OPTIMAL SEQUENTIAL DISASSEMBLY IN THE PLANE

Tony C. Woo
Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

Technical Report NO. 87-4

December 1986

OPTIMAL SEQUENTIAL DISASSEMBLY IN THE PLANE

Tony C. Woo

Department of Industrial and Operations Engineering

The University of Michigan

Ann Arbor, Michigan 48109-2117

U.S.A.

December 1986

ABSTRACT

Generating a sequence of motions for removing components in an 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.

For certain repair and maintenance applications, it may be required that several defective components be removed. An algorithm is developed for the generation of a DT, the traversal of which corresponds to the removal of more than one component. It is shown to take, on the average, O(N) time, where N is the total number of mating edges in the assembly.

1. INTRODUCTION

Motion planning in robotics can be described informally as navigating an object in a room with obstacles. The "Sofa Moving Problem" [4] is an instance in which a collision-free path from source to destination is sought. If we associate a sofa in a warehouse with a component in an assembly then motion planning is concerned with the generation of a geometric solution that achieves some objective function (shortest path from initial to final position) under certain constraints (without collision). Assembly planning is concerned with the generation of a logical sequence of steps each of which is a motion plan. The "Warehouseman's Problem" [2] is an apt characterization of the assembly planning problem in which several sofas are to be unstacked in order to reach a particular sofa, which is proven to be PSPACE-complete.

The difficulty of motion planning and assembly planning lies in the degrees of freedom of the object under consideration. If we restrict the motion to translation, significant improvement can be made in the efficiency of the algorithm from doubly exponential [5] to polynomial time. For example, if N is the number of walls, the motions for a circular disk can be planned in 0 (N log N) time [3], and for two disks in 0 (N^3) time [6]. We are motivated by the practicality of restricting our assembly robot to translational motions (with two degrees of freedom in the plane). For an assembly of k components with a total of N

edges, we show that it can be constructed, one component at a time, in O(N) time.

We study the problem of assembly 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 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.)

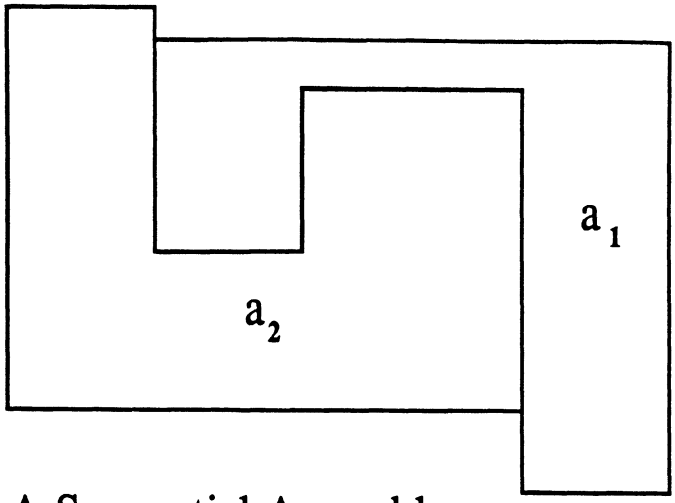
To define the domain, we characterize the inherent difficulty of the problem by the logical complexity of the solution. If the components of an assembly can be removed one at a time we call it a sequential assembly. If, on the other hand, several components must be removed in parallel, we call it a parallel assembly. Consider the assembly shown in Figure 1.1(a). Component a_1 can be removed by a manipulator while component a_2 is held by a fixture, or vice versa. Now, consider the assembly shown in Figure 1.1(b). None of the components can be removed individually. However, disassembly is possible if components b_1 and b_2 are removed "simultaneously". (We can consider a parallel assembly of k components as requiring k manipulators operating in parallel or as requiring one manipulator that removes a fixture that keeps the k components together during the removal.) Thus, in parallel, components b_1 and b_2 can be removed either by two manipulators or by assigning them to a

temporary subassembly variable b_{12} and removing it by a single manipulator. In this paper, we deal with assemblies that are sequential.

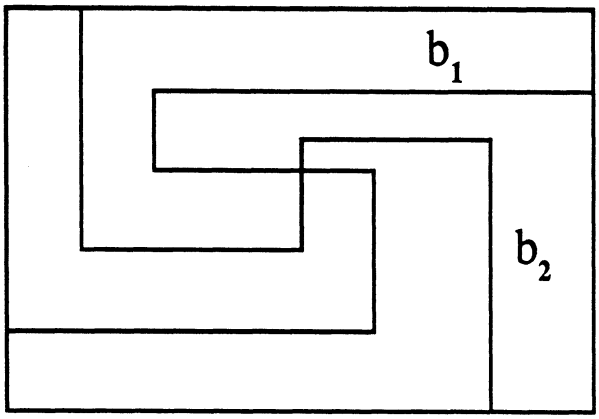
< Insert Figure 1.1 >

Let us take a closer look at sequential assemblies by comparing component a_1 in Figure 1.1(a) to component c_1 in Figure 1.1(c). The component a_1 can be removed in one step from the assembly by translating it in the positive Y direction. The removal of c_1, on the other hand, involves two translations -- one in the negative X direction and another in the positive Y direction. We say that c_1 is 2-disassemblable. It is not difficult to see that the sofa in the Sofa Moving Problem is a k-disassemblable component, where k>1. In this paper, the components in a sequential assembly are all assumed to be 1-disassemblable.

