of the assembly). Compare the two examples in Figure 2.1 (a) and (b), whose exploded views are given by Figure 2.1 (c) and (d), respectively. It is clear that component c_1 is disassemblable with respect to subassembly S_1 while component c_2 is not disassemblable with respect to S_2. We can distinguish these two types by computing a range of directions along which a component can be translated such that it does not interfere with the subassembly.

< Insert Figure 2.1 >

Definition 2.1 A component c is 1-disassemblable (or just disassemblable, for short) from a subassembly S in the direction d if, for all points p in c, a RAY (p,d), which is a semi-infinite line from p in the direction d, does not intersect S.

If a robot can "see" all points p of component c, from infinity, in the opposite direction of RAY (p,d) then an interference-free disassembly is ensured. We illustrate the notion of disassemblability by RAY (p,d) in Figure 2.2. (For ease of illustration, a two-dimensional cross-section is given.)

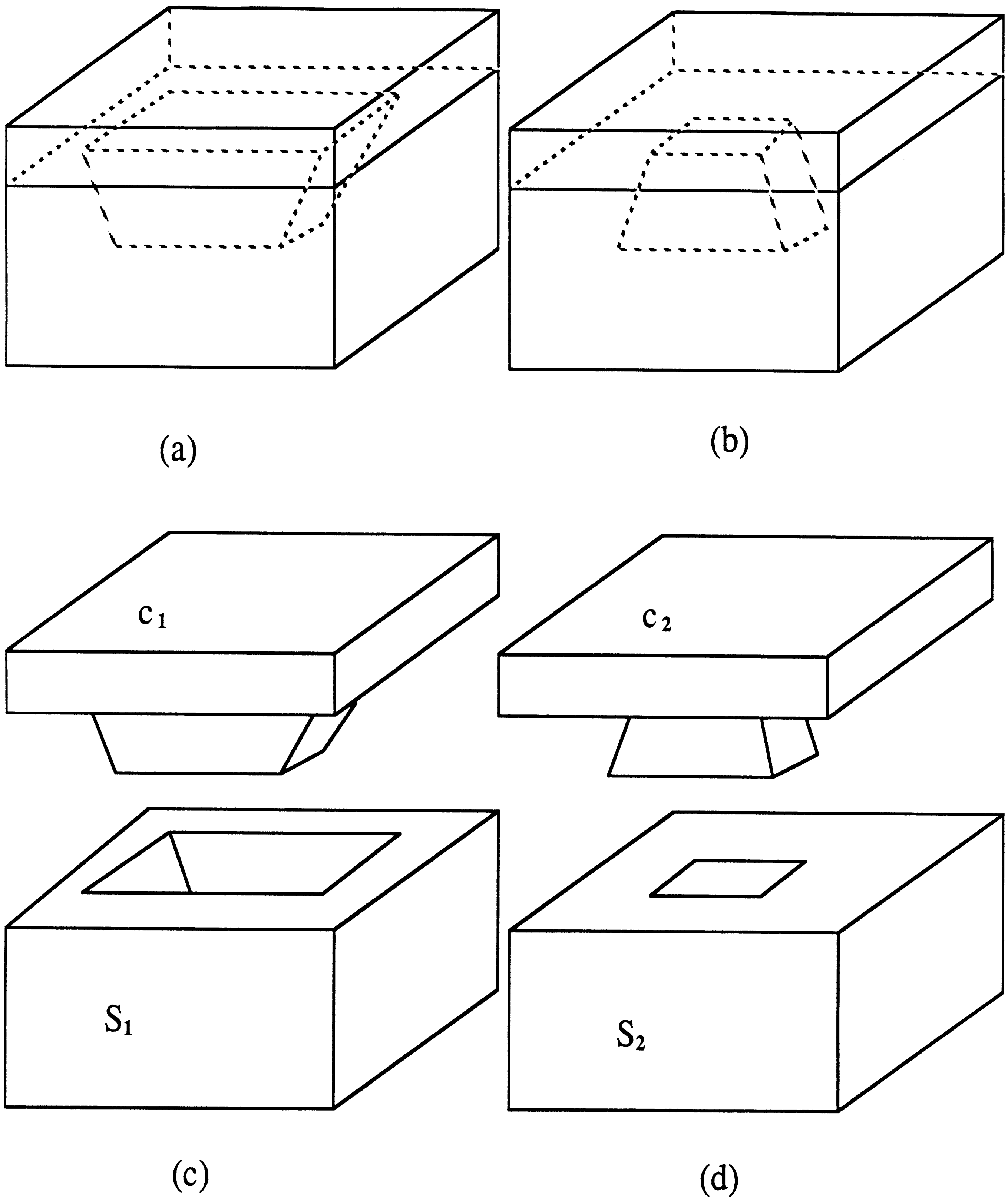


Figure 2.1 Disassemblability

In Figure 2.2(a), directions d_1 and d_2 satisfy Definition 2.1, for all points p in component c_1. By contrast, directions d_3 and d_4 do not, for all p in c_1. In Figure 2.2 (b), there does not exist a direction d such that RAY (p,d), for all p in c_2, satisfies Definition 2.1. While Ray ($p_1 d_1$) seems satisfactory, the same direction d_1 applied to point p_2 causes interference. Similarly, RAY (p_3,d_2) does not satisfy disassemblability though d_2 is satisfactory for p_2.

