A General Framework for Assembly Planning: The Motion Space Approach

Dan Halperin* Tel Aviv University

Jean-Claude Latombe† Stanford University

Randall H. Wilson‡ Eastman Kodak Company

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.

In the first part of the paper we give background material, present the motion space approach and describe applications of the approach to assembly motions of several useful types, including one-step translations, multi-step translations, and infinitesimal rigid motions. Several efficiency improvements over the basic algorithms are described. The resulting algorithms have been implemented and tested extensively on complex assemblies and we describe some of the experimental results. Most of the results described in this part have been published separately before and we integrate them here under the unifying concept of motion space.

In the second part (the last section) of the paper we survey major open problems in this domain. We discuss four topics where some progress has recently been achieved but which are in general still largely open: assembly planning with compound motions, with toleranced parts, and with additional practical constraints on assembly plans, and optimizing cost measures of assembly sequences.

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. But, 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 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 *non-directional 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 co-authored. 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.

*Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. halperin@math.tau.ac.il. 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.

†Department of Computer Science, Stanford University, Stanford, CA 94305, USA. latombe@cs.stanford.edu. Work on this paper by J.C. Latombe has been supported by Army MURI grant DAAH04-96-1-007.

‡Eastman Kodak Company, Albuquerque, NM 87112-1881, USA. rwilson@kodak.com.

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 and surveys related work. Section 3 presents the general principles of the motion space approach. Section 4 exemplifies the motion space approach for what we call one-step and multi-step motions, presenting polynomial-time algorithms to compute and use NDBGs for many useful families of motions, describing improvements of the efficiency of the basic algorithms, and surveying implementations of these algorithms. In Section 5 we present what we consider to be major challenges in the area of assembly planning. We concentrate on four topics where some results have already been obtained but which are still largely open: assembly planning with compound motions, assembly of toleranced parts, planning with additional constraints, and optimizing cost measures of assembly sequences.

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[1] 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 forms a connected subset of the workspace. However, except when 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 [16] 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 [35]. In the rest of this paper, *we will only consider two-handed assembly sequences.*

[1] 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 $\Re^k$.

With the above definitions, an assembly sequence only generates subassemblies of the final assembly. Such a sequence is said to be **monotone** [47]. 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. Such a sequence is called **non-monotone**. 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 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 5.3 we discuss the addition of other constraints.

Finally, it is convenient to introduce the following categories of motions. Let us 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 **multi-step 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, both for assemblies of polyhedra each with a constant number of vertices [35] and for assemblies of polygons [47]. 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 [44]. The problem remains NP-complete when motions are restricted to multi-step translations and other variations [27].

The assembly-by-disassembly strategy prevails throughout the literature on assembly planning [2, 3, 9, 21, 23, 29, 48]. 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, 23]), 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. Most often p is a simple path (e.g., a translation) that is inferred from contacts between parts [22, 48]. But 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 [21, 29, 47]. Others incorporated heuristics, for example, to drastically reduce the number of separating motions considered by the planner at each partitioning step [9, 21]. Some were quite general and hence limited to assemblies with small part count [3]. The NDBG was introduced to avoid this combinatorial trap [42, 45].

Arkin, Connelly, and Mitchell [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 [6]. 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 $\varepsilon > 0$ [8]. 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—see Section 5.4.

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 non-directional preimage is also based on a decomposition of a motion space [10, 32]. 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

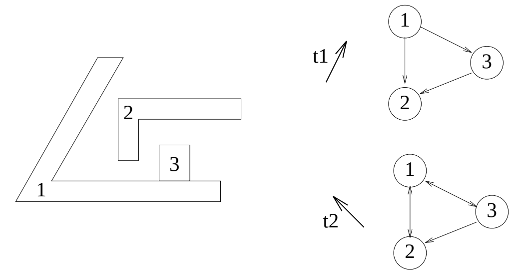

Figure 1: Examples of directional blocking graphs

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 3-D, 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 $p \in \mathcal{P}_{ij}$ induces an arc of $G(p)$ directed from P_i to P_j. We call this graph the **directional blocking graph**[2] (or DBG) of A for path p. See Figure 1 for an illustration of DBG's for one-step translation in 2-D.

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 bound-

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

ary [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 [18] 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 **non-directional blocking graph** (or NDBG) of A.

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 [42]. 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, it is computed once, before running `disassemble`. 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 dimension of M-space is independent of the number of parts in A. But 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.

4 Applications, Efficiency Improvements, and Implementations

In this section we review applications of the motion space approach, together with techniques to speed up the running time of the basic algorithms in certain instances. We then survey software implementations of the approach and experiments carried out with them.