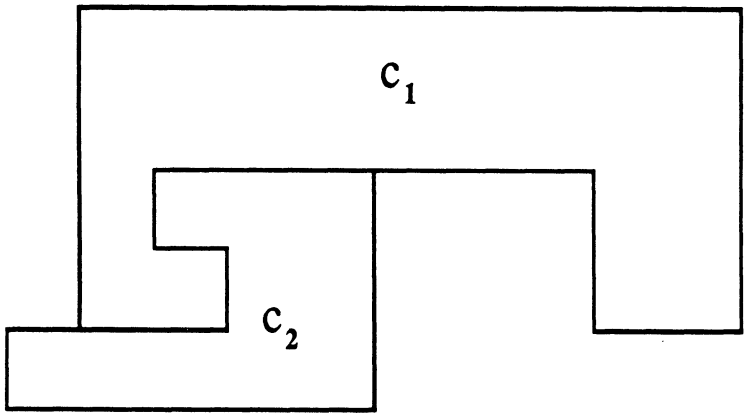
While we have defined our problem domain to be sequential assemblies whose components are 1-disassemblable, the objective of a fast algorithm for disassembly is not necessarily within immediate reach. Consider the assembly with L + R components stacked as shown in Figure 1.2. Suppose there are L of them to the left and R of them to the right. If the R components are removed first, then the L components become 1-disassemblable. However, if the L components are attempted first, they are at best 2-disassemblable. But, to determine which one of the k = (L + R) components is 1-disassemblable suggests a decision pro-



(a) A Sequential Assembly



(b) A Parallel Assembly



(c) A 2-Disassemblable Sequential Assembly

Figure 1.1 · Types of Assemblies

cedure that could take k + (k-1) + (k-2) + ... steps resulting in an $O(k^2)$ time algorithm. In this paper, we give a linear time algorithm for constructing a disassembly tree. An example is given in Figure 1.3.

<Insert Figures 1.2 and 1.3>

Traversing a disassembly tree depth-first gives the level of the component. In a disassembly tree with k nodes, the height (the level of a terminal node) is not necessary log k. Revisiting the assembly in Figure 1.2, we see that the height of its corresponding disassembly tree is O(k) if there are k/2 components on the left. The observation that a disassembly tree is not necessarily balanced leads to the requirement in its construction: that a node corresponding to a component resides at the minimum level possible. Our O(N) algorithm ensures the minimality of the path from the root of the disassembly tree to any node.

