GEOMETRIC REASONING ABOUT MECHANICAL ASSEMBLY

Randall H. Wilson
rwilson@cs.stanford.edu

Jean-Claude Latombe
latombe@cs.stanford.edu

Robotics Laboratory
Department of Computer Science, Stanford University
Stanford, CA 94305, USA

Abstract

In which order can a product be assembled or disassembled? How many hands are required? How many degrees of freedom? What parts should be withdrawn to allow the removal of a specified subassembly? To answer such questions automatically, important theoretical issues in geometric reasoning must be addressed. This paper investigates the planning of assembly algorithms specifying (dis)assembly operations on the components of a product and the ordering of these operations. It also presents measures to evaluate the complexity of these algorithms and techniques to estimate the inherent complexity of a product. The central concept underlying these planning and complexity evaluation techniques is that of a "non-directional blocking graph," a qualitative representation of the internal structure of an assembly product. This representation describes the combinatorial set of parts interactions in polynomial space. It is obtained by identifying physical criticalities where geometric interferences among parts change. It is generated from an input geometric description of the product. The main application considered in the paper is the creation of smart environments to help designers create products that are easier to manufacture and service. Other possible applications include planning for rapid prototyping and autonomous robots.

Acknowledgments: This research was funded by ARPA contract N00014-88-K-0620, NSF grant IRI-9306544-001, and a grant of the Stanford Integrated Manufacturing Association (SIMA). The authors also thank Leo Guibas, Rajeev Motwani, and Achim Schweikard for their encouragement and comments.

1. INTRODUCTION

Reasoning about mechanical assembly (and disassembly) is an important research topic which has attracted interest from researchers in both artificial intelligence [38] and robotics [27]. In which order can a product be assembled or disassembled? How many hands are required? How many degrees of freedom? What parts should be removed from the assembled product to allow replacement of a specified subassembly? To automatically answer questions like these, interesting theoretical issues in geometric reasoning must be addressed and computational techniques of broad interest must be developed. This investigation will also benefit many applications. The one considered in this paper is the creation of smart interactive environments to help designers create products that are easier to manufacture and service. Other important applications include planning for rapid prototyping and autonomous robots.

Given a mechanical assembly product (such as a toaster, an automobile, or a plane), an *assembly algorithm* specifies assembly and/or disassembly operations on its components and the ordering of these operations. The manufacturing, maintenance, and repair procedures for the product are all instances of assembly algorithms. These algorithms may be executed by humans (where the algorithm takes the form of an instruction sheet), robots (here expressed as a robot program), or specific machines (here an input to the design of these machines). Assembly algorithms are the key link between design and manufacturing/servicing. If we can measure the cost of executing an assembly algorithm, then the product's "inherent complexity", with respect to manufacturing and servicing, is given by the lower bound cost of all the assembly algorithms that are possible for this product.

Here we investigate the following two related problems: (1) the automatic generation of assembly algorithms from CAD data, and (2) the characterization of the complexity of assembly designs. Our main goal is to provide efficient computational support for the "concurrent engineering" approach to design, in which constraints arising from manufacturing and servicing are taken into account at design time [9]. As products are designed with more parts of various sorts (e.g., machined, composite, electrical, electronic) densely packed to provide more functions per cubic inch, the need for powerful assembly support tools will increase dramatically.

A typical CAD model of an assembly describes the geometry of the parts composing the assembly and the spatial relations among these parts. It is appropriate for graphic rendering and some man-machine interaction, but it does not directly provide the information that is needed to easily plan assembly algorithms. Indeed, to synthesize such algorithms one must first analyze how the various components in an assembly constrain their respective motions. However, there is a combinatorial set of potential interactions among parts. This suggests that the CAD model be converted into another representation making these constraints explicit. We propose one such representation, called the *non-directional blocking graph*, or NDBG, which describes the potential interactions among parts in polynomial space. Its construction derives from the observation that infinite families of motions can be partitioned into finite collections of subsets such that the interferences among the parts are constant over every subset. Once computed, the NDBG can be exploited for a variety of purposes, including the efficient (polynomial) generation of assembly algorithms.