4.1 One-Step motions

4.1.1 One-Step Translations in 2-D and 3-D

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 is defined by a single angular parameter representing the direction of the translation. Therefore, the motion space is represented by the unit circle $\mathcal{S}^1$.

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

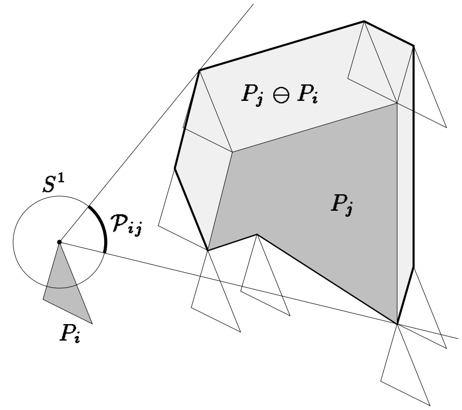

Figure 2: Construction of the M-region $\mathcal{P}_{ij}$

$-P_i$. See Figure 2, 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 non-convex 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 [42]. 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 $O(n^5)$.

The above techniques easily generalize to polyhedral assemblies. Then the motion space is the unit sphere $\mathcal{S}^2$ in $\Re^3$, and each region $\mathcal{P}_{ij}$ is obtained by "projecting" $P_j \ominus P_i$ onto $\mathcal{S}^2$, for pairs of polyhedral parts P_i and P_j. For further details see [42].

4.1.2 Infinitesimal Motions in 3-D

The direction of a one-step motion is given by a unit vector in 6-D. Hence, the set of all possible directions of motion make up the unit 5-D sphere $\mathcal{S}^5$. Computing the M-regions requires projecting 6-D regions of complex shapes onto $\mathcal{S}^5$. This computation is feasible in principle (as long as all parts of A are described as semi-algebraic sets), but it would be much too expensive to be practical.[3] 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

[3] However, this computation should be more reasonable in a 2-D workspace.

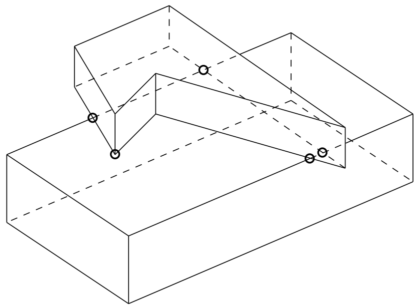

Figure 3: The face-face contact is equivalent to a set of point-plane contacts at the vertices V_k of the convex hull of the contact area (circled).

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.

As mentioned above, 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 6-D 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 that 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 in a piece of planar surface (see Figure 3). The set of motions dX 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 [20, 46]. 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 5-D hyperplane in the 6-D 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 $0, 1, \ldots, 5$ on $\mathcal{S}^5$. The DBG is fixed over each such region.

From this point on applying the general scheme is fairly straightforward (see [46] for details). In [17] we show that for the purpose of infinitesimal partitioning it is not necessary to build the entire NDBG, and this observation yields big savings in computation time; see also Section 4.3.2 below.

4.2 The Case of Multi-Step Motions

We present two ways to cope with multi-step motions. First, in Section 4.2.1 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. The interference diagram is an intermediate structure from which one can compute the NDBG. It provides a conceptually general method for constructing NDBGs for multi-step motions (in fact, for motions of any type), but the resulting algorithm may not be very efficient. In Section 4.2.2 we propose an efficient direct M-space algorithm that handles the specific case of two-translations in the plane.

4.2.1 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 multi-step 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 are bounded, 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 $I(A)$.

Let us now consider the M-space of all multi-step translations. We are only interested in paths that partition A into two subassemblies. Each such path p must be a curve[4] 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 $G(p)$.

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 semi-algebraic sets, for

[4] We can consider this curve to be a polygonal line without loss of generality.

general multi-step motions. However, even for multi-step translations in 2-D, the resulting NDBG has impractically large size. This leads us to investigate the case of two-step translations in 2-D in a different way.

4.2.2 Two-Step Translations in 2-D

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 $0, 1, 2$, and 3, 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 [19] we give a detailed account of how to compute the family $\mathcal{S}$; we omit the details here for lack of space. 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 [7] whose output enables us to traverse the arrangement cell by adjacent cell. In [19] we also extend this approach to handle paths consisting of three or more translations in the plane.

4.3 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 combinatorial: 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.

4.3.1 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 $G_0, \ldots, G_{K-1}$.[5] 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 $G_i \setminus E = G_0 \setminus E$ and an active subgraph $G_i \cap E$. Preprocess the common subgraph $G_0 \setminus E$ 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.