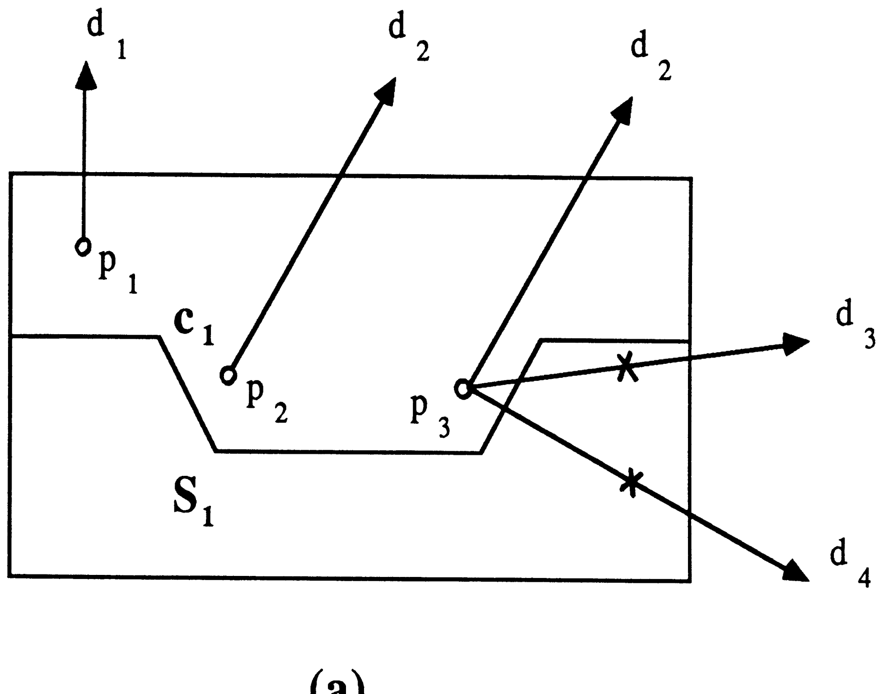
<Insert Figure 2.2>

The crucial test for disassemblability, according to Definition 2.1, is that the direction d must yield a non-intersecting RAY (p,d) for <u>all</u> points p in the component under consideration. As there are uncountably many points in each component, we need to identify a <u>finite</u> set based on which we compute disassemblability.

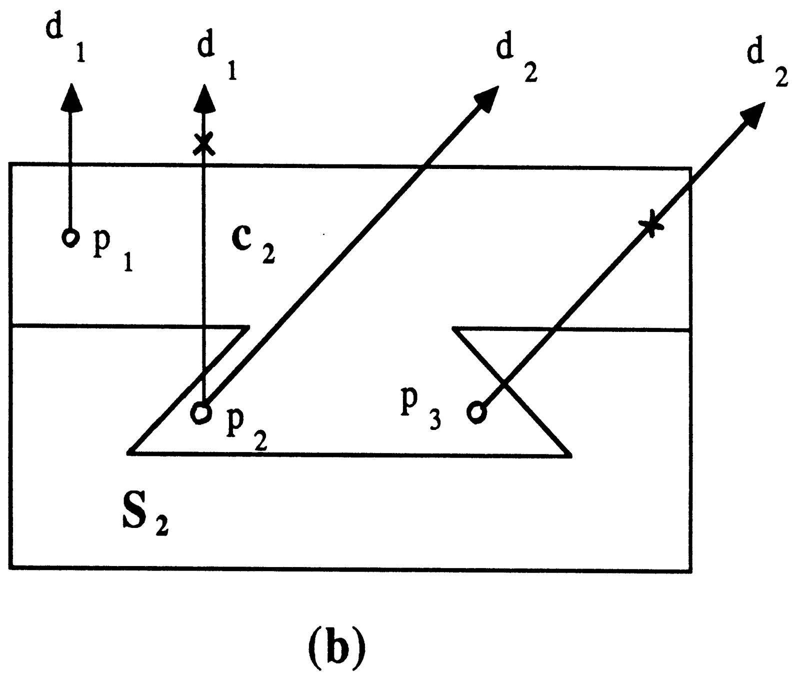
In the following sections, we compute the directions for disassembly from the normals of the <u>mating faces</u>, a subset of the boundary shared by a component and the subassembly. (Readers with intuition may choose to go directly to Section 2.2 in which the procedure is given.) For completeness, we need to show that consideration of mating face normals is not only necessary but also sufficient.

2.1 <u>Mating Faces</u>

Disassembly, according to Definition 2.1, involves two



(a)



(b)

Figure 2.2 Direction of Disassembly

entities: a point and a direction. For a given direction, we need to verify that it satisfies all the points in the component under consideration. The following lemma shows the sufficiency of considering only the boundary.

Lemma 2.1 For a given direction d, component c is disassemblable if RAY (p,d) does not intersect subassembly S, for all points p on the boundary of c.

> [Proof] Let a point p be in the interior of component c. The RAY (p,d) makes an odd number of intersections with the boundary of c. Let the intersections by p_1, p_2, p_3 ... p_{n+1}, where n is an even integer starting at 0. Since p is in c, it is necessarily outside S. The RAY (p,d) makes an even number of intersection with the boundary of S, possibly none. Let these intersections be q_1, q_2, ... q_{m-1}, where m is an odd integer starting at 1. Referring to Figure 2.3, we see that it is not possible to have q_i between p and p_1, and in general between p_{even} and p_{odd}, in the direction of RAY (p,d). Thus, there is no loss of generality by treating p as p_1 and as being on the boundary of c, since p lies between p_{even} and p_{odd} and intersection with S cannot occur in that interval.