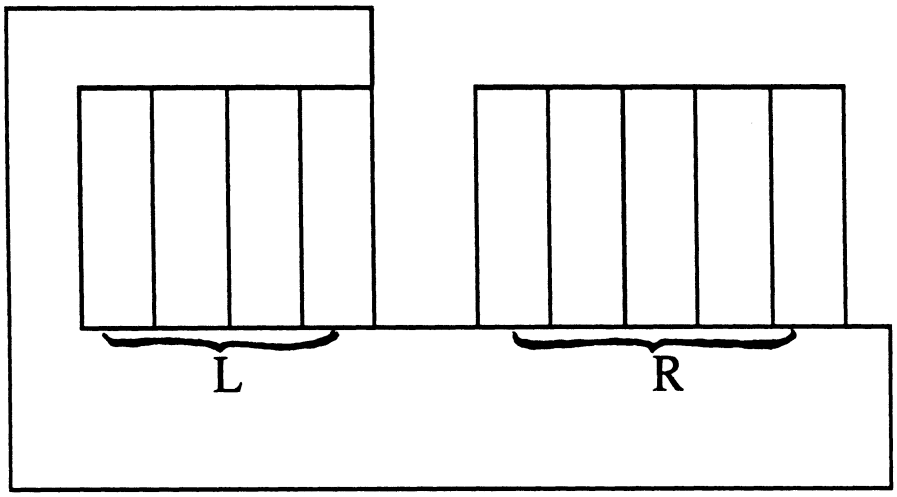


Figure 1-2. A 1-Disassemblable Sequential Assembly

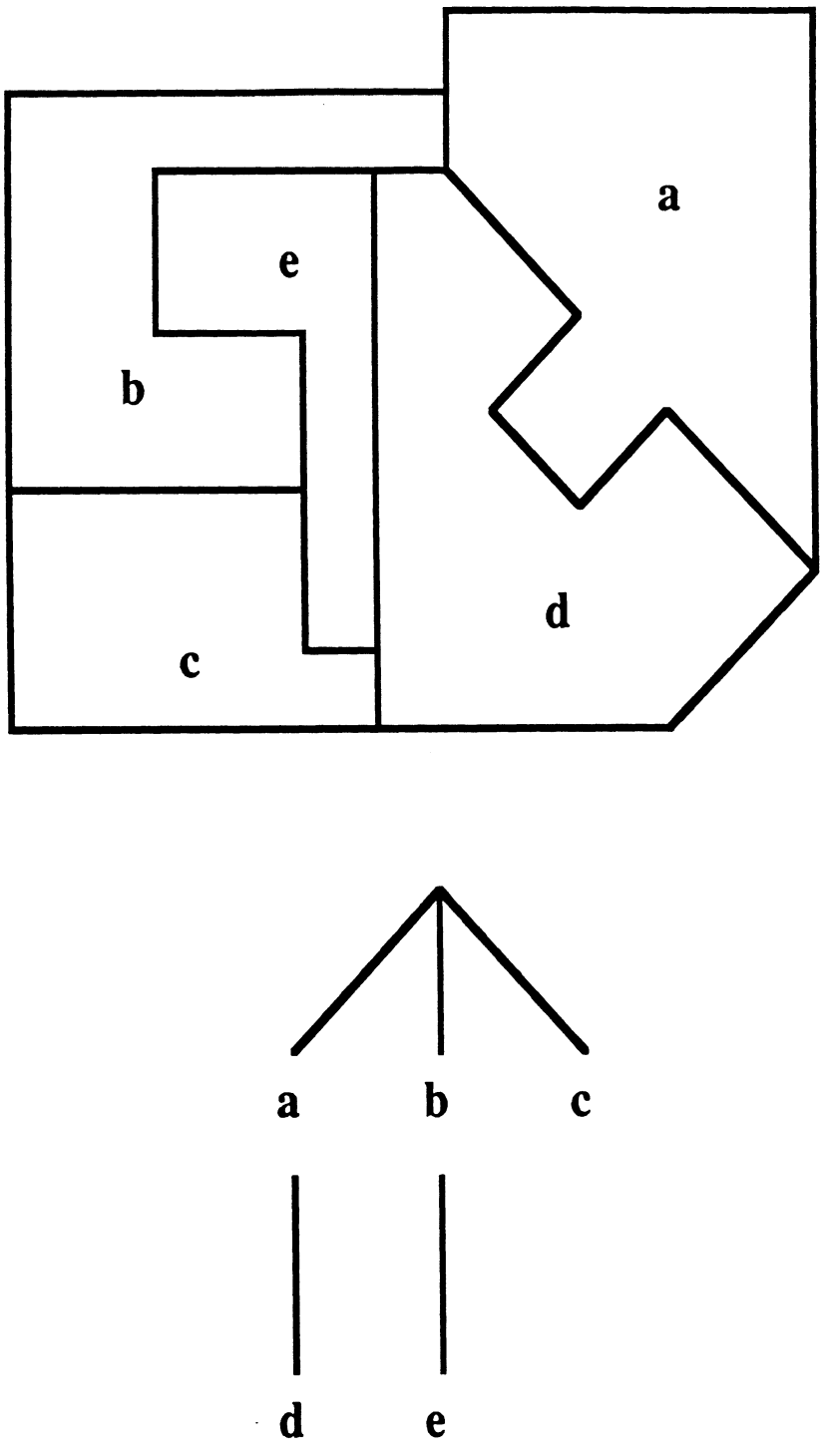


Figure 1.3 An Assembly and Its Disassembly Tree

2. DISASSEMBLABILITY

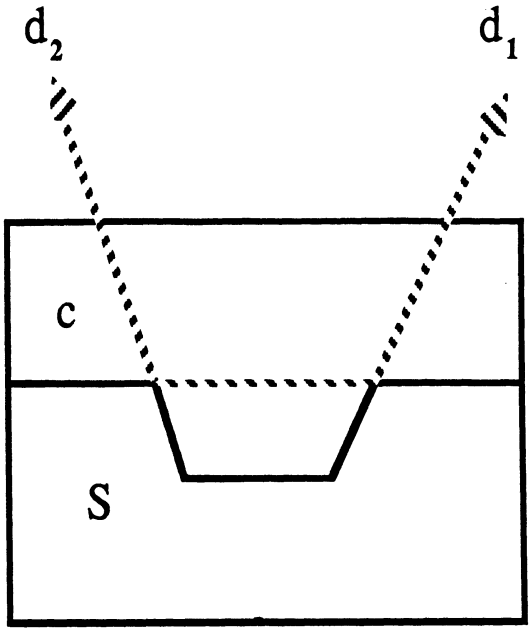
For simplicity, we shall first consider assemblies with only two components -- one being the component under consideration for removal and the other representing the subassembly (the rest of the assembly). Observe the two examples in Figure 2.1. It is clear that the component c in Figure 2.1(a) is disassemblable while the one in Figure 2.1(b) is not. We distinguish these two cases by computing a range of directions along which a component can be translated such that it does not interfere with the subassembly S.

< Insert Figure 2.1 >

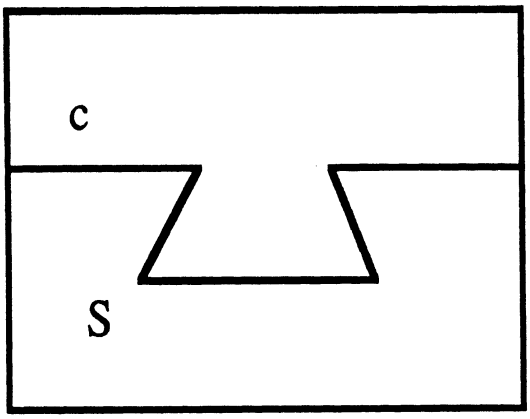
Definition 2.1 A component c is said to be linearly disassemblable (or just disassemblable, for short) from a subassembly S in the direction d if, for all points p in c, a RAY (p,d), which is a semi-infinite line from p in the direction d, does not intersect S.

Definition 2.1 implies that component c, if disassemblable, can be gripped at any point p in c and pulled in the direction d without interference. If p is the center of mass for c then there is no angular moment when c is pulled. If the center of mass lies outside c (or lies in a hole of c), two or more grip points will ensure an interference-free removal [8]. In Figure 2.1(a), the direction d lies between d_1 and d_2.