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 $G_0 \setminus E$. 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.

4.3.2 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.

In M-space we let Q_i denote the complement of an M-region $\mathcal{P}_i$, and call it the Q-region. Thus for an ordered pair of parts P_i, P_j, the corresponding Q-region denotes motions that do *not* cause P_i to overlap P_j. The key concept of the improvement is that of 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 [17] 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 full-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

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

for NDBGs induced by infinitesimal motions. We have shown the following result in [17]:

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 $O(K^{d-1}N)$, where N denotes the number of "equivalent" point-plane contacts (Section 4.1.2) in the assembly.

For infinitesimal motions in 3-D, the dimension of the motion space is 5. The maximally covered cells approach leads to significant savings over the best previously known algorithm [46] 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 4.4.

4.4 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 [42, 45]. GRASP implemented an infinitesimal translation NDBG in 3-D for polyhedral parts. 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 [23] 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.

Wilson and Schweikard [40] describe experiments with an extended 3-D 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 [37] to construct the 2-D arrangement of regions and corresponding blocking graphs. The implementation was tested on hand-generated and random examples.

Romney *et al* [39] reimplement and refine the methods of [42] 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 3-D [12].

The observation regarding maximally covered cells (see Section 4.3.2) has been exploited in an implementation of an algorithm for polyhedral assembly partitioning under infinitesimal rigid motions (translation and rotation) [17]. 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 [17]. In Figure 4 we display a model aircraft engine with twelve parts inducing 1,112 point-plane contacts that has been successfully handled by this program.

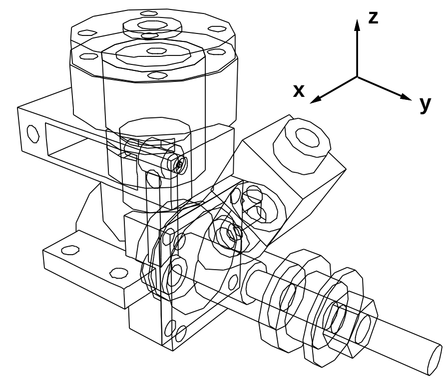

Figure 4: Assembled engine with 12 parts inducing $K = 24$ ordered pairs of parts in contact and $N = 1,112$ point plane contacts [17]: the program finds infinitesimal partitionings under translation and rotation. Previous solutions required time $O(N^5K^2)$, while in the novel approach the required time is $O(K^4N + K^6 \log N + N^{2+\varepsilon})$, for any $\varepsilon > 0$.

The special NDBG mentioned below in Section 5.2 to deal with toleranced parts was implemented in an assembly planner described in [31]. 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 [25, 26] 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 NDBG extensions to handle additional constraints (such as grasping—see Section 5.3), 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 5, consisting of 472 parts described by over 55Mb of CAD data.

A method which concurrently plans an assembly sequence and generates a fixture that stabilizes all intermediate subassemblies (see [38] and Section 5.3 below) is implemented in a system called Atlas 3 written in C. It has been successfully tested on many planar assemblies having 5 to 11 parts [38].

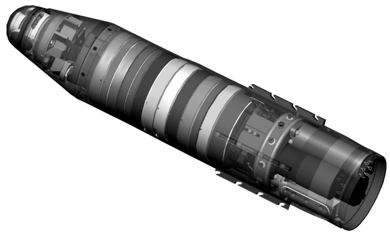

Figure 5: Guidance section of the Hughes AIM-9X air-to-air missile. (Figure used by permission of Hughes Aircraft and the U.S. Naval Air Systems Command.)

5 Open Problems

In this section we focus on four problems in assembly planning that, in spite of progress achieved in recent years, are still largely open.

5.1 Compound Motions

In most existing implementations of assembly planners the allowable motions of the parts in each assembly operation are rather limited (e.g., each is a single translation). Two reasons can be pointed out for this limitation: (i) simple motions are favorable in manufacturing, (ii) the algorithms for generating the sequence become too complicated if the allowable motions are not restricted.

However, there are occasions where more complex motions are required. In particular when dealing with product maintenance, namely with service and repair. For example, in aircraft engines certain parts need to be serviced every several dozens of flight hours. It is required that these parts can be accessed and removed without an overhaul of the engine.