<Insert Figure 2.3>

While Lemma 2.1 provides the basis for computing

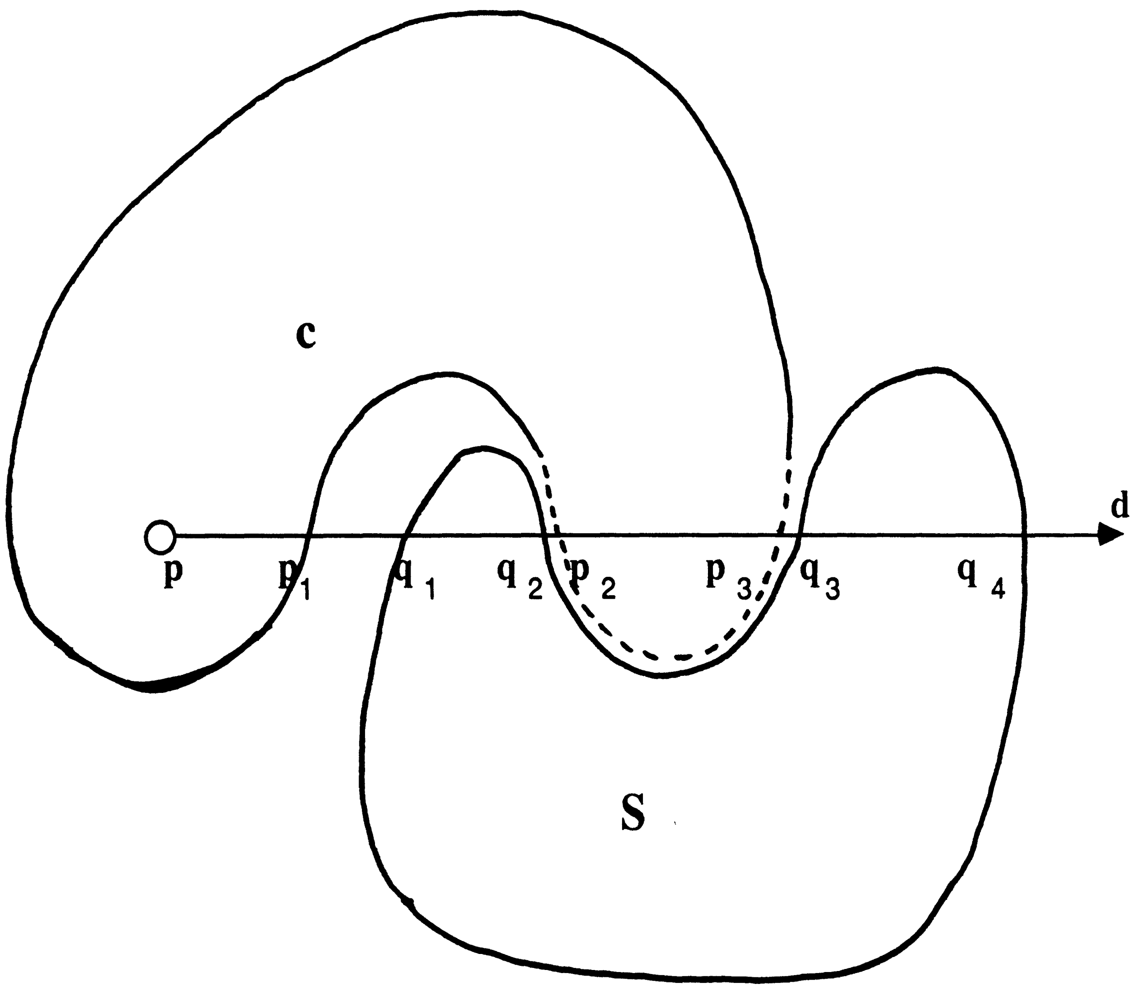


Figure 2.3 Proof of Lemma 2.1

disassemblability (by replacing the interior by the boundary), we still need to show that only the portion of boundary in common between the component and the subassembly is necessary for computation.

Lemma 2.2 Disassembly in the sense of Definition 2.1 can be determined if and only if there is a common boundary between component c and subassembly S.

[Proof] Divide the boundary of c into two disjoint subsets, one of which may be empty. In Figure 2.4, we denote the subset in common with that of the subassembly S by B_{cs} and the other B'_{cs}. We show that B_{cs} dominates B'_{cs} by the following case analyses.

Case 1. [Ray from a point in B'_{cs} does not intersect S.] Let p_1 be in B'_{cs}. Suppose RAY (p_1,d_1) does not intersect S. But a ray in the same direction from a point in B_{cs}, p_3 for example, will intersect S.

Case 2. [Ray from a point in B'_{cs} intersects S.] Let p_2 be a point in B'_{cs}. Suppose RAY(p_2,d_2) intersects S. A ray in the same direction from any point in B_{cs}, p_4 for example, must also intersect S.

Case 3. [Ray from a point in B_{cs} does not intersect S.] Suppose RAY (p_3,d_3) does not intersect S, where p_3 is in B_{cs}. By the same reasoning in the

Proof for Lemma 2.1, a ray in the same direction from another point in B'_{cs}, p_1 for example, cannot intersect S.)

Case 4. [Ray from a point in B_{cs} intersects S.] Suppose RAY (p_4,d_4) intersects S, where p_4 is in B_{cs}. A ray in the same direction from a point in B'_{cs} does not have to intersect S -- RAY (p_1,d_1), for example.

<Insert Figure 2.4>

2.2 Monotonicity

Having identified the mating faces as being necessary and sufficient for computing the direction for disassembly, we proceed to describe the procedure.

Suppose the unit normal of a face point inward (towards the material side of the component). We can map the unit normals of the mating faces to the origin of a coordinate system. This idea is illustrated (using the same two assemblies) in Figures 2.5 (a), (c), and (e) for the one that is disassemblable. Figures 2.5 (b), (d) and (f) illustrate the one that is not disassemblable. The notion of montonicity [4] of faces enables the determination of disassemblability.

