Algorithmica (2000) 26: 577–601
DOI: 10.1007/s004539910025

Algorithmica
© 2000 Springer-Verlag New York Inc.

A General Framework for Assembly Planning: The Motion Space Approach[1]

D. Halperin,[2] J.-C. Latombe,[3] and R. H. Wilson[4]

Abstract. Assembly planning is the problem of finding a sequence of motions to assemble a product from its parts. We present a general framework for finding assembly motions based on the concept of *motion space*. Assembly motions are parameterized such that each point in motion space represents a mating motion that is independent of the moving part set. For each motion we derive blocking relations that explicitly state which parts collide with other parts; each subassembly (rigid subset of parts) that does not collide with the rest of the assembly can easily be derived from the blocking relations. Motion space is partitioned into an arrangement of cells such that the blocking relations are fixed within each cell. We apply the approach to assembly motions of several useful types, including one-step translations, multistep translations, and infinitesimal rigid motions. Several efficiency improvements are described, as well as methods to include additional assembly constraints into the framework. The resulting algorithms have been implemented and tested extensively on complex assemblies. We conclude by describing some remaining open problems.

Key Words. Assembly planning, Assembly sequencing, Motion space, Nondirectional blocking graph, Manufacturing.

1. Introduction.

Assembly planning is the problem of finding and sequencing the motions that put the initially separated parts of an assembly together to form the assembled product. This is a problem of considerable importance. Its solution would allow CAD systems to provide better feedback to designers, in order to help them create products that are more cost-effective to manufacture. Assembly sequences are also useful to select assembly tools and equipment, and to layout manufacturing facilities.

In this paper we present a general framework for assembly planning based on the concept of motion space. This framework is similar in spirit to the configuration space framework used for path planning. However, while in path planning one represents any possible placement of the moving object as a point in a configuration space, here we represent each possible motion of a subassembly as a point in a motion space. For each motion in the motion space, certain parts collide with others. These pairs of parts constitute constraints on which subassemblies may perform the motion. Then the idea

[1] Work on this paper by D. Halperin has been supported in part by an Alon Fellowship, by ESPRIT IV LTR Project No. 21957 (CGAL), by the USA–Israel Binational Science Foundation, by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities, and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University. Work on this paper by J.-C. Latombe has been supported by Army MURI Grant DAAH04-96-1-007.
[2] Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. halperin@math.tau.ac.il.
[3] Department of Computer Science, Stanford University, Stanford, CA 94305, USA. latombe@cs.stanford.edu.
[4] Eastman Kodak Company, Albuquerque, NM 87112-1881, USA. rwilson@kodak.com.

Received November 15, 1996; revised January 15, 1998. Communicated by R. Motwani and P. Raghavan.

is to precompute a decomposition of motion space into regions such that the constraints among the parts in the subassembly are the same for all the motions in the same region. We represent the constraints over a region in the form of a graph called the *directional blocking graph*, or DBG. We call the collection of all regions in motion space together with their associated DBGs the *nondirectional blocking graph*, or NDBG, of the assembly. Once the NDBG has been computed, it can be queried to generate one assembly sequence, or several, or all. Throughout this paper we present several techniques to efficiently compute and query NDBGs for a variety of motion families.

Most of the techniques presented here have already been described in a series of recent papers and reports, many of which we have authored or coauthored, but the connections among these techniques and the underlying general framework were only partially noticed. The present paper outlines the motion space framework in its full generality and re-presents (without all the details) previous techniques within this framework. It provides foundations for future research in assembly planning and, along the way, it identifies several problems that remain open.

The paper is organized as follows. Section 2 defines the assembly planning problem in more formal terms and introduces the assembly-by-disassembly approach that is used in the motion space framework. It references some of the previous work that has also used the assembly-by-disassembly approach, and surveys related work. Section 3 presents the general principles of the motion space approach, but stops short of giving effective algorithms. Sections 4 and 5 exemplify the motion space approach for what we call one-step and multistep motions. These two sections give polynomial-time algorithms to compute and use NDBGs for many useful families of motions. Section 6 describes two improvements of the efficiency of these algorithms. Section 7 extends the motion space approach to the case where the parts of an assembly are toleranced, i.e., multiple instances of the same parts can have slightly different geometries. Section 8 extends the approach to handle considerations such as stability and tool use not addressed in previous sections. Section 9 briefly presents the existing implementations of the techniques described in this paper. Finally, in Section 10 we mention several open problems.

2. Assembly Planning

2.1. *Definitions.* A **workspace** is a subset of the two or three-dimensional physical space modeled by the Euclidean space $\Re^k$ ($k = 2$ or 3). A **body** is a rigid physical object modeled as a compact manifold with boundary[5] in $\Re^k$. We say that two bodies **overlap** if their interiors intersect. We say that they **touch** each other if they intersect without overlapping.

An **assembly** is a collection of bodies (also called **parts**) in some given relative placements in the workspace, such that no two bodies overlap. A **subassembly** is a subset of the bodies composing an assembly A in their relative placements in A. In most practical cases, each part of an assembly is in contact with a few other parts so that the union of all the parts form a connected subset of the workspace. However, except when

[5] A subset M of $\Re^k$ is a manifold with boundary if every point of M has a neighborhood V such that $V \cap M$ is homeomorphic to either an open ball of $\Re^k$ or a closed half-space of $\Re^k$.

we indicate otherwise, we do not make such an assumption. The motion space approach described in this paper applies to both the case where there are contacts among parts of an assembly and the case where there are no such contacts.

We say that two subassemblies are **separated** if they are arbitrarily far apart from one another. Given an assembly A, an **assembly operation** is a motion that merges s ($s \geq 2$) pairwise separated subassemblies of A into a new subassembly of A. During this motion, each subassembly moves as a single body and no overlapping between bodies is allowed. The parameter s is called the **number of hands** of the operation. We call the reverse of an assembly operation a **partitioning** operation.

An **assembly sequence** is a total ordering on assembly operations that merges the separated parts composing an assembly into this assembly. The maximum, over all the operations in the sequence, of the number of hands required by an operation is called the number of hands of the sequence. A **two-handed** assembly sequence is one that requires two hands only.

It has been shown in [17] that every assembly of convex polygons in the plane admits a two-handed assembly sequence of translations; but, in the worst case, s hands are necessary and sufficient for assemblies of s star-shaped polygons/polyhedra [34]. In the rest of this paper *we will only consider two-handed assembly sequences*.

With the above definitions, an assembly sequence only generates subassemblies of the final assembly. Such a sequence is said to be **monotone** [45]. A more general assembly sequence is one in which some operation brings a body to an intermediate placement (relative to other bodies), before another operation transfers it to its final placement. See Figure 1. Such a sequence is called **nonmonotone**. In the rest of this paper *we will only consider monotone assembly sequences*.

An **assembly sequence planner** (or **assembly planner**) is an algorithm that takes the description of an assembly as input and computes the assembly sequences for this assembly.

A classical strategy for assembly planning is called **assembly-by-disassembly**. It consists of partitioning, first the input assembly A into subassemblies and then, recursively, the generated subassemblies that are not individual parts. This strategy is implemented

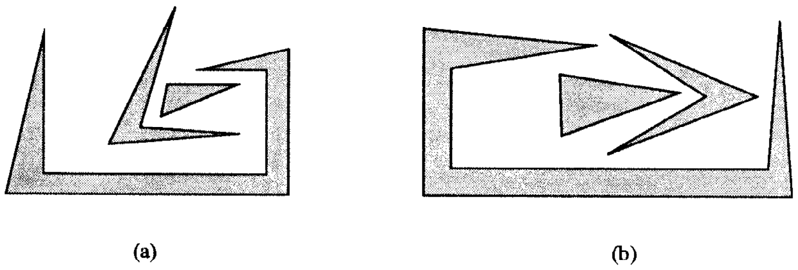

Fig. 1. Both assemblies above admit two-handed sequences with translational motions only. While (a) accepts a monotone such sequence, (b) does not. To disassemble (a) one can first remove the subassembly consisting of the two "inner" parts, then break this subassembly into its two components. To disassemble (b) the triangle must be translated to an intermediate position, since the two inner parts cannot be removed as a subassembly. If general motions are accepted, there exists a monotone two-handed sequence for (b). A monotone three-handed sequence with translations only is also possible.

by two key procedures: `partition` and `disassemble`. The procedure `partition` takes the description of an assembly S as input and generates two subassemblies S_1 and S_2 ($S_1 \cup S_2 = S$), along with a path p such that moving S_1 along p separates it from S_2. Whenever such subassemblies and direction do not exist, the procedure returns failure. The procedure `disassemble` applies `partition` to the given assembly A and, recursively, to the generated subassemblies. The motion space approach described in this paper provides general and efficient means for designing `partition`.

Note that the above definitions ignore the constraints introduced by the mechanical system manipulating the parts; in other words, they assume that the parts of an assembly can move and maintain stability without any external help. In Section 8 we will consider the addition of other constraints.

Finally, it is convenient to introduce the following categories of motions. We say that a motion step translates a body along a single direction t by some distance ℓ, while rotating it at constant rate about an axis a that is fixed relative to the body. A **one-step motion** consists of a single motion step in which ℓ is arbitrarily large. If the rotation rate is null (i.e., the body has fixed orientation), the motion is a **one-step translation**. A **multistep motion** is the concatenation of several motion steps, in which the last step has arbitrarily large ℓ. An **infinitesimal motion** consists of a single motion step in which ℓ is arbitrarily small.