Many new products are densely packed with a large number of parts of complex and diverse shapes. Answering maintainability questions for such products can be a very difficult task—it often calls for planning a single operation which may be extremely complicated. Motion planning techniques have already been applied to solve problems of this nature [5]. However, recent experiments have shown that current motion planning technology can offer only partial solution to these problems. Even the most advanced practical motion planners take enormous time to solve maintainability problems. Furthermore, their running time varies greatly and unpredictably. Note that if we need to remove a single part, then this is a variant of the motion planning problem where the obstacle environment is cluttered. If we need to find a removable subset along a complicated path, then this is the partitioning problem which is even harder.

Can the motion space approach be of help when compound assembly motions are necessary? We are currently examining a randomized variant of the motion space approach [4]. We aim to solve the assembly partitioning problem for polygonal parts with multi-translation paths, with about five-to-ten segments in a path. The algorithm for solving the problem as described in Section 4 would require the construction of an arrangement in ten (or more) dimensional space—far beyond what current geometric software technology offers. In [4] instead of constructing the full arrangement, we sample vertices, namely 0-dimensional cells, of this arrangement. This is in contrast with prevailing randomized motion planners that sample with very high probability points inside d-dimensional cells in d-dimensional arrangements. The reason to sample vertices rather than d-dimensional cells is that in tight assemblies a solution to the assembly planning problem may only lie in vertices.

While preliminary experimental results with sampling vertices are encouraging, they are still rather limited. The maintenance problem mentioned above may require dozens of steps in a single operation. Also, it is desirable to extend the approach to three-dimensional space and to compound motions involving rotations.

In summary, assembly planning raises natural problems with many degrees of freedom that call for the development of new and more advanced machinery. Unlike typical motion planning problems, assembly planning problems often require handling "tight" situations, difficult for existing motion planners.

5.2 Toleranced Assemblies

Tolerancing addresses the fact that manufacturing is inherently imprecise and produces parts of variable shapes. Tolerancing languages 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.

Consider an assembly A made of n polygonal parts $P_1, \ldots, P_n$. Each part 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 so 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 relation fixes the 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 non-directed link if and only if a spatial relation has been specified between P_i and P_j. This graph is assumed to be connected and without cycles.

Let us allow one-step translations only, 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. We define $\mathcal{P}_{ij}$ as the *union* of all such arcs, when the parts in A span all their possible geometries. Using the various $\mathcal{P}_{ij}$'s, we create the NDBG of A exactly as in Section 4.1.1. 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 novel issue here is the computation of $\mathcal{P}_{ij}$. In [31] 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 $O((nr)^2 q^4 \log(nq))$.

The algorithms in [31] assume a relatively simple tolerancing language. In particular, it allows no tolerancing on the orientation of an edge/face of a polygonal/polyhedral part. Extending the algorithms effectively to handle such angular tolerances, as well as developing strategies for different tolerancing languages, remain open problems.

5.3 Additional Constraints

The motion space approach presented here addresses the motions of the parts of an assembly, but many other constraints apply to the assembly process [24]. For example, fixtures are required to support unstable subassemblies; human hands or robot grippers must manipulate the parts; and various tools must be applied to fasten, move, or test parts [43]. These are active research areas by themselves, but the application to assembly raises important additional issues. A human-interactive approach to various assembly constraints that integrates well with the motion space approach is presented in [26], but we will focus here on computational solutions. We describe some characteristic open problems in fixturing and grasping for assembly.

Fixturing An assembly fixture must locate and support all partial assemblies in a chosen assembly sequence, but cannot interfere with any of the assembly operations. The sequence and fixture can be very dependent on each other. Hence solving the assembly fixturing problem efficiently requires considering fixture design and assembly sequencing together.

An initial step in this direction is described in [38]. The method models a fixture as a collection of frictionless points called **fixels** (for "fixture elements"). An initial fixture is created by distributing fixels at random on the boundary of the assembly. As assembly operations are generated, fixels that interfere with the part motions are removed. The remaining fixels must be sufficient to guarantee stability of all partial assemblies that appear in the sequence. Linear programming techniques test for assembly stability.

The approach described in [38] is limited in several ways. A single fixture must be used for the entire assembly sequence, and the set of fixels specified may not be physically realizable. It only allows adding a single part at a time to the assembly. Finally, a worst-case exponential-time algorithm searches the space of possible sequences and fixtures. Improvements in these areas will require careful representation and integration of the various geometric constraints.

Grasping Grasping and gripper design for single parts has been well studied (see e.g. [15, 34, 36, 41]). When applied in an assembly context, two important open problems arise: grasping sets of parts, and grasping sequences of parts. The former problem is closely related to fixture design for a static assembly. For the second problem, consider a single low-volume assembly robot that must assemble an entire product or a subset of a product. The robot can have several grippers available, each with a different set of parts it can grasp. However, grippers add to the cost of the cell and gripper changes are slow. A gripper can place a part if it can grasp it stably and neither the gripper nor the part collide with already-placed parts during the mating operation. Most industrial grippers are quite simple, such as parallel jaws of varying size and travel distance, configurations of suction cups of different sizes, and chucks that hold vertical cylindrical features of a certain diameter range.