The range of directions can be determined from the mating



(a) Component c is disassemblable from subassembly S



(b) Component c is not disassemblable

Figure 2.1 Linear Disassemblability and Visible Region

edges, the sequence of edges shared by c and S. If a viewer (the robot) can "see" (from infinity in the opposite direction of the ray) all points p on the mating edges without having to go through S then c is removable. We need the notion of a region in which the viewer should reside.

Definition 2.2 A visible region VR ($\{e_i\}$) for a sequence of edges $\{e_i\}$ is the intersection of half space HS (e_i) induced by edge e_i.

We are ready to state the condition for disassembling c from S and the determination for the existence of such a condition.

Lemma 2.1 A component c is disassemblable from a subassembly S if the visible region VR ($\{e_i\}$) is unbounded.

[Proof] If the visible region is bounded, by Definitions 2.1 and 2.2, there cannot be a point at infinity from which a ray reaches all points x on the edges $\{e_i\}$, and vice versa.

An edge e_i of a polygon naturally partitions a two-dimensional space into two half spaces -- the inside and the outside of the polygon. When the notion of a visible region is applied to the inside of a polygon and when the sequence of edges is the entire boundary of the polygon, we recall the definition of a kernel [1].

Definition 2.3 A kernel with respect to a set of edges $\{e_i\}$ is a region R such that, for any point p in R, and another point q on

any edge e_i, the line segment pq lies entirely in R.

A kernel applies to "internal" visibility because it always lies inside a polygon. A visible region, by contrast, applies to external visibility as the viewer is assumed to be at infinity. Since a kernel can be constructed in time linear in the number of edges [1], we have a linear construction time for a visible region.

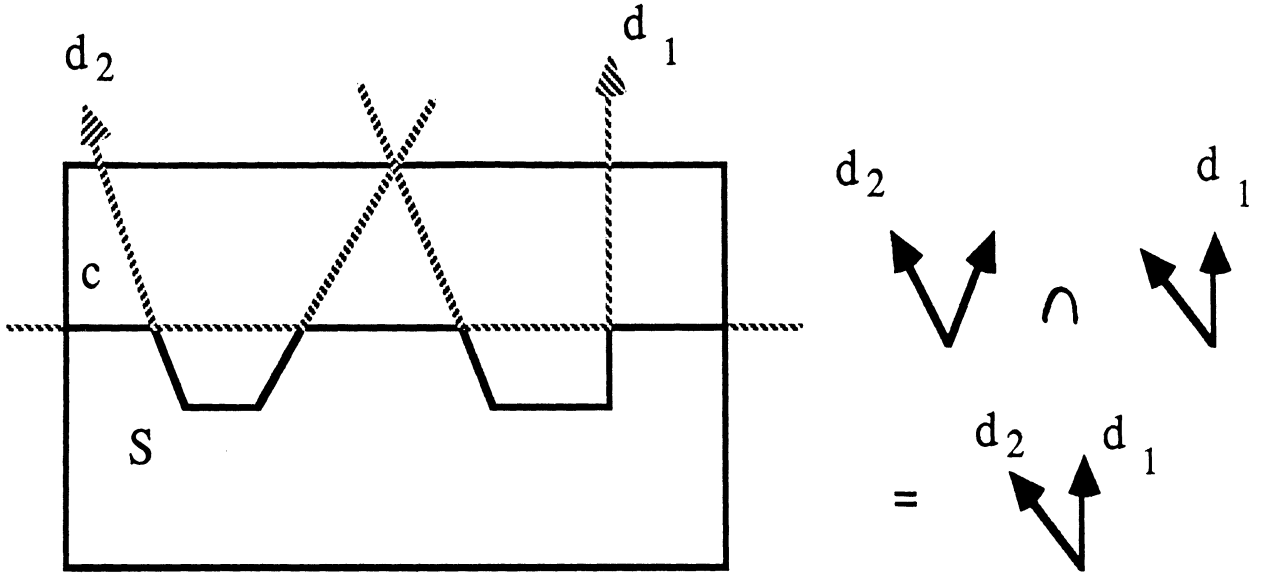
Lemma 2.2 A visible region VR ($\{e_i\}$) can be constructed in $O(n)$ time, where n is the number of mating edges in $\{e_i\}$.

Since an unbounded VR is necessarily convex, there must be two half-lines or rays extending to infinity. As illustrated in Figure 2.2(a), we denote these two rays by their polar angles in the counter-clockwise direction.