2.2. *Previous and Related Work.* The complexity of assembly sequencing has been investigated in several papers. When arbitrary sequences are allowed, assembly sequencing is PSPACE-hard, even when the bodies are polygons, each with a constant number of vertices [34]. When only two-handed monotone sequences are permitted, deciding if an assembly can be partitioned into two subassemblies by an arbitrary motion is NP-complete [42]. The problem remains NP-complete when motions are restricted to multistep translations and other variations [26].

The assembly-by-disassembly strategy prevails throughout the literature on assembly planning [2], [3], [7], [20], [22], [29], [46]. It is motivated by the observation that a product in its assembled state exhibits far more constraints on its components than in its disassembled state. These constraints drastically reduce the range of assembly motions which a planner must consider. Moreover, when all parts are rigid, there is a bijection between assembly and disassembly sequences, as far as geometric constraints are concerned. (This bijection would also be broken when physics, e.g., gravity, and motion control uncertainty are taken into account.)

In many planners (e.g., [3] and [22]), the procedure `partition` is based on generate-and-test: given S, enumerate all candidate partitions $\{S_1, S_2\}$ of S, until a path p is found that separates S_2 from S_1. The partitions are usually generated from cutsets of the assembly connection graph, which in many cases lowers the number of partitions to consider and ensures S_1 and S_2 are both connected (see Section 8.1). Most often p is a simple path (e.g., a translation) that is inferred from contacts between parts [21], [46]. However, even with the cutset approach the number of candidate partitions is exponential in the number of parts in S, while the number of feasible partitions is usually much smaller. Several effective planners were nevertheless built along this principle. Some reduced time complexity by only allowing partitioning operations in which one subassembly is a single part [20], [29], [45]. Others incorporated heuristics, for example, to reduce drastically the number of separating motions considered by the planner at each partitioning step [7], [20]. Some were quite general and hence limited to assemblies with small part count [3]. The NDBG was introduced to avoid this combinatorial trap [39], [43].

Arkin et al. [2] use the concept of a monotone path among obstacles to derive a polynomial-time algorithm for partitioning an assembly of polygons in the plane with a one-step translation. Their algorithm for this special case is more efficient than a straightforward application of the NDBG approach, but does not generalize to other motions.

If we restrict ourselves to moving one part at a time and allow only a single direction d of translation, we obtain a very simple assembly sequence (and hence desirable, whenever possible). Such a sequence, when one exists, defines a depth order of the parts in the assembly along the direction d. Depth orders were extensively studied, mainly in relation to computer graphics [4]. A depth order for an assembly of polyhedral parts with a total of N vertices can be computed in $O(N^{4/3+\varepsilon})$, for any $\varepsilon > 0$ [6]. If we are allowed to move the parts along k different directions (still one part at a time and each move is a single translation to infinity), then a more elaborate algorithm needs to be used [1] which runs in $O(kN^{4/3+\varepsilon})$ time. The latter algorithm has been also extended to deal with rotational motions along a small fixed set of axes, and with a slightly higher running time. For details see [1].

The motion space approach that we present below is primarily concerned with determining one or several (or all) assembly sequences of a type, if these exist. More involved questions arise when one attempts to optimize certain cost measures of an assembly sequence. Recently it has been shown that several variants of cost measure optimization in assembly planning are hard, even in their approximate versions [13], [15]. We cite here one hardness result from [14], for a problem that at first glance seems very simple. We are given n pairwise interior-disjoint unit disks in the plane. One disk is marked as the *key* disk that we need to access, and we are allowed to remove disks in one-step translational motions. The goal is to minimize the number of the disks removed before the key disk can be removed. It is shown that obtaining a $2^{\log^{1-\gamma} n}$-approximation to this problem is hard for any $\gamma > 0$. We refer the reader to [13], [14], and [15] for full technical details on this and other hardness results in this area.

The representation of a motion planning problem in a motion space is not completely new. In motion planning with uncertainty in control and sensing, the notion of a nondirectional preimage is also based on a decomposition of a motion space [8], [31]. The part-orienting technique proposed in [11] makes use of a similar decomposition.

3. The Motion Space Approach.

A widespread representation tool used in robot motion planning [30], [33] is **configuration space**. The configuration space of a given robot is the space of parametric representations of the robot configurations: every point in the configuration space defines a unique placement of the robot in the physical space. The dimension of the configuration space is the number of **degrees of freedom** of the robot. It is finite for robots made of rigid components, but infinite when components are flexible. Each obstacle B in the physical space maps into a configuration-space obstacle, or **C-obstacle**, CB in configuration space: every point in CB corresponds to a configuration of the robot where it intersects the obstacle B.

Using a similar approach for assembly planning, we define the **motion space** (or **M-space**) to be the space of parametric representations of all allowable motions for partitioning operations: every point in M-space uniquely defines a path of the subassemblies moved by an operation. Hence, the dimension of the motion space is the minimal number of parameters required to define a path with a fixed starting point (the initial position of the corresponding subassembly). If no restriction is placed on assembly motions, the dimension is infinite. If, for instance, motions are restricted to be one-step translations in three dimensions, the motion space has dimension 2, corresponding to the two parameters describing the direction of a translation.

Note that the motion space must be parameterized in such a way that the representation of a motion is independent of the subassembly that will eventually be moved away from the rest of the assembly. Usually, this is accomplished by attaching coordinate frames to the parts such that all the coordinate frames coincide with a universal frame $\mathcal{U}$ when the parts are in their assembled configurations. This allows us to describe a motion as a displacement from the universal frame, independently of the parts that perform it.

The analog of a C-obstacle in motion space is the M-space region, or **M-region**: for every ordered pair of parts P_i and P_j in an assembly we define their M-region $\mathcal{P}_{ij}$ to be the collection of points p in motion space such that if we move P_i along the path that p represents, P_i will overlap with P_j at some point. For each path p in $\mathcal{P}_{ij}$ we say that P_j **blocks** P_i.

Given an assembly A made of n parts $P_1, \ldots, P_n$, we associate a directed graph $G(p)$ with every point p of motion space. The nodes of $G(p)$ are the n parts $P_1, \ldots, P_n$ composing the assembly, and each ordered pair (P_i, P_j) such that $p \in P_{ij}$ induces an arc of $G(p)$ directed from P_i to P_j. We call this graph the **directional blocking graph**[6] (or DBG) of A for path p.

Let $\partial\mathcal{P}_{ij}$ denote the set of all paths p such that if P_i moves along p it will eventually touch P_j (i.e., the boundaries of P_i and P_j will intersect) without overlapping it. In most cases $\partial\mathcal{P}_{ij}$ and the boundary of $\mathcal{P}_{ij}$ are identical; in some cases, however, $\partial\mathcal{P}_{ij}$ is a superset of $\mathcal{P}_{ij}$'s boundary [30]. The sets $\partial\mathcal{P}_{ij}$ for all $i, j \in [1, n], i \neq j$ decompose the motion space into an arrangement of cells such that the DBG of A remains fixed over each cell. The arcs of the DBG on any cell c in this arrangement correspond exactly to the M-regions that contain c. We call the arrangement of cells thus defined, along with the DBG of each cell, the **nondirectional blocking graph** (or NDBG) of A. See Figure 2 for an illustration.

Consider a DBG $G(p)$ of A. We can partition A into two subassemblies A_1 and A_2 by moving A_1 along p if and only if there exists no arc in $G(p)$ connecting a part of A_1 to a part of A_2. Hence, A can be partitioned by a motion along p if and only if $G(p)$ is not strongly connected. If $G(p)$ is not strongly connected, then one strong component must have no outgoing arcs, and is therefore removable along p. A straightforward computation on the strong components of $G(p)$ yields all possible partitions of A for path p if desired [39]. Notice also that the NDBG of any subassembly S of A is obtained by restricting every DBG to the parts of S; it does not have to be recomputed from scratch.

[6] In the earlier papers on the NDBG, we only considered paths that were directions of motion, hence the term "directional blocking graph." We retain the terminology for consistency with the existing work.

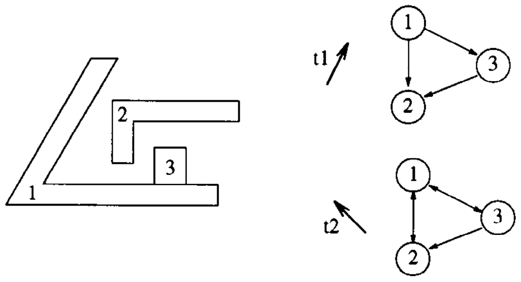

Fig. 2. Examples of directional blocking graphs.

We define `partition` as follows:

```
procedure partition($S$);
  for every cell $c$ in the NDBG of $S$ do:
    if the DBG associated with $c$ is not strongly connected
      then return $c$ and a feasible partition of $S$;
  return failure;
```

The NDBG of A is computed once, before running `disassemble`.

Assembly planning can be seen as the problem of computing coordinated paths for the parts composing an assembly A. By treating each part as an independent moving object, this problem corresponds to generating a path in the composite configuration space of the parts [30], but the number of dimensions of this space grows linearly with the number of parts in A. Though randomized path planning techniques could possibly be used in such a high-dimensional space [27], they would not allow systematic exploration of the set of feasible assembly sequences for given families of partitioning motions. In contrast, the number of dimensions of M-space is independent of the number of parts in A. However, it grows with the complexity of the allowable motions. As will become apparent in the following sections, the motion space approach to assembly planning yields efficient algorithms only when the allowable motions can be described with relatively few parameters.