<Insert Figure 2.5>

Definition 2.2 A set of faces $\{f_i\}$ with unit normals $\{d_i\}$ is

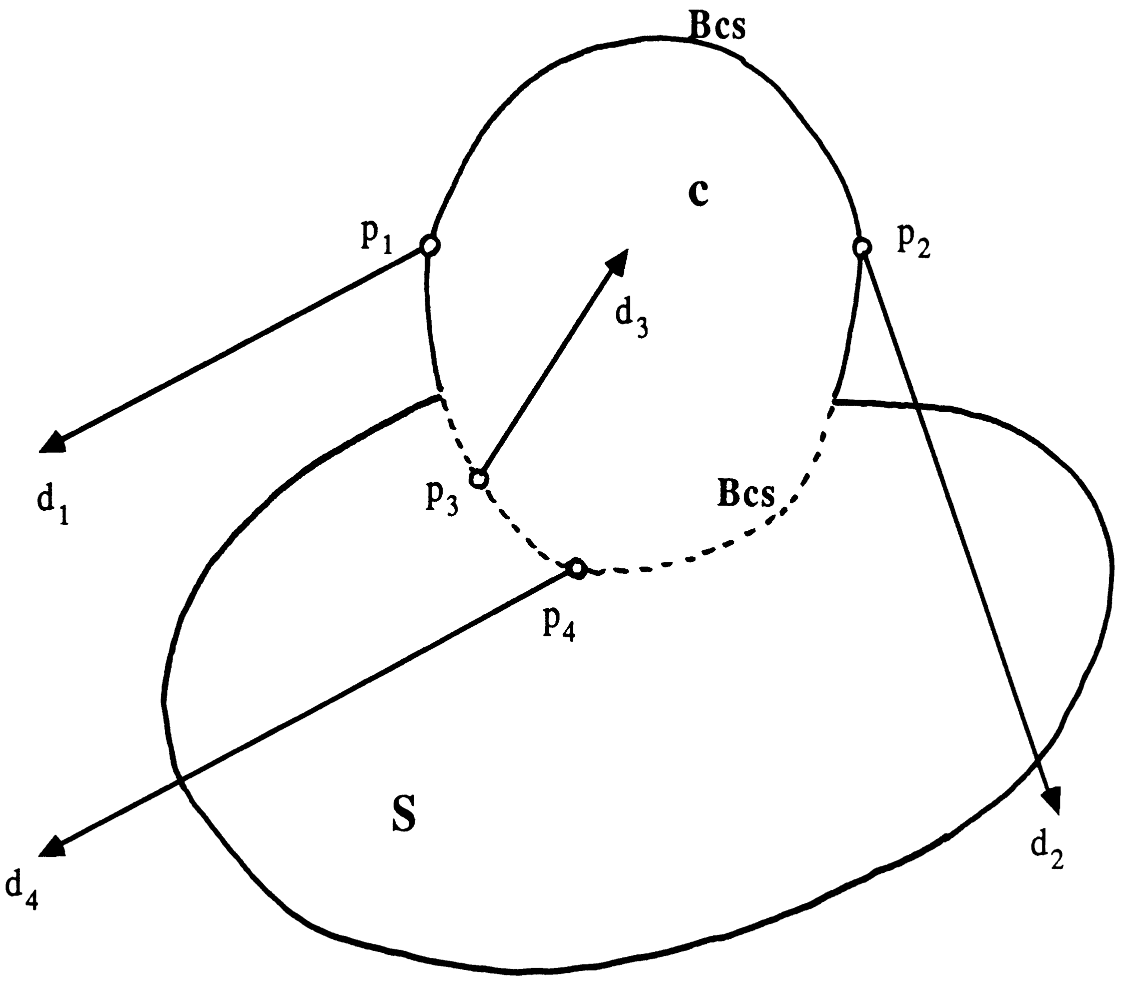


Fig 2.4 Proof of Lemma 2.2

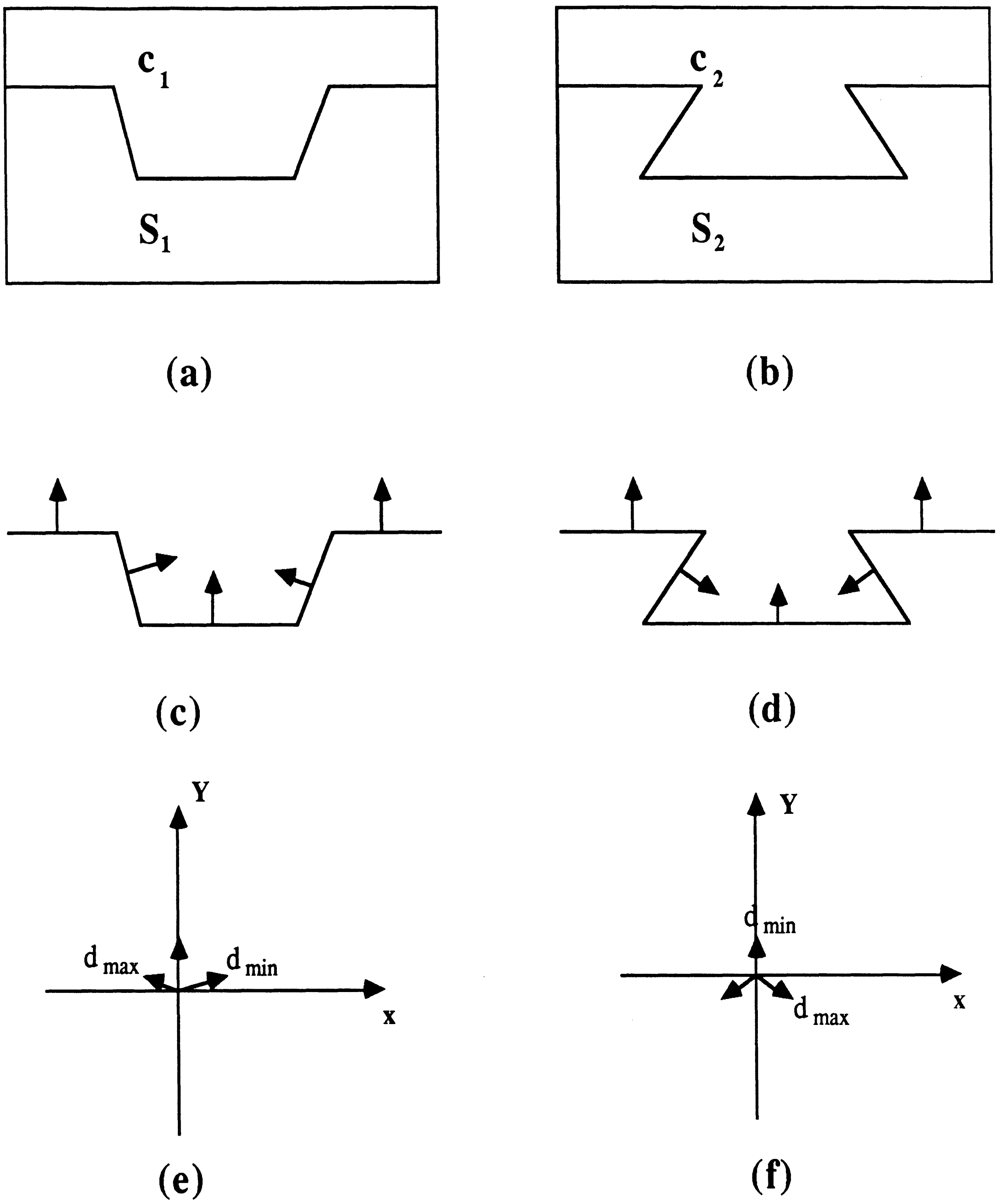


Figure 2.5 Monotonicity of Mating Faces

<u>monotone</u> if all the normals lie on one side of a plane.

As shown by Figure 2.5 (e), the unit normals of the mating faces of the component in Figure 2.5 (a) lie on one side of the plane containing the x-axis, for example. Note that there are only two delimiting normals -- d_{min} and d_{max}, where "min" and "max" denote the minimum and the maximum of the Euler angles. The following procedure determines monotonicity as well as a range of directions for a given set of mating faces.

```
Function Disassemblable (c)
    1. $\{f_i\}$  <--  mating faces of c
    2. $\{d_i\}$  <--  unit normals of $\{f_i\}$
    3. $d_{min}$  <--  MIN ($\{d_i\}$)
    4. $d_{max}$  <--  MAX ($\{d_i\}$)
    5. if $d_{max} - d_{min} \geq 180^o$
            then return 'false'
            else return $\{d_{min}, d_{max}\}$
End
```

The following lemma asserts the linear time behavior of the procedure for determining component disassemblability.

<u>Lemma 2.3</u> The disassemblability of a component c with respect to a subassembly S can be determined in O(n) time, where n is the number of mating faces in c.

[Proof] Let the cardinality of $\{f_i\}$ be n. Clearly, steps 1 and 2 in Function Disassemblable take O(n) time.

Steps 3 and 4 also take O(n) time since no sorting is required.