There are often many algorithms to assemble or disassemble a product, and we may not want to produce all. To deal with this issue, we propose to measure the complexity of an algorithm along various axes, e.g., the number of hands it requires, the longest sequence of operations that cannot be parallelized, the total number of elementary motions. Along each axis, we define the algorithmic complexity of a product as the lower bound on the complexity of all assembly algorithms for this product. This definition immediately yields various classes of products, for instance the class of products that can be (dis)assembled with single translations only. The complexity of a product and its class provide concise pertinent information to feed back to the designers during design; it can also be used as a reference to select the most interesting assembly sequences. We will see that the NDBG is directly exploitable to estimate various complexity measures of a design.

The NDBG is a qualitative representation of the internal structure of an assembly product, which is obtained by identifying physical criticalities where interferences among parts change. Constructing the NDBG of an assembly design is a precomputation step allowing subsequent computations (synthesis of assembly algorithms, evaluation of complexity measures) to be performed in efficient query time. We will describe several types of NDBGs (for different families of motions) whose computation takes polynomial time in the size of the CAD input (number and complexity of parts). This computation can be done from a full assembly model, or incrementally, while the assembly is being designed. The exploitation of the NDBGs can be aimed at answering specific requests from the designers (e.g., how many parts must be removed from the assembled product before a specified subassembly can itself be extracted?) and/or posting warnings to the designers (e.g., when the product changes from one complexity class to another).

Notation: The following notation will be used throughout the paper:
- n: number of parts (polygons or polyhedra) in an assembly A,
- v: total number of vertices of the parts in A,
- r: number of pairs of parts in contact in A,
- c: number of contacts (edge-edge or plane-plane) in A,
- k: total number of vertices in the convex hulls of the c contacts in A.

All other notations are defined and used locally.

2. RELATED WORK

The automatic planning of assembly and disassembly operations has attracted the interest of AI researchers for a long time. The classical blocks world can be seen as a primitive assembly planning domain [15, 38]. Moreover, some AI planners have considered more complex domains. For instance, NOAH [42] was originally aimed at supplying instructions to a human apprentice to repair an air compressor, including disassembly and assembly plans. However, because they are usually interested in more general planning issues than just assembly planning, virtually all AI planners make use of a very abstract geometric description of the objects and their relations expressed in logical notation, e.g. `ON(A,B)`. Additional geometric knowledge is implicitly coded in the operators representing the actions that can be executed (e.g., stacking a block on top of another). A noticeable exception is BUILD [13], which includes a simple treatment of such notions as stability and friction. The interest of AI in general planning is still very high [35].

The geometric approach to assembly planning originated in robotics with the work reported in [29] (AUTOPASS), [31] (LAMA), and [45]. It is more limited in scope than traditional AI planning and focuses specifically on issues raised by the manipulation of physical objects. It has motivated various research in basic path planning, motion planning with uncertainty, manipulation planning with movable objects, and grasp planning [27]. However, the high complexity of assembly planning when viewed as a general motion planning problem has led researchers to turn their attention toward a simpler subproblem known as *assembly sequence planning*, or simply *assembly sequencing* [1, 22]. In this problem, only the constraints (mainly geometric ones) arising from the assembly itself are considered; the manipulation system (e.g., the robots) executing this plan is simply ignored by assuming that the parts composing the assembly are flee-flying objects.

The early assembly sequencers were mainly sequence editors. Geometric reasoning was supplied by a human who answered questions asked by the computer systems; the assembly sequences were inferred from the answers to these questions [7, 10]. Automated geometric reasoning was later added to answer these questions automatically [5, 19, 20, 28, 30, 47, 51]. This development resulted in generate-and-test assembly sequencers, with a module guessing candidate sequences and generating questions to check their feasibility, and geometric reasoning modules answering these questions. This approach tends to repeat the same geometric computations many times. Mechanisms for saving and reusing previous computations, such as the "precedence expressions" [47], have been proposed to overcome this drawback, but with limited success. In practice, the generate-and-test paradigm remains relatively inefficient (it takes time exponential in the number of parts) and is applicable only to assemblies with few parts. The NDBG avoids this combinatorial trap. Though there is a combinatorial set of potential parts interactions, the NDBG represents them in polynomial space, allowing valid operations and assembly sequences to be directly generated in polynomial time.

Originally, the research in assembly planning was aimed at assisting process planning in order to reduce delays between design and manufacturing, and eventually produce better plans [10]. This goal is still valuable, especially for rapid prototyping and even mass production. Recently, however, the interest has shifted toward generating assembly sequences to evaluate assembly designs and help designers create products that are easier to manufacture [44, 48]. In this new context, automated geometric reasoning and computational efficiency of assembly planning are critical issues that must be thoroughly explored. The synthesis of pertinent information to feed back to designers is another important issue. The concept of the algorithmic complexity of an assembly design presented in this paper directly addresses this issue. It derives in part from informal complexity measures currently in use in several companies (e.g., see [6]).