The following two sections exemplify the motion space approach to solving the partitioning problem for polygonal and polyhedral assemblies. We consider two types of motions: one-step and multistep motions.

4. The Case of One-Step Motions.

In this section we give efficient algorithms to compute NDBGs for rather simple families of motions. We first consider polygonal assemblies. Next, we generalize to polyhedral assemblies.

4.1. *One-Step Translations in Two Dimensions.*

Let A be a planar assembly made of n polygonal parts $P_1, \ldots, P_n$, which we want to assemble with one-step translations only. Each possible path for a part is defined by a single angular parameter representing the direction of the translation. Therefore, the motion space is the unit circle $\mathcal{S}^1$.

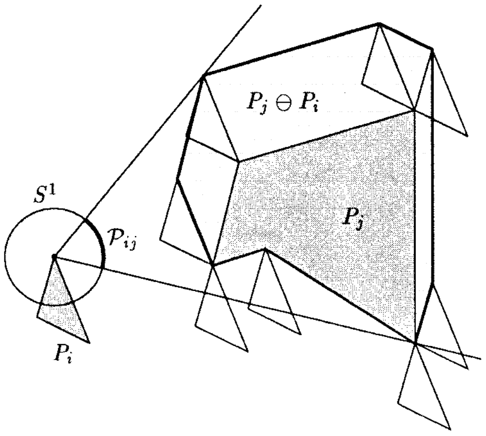

Fig. 3. Construction of the M-region $\mathcal{P}_{ij}$.

The M-region $\mathcal{P}_{ij}$ of translations along which P_i is blocked by P_j is an arc of $\mathcal{S}^1$. The endpoints of all the arcs $\mathcal{P}_{ij}$ decompose $\mathcal{S}^1$ into an arrangement of cells (open arcs and points) of complexity $O(n^2)$, such that the DBG of A remains fixed over each cell of this arrangement. The circular sequence of cells and their DBGs form the NDBG of A.

Assume that there are no "tight" insertions in A, i.e., no insertion like a peg in a hole that leaves a single translation along which a part is not blocked by another. Each arc $\mathcal{P}_{ij}$ can be constructed by extending the two extreme rays from the origin O of the coordinate system embedded in the plane containing the assembly and tangent to $P_j \ominus P_i = \{p_j - p_i | p_i \in P_i, p_j \in P_j\}$, the Minkowski sum of P_j and $-P_i$. See Figure 3, where the polygon in bold contour is $P_j \ominus P_i$. If P_i and P_j touch each other, then O lies in the contour of $P_j \ominus P_i$. If we allowed P_i to be tightly inserted into P_j, we would have to be more careful, since the set of positions where P_i touches P_j would then be a superset of the boundary of $P_j \ominus P_i$ [30]. In the general case where P_i and P_j are nonconvex polygons with q_i and q_j edges, respectively, the computation of $\mathcal{P}_{ij}$ can easily be done in total time $O(q_i q_j)$.

Let q be the maximal number of edges in a single part of A. The $O(n^2)$ arcs $\mathcal{P}_{ij}$ are computed in time $O((nq)^2)$. They determine $O(n^2)$ points in $\mathcal{S}^1$ that are sorted in time $O(n^2 \log n)$. The DBG in any cell can be obtained in time $O(n^2)$. However, as we scan all the cells in $\mathcal{S}^1$, the DBG undergoes $O(n^2)$ changes that can be handled each in constant time. Thus, once a DBG has been computed, all other DBGs can be computed in total time $O(n^2)$ by scanning the sequence of cells in $\mathcal{S}^1$ and, for each cell, modifying the DBG constructed for the previous cell [39]. The complete NDBG takes time $O(n^2(\log n + q^2))$ to compute.

Computing the strong components of a DBG of A takes time $O(n^2)$. Hence, `partition` runs in time $O(n^4)$ and `disassemble` generates an assembly sequence in time $O(n^5)$.

4.2. *One-Step Translations in Three Dimensions.* The above techniques easily generalize to polyhedral assemblies. Then the motion space is the unit sphere $\mathcal{S}^2$ in $\Re^3$. Each region $\mathcal{P}_{ij}$ is obtained by "projecting" $P_j \ominus P_i$ onto $\mathcal{S}^2$. Since each region $P_j \ominus P_i$ is a polyhedron, $\mathcal{P}_{ij}$ is a surface patch of $\mathcal{S}^2$ bounded by arcs of great circles. Let v be the total number of vertices in the parts of A. The arrangement on $\mathcal{S}^2$ is formed by $O(v^2)$ arcs, hence has size $O(v^4)$ and is computed in time $O(v^4)$. The total NDBG (including the DBGs) is also constructed in time $O(v^4)$ [39]. Each DBG has $O(n^2)$ arcs, so that finding its strong components takes time $O(n^2)$. Hence, `partition` has complexity $O(n^2 v^4)$.

4.3. *Infinitesimal Motions in Three Dimensions.* The direction of a one-step motion is given by a unit vector in six dimensions. Hence, the set of all possible directions of motion make up the unit five-dimensional sphere $\mathcal{S}^5$. Computing the M-regions requires projecting six-dimensional regions of complex shapes onto $\mathcal{S}^5$. This computation is feasible in principle (as long as all parts of A are described as semialgebraic sets), but it would be much too expensive to be practical.[7] This leads us to consider infinitesimal motions, instead. The directions of infinitesimal motion still span $\mathcal{S}^5$, but computing the M-regions is much easier.

Considering infinitesimal motions requires us to adapt some of our early definitions since no infinitesimal motion can truly separate two subassemblies. For the rest of this section we will say that an infinitesimal motion separates two subassemblies if it displaces one relative to the other, by an arbitrarily small amount, without overlapping of their interiors. Clearly, the existence of a separating infinitesimal motion is a necessary condition for the existence of an arbitrarily complex extended motion that would move the two subassemblies far apart, but it is not a sufficient condition. Note also that considering infinitesimal motions makes sense only in the case where the assembly forms a connected body. In an assembly with no contacts among parts, any assembly sequence with infinitesimal motions would be valid.

To ensure that motion space is parameterized independently of which parts follow a path, we attach a Cartesian frame to each part in A, such that all the frames coincide in the assembled configuration. We call the initial placement of the frames the universal frame and denote it by $\mathcal{U}$. An infinitesimal motion of any part P_i is described by a six-dimensional vector $dX = (dx, dy, dz, d\alpha, d\beta, d\gamma)$. The components dx, dy, and dz give the translation of the frame of P_i with respect to the frame $\mathcal{U}$. The components $d\alpha$, $d\beta$, and $d\gamma$ are the infinitesimal angles by which the frame of P_i rotates about the axes of $\mathcal{U}$.

Let V be a vertex of P_i. The motion described by dX causes V to undergo a translation $J_V\ dX$, where J_V is a constant 3×6 Jacobian matrix: each column of J_V gives the translation V experiences due to a unit motion of P_i in the corresponding parameter of dX. Assume that P_i and P_j are in contact such that the vertex V of P_i is contained in the face F of P_j. Let n_F be the outgoing normal vector to F. The motion dX causes V to penetrate F when $n_F J_V dX < 0$, to break the contact with F when $n_F J_V dX > 0$, and to slide in F when $n_F J_V dX = 0$. The set of motions dX of P_i that are blocked by the contact between V and F are those satisfying $n_F J_V dX < 0$.

Now let P_i and P_j be two parts in contact such that a face F_i of P_i and a face F_j of P_j intersect at a piece of planar surface (plane–plane contact). The set of motions dX

[7] However, this computation should be more reasonable in two-dimensional workspace.

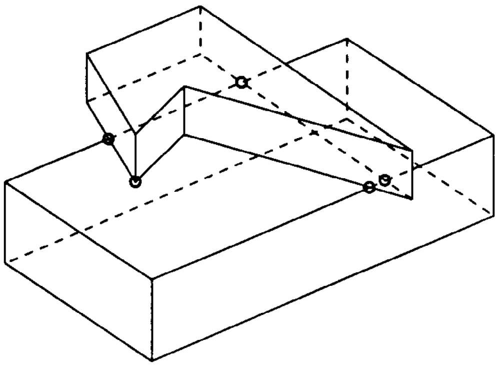

Fig. 4. Plane–plane contact expressed as point–plane contacts.

allowed by this contact is the intersection of all the closed half-spaces $n_{F_j} J_{V_k} dX \geq 0$ computed for the vertices V_k of the convex hull of the intersection of F_i and F_j [19], [44]. For example, in Figure 4, the vertices V_k are circled.

For each vertex V_k of the convex hull of the intersection of two parts, the equation $n_F J_{V_k} dX = 0$ defines a five-dimensional hyperplane in the six-dimensional space of infinitesimal motions, which partitions $\mathcal{S}^5$ into two open half-spheres and a great circle. The set of all such hyperplanes, determined by the vertices of the convex hulls of the planar contacts, induces an arrangement of cells of dimensions $5, \ldots, 1, 0$ on $\mathcal{S}^5$. The DBG is fixed over each such region.