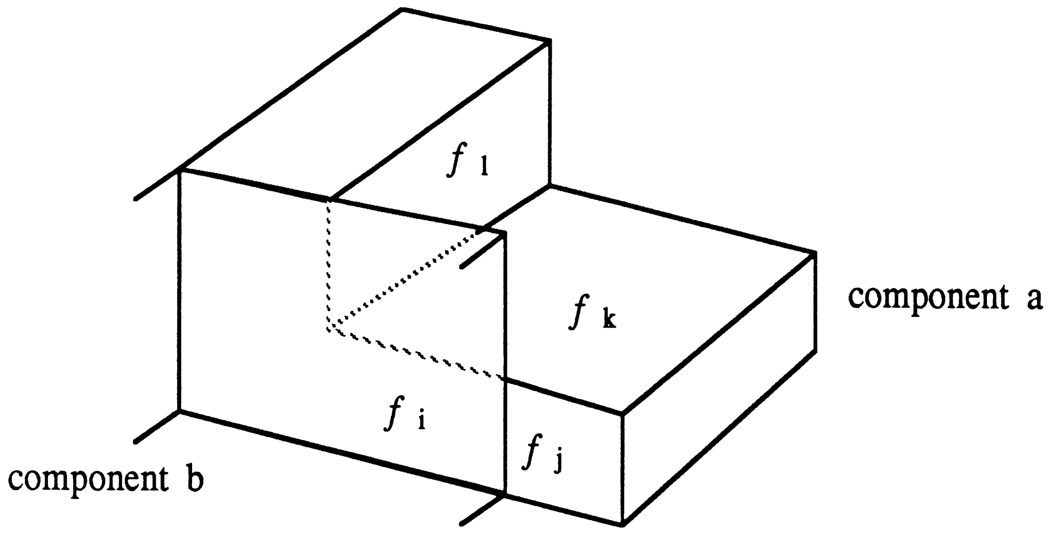
We are ready to consider the generation of DT with more than two components.

3. DISASSEMBLY TREE

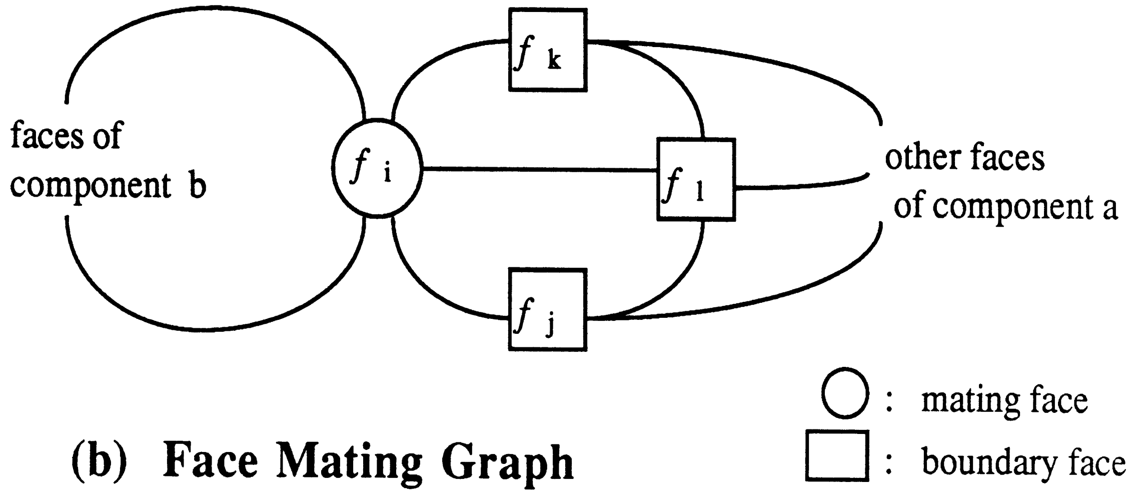
The input to the algorithm consists of two data structures. A face mating graph FMG contains nodes that are faces of each component and arcs that denote mating. Each face belongs to exactly two mates. If one of the two mates is the background, then the face is a boundary face. Otherwise, it is a mating face. FMG is used as input to Function Disassemblable. In Figure 3.1 (a), faces of two mating components are shown. The L-shaped face is split into two faces -- f_i is a mating face and f_j is a boundary face. In Figure 3.1 (b), a mating face is denoted by a circle and a boundary face by a square in FMG. A second data structure, component mating graph CMG, is a weak dual of FMG. Its nodes are components and its arcs denote mating. CMG is used in determining the ordering. Figure 3.1 (c) shows two components a and b mating at face f_i. Mating with boundary B is denoted likewise. Both FMG and CMG can be constructed from a boundary representation of a solid model such as [9] in linear time.

<Insert Figure 3.1>

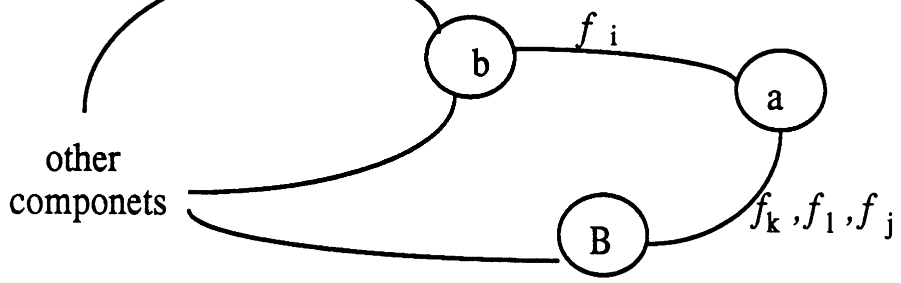
Our strategy is to begin from the outside. We take a component with a boundary face and test it for disassemblability. If it is disassemblable, we delete its boundary faces from FMG, change its mating faces to boundary faces, delete the component from CMG and insert it in DT. Next, we exploit the adjacency relation in the CMG. If a component is found to be disassem-



(a) Faces of Two Components



(b) Face Mating Graph



(c) Component Mating Graph

Figure 3.1 Data Structure

blable, its mating components are the next candidates. If each component has m mates then the algorithm would run in O(mk) time. We need to show that m, on the average, is a constant.

Lemma 3.1 Each node in the component mating graph CMG with k nodes has, on the average, (3 - 6/k) nodes that are adjacent.