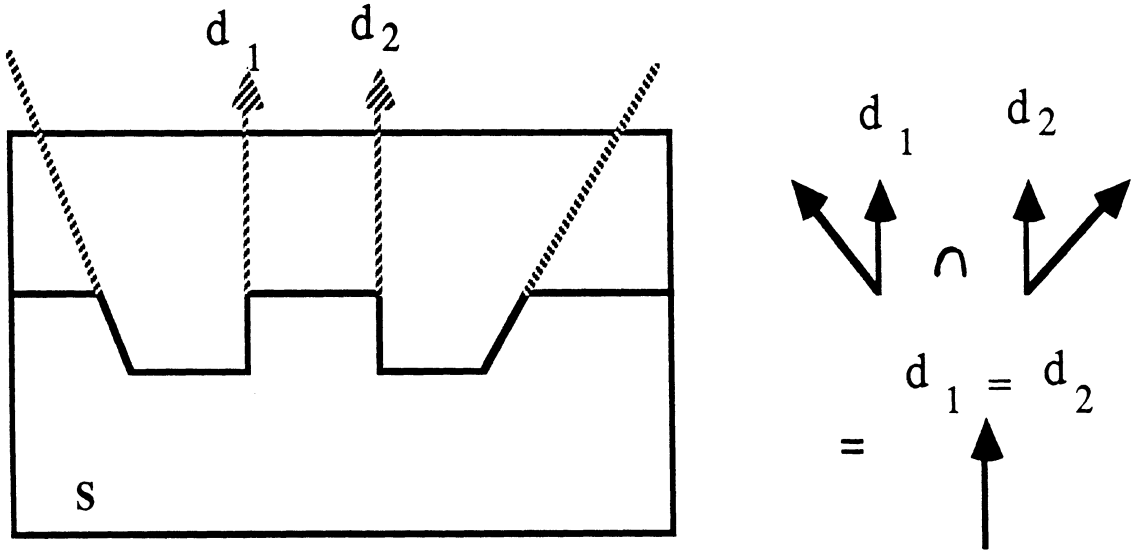
Definition 2.3 If c is disassemblable from S the visible region VR ($\{e_i\}$) is said to span a cone of visibility CV (d_1, d_2) with angle $d_2 - d_1$.

< Insert Figure 2.2 >

A special case for computing CV (d_1, d_2) arises when $d_1 = d_2$. This situation as illustrated in Figure 2.2 (b) occurs when two half-planes $HS(e_i)$ and $HS(e_j)$ are parallel (intersecting at infinity). As an alternative, we can decompose the mating edges $\{e_i\}$ into subsequences, find their respective CVs, and compute the intersection of the CVs. This motivates the following lemma.



(a) $d_1 < d_2$



(b) $d_1 = d_2$

Figure 2.2 Intersection of Cones of Visibility

Lemma 2.3 A cone of visibility CV (d_1, d_2) can be computed from the intersection of CV_i (d_1, d_2), each corresponding to the cone of visibility of a subsequence in $\{e_i\}$, if they exist.

[Proof] We are concerned with the efficiency of finding the maximum and the minimum of the spanning angles of the individual cones. Given a set of d_1s and d_2s, the maximum of the d_1s and the minimum of the d_2s can be found in 0(n) time. Hence, the method by Lemma 2.3 has the same time complexity as the one inferred by Lemma 2.2.

Disassemblability of a component c with respect to a subassembly S can now be expressed as a predicate.

Function Disassemblable (c,S)

1. From the mating edges of c and S, construct a visible region VR.
2. If the VR is bounded, return 'false'. Otherwise, return a cone of visibility CV(d_1,d_2).

Step 1 of the function can be done in 0(n) time, where n is the number of mating edges for c with respect to S. Step 2 involves traversing the boundary of the VR to detect closure which is again 0(n) time. Thus, we have the following result:

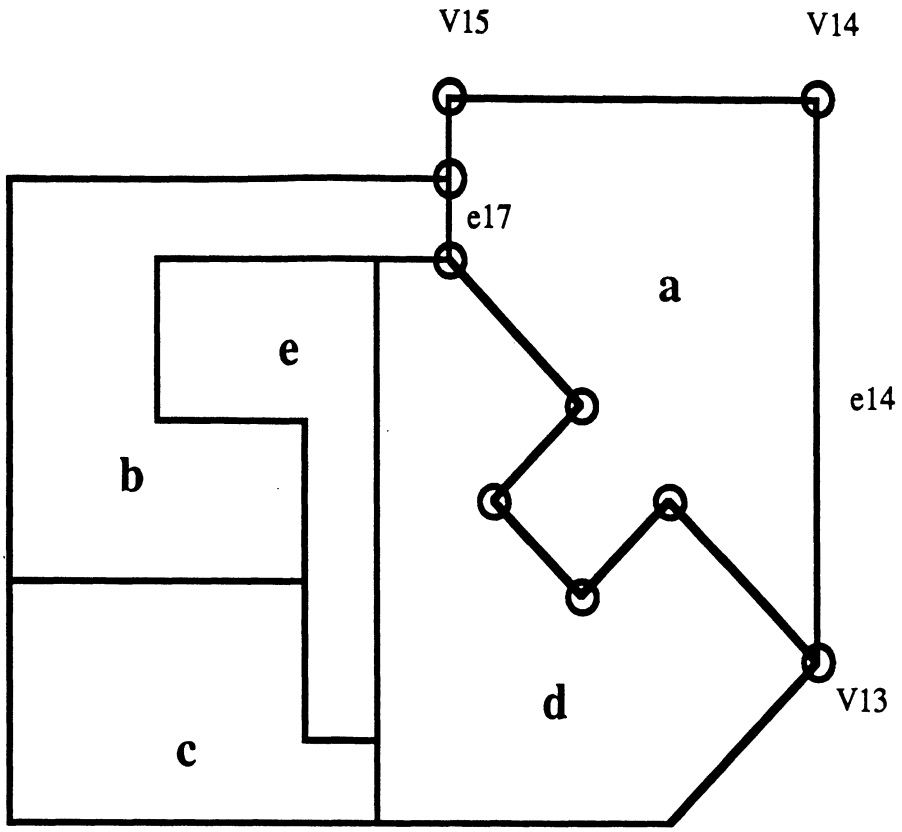
Lemma 2.4 The disassemblability of a single component c can be determined in 0(n) time, where n is the number of mating edges between c and the subassembly S.