The intersection of two nonconvex faces F_i and F_j having v_i and v_j vertices, respectively, has $O(v_i v_j)$ vertices and can be computed in $O(v_i v_j)$ time. Its convex hull is constructed in $\Theta(v_i v_j \log v_i v_j)$ time [35] and has $O(v_i + v_j)$ vertices. Indeed, all the intersection vertices of $F_i \cap F_j$ lie on edges of F_i and F_j, but each particular edge can contribute at most two vertices of the convex hull. Let k be the total number of vertices in the convex hulls of the c contacts. We have $k \in O(cv)$, where v is the total number of vertices in the parts of A; in practice, however, k is much smaller.

Since a singleton in the above arrangement arises whenever five hyperplanes intersect, the arrangement contains $O(k^5)$ regions. It is constructed in $O(k^5)$ time [9], [10]. Constructing a DBG in any region is done in time $O(k)$. Again, by noticing that the DBG undergoes a small number of changes between any two adjacent cells, the entire NDBG can be computed in $O(rk^5)$ time, where $r \in O(n^2)$ is the number of pairs of parts in contact. Given this NDBG, computing a candidate partitioning of A into two subassemblies takes $O(rk^5)$ time.

We will see in Section 6.2 that for the purpose of assembly planning it is not necessary to build the entire NDBG, and this observation yields a more efficient computation than above.

5. The Case of Multistep Motions.

In this section we present two ways to cope with multistep motions. First, in Section 5.1 we describe a structure called the interference diagram. This structure explicitly represents the overlapping relations between the parts in the assembly, for all possible relative positions of the parts. It is based on a careful parametrizaton of the configuration space of the parts composing the assembly. A curve γ in the interference diagram defines a possible partitioning path, and the blocking relations along this path are derived from the cells of the diagram that γ crosses. The interference diagram is therefore an intermediate structure from which one can compute the NDBG. It provides a conceptually general method for constructing NDBGs for multistep motions (in fact, for motions of any type), but the resulting algorithm may not be very efficient. In Section 5.2 we propose an efficient direct M-space algorithm that handles the specific case of two-translations in the plane.

5.1. *The Interference Diagram.* Let A be a planar assembly of n polygons $P_1, \ldots, P_n$. We wish to solve the partitioning problem for A when the allowable motions are multistep translations.

We select an arbitrary placement of A and we place a coordinate system of origin O in the plane containing A. We then describe each part of A as a set of points in this coordinate system. With each part P_i we associate a point O_i such that when P_i is at its placement in A, O_i coincides with O. Note that O_i does not have to be a point of P_i, but its position relative to P_i's points is fixed. Since we restrict part motions to be translations, the position of O_i always determines the placement of P_i in the plane.

For every pair of parts (P_i, P_j) we define $P_i/P_j = P_j \ominus P_i$. Assuming no tight insertion of P_i in P_j, the interior of the region P_i/P_j is the set of all positions of O_i where P_i overlaps P_j. Note that P_j/P_i is simply P_i/P_j rotated by π radians.

The boundaries of all the regions P_i/P_j define a polygonal arrangement in $\Re^2$, which we call the **interference diagram** $I(A)$ of A. We label each cell c of this arrangement with the list of all regions P_i/P_j that contain c. All cells have bounded size, except one which surrounds all others and have an empty label; we call this cell the **external** cell of $I(A)$. We call the cell that contains O the **central** cell of $I(A)$. To illustrate, Figure 5

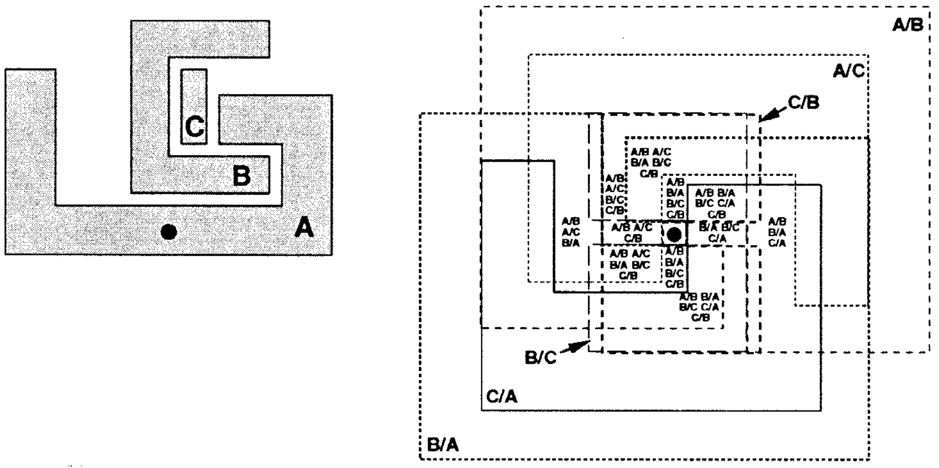

Fig. 5. A simple assembly and its interference diagram.

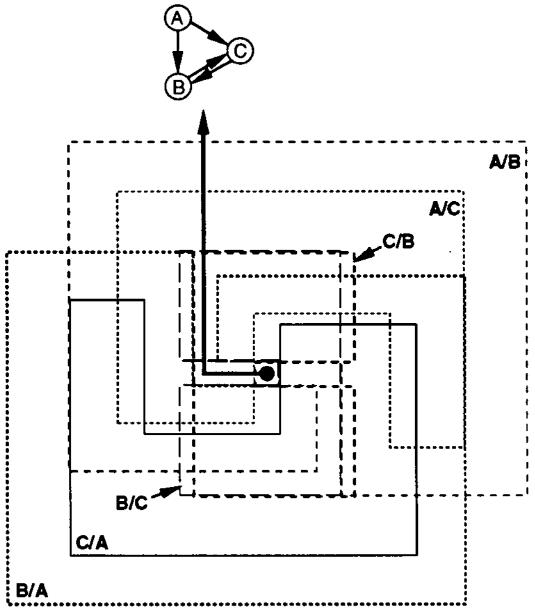

Fig. 6. A two-step translation and the corresponding DBG.

shows a simple assembly made of three parts, A, B, and C, and its interference diagram. The central cell is in the middle and is marked with a thick dot.

We now consider the M-space of all multistep translations. We are only interested in paths that partition A into two subassemblies. Each such path p must be a polygonal line connecting O to the external cell of $I(A)$. The DBG $G(p)$ for this path can be constructed as follows. Initialize the set of arcs in $G(p)$ to the empty set. Then, for every cell c of $I(A)$ traversed by p, for every P_i/P_j in the label of c, add (P_i, P_j) to the set of arcs of $G(p)$. Figure 6 shows a two-step translation path in the interference diagram of Figure 5 and the DBG associated with this path.

This construct yields a decomposition of the M-space into an arrangement of path cells: each path cell is defined by a sequence $(c_1, \ldots, c_r)$ of cells of $I(A)$, such that c_1 is the central cell, c_r is the external cell, and c_k and c_{k+1} are adjacent for all $k = 1, \ldots, r-1$. The DBG remains fixed over each path cell.

This construction is very general, with the interference diagram used as an intermediate structure toward the NDBG. The same construction can be made for polyhedral assemblies. More generally, it can also be extended to any assembly whose parts are modeled as semialgebraic sets, for general multistep motions. However, even for multistep translations in two dimensions, the resulting NDBG has impractically large size. This leads us now to investigate the case of two-step translations in two dimensions in a more specific way.

5.2. *Two-Step Translations in Two Dimensions.*

Let A be a planar assembly of n polygons $P_1, \ldots, P_n$. We wish to solve the partitioning problem for A when the allowable motions are two-step translations.

As in the construction of the interference diagram we place a coordinate system of origin O in the plane and we describe each part P_i as a point set in this system. A partitioning two-step translation consists of a line segment s connecting O to some point (x, y) and a ray r originating at this point. We denote the direction of r (i.e., the angle it makes with the positive x-axis) by θ. Since each possible partitioning path can be represented by the parameters (x, y, θ), the motion space in this case is three-dimensional.

We consider every pair (P_i, P_j) of parts and compute the boundaries of the M-region $\mathcal{P}_{ij}$. We first compute the boundary patches induced by the first translation s, and then the boundary patches induced by the second translation r to infinity. The union of these two sets of patches constitutes the boundary of $\mathcal{P}_{ij}$.

Constructing the M-regions precisely may be a rather difficult task. Therefore, we simplify this step by computing a family $\mathcal{S}$ of surface patches (here called surfaces, for short) whose union contains all the boundaries of all the M-regions. Note that by this simplification we do not give up the completeness of the assembly planner. $\mathcal{S}$ induces an arrangement that decomposes M-space into cells of dimensions 3, 2, 1, and 0, such that the DBG is fixed over each cell. The resulting NDBG is a refinement of the NDBG that would have been computed if we had only computed the boundaries of the M-regions.

In [18] we give a detailed account of how to compute the family $\mathcal{S}$. Here we briefly demonstrate how to construct the surfaces induced by the first translation for a pair of polygonal parts P_i and P_j; see Figure 7.