The problem is to choose an assembly sequence and design a set of grippers so as to minimize the number of grippers, the number of gripper changes, or a given weighted sum of the two. A more complex problem is to choose a sequence to assemble the product on a given number of sequential robot workcells, so as to minimize the maximum assembly time taken by any workcell and/or the total number of grippers in the workcells. Solutions to these problems could significantly impact the economics of low-volume robotic assembly.

5.4 Cost Measures of Assembly Sequences

The motion space approach 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. For example, one may wish to minimize the number of directions of extended motion (one-step translation) used in 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, 14]. We cite here one hardness result from [14], for a problem that at first glance seems very simple. We are give 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] for full technical details on this and other hardness results in this area. In general the problem of efficient computation of cost measures is open.

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. on Robotics and Automation*, 7(1):78–94, 1991.

[4] N. Birnbaum and D. Halperin. A randomized approach to assembly planning along paths. Manuscript, 1998.

[5] H. Chang and T. Li. Assembly maintainability study with motion planning. In *Proc. IEEE Int. Conf. on Rob. and Autom.*, pages 1012–1019, 1995.

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

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

[8] M. de Berg, M. Overmars, and O. Schwarzkopf. Computing and verifying depth orders. *SIAM Journal on Computing*, 23:437–446, 1994.

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

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

[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. 11th Annu. ACM Sympos. Comput. Geom.*, pages C32–C33, 1995.

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

[14] M. Goldwasser and R. Motwani. Complexity measures for assembly sequences. *International Journal of Computational Geometry and Applications.* To appear.

[15] R. Grupen, T. Henderson, and I. McCammon. A survey of general-purpose manipulation. *Intl. J. of Robotics Research*, 8(1):38–62, 1989.

[16] L. Guibas and F. F. Yao. On translating a set of rectangles. *Advances in Computing Research*, 1:235–260, 1983.

[17] 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 Intl. Conf. on Robotics and Automation*, pages 2553–2560, 1995.

[18] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, 1997.

[19] D. Halperin and R. H. Wilson. Assembly partitioning along simple paths: the case of multiple translations. *Advanced Robotics*, 11:127–145, 1997.

[20] 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. on Robotics and Automation*, 10(3):309–322, 1994.

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

[22] L. S. Homem de Mello. *Task Sequence Planning for Robotic Assembly.* PhD thesis, Carnegie-Mellon Univ., 1989.

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

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

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

[26] R. E. Jones, R. H. Wilson, and T. L. Calton. Constraint-based interactive assembly planning. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 913–920, 1997.

[27] L. Kavraki and M. N. Kolountzakis. Partitioning a planar assembly into two connected parts is NP-complete. *Information Processing Letters*, 55(3):159–165, 1995.

[28] S. Khanna, R. Motwani, and R. H. Wilson. On certificates and lookahead in dynamic graph problems. In *Proc. 7th 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, 1991.

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

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

[33] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[34] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2:541–558, 1987.

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

[36] J. Ponce, S. Sullivan, A. Sudsand, J.-D. Boissonnat, and J.-P. Merlet. On computing four-finger equilibrium and force-closure grasps of polyhedral objects. *Intl. J. of Robotics Research*, 1997. To appear.

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

[38] B. Romney. *On the Concurrent Design of Assembly Sequences and Fixtures.* PhD thesis, Dept. of Computer Science, Stanford Univ., 1997.

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

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

[41] C. Wentink, A. F. van der Stappen, and M. Overmars. Algorithms for fixture design. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation: 1996 WAFR*, pages 321–346. A. K. Peters, 1997.

[42] R. H. Wilson. *On Geometric Assembly Planning.* PhD thesis, Stanford Univ., March 1992. Stanford Technical Report STAN-CS-92-1416.

[43] R. H. Wilson. Geometric reasoning about assembly tools. Technical Report SAND95–2423, Sandia National Labs, 1996. To appear in *Artificial Intelligence.*

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

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

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

[47] J. D. Wolter. *On the Automatic Generation of Plans for Mechanical Assembly.* PhD thesis, Univ. of Michigan, 1988.

[48] T. C. Woo. Automatic disassembly and total ordering in three dimensions. In *Conf. on Intelligent and Integrated Manufacturing Analysis and Synthesis*, pages 291–303, 1987.