3. DISASSEMBLY TREE

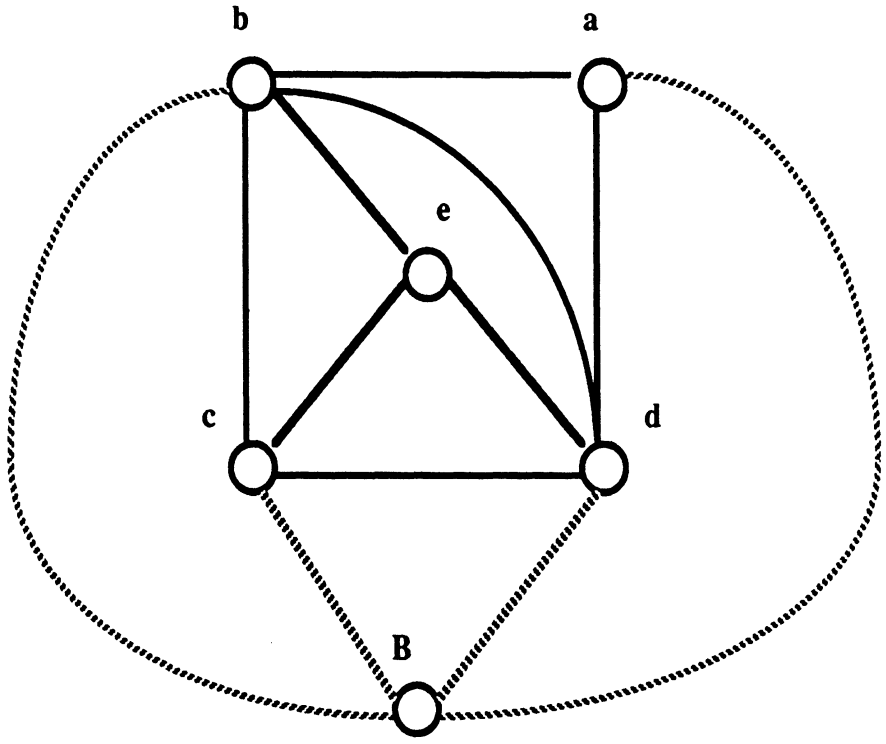
We are ready to consider an assembly with more than two components. First, we describe the data structure to be used by the algorithm. The assembly is given as a boundary graph BG of vertices, edges and faces. Each vertex points to all of its incident edges. Each face points to all if its surrounding edges. Each edge points to its two end points that are vertices, its two faces that are mating components, and four edges with two at each end [5]. If one of the two faces of an edge is the background then it is a boundary edge. Otherwise, it is a mating edge. In Figure 3.1(a), edge e_{14} is a boundary edge while edge e_{17} is the mating edge for components a and b. Note that an edge with more than one mate is split. Thus, every edge in the BG has exactly two mates. From the BG, we construct a dual mating graph MG. The arcs in MG connect nodes that are mating components. As illustrated in Figure 3.1(b), component a mates with components b and d as well as with the background B. As there are k components, and each access in the boundary graph involves constant time [5], the planar graph MG can be constructed from BG in time linear in k.

< Insert Figure 3.1 >

We begin the construction of disassembly tree DT by determining if components with boundary edges can be disassembled first. Using the example from Figure 3.1, we find four candidates a, b, c and d. We next determine if these candidates



(a) Boundary Graph



(b) Mating Graph

Figure 3.1 Data Structures

should be attached as nodes to the root of the tree denoted by A, the assembly. In Figure 3.2(a), each of the candidates is tested for disassemblability with respect to a subassembly. For component d, the subassembly is S = A - d. If it is found to be false after calling the function Disassemblable (d,S) then it is 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.

We next expand the tree by retrieving the mates of each of the leaf nodes and by testing their disassemblability. From the mating graph, the non-background mates are retrieved for each of the newly attached leaves. If a retrieved mate occurs elsewhere in the tree, it is discarded from further consideration. In Figure 3.2(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 against subassembly S = A - a - d. (The new boundary for S is updated by replacing all occurrences of a and d by background B for all the edges in the boundary graph.) 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 3.2(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