We start by considering the motion space of the first translation which is two-dimensional. In Figure 7(a) the two parts are displayed in their relative placement in the assembly. Figure 7(b) displays the set $P_j \ominus P_i$. The bold-line edges in this figure are the central envelope of $P_j \ominus P_i$, namely, the first point of intersection of any ray from the origin with $P_j \ominus P_i$. Any translation that ends in a point (x, y) causes P_i to collide with P_j if and only if the segment s from the origin to (x, y) intersects the central envelope of $P_j \ominus P_i$. The shaded area in Figure 7(c) is the collection of points (x, y) such that the segment connecting them to the origin crosses the central envelope. Consequently, this shaded area is exactly the two-dimensional M-region $\hat{\mathcal{P}}_{ij}$. The central envelope together

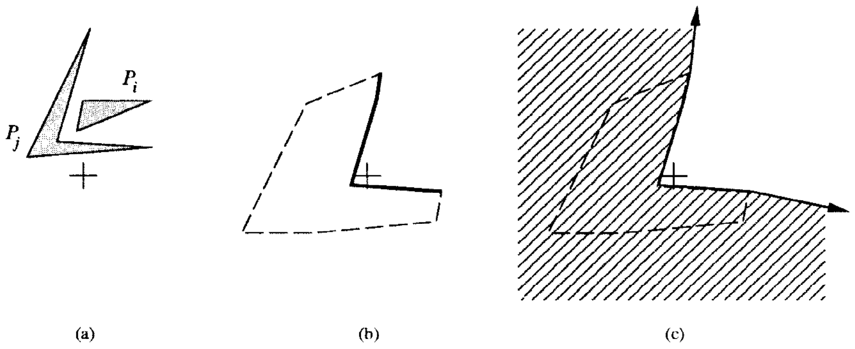

Fig. 7. Two-step translation: (a) two parts P_i and P_j in their relative placement in the assembly, (b) the set $P_j \ominus P_i$ (dashed) and its central envelope (bold), and (c) the M-region $\hat{\mathcal{P}}_{ij}$ (shaded) and its boundary.

with the two rays that extend from its endpoints and supported by the lines through the origin constitute the boundary curve of $\hat{\mathcal{P}}_{ij}$.

Next, we note that no matter what the second translation is, the same two-dimensional M-region induced by the first translation persists in every θ-cross section of the three-dimensional path space. Therefore we extend the boundary curve $\partial\hat{\mathcal{P}}_{ij}$ into a surface by extending a line in the θ direction from every point on the curve. This way we obtain the contribution of the first translation to the boundary surfaces of $\mathcal{P}_{ij}$.

The description of the surfaces induced by the second translation is more involved (we refer the reader to [18] for the details). We obtain these surfaces by first fixing a direction θ and analyzing the boundary curve for θ. Then we let θ vary and define the required surface as the union of boundary curves for all possible θ values.

We construct the boundary surfaces for every M-region $\mathcal{P}_{ij}$ and thus obtain the family $\mathcal{S}$. The next step is to construct the arrangement induced by $\mathcal{S}$. For a three-dimensional motion space, this can be done by a fairly simple algorithm [5] whose output enables us to traverse the arrangement cell by adjacent cell.

In [18] we extend this approach to handle paths consisting of two or more translations in the plane.

6. Improved Efficiency.

This section describes two improvements to the efficiency of the motion space approach that apply in certain cases. The first efficiency gain is purely combinatoric: it notes that successive DBGs vary only slightly, and amortizes some shared costs of testing their connectivity. The second gain is geometric: it reduces the number of DBGs that must be considered and finds them without computing the entire arrangement of M-regions, based on a property of that arrangement for certain types of motions.

6.1. *Amortizing Strong Connectivity Tests.*

In some representations of NDBG, the DBGs are most efficiently computed as one DBG G_0 for a first cell in the NDBG and a list of arc insertions and deletions (stepping into and out of M-regions) that generate all other DGBs $G_i, i > 0$. Khanna et al. [28] derive a more efficient algorithm to test all the DBGs for strong connectivity that is applicable in cases where the number of arcs in a blocking graph can be $\Omega(n^2)$, such as in one-step translation NDBGs. They group the graphs into **phases**, preprocess the common subgraph for each phase, then compute the strong connectivity of the full graphs.

Let a phase consist of some number K of arc operations in sequence, and assume that we are testing the first phase $G_0, \ldots, G_{K-1}$.[8] The total number of nodes involved in the K arc operations within the phase cannot exceed $2K$; call these the **active** nodes. Let E be the set of all possible arcs between the active nodes. Each graph G_i can now be divided into a common subgraph $G_i \backslash E = G_0 \backslash E$ and an active subgraph $G_i \cap E$. Preprocess the common subgraph $G_0 \backslash E$ for the phase by computing its transitive closure and storing the result in a subgraph H restricted to active nodes. Khanna et al. [28] prove that G_i is strongly connected if and only if both $G_0 \cup E$ and $H_i = H \cup (G_i \cap E)$ are strongly connected.

[8] We will also use the notation G_i somewhat loosely to mean the set of arcs of graph G_i.

The graph H_i has at most $2K$ nodes, so testing its strong connectivity requires $O(K^2)$ time. The preprocessing for each phase is dominated by computing the transitive closure of $G_0 \backslash E$. Let $T(N)$ denote the time complexity of computing transitive closure on an N-node graph. Then the running time for the phase is $O(n^2 + T(n) + K^3)$. This expression is minimized when $K = T(n)^{1/3}$.

However, observe that testing the sequence of graphs H_i for strong connectivity is equivalent to our original problem. This leads to a recursive approach with further improved running time. Using fast matrix multiplication to compute transitive closure, [28] achieve an amortized cost of $O(T(n)/n)$ or $O(n^{1.376})$ per graph tested.

6.2. *Maximally Covered Cells.* Let $\Gamma(A)$ be an NDBG for an assembly A. Suppose G_1 and G_2 are two DBGs of $\Gamma(A)$ such that $E_1 \subseteq E_2$ for their respective arc sets E_1 and E_2. Then it suffices to test G_1 for strong connectivity and we do not need to test G_2 for strong connectivity as well. This follows from the monotonicity of strong connectivity: if G_1 is strongly connected then so is G_2, and if G_2 is *not* strongly connected then neither is G_1.

This simple observation implies that if we can identify subset containment between DBGs we could spare strong connectivity tests. Moreover, if we could predict such containment relationship without even constructing the DBG for the "containing" set (E_2 in the example above), then the saving becomes more meaningful; this is because asymptotically, the time to test the DBG for strong connectivity is subsumed by the time to construct the DBG.

This idea leads to a substantial saving in the time for NDBGs induced by infinitesimal motions (see Section 4.3). We first exemplify the idea for infinitesimal translations in three dimensions, and then discuss it in the context of infinitesimal translations and rotations. Full details of this approach as well as experimental results can be found in [16].

Let A be an assembly of polyhedral parts $P_1, \ldots, P_n$, and let $\Gamma(A)$ denote the NDBG of A for infinitesimal translations. We represent the motion space as the unit sphere $\mathcal{S}^2$ in $\Re^3$. The M-region $\mathcal{P}_{ij}$ for every ordered pair of polyhedral parts (P_i, P_j) in A is a spherical polygon on $\mathcal{S}^2$, i.e., a region bounded by arcs of great circles (see Section 4.3). For the rest of this section, and to keep our notations compatible with those in [16], we define $\mathcal{Q}_{ij}$ as the complement of $\mathcal{P}_{ij}$ in M-space. Hence, $\mathcal{Q}_{ij}$ is the set of all directions such that an infinitesimal translation of P_i is possible without overlap with P_j.

It is convenient to project the arrangement induced by the boundaries of the regions $\mathcal{Q}_{ij}$ onto a plane Π tangent to $\mathcal{S}^2$ from the center O of the sphere. This covers only the regions (or portions thereof) on a hemisphere, but it is easily verified that the missing hemisphere contains precisely symmetric information and hence we do not lose any information by the projection. Some caution, however, is needed in choosing the plane Π so that we do not lose information that appears on the great circle that separates the hemispheres.

Consider the following scenario. The given assembly induces three Q-regions $\mathcal{Q}_{ij}$ on Π; see Figure 8. The boundaries $\partial\mathcal{Q}_{ij}$ of the Q-regions divide the plane into eight faces (two-dimensional cells) $f_0, \ldots, f_7$, having eight distinct DBGs. We claim that we need only examine DBG(f_0)—the DBG corresponding to the face f_0 where the three regions intersect. The DBG corresponding to any other face contains between one to three additional arcs that are not present in DBG(f_0). For example, DBG(f_1) has the same arcs as

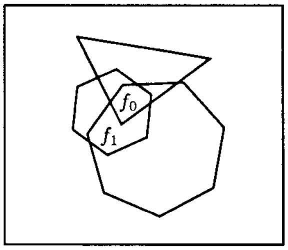

Fig. 8. It suffices to test DBG(f_0) for strong connectivity because no other cells are maximally covered.

DBG(f_0) plus the arc corresponding to the triangular Q-region (that does not contain f_1). In fact. the arrangement induced by the boundaries of the three Q-regions also contains cells of dimensions 1 and 0, but because no two Q-regions intersect without having their interiors overlap, no DBG associated with these lower-dimensional cells contains a subset of the arcs of every DBG(f_i).

We call the face f_0 a **maximally covered cell**. Informally, a cell is maximally covered if it is covered by (or contained in) more Q-regions than any of its immediate neighbors. Being maximally covered implies that the set of DBG arcs of the cell is locally minimal. In [16] we show that it suffices to test the DBGs of maximally covered cells for strong connectivity in order to solve the partitioning problem. Note that if some Q-regions touch one another without overlapping, the maximally covered cell may not be two-dimensional.