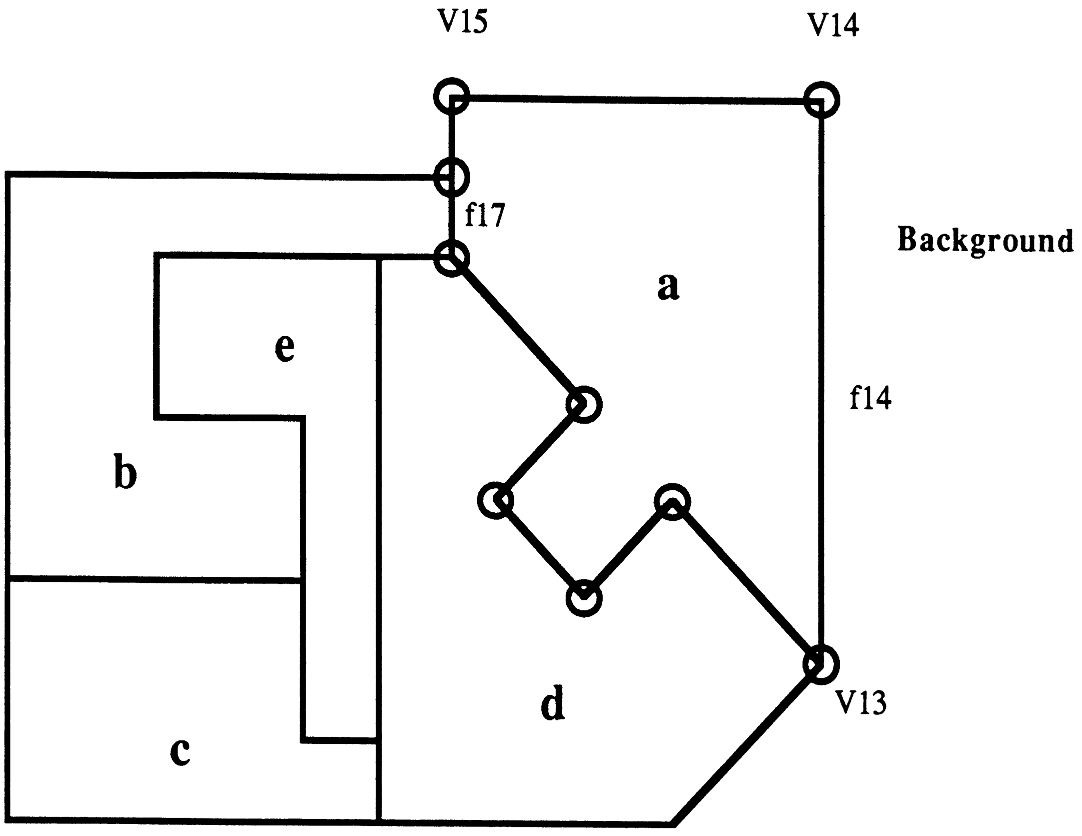
> [Proof] The CMG with k nodes is a three-dimensional polyhedron P with k vertices. If the faces of P are triangulated three are a maximum of (3k - 6) edges in P[10]. Averaged over k vertices, each vertex has (3 - 6/k) edges in CMG. Hence, each component has, on the average, (3 - 6/k) mating components.

Thus, the first "layer" of components disassembled are the ones with boundary faces. The second and subsequent layers are the mates of the ones that have been disassembled. We are ready to illustrate the algorithm.

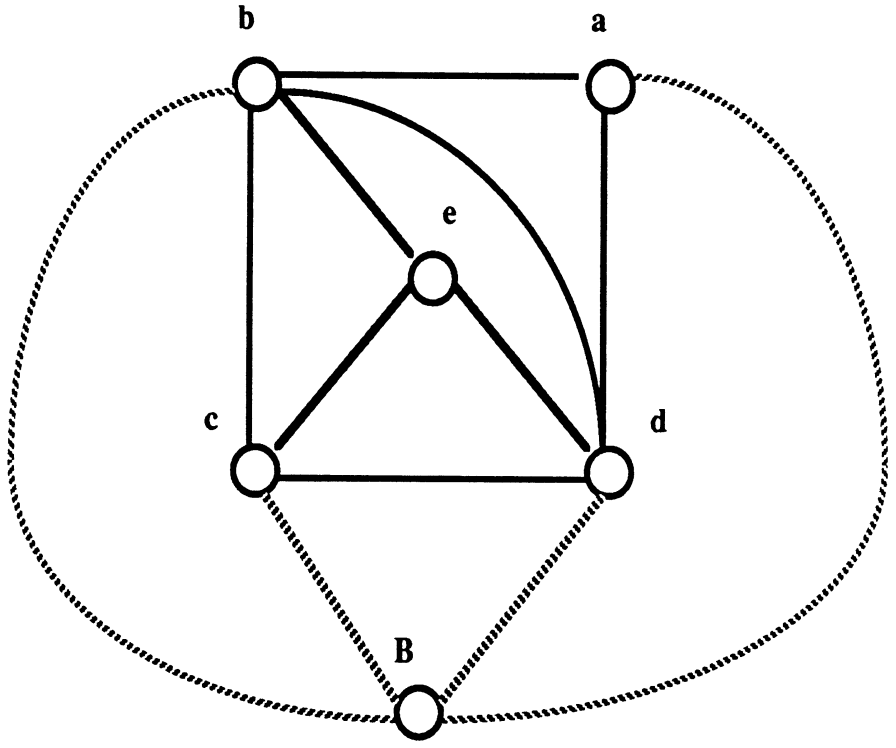
3.1 Single Disassembly

For ease of visualization, we shall take the cross-section of a three-dimensional assembly and combine it with FMG. In Figure 3.2 (a), we see five components: a, b, c, d and e. Face f_{17} is a mating face between a and b. Face f_{14} is a boundary face belonging to a. The CMG in Figure 3.2 (b) shows the mating of the components with each other as well as with background B.

<Insert Figure 3.2>



(a) Face Mating Graph



(b) Component Mating Graph

Figure 3.2 Example

The generation of a disassembly tree for the example is illustrated in Figure 3.3. The empty disassembly tree DT is initialized with a root node A (denoting the entire assembly). From FMG, we find four candidate components a, b, c, and d that have boundary faces as shown in Figure 3.3 (a). Each of the candidates is tested for disassemblability with respect to a subassembly. (For component d, the subassembly is A - d.) It is found to be false after calling the function Disassemblable, therefore, not attached to the tree as illustrated by a shaded circle. The total number of attached nodes, not counting the root node A, is recorded in an integer variable NODES.