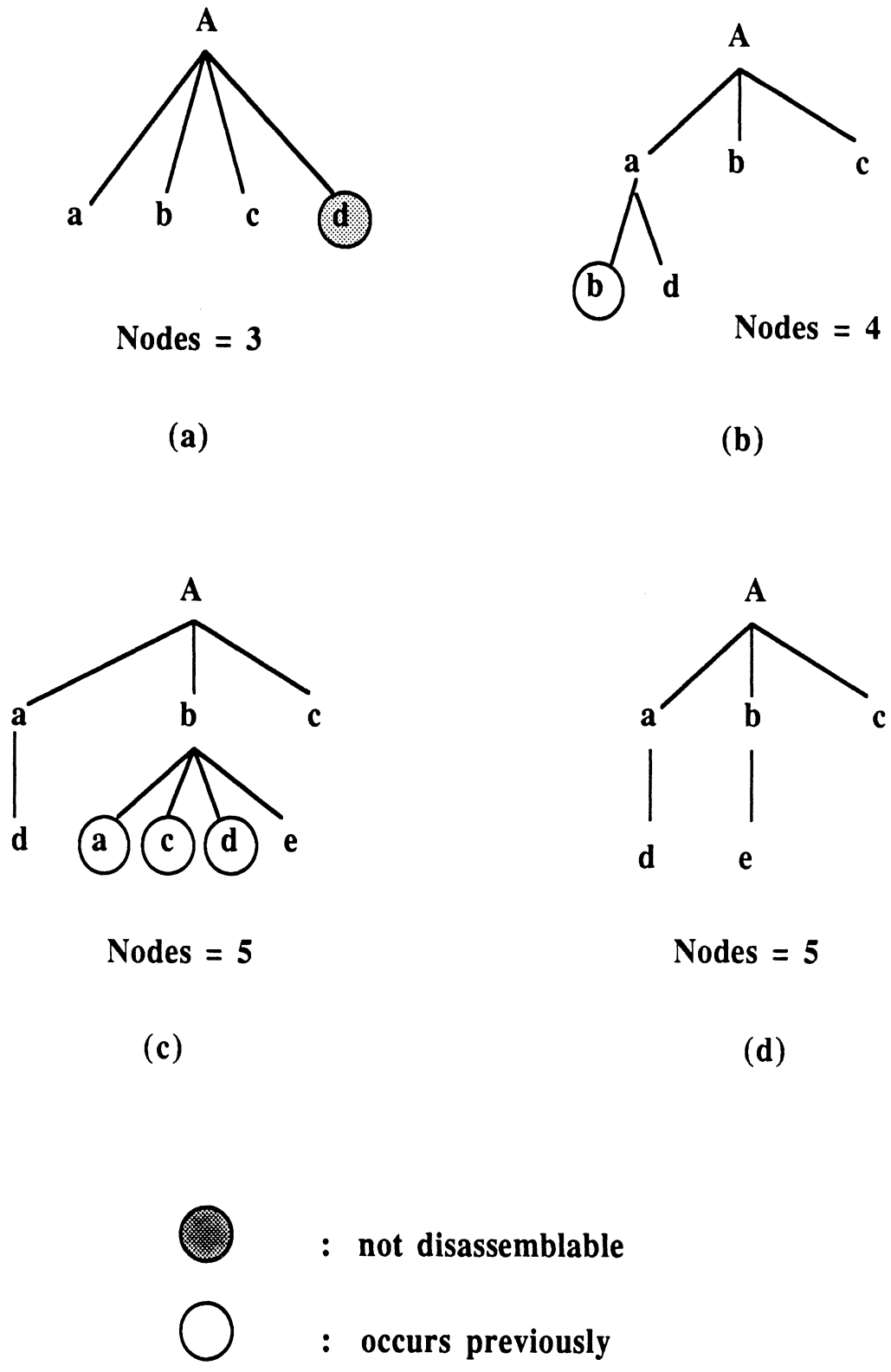


Figure 3.2 Generation of the Disassembly Tree

number of components in the assembly and the process terminates. The resulting tree is shown in Figure 3.2(d). The algorithm for generating such a disassembly tree can be described as follows:

Algorithm Disassemble

input: boundary graph BG and mating graph MG

output: disassembly tree DT

[Initialization]

1. NODES <-- 0

   TOTAL <-- total number of non-background nodes in MG.

2. Make a root node A for the empty tree DT.

[First level]

3. Identify all components in MG that are connected to the boundary B.

4. For each such component c

   If c is disassemblable

   then insert c as a leaf in DT

   NODES <-- NODE + 1

   else return 'failure'.

[Subsequent levels]

5. While NODES < TOTAL do

   5a. For each leaf L in DT

   retrieve its mates from MG

5b. For each mate M

If M is not in DT and is disassemblable

then insert M as a leaf node of L

NODES <--- NODES + 1

else return 'failure'.

We show that the DT constructed by Algorithm Disassemble gives a minimal sequence. This is done by induction on the height of the tree. Steps 1 and 2 of the algorithm give level 0 of the tree. Steps 3 and 4 give level 1 of the tree. Components in level 1 cannot occur in level 0, by construction. Now, suppose there are (v-1) levels in the tree and we examine a leave node L at level v. There are three possibilities.

Case 1. [L occurs at level < v]

By Step 5b, this is not possible. For a mate M to become a leaf L, it must not already be in DT.

Case 2. [L occurs at level v]

Again, by Step 5b, this is not possible, unless it is disassemblable. And, if so, there is only one occurrence of L anywhere in the tree, including level v.

Case 3. [L occurs at level > v]

A leaf L that should occur later in the tree gets mistakenly placed at level v is not possible. By Step 5a, only mates that are leaf candidates at level v are retrieved.

This proves the minimality of the sequence obtained from traversing a DT, hence the following theorem.