The discussion of maximally covered cells so far applies to any NDBG. Although we do not know how to exploit it in arbitrary NDBGs, we have been able to make good use of it for NDBGs induced by infinitesimal motions. We have shown the following result in [16]:

Given an assembly A of n polyhedral parts, the NDBG $\Gamma(A)$ of the assembly for infinitesimal motions, where any direction of motion is defined by d parameters, has at most $O(K^d)$ maximally covered cells, where K is the number of ordered pairs of parts in contact in the assembly. A sample direction in each maximally covered cell can be computed in total time $O(K^{d-1}k)$, where k denotes (as in Section 4.3) the number of "equivalent" point–plane contacts in the assembly.

For infinitesimal motions in three dimensions, the dimension of the motion space is 5. The maximally covered cells approach leads to significant savings over the best previously known algorithm [44] to solve this partitioning problem. In addition, the algorithm is based on linear programming techniques and hence simpler to implement robustly than the arrangement computations required by most NDBGs. In particular, we have implemented an algorithm based on this approach and have been able to solve the partitioning problem for sizable instances; see more details in Section 9.

7. Toleranced Assemblies.

Part tolerancing addresses the fact that manufacturing processes are inherently imprecise and produce parts of variable shapes. Tolerancing languages (e.g., Y14.5) provide designers with symbolic means to specify acceptable variations. Given an assembly made of toleranced parts, one would like to compute assembly sequences that are valid for all possible instances of parts manufactured according to the specified tolerances. The NDBG concept can be adapted to solve this problem.

To illustrate, consider a planar assembly A made of n parts $P_1, \ldots, P_n$. Each part is a polygon in which each edge e is supported by a line of fixed orientation located anywhere in a narrow strip called the **tolerance zone** of e. The tolerance zones are narrow enough to guarantee that all instances of a given part have the same topology. The placement of the parts in A is uniquely defined by spatial relations. Each spatial relation fixes relative placement of two parts P_i and P_j. It consists of elementary relations between edges and vertices of P_i and P_j, such as: vertex v_i of P_i is at some distance from edge e_j of P_j.

Let the **relation graph** of A be defined as follows: its nodes are the parts $P_1, \ldots, P_n$; each pair (P_i, P_j) is a nondirected link if and only if a spatial relation has been specified between P_i and P_j. A necessary and sufficient condition for the set of spatial relations to be complete and nonredundant is that the relation graph of A be connected and without cycles.

Assume that we only allow one-step translations, so that the M-space is $\mathcal{S}^1$. Consider any two parts P_i and P_j. Due to the toleranced geometry of the parts in A, the arc of $\mathcal{S}^1$ representing the translations along which P_i is blocked by P_j is not fixed over multiple instances of A. This leads us to define $\mathcal{P}_{ij}$ as the *union* of all such arcs, when the parts in A span all their possible geometries. Using the arcs $\mathcal{P}_{ij}$ so defined, we can create the NDBG of A exactly in the same way as in Section 4.1. Any partitioning of A that can be computed from this NDBG is guaranteed to be feasible for any geometry of the parts in A.

The main novel issue here is the computation of $\mathcal{P}_{ij}$. In [32] we describe an algorithm that performs this computation in time $O(r_{ij}(q_i q_j + \log r_{ij}))$, where q_i and q_j are the number of edges of P_i and P_j, respectively, and r_{ij} is the length of the path connecting P_i and P_j in the relation graph of A. Hence, if q is the maximal number of edges in a part of A and r is the length of the longest path in the relation graph, the NDBG of A is obtained in time $O(n^2 r(q^2 + \log r))$. The algorithm generalizes to polyhedral assemblies; in that case, the NDBG is computed in time $O((nr)^2 q^4 \log(nq))$.

In [32] we named the above NDBG the **strong** NDBG of A. By defining $\mathcal{P}_{ij}$ as the *intersection* of all arcs of $\mathcal{S}^1$ representing the translations along which P_i is blocked by P_j over all possible geometries of the parts of A, we also introduced the **weak** NDBG of A. If no partitioning can be derived from the weak NDBG, then it is guaranteed that no assembly sequence exists whatever the geometry of the parts. The algorithm proposed to compute the weak NDBG takes time $O(n^2 r(q^9 + \log r))$.

The algorithms given in [32] assume a relatively simple tolerancing language. In particular, it allows no tolerancing on the orientation of an edge of a polygonal part or a face of a polyhedral part. Extending the algorithms effectively to handle such angular tolerances remains an open problem.

8. Additional Constraints.

The NDBG as presented thus far provides the ability to generate efficiently assembly operations in which no parts will collide. However, assembly plans must satisfy many other constraints regarding considerations as diverse as stability, tool use, assembly line layout, supplier relationships, testing requirements, and so on [23].

Blocking constraints alone often reduce the number of feasible operations generated by the NDBG to a small number that can be tested against these additional constraints. However, in some cases a very large number of assembly operations are geometrically feasible but only a few satisfy the additional constraints. Using a generate-and-test paradigm in these cases will be very inefficient.

The following subsections describe methods to integrate some additional constraints on assembly plans into the motion space approach. In each case, the DBGs are processed in a modified way or additional nodes and arcs are inserted into them in order to encode the constraints. In general, the methods below are compatible and can be combined with some care in implementation.

8.1. *Connected Subassemblies.* The **connection graph** $G_C(A)$ of an assembly A is an undirected graph with a node for each part of A and an edge connecting two nodes if and only if the corresponding parts are in contact in A. Then a subassembly S is called **connected** when the nodes of S induce a connected subgraph of $G_C(A)$. Unconnected subassemblies are in general harder to fixture, manipulate, and mate with other subassemblies, and real assembly plans rarely include them. Requiring connected subassemblies is a common constraint in assembly planning systems and is implicit in cutset-based methods (see Section 2.2).

When an NDBG has been computed for infinitesimal motions, then any arc (P_i, P_j) in a DBG has a corresponding undirected edge (P_i, P_j) in $G_C(A)$, because one part can only block another if they are in contact. This fact allows us to extend the `partition` algorithm with a postprocessing step that guarantees that the two generated subassemblies are connected.

We assume that A is connected. Let G be a DBG of A and let S be a strong component of G that is blocked by no other parts. Then S must be connected, because S is a strong component and every arc between parts in G has a corresponding undirected edge in $G_C(A)$. However, the partitioning $(S, A\backslash S)$ is only valid if the remaining subassembly $A\backslash S$ is also connected. If it is, then we are done.

If $A\backslash S$ is not connected, then it consists of a number of connected components C_i. Choose any connected component C_1 of $A\backslash S$. We prove in [39] that the partitioning $(A\backslash C_1, C_1)$ is feasible in G and that both C_1 and $A\backslash C_1$ are connected. Furthermore, [39] shows how to generate all removable subassemblies under the connected subassembly constraint.

For types of motion whose blocking relations are not restricted to part contacts, efficiently generating free partitionings where both subassemblies are connected is an open problem.

8.2. *Tool Access.* Many mechanical assembly operations require the use of various tools to manipulate, attach, and test parts and subassemblies. Let a **post-tool** be an assembly tool that is applied after the corresponding parts are mated; the great majority of common assembly tools are post-tools [41]. Examples of post-tools include welders, screwdrivers, wrenches, and most inspection and measurement operations. The NDBG framework can be extended in a simple way to ensure post-tool access. We assume that a feasibility predicate $\mathcal{F}(S)$ determines whether a given post-tool can be applied in subassembly S.

A tool must be used in the first assembly operation that results in a subassembly that includes all the parts in a **target part set**. To be more specific, consider an assembly operation joining subassemblies S_1 and S_2 to make a larger subassembly $S = S_1 \cup S_2$. If a tool has target part set T, then it must be used in the operation if and only if

$$T \subseteq S \;\wedge\; T \not\subseteq S_1 \;\wedge\; T \not\subseteq S_2.$$

For instance, the target part set for a wrench that turns a nut P_1 onto a bolt P_2 will usually be the set $\{P_1, P_2\}$. If either P_1 or P_2 is missing in the joined subassembly, or if P_1 and P_2 are in the same subassembly before the operation, then the wrench is not needed.

Consider now partitioning a subassembly S with NDBG $\Gamma(S)$. If the target part set T for a tool is a subset of S, then the tool is required for any partitioning of S that splits the parts of T. If $\mathcal{F}(S)$ returns true, then use of the tool presents no additional constraint on partitioning S. On the other hand, if $\mathcal{F}(S)$ returns false, then T cannot be split by any partitioning of S at this stage of assembly. In this case, we simply add bidirectional arcs between all pairs of parts in T in all the DBGs of $\Gamma(S)$. This modification places the parts of T in the same strong component of each DBG, so that they cannot be split by any partitioning generated from those graphs. The extension can be performed for any number of tools that might be required to build S. Then applying the standard partitioning algorithm to the modified $\Gamma(S)$ will generate exactly those partitionings (S_1, S_2) such that the tools required to mate S_1 with S_2 are feasible.

We give more detail on these methods in [40] and [41], including feasibility predicates $\mathcal{F}(S)$ for geometric access of a variety of mechanical assembly tools and methods to amortize the cost of computing the predicates over many invocations in assembly planning. Repeated application of an NDBG partitioning algorithm with these feasibility predicates for post-tools never requires backtracking, resulting in polynomial-time assembly planning.