<Insert Figure 3.3>

We next expand the tree by retrieving from CMG the mates of each of the leaf nodes and by testing their disassemblability. If a retrieved mate occurs elsewhere in the tree, it is discarded from further consideration. In Figure 3.3 (b), the component a has two mates b and d. Since b occurs previously, it is marked by a circle and not considered for further testing. The mate d has not occurred previously and is tested for disassemblability. Following the example, it is found that d is disassemblable, hence inserted in the tree, and NODES is incremented by 1. Next, the mates for component b are retrieved. They are shown as a, c, d, e in Figure 3.3 (c). Since a, c, d, has occurred previously, they are excluded from further consideration. The mate e is tested and is found to be disassemblable. At this point, the count for NODES equals the total number of components in the

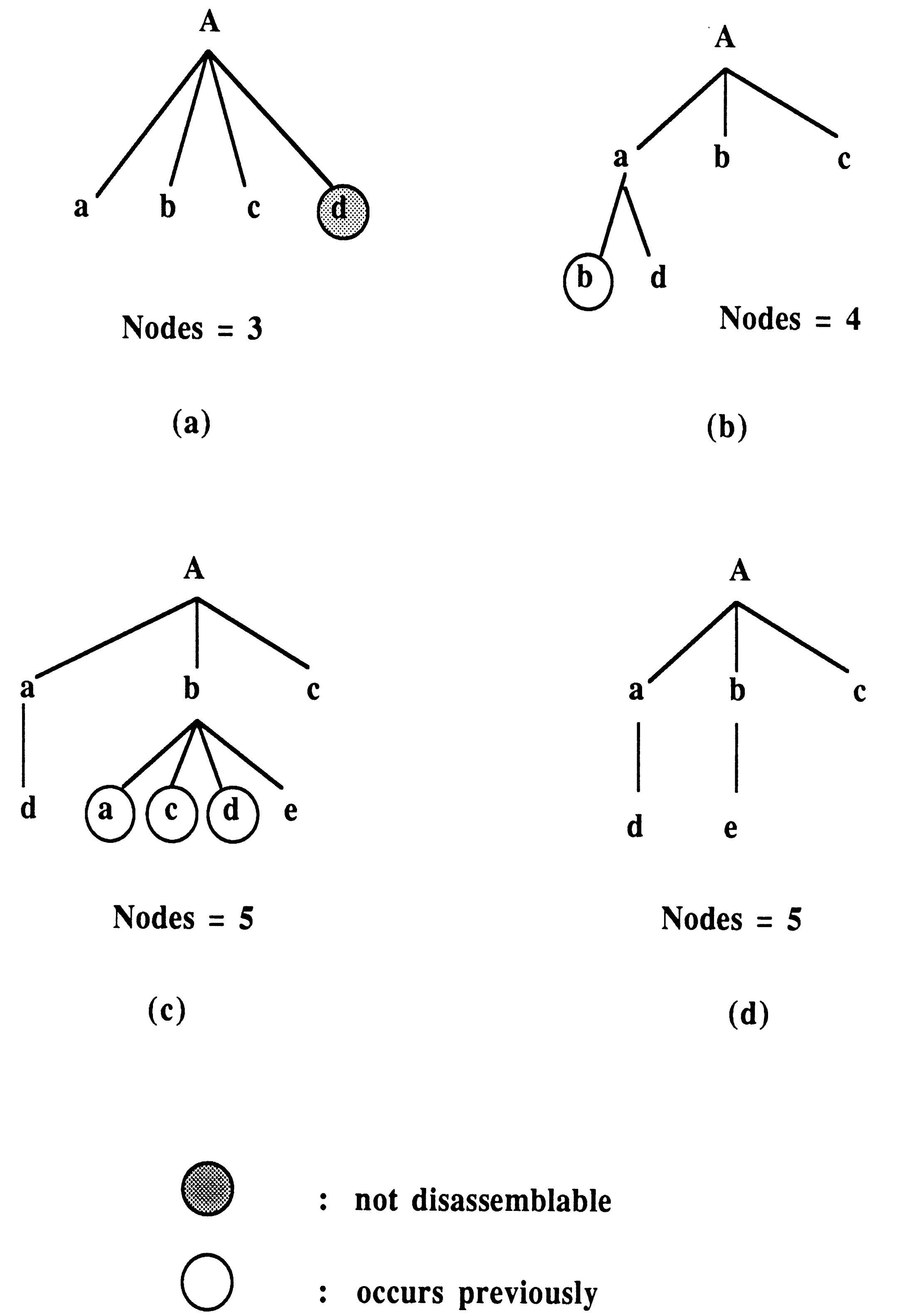


Figure 3.3 Generation of a Disassembly Tree

assembly and the process terminates. The resulting tree is shown in Figure 3.3 (d).

It is useful to show that the DT thus constructed yields a minimum number of removals (motions)in order to have access to a certain component. For example, suppose component e in Figure 3.2 (a) is to be removed. According to the DT generated in Figure 3.3 (d), only component b has to be removed (rather than components a and d, for example). The following lemma which asserts the minimality of the number of removals is shown by induction on the height of DT.

Lemma 3.2 The disassembly tree DT is optimal in that the path from the root of DT to any node is the shortest possible when traversed top-down (in in-order).

[Proof] At initialization, DT has a root node and its height is 0. The first layer of components with boundary faces gives DT a height of 1. Now, suppose the height of DT is (h - 1). We examine the cases in which a new node (corresponding to a component c) is to be inserted in DT.

Case 1. [Component c appears elsewhere in DT]
If the component appears elsewhere in DT, it will not be considered again, by construction. Hence the number of removals to disassemble component c is no greater than (h - 1).

Case 2. [Component c is not yet in DT]

If the component has not appeared before, its position in the tree can be at most height h.

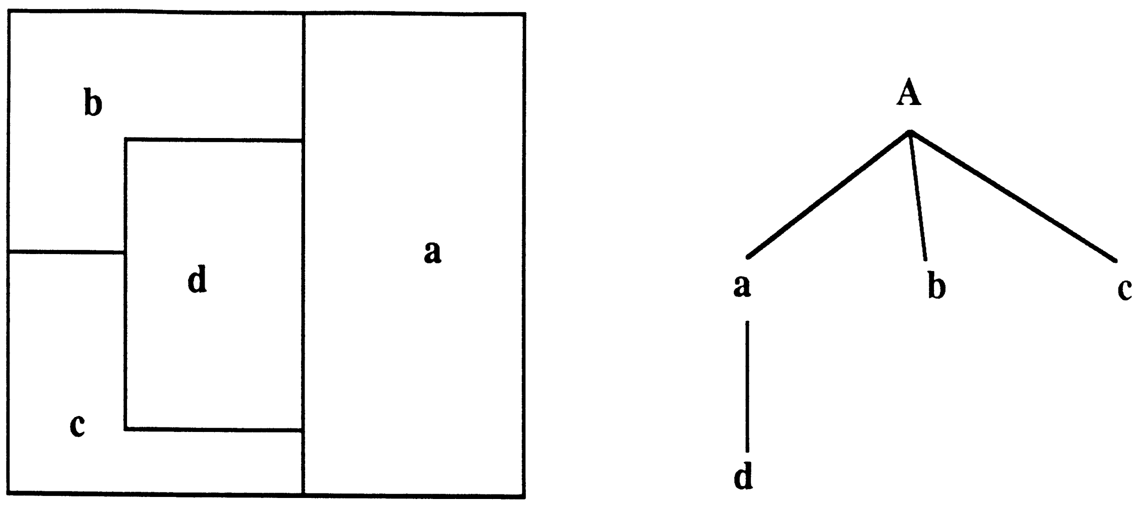
Since a node appears exactly once in DT and since its position is as near the root as possible, the number of removals is minimum. Hence, DT is optimal.