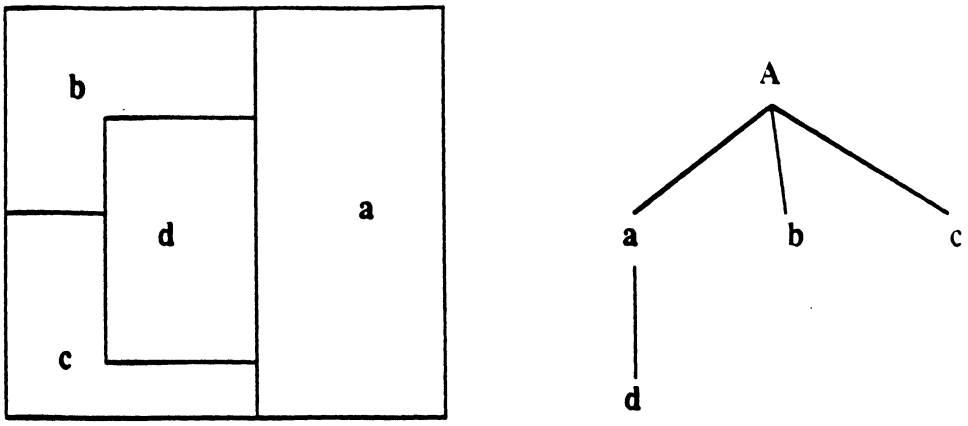
Theorem 3.1 Algorithm Disassemble generates a DT the traversal of which yields a minimal number of removals for the disassembly of a single component.

4. ALL DISASSEMBLY SEQUENCES

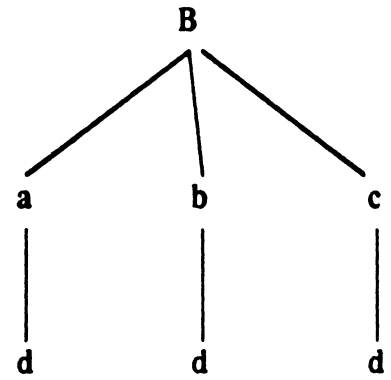
The DT generated by Algorithm Disassemble gives a minimal sequence if a single component is to be removed. However, if multiple components are to be removed from the assembly, Algorithm Disassemble may not give the optimal solution. Consider the assembly in Figure 4.1(a) and suppose that two components c and d are to be removed from it. The Algorithm Disassemble could generate a tree with root A requiring 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 tree with root B, as shown in Figure 3.3(b), only two removals are necessary for this instance.

< Insert Figure 4.1 >

To generate all optimal sequences for multiple component removal, two modifications to Algorithm Disassemble are necessary. First, disqualifying a retrieved mate M from disassemblability test should be postponed. Second, the termination condition for the algorithm should not be the total number of components. The first modification can be effected by changing, in step 5b of Algorithm Disassemble, "If M is not in DT and ..." to "If M is not <u>at a higher level</u> in DT and ...". The second change can be effected by noting the situation when the tree ceases to grow. In other words, if none of the retrieved mates M at that level are added to the tree then the algorithm should terminate.



(a) One Disassembly Sequence



(b) All Disassembly Sequences

Figure 4.1 Multiple Disassembly

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 the boundary B.

```
Algorithm Multiple_Disassemble
     input:  boundary graph BG and mating graph MG
     output:  disassembly tree 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 MG
               LEVEL <-- LEVEL + 1
          For each mate M 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. However, we shall show that its time complexity is still O(N), on

the average.

Within the same level, a component can appear as many times in the tree as the number of mates it has in the mating graph MG. Given k components in an assembly, it is known that its MG is maximally connected if the k nodes form a triangulation graph. In other words, there can be a total of

$$m = 3k - 6$$

possible edges in MG. Averaging m over k components, we have

$$\frac{m}{k} = \frac{3k - 6}{k} = 3 - \frac{6}{k} .$$

Hence the average number of edges per node in MG is bounded by 3. Since each mating edge is tested at most twice (for its two mating components), the total work for Algorithm Multiple_Disassemble is at most 6N or 0(N). This proves our next theorem.

Theorem 4.1 All minimal sequences for multiple disassembly can be constructed on the average in 0(N) time, where N is the total number of mating edges in the assembly.

5. DISCUSSION

We have presented two algorithms 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. We showed that any such sequence is the shortest possible, hence optimal. Algorithm Disassemble constructs a DT for the case in which one component is to be removed from the assembly. Algorithm Multiple_Disassemble constructs a DT for the case in which the removal of more than one component is desired. The latter is shown to run in time linear in the number of mating edges in the assembly, on the average. If there are k components with a total of N mating edges, it is possible that in the worst case it takes O(k) time to construct a DT. Since it handles multiple components, O(k) time is the best case as well. Since multiple disassembly subsumes single disassembly, our algorithms are optimal.

The two algorithms extend naturally to three dimensions. A visible region for a set of n mating faces involves the intersection of n half-planes and a visible cone will be a solid angle bounded by at most n such half-planes. However, computing a disassembly sequence for a parallel assembly is expected to take more than linear time since the components must be grouped in combinations of two, three, ... up to (k-1), where k is the total number of components.

ACKNOWLEDGEMENT

The author wishes to thank J.D. Wolter of The University of Michigan for his keen insights. The author also acknowledges Professor T. L. Kunii of the University of Tokyo who initiated the author to the idea of disassembly during one of his visits to Ann Arbor.

REFERENCES

1. Lee, D.T. and Preparata, F.P., "An Optimal Algorithm for Finding the Kernel of a Polygon", Journal of the ACM, Vol. 26, 1979, pp. 415-421.

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

5. 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.

6. 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.

7. Wolter, J.D., Volz, R.A., and Woo, T.C., "Automatic Generation of Gripping Positions", IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-15, No. 2, 1985, pp. 204-213.

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