8.3. *Fixturing.* In real assembly operations, fixtures are needed to keep intermediate subassemblies stable. An adaptation of the NDBG to deal with fixturing has been proposed in [37].

The method in [37] models a fixture as a collection of frictionless points called fixels (for “fixture elements”). Each assembly operation in a computed assembly sequence consists of adding a single part to the subassembly built so far (initially, only the fixture). The fixture is *unchanged* throughout the whole sequence; hence, it must guarantee the stability of all intermediate subassemblies. No fixel is ever allowed to penetrate the interior of a part.

The method creates an initial fixture by distributing fixels at random on the boundary of the assembly A. To test for the stability of A (and later the stability of any given subassembly), it invokes a stability predicate $\mathcal{S}(A)$ whose implementation uses linear-programming techniques. The initial set of fixels, which contains many more fixels than needed to insure the stability of A, will be pruned during the computation of an assembly sequence.

The method first constructs the NDBG Γ of the assembly that is defined as the union of A and all the initial fixels, for one-step translations. Then, to compute an assembly sequence, it performs the following operations iteratively, until all parts of A have been removed from Γ or instability has been detected:

1. Traverse the cells of Γ until a DBG G is found in which there exists a part P such that no arc connects P to another part of A in G.
2. If no arc of G connects P to a fixel,
   - then remove P from Γ,
   - else
     (a) for every fixel F such that (P, F) is an arc of G, remove F from the set of fixels and from Γ;
     (b) test the stability of all the subassemblies generated so far, including A, with the new (reduced) set of fixtures;
     (c) if all subassemblies remain stable, remove P from Γ.

Several removable parts P and multiple translations for P may be identified at Step 1. Whenever the algorithm detects that a subassembly becomes unstable at Step 2(b), it backtracks, and selects another P and/or another translation. For every sequence eventually computed by the algorithm, the remaining set of fixels may still be redundant and can be simplified in a postprocessing step. This algorithm runs well and efficiently in practice [37].

The problem of simultaneously computing assembly sequences in which each operation may merge two subassemblies S_1 and S_2, neither of which is a single part, and generating the two fixtures that stabilize S_1 and S_2 is still an open problem.

8.4. *Other Constraints*. The above constraints constitute only a sampling of the constraints commonly imposed on assembly plans [23]. In [25] we describe an approach to interactive assembly planning that provides a library of 19 additional constraint types that users can impose on assembly sequences. Of these, six are integrated into the NDBG approach to avoid the inefficiencies of generate-and-test. We present two examples to give the flavor of these methods here.

In some cases the user wishes to require that a certain subassembly S_R appear in the assembly plan for a product A, due to a manufacturing constraint unknown to the planning system. When the subassembly S to be partitioned is a strict subset of S_R, then the constraint disallows splitting the parts of S_R until the extra parts have been removed. We encode this constraint by adding bidirectional arcs between all pairs of parts in S_R in each DBG of $\Gamma(S)$. When the subassembly to be partitioned includes only parts in the required subassembly S_R, no arcs are added, allowing S_R and its subassemblies to be partitioned.

A slightly more complex example results when the user specifies that certain parts, called **fasteners**, are to be added to the assembly as soon as another set of parts, the **fastened parts**, are all brought together for the first time. Fasteners (such as screws, nuts and bolts, and rivets) are very common features in mechanical assemblies, so handling them efficiently is important. Before partitioning a subassembly S, all fasteners in S are checked for whether they could be placed. For each that cannot, the corresponding fastened parts are bound together with bidirectional arcs in all DBGs in $\Gamma(S)$. The partitioning algorithm is then applied to the NDBG $\Gamma(S \backslash F)$, where F is the set of all fasteners in S, and the fasteners are added back implicitly after partitioning.

The fastener method is in fact a heuristic based on the fact that fasteners are usually small and do not interfere with operations of parts other than their fastened parts. The resulting partitionings must be checked to make sure that this assumption holds, i.e., that the fasteners do not collide with other parts in the partitioning. However, the effective reduction in the number of parts in S (often fasteners outnumber other parts) usually far outweighs this concern.

9. Implementations.

Several forms of NDBG have been implemented and tested. Some of these were part of experimental assembly planning systems, while others were stand-alone implementations. In this section we give brief descriptions and pointers to those implementations and corresponding experiments.

The NDBG was initially implemented in the assembly planner GRASP, described in [39] and [43]. GRASP implemented an infinitesimal translation NDBG in three dimensions for polyhedral parts, including the connected subassemblies extension of Section 8.1. Its implementation was simple but inefficient. It created and analyzed a DBG for the direction of translation given by the intersection of every pair of contact planes in the target assembly. This was supplemented by DBGs for additional directions of motion identified from feature-based part contacts, such as cylinder–cylinder and threaded contacts. The resulting one-step translations were then checked for collisions. GRASP explicitly compared NDBG-based partitioning against the cutset approach [22] for assemblies of up to 42 parts. The NDBG was somewhat faster in most tests; however, one assembly constituted a worst case for the cutset approach, which not surprisingly took a very long time to partition that assembly.

Schweikard and Wilson [38] describe experiments with an extended three-dimensional translational NDBG for assemblies of polyhedral parts. They calculate the M-region for a pair of parts by triangulating the faces of both polyhedra, computing the M-region for every pair of triangles, and taking their union. They use a standard line sweep [35] to construct the two-dimensional arrangement of regions and corresponding blocking graphs. The implementation was tested on hand-generated and random examples.

Romney et al. [36] reimplement and refine the methods of [39] and apply them to determine various measures of assembly complexity. They pay particular attention to issues affecting average-case performance. Their program, STAAT, explicitly constructs an NDBG for infinitesimal translations on the unit sphere, resulting in a much smaller set of translation directions to analyze than GRASP analyzes. STAAT not only generates assembly sequences, but also analyzes the NDBG directly to detect certain assembly properties, such as whether the product is a stack assembly (i.e., all parts can be assembled from a single direction). One of the distinguishing features of STAAT is an efficient implementation of the decomposition of $\mathcal{S}^2$ induced by infinitesimal translations in three dimensions [12].

The observation regarding maximally covered cells (see Section 6.2) has been exploited in an implementation of an algorithm for polyhedral assembly partitioning under infinitesimal rigid motions (translation and rotation) [16]. This implementation uses efficient linear programming to access the maximally covered cells. The program has been successfully run on elaborate examples, and experimental results are reported in [16].

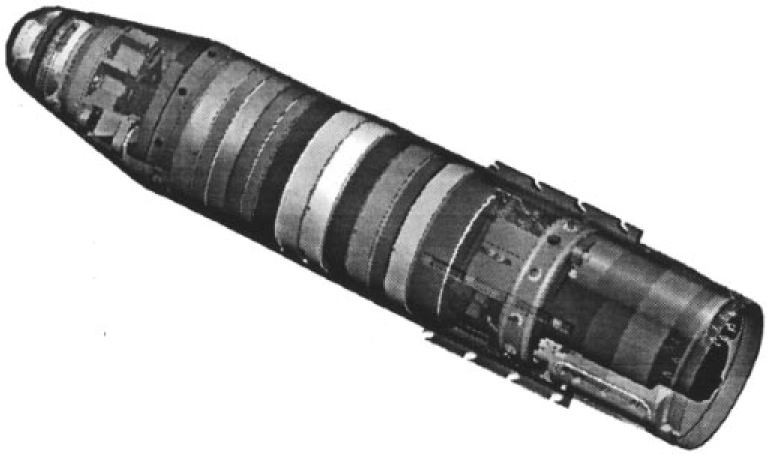

Fig. 9. Guidance section of the Hughes AIM-9X air-to-air missile. (Figure used by permission of Hughes Aircraft and the U.S. Naval Air Systems Command.)

The strong NDBG presented in Section 7 to deal with toleranced parts was implemented in an assembly planner described in [32]. This planner is written in C++. It was tested on several polygonal assemblies of small size. Experiments with this planner show that tolerances can have profound impact on the set of feasible assembly sequences.

The Archimedes computer-aided assembly planning system [24], [25] was designed to bring NDBG-based assembly sequence generation to practical use. Its NDBG implementation is very similar to GRASP's (above), but it includes all of the NDBG extensions described in Section 8 except fixturing, including a library of over 100 common hand and robotic tools. Despite the inefficient NDBG implementation, planning time is dominated by other computations, including contact finding, collision detection, and determining feasible tool placements. Archimedes can input CAD data from several commercial mechanical design systems, and has been applied to over 20 products at this time. The most complex is the guidance section of the Hughes AIM-9X air-to-air missile shown in Figure 9, consisting of 472 parts described by over 55 Mb of CAD data.

The method of Section 8.3 which concurrently plans an assembly sequence and generates a fixture that stabilizes all intermediate subassemblies is implemented in a system called Atlas 3 written in C. It has been successfully tested on many planar assemblies having 5–11 parts [37].

10. Conclusion. We have presented a general framework for assembly planning that consists of partitioning the space of possible separating paths into an arrangement of cells such that the part blocking constraints are fixed within each cell. We have applied this approach to assembly paths of several useful types, including one-step translations, multistep translations, and infinitesimal rigid motions. While the simple application of the motion space framework yields polynomial-time planning algorithms in each case, we have also described efficiency improvements for several cases. A variety of additional assembly constraints can be incorporated into the framework efficiently. This approach has proved powerful for both theoretical and practical purposes.