3.2 Multiple Disassembly

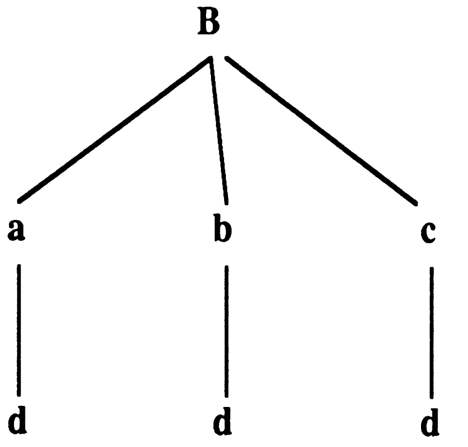
The DT generation illustrated previously gives an optimal solution in storage. However, if multiple components are to be removed from the assembly, it may not give the optimal solution in time. Consider the assembly in Figure 3.4 (a) and suppose that two components c and d are to be removed from it. (This is quite conceivable in a repair and maintenance application in which there are multiple defective components.) The DT with root A requires the removal of three components a, d, and c. On the other hand, if the components to be removed (c and d) are in the same path in the DT with root B, as shown in Figure 3.4 (b), only two removals are necessary.

< Insert Figure 3.4 >

To generate all optimal sequences for multiple component removal, two modifications are necessary. First, disqualifying a retrieved mate from disassemblability test should be postponed. Second, the termination condition for the algorithm should not be the total number of components. In other words, if none of the



(a) One Disassembly Sequence



(b) All Disassembly Sequences

Figure 3.4 Multiple Disassembly

retrieved mates at that level are added to the tree then the algorithm should terminate.

We now present Algorithm Multiple_Disassemble reflecting the two modifications. It is made compact by treating the first level of the tree as components that mate with the boundary. Thus, the root (the first new leaf of an empty tree) of the DT is B.

```
Algorithm Multiple_Disassemble
     input:  FMG and CMG
     output: DT with multiple occurrences of components


[Initialization]
     1.  Insert boundary B as NEW_LEAF in DT
     2.  LEVEL <-- 0
[Grow tree]
     3.  While NEW_LEAF at LEVEL is not empty do
          For each NEW_LEAF at LEVEL
               retrieves its mates from CMG
               LEVEL <-- LEVEL + 1
          For each mate M of NEW_LEAF at LEVEL
               If M is not at level < LEVEL and M is Disassemblable
                    then insert M as NEW_LEAF in DT
                    else return 'failure'.
```

As a component can now appear more than once at the same level in the tree, Algorithm Multiple_Disassemble performs more work than in the case of single disassembly. However, by Lemma 3.1, the average number of edges per node in CMG is bounded by 3. Since each mating face is tested at most twice (for its two mating components), the total work for Algorithm Multiple_Disassemble is at most 6N or O(N), where N is the sum of n mating faces over k components. This proves our next theorem.

Theorem 4.1 All minimal sequences for multiple disassembly can be constructed on the average in O(N) time, where N is the total number of mating faces in the assembly.

4. CONCLUSION

We have presented an algorithm for computing disassembly sequences by generating a disassembly tree DT. Traversing a DT in in-order yields a disassembly sequence. Traversing it in pre- or post-order yields an assembly sequence.

If there are k components each with m mates, it is possible that in the worst case it takes O(mk) time to construct a DT. Since the expected value of m is bounded by a constant, our algorithm runs in O(k) time on the average. To compute disassemblability, we involve n, the number of mating faces for each component. Summed over k components, the time complexity for constructing a DT is O(N), where N is the total number of mating faces in the assembly.

REFERENCES

1. Aho, A.V., Hopcroft, E., and Ullman, J.D. Data Structures and Algorithms, Addison-Wesley, 1983, pp. 306-306.

2. Hopcroft, J.E., Schwartz, J.T. and M. Sharir, "On the complexity of Motion Planning for Multiple Independent Objects: PSPACE Hardness of the Warehouseman's Problem", Int. J. of Robotics Research, Vol. 3, No. 4, 1984, pp. 76-88.

3. O'Dunlaing, C. and Yap, C., "A Retraction Method for Planning the Motion of a Disc", J. of Algorithms, Vol. 6, 1985, pp. 104-111.

4. Preparata, F.P. and Supowit, K., "Testing a Simple Polygon for Montonicity", Information Processing Letters, Vol. 12, No. 4, 1981, pp. 161-163.

5. Reif, J., "Complexity of the Mover's Problems and Generalizations", Proc. 20th IEEE Symp. on Foundations of Computer Science, 1979, pp. 421-427.

6. Schwartz, J.T. and Sherir,M., "On the Piano Mover's Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds", Advances in Appl. Math, Vol. 4, 1983, pp. 298-351.

7. Schwartz, J.T. and Sharir, M., "On the Piano Movers' Problem: III. Coordinating the Motion of Several Independent Bodies: The Special Case of Circular Bodies Moving Amidst Polygonal Barriers", Int. J. of Robotics Research, Vol. 2, No. 3, 1983, pp. 46-75.

8. Winston, P.H., Artificial Intelligence, Addison-Wesley, 1977, pp. 157-164.

9. Woo, T.C., "A Combinatorial Analysis of Boundary Data Structure Schemata", IEEE Computer Graphics & Applications, Vol. 5, No. 3, 1985, pp. 19-27.

10. Woo, T. C. and Wolter J.D. "A Constant Average Time and Linear Storage Data Structure for Three-Dimensional Objects", IEEE Trans. Systems, Man and Cybernetics, Vol. SMC-14, No. 3, 1984, pp. 510-515.