A number of open problems remain. The most obvious is whether assembly partitionings can be generated more efficiently for certain classes of motion than yielded by straightforward application of the motion space framework. We presented two improvements in Section 6, but the NDBG remains impractical to apply to complex classes of motions.

As mentioned in Section 8.1, the following is an open problem: for a class of finite (i.e., noninfinitesimal) separating motions, given an assembly A, identify in polynomial time a removable subassembly S such that both S and $A\backslash S$ are connected subassemblies, or report that no such subassembly exists.

Generating assembly sequences that satisfy tolerancing requirements is a very important problem, and one that is quite difficult for humans to solve in many cases. Our initial efforts in this direction (Section 7) are limited to very simple, nonrotational tolerance relations and an acyclic relation graph. More general methods are needed to apply these techniques to real assemblies and tolerancing systems.

Similarly, more sophisticated techniques are needed to merge assembly planning and fixture planning. Among the issues not solved by the work reviewed in Section 8.3 are refixturing, guaranteed polynomial-time fixture and assembly sequence co-design, and generating assembly fixtures for assembly sequences in which some operations mate two subassemblies with more than one part.

One very interesting application of assembly planning is computing physical measures of the "complexity" of an assembly. Assembly complexity measures include the number of hands, subassemblies, and different paths required to assemble a product, as well as other measures. We give methods to compute some complexity measures in [36] and [43]. As mentioned in Section 2.2 there are several hardness proofs on computing complexity measures. However, in general the problem of efficient computation of many cost measures is open.

Finally, one of the most valuable application areas of assembly planning is providing early design-for-assembly feedback on product designs. At present such feedback from automatic planners is limited to plan existence, visualization, assembly complexity measures, and incidental knowledge gained from applying an assembly planner to a product. Methods to generate more direct feedback, such as a list of "reasonable" changes to a product to make it easier to assemble, would be of great value.

References

[1] P. Agarwal, M. de Berg, D. Halperin, and M. Sharir. Efficient generation of k-directional assembly sequences. In *Proc. 7th ACM–SIAM Symp. on Discrete Algorithms*, pages 122–131, 1996.

[2] E. M. Arkin, R. Connelly, and J. S. B. Mitchell. On monotone paths among obstacles, with applications to planning assemblies. In *Proc. 5th ACM Symp. on Computational Geometry*, pages 334–343, 1989.

[3] D. F. Baldwin, T. E. Abell, M.-C. M. Lui, T. L. De Fazio, and D. E. Whitney. An integrated computer aid for generating and evaluating assembly sequences for mechanical products. *IEEE Trans. Robotics Automat.*, 7(1):78–94, 1991.

[4] M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal*. Volume 703 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1993.

[5] M. de Berg, L. J. Guibas, and D. Halperin. Vertical decompositions for triangles in 3-space. *Discrete Comput. Geom.*, 15:35–61, 1996.

[6] M. de Berg, M. Overmars, and O. Schwarzkopf. Computing and verifying depth orders. *SIAM J. Comput.*, 23:437–446, 1994.

[7] A. Delchambre and P. Gaspart. KBAP: An industrial prototype of knowledge-based assembly planner. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 2404–2409, 1992.

[8] B. R. Donald. The complexity of planar compliant motion planning under uncertainty. *Algorithmica*, 5:353–382, 1990.

[9] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, New York, 1987.

[10] H. Edelsbrunner, R. Seidel, and M. Sharir. On the zone theorem for hyperplane arrangements. *SIAM J. Comput.*, 22(2):418–429, 1993.

[11] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.

[12] M. Goldwasser. An implementation for maintaining arrangements of polygons. In *Proc.* 11*th Ann. ACM Symp. on Computational Geometry*, pages C32–C33, 1995.

[13] M. Goldwasser, J. C. Latombe, and R. Motwani. Complexity measures for assembly sequences. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 1581–1587, 1996.

[14] M. Goldwasser and R. Motwani. Intractability of assembly sequencing: unit disks in the plane. *Proc.* 5*th Internat. Workshop on Algorithms and Data Structures* (*WADS* '97). Volume 1272 of Lecture Notes in Computer Science. Springer, Berlin, pages 307–320, 1997.

[15] M. Goldwasser. Complexity Measures for Assembly Sequencing. Ph.D. thesis, Computer Science Department, Stanford University, Stanford, CA, 1997.

[16] L. J. Guibas, D. Halperin, H. Hirukawa, J. C. Latombe, and R. H. Wilson. A simple and efficient procedure for polyhedral assembly partitioning under infinitesimal motions. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 2553–2560, 1995. *Internat. J. Comput. Geom. Appl.*, 8(2): 179–199, 1998.

[17] L. J. Guibas and F. F. Yao. On translating a set of rectangles. In *Proc.* 12*th ACM Symp. on Theory of Computing*, pages 154–160, 1980.

[18] D. Halperin and R. H. Wilson. Assembly partitioning along simple paths: the case of multiple translations. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 1585–1592, 1995. To appear in *Advanced Robotics*.

[19] H. Hirukawa, T. Matsui, and K. Takase. Automatic determination of possible velocity and applicable force of frictionless objects in contact from a geometric model. *IEEE Trans. Robotics Automat.*, 10(3):309–322, 1994.

[20] R. L. Hoffman. Automated assembly in a CSG domain. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 210–215, 1989.

[21] L. S. Homem de Mello. Task Sequence Planning for Robotic Assembly. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, PA, 1989.

[22] L. S. Homem de Mello and A. C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Trans. Robotics Automat.*, 7(2):228–240, 1991.

[23] R. E. Jones and R. H. Wilson. A survey of constraints in assembly planning. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 1525–32, 1996.

[24] R. E. Jones and R. H. Wilson. An interactive assembly planning system. In *Video Proc. IEEE Internat. Conf. on Robotics and Automation*, 1997.

[25] R. E. Jones, R. H. Wilson, and T. L. Calton. Constraint-based interactive assembly planning. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 913–20, 1997.

[26] L. E. Kavraki and M. N. Kolountzakis. Partitioning a planar assembly into two connected parts is NP-complete. *Inform. Process. Lett.*, 55(3):159–165, 1995.

[27] L. E. Kavraki, P. Švestka, J.C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics Automat.*, 12(4):566–580, 1996.

[28] S. Khanna, R. Motwani, and R. H. Wilson. On certificates and lookahead in dynamic graph problems. In *Proc.* 7*th ACM–SIAM Symp. on Discrete Algorithms*, pages 222–231, 1996.

[29] H. Ko and K. Lee. Automatic assembling procedure generation from mating conditions. *Computer-Aided Design*, 19(1):3–10, 1987.

[30] J. C. Latombe. *Robot Motion Planning*. Kluwer, Dordrecht, 1991.

[31] A. Lazanas and J. C. Latombe. Landmark-based robot navigation. *Algorithmica*, 13:472–501, 1995.

[32] J. C. Latombe, R. H. Wilson, and F. Cazals. Assembly sequencing with toleranced parts. *Computer-Aided Design*, 29(2):159–174, Feb. 1997.

[33] T. Lozano-Pérez. Spatial planning: a configuration space approach. *IEEE Trans. Comput.*, 32(2):108–120, 1983.

[34] B. K. Natarajan. On planning assemblies. In *Proc. 4th ACM Symp. on Computational Geometry*, pages 299–308, 1988.

[35] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[36] B. Romney, C. Godard, M. Goldwassser, and G. Ramkumar. An efficient system for geometric assembly sequence generation and evaluation. In *Proc. ASME Internat. Computers in Engineering Conf.*, pages 699–712, 1995.

[37] B. Romney. On the Concurrent Design of Assembly Sequences and Fixtures. Ph.D. thesis, Electrical Engineering Department, Stanford University, Stanford, CA, 1997.

[38] A. Schweikard and R. H. Wilson. Assembly sequences for polyhedra. *Algorithmica*, 13(6):539–552, 1995.

[39] R. H. Wilson. On Geometric Assembly Planning. Ph.D. thesis, Technical Report STAN-CS-92-1416, Stanford University, Stanford, CA, March 1992.

[40] R. H. Wilson. A framework for geometric reasoning about tools in assembly. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 1837–44, 1996.

[41] R. H. Wilson. Geometric Reasoning About Assembly Tools. Technical Report SAND95–2423, Sandia National Labs, Albuquerque, NM, 1996. Submitted to *Artificial Intelligence*.

[42] R. H. Wilson, L. Kavraki, T. Lozano-Perez, and J. C. Latombe. Two-handed assembly sequencing. *Internat. J. Robotics Res.*, 14(4):335–350, 1995.

[43] R. H. Wilson and J. C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71(2):371–396, 1994.

[44] R. H. Wilson and T. Matsui. Partitioning an assembly for infinitesimal motions in translation and rotation. In *Proc. Internat. Conf. on Intelligent Robots and Systems*, pages 1311–1318, 1992.

[45] J. D. Wolter. On the Automatic Generation of Plans for Mechanical Assembly. Ph.D. thesis, University of Michigan, 1988.

[46] T. C. Woo and D. Dutta. Automatic disassembly and total ordering in three dimensions. *J. Eng. Industry*, 113(2):207–213, 1991.