The field of computational geometry has also explored issues relevant to assembly planning, for example, set separation problems [46]. Given a 2D polygonal assembly A, the problem of deciding whether there is a direction d and a subassembly $S \subset A$ such that a translation along d separates S from the rest of A is addressed in [4]. An algorithm to construct a sequence of translations separating two polygonal parts is given in [39]. Several techniques presented in this paper have been influenced by the work in computational geometry.

The construction of an NDBG is based on the identification of physical criticalities to decompose a continuous set into a finite number of regions that are treated as single entities. This approach relates to the general interests of qualitative physics [11] and, more specifically, qualitative kinematics [14, 24], which studies the internal motions of parts in an operational device. It yields more meaningful decompositions than blind discretizations not based on any sort of criticality (discontinuity, singularity, or event). The notion of an "aspect graph" used in computer vision also has the same qualitative flavor as the NDBG. The aspect graph of an object describes the appearance of the object from all possible points of view. Aspect graphs were first computed by discretizing the set of viewing directions. Recent algorithms take advantage of the fact that, except at critical viewing directions, the occluding contours of an object remain qualitatively (i.e., topologically) the same for small changes in the viewpoint (e.g., see [25]). In [17] a criticality-driven approach makes it possible to plan a sensorless sequence of squeezing operations to achieve some specified orientation of a polygonal part independent of its initial orientation.

3. Virtual Manipulation Systems

Assume that we are given the description of a robot system and the design of an assembly product. We wish to plan an algorithm to make the robot system assemble the product from its individual parts. This problem can be formulated as a manipulation problem whose solution is a path in a large-dimensional space, the composite configuration space of the robot and all the movable parts [27]. A point (configuration) in this space fully represents a spatial placement of the robot and the other parts. A solution path describes all the motions that are necessary to construct the assembly. It is a continuous curve connecting an initial configuration where the parts are separated, to a goal configuration where they are assembled together according to the product design. The curve must satisfy certain physical constraints (e.g., the parts cannot move on their own). This view, however, makes assembly planning a highly complex motion planning problem. It also raises several conceptual issues that are still poorly understood, for example, the interaction between stability, fixturing, and grasping. Moreover, the manipulation system may not be known in advance. In fact, assembly planning is often used to help specify this system.

This leads to approaching assembly planning hierarchically, by solving a succession of simplified, but increasingly more realistic planning problems. The assembly algorithms for one problem are used to prune large subsets of the solution spaces of subsequent problems and to guide the search of these spaces. Each problem can then be seen as a planning problem for an abstraction of the (possibly not yet known) real manipulation system. We call this abstraction a *virtual manipulation system.* For example, at a high level of abstraction the parts composing the assembly are considered free-flying geometric objects; the corresponding constraints on assembly algorithms arise only from the product itself. At lower levels of abstraction, grippers, fixtures, and machines are introduced, along with uncertainties. As the virtual manufacturing system becomes more realistic, more detailed assembly algorithms can be generated. Just as one can write computer algorithms without knowing the details of their implementation, one can plan assembly algorithms without knowing which machines (or humans) will perform the manipulation.

Throughout the rest of this paper we will assume a very abstract virtual manufacturing system, in which individual parts are free-flying rigid geometric objects. This assumption corresponds to the most frequent situation in the early phase of the geometric design of a product and also underlies previous work in assembly sequencing (see Section 2). At this level of abstraction, we define an *assembly instruction* as the specification of the relative motions of m subsets of assembly parts, with $m \in \{2, 3, \ldots\}$, between an initial and a goal geometric arrangement of the parts. The m subsets are called the *moved sets* of the instruction. All the parts in the same moved set remain in constant relative position during the motion specified by the

Figure 1: Tree of an assembly algorithm for a simple product

instruction; usually, each moved set forms a connected composite object, but this is not required.

An algorithm to assemble a product from its parts is a partial ordering of assembly instructions in time. The graph of this relation is a tree whose maximal element (the root) is the instruction generating the product itself (see Fig. 1). Furthermore, no two instructions that are not comparable in the ordering move the same assembly component. In the virtual assembly system considered here, an algorithm to disassemble a product is obtained by simply inverting the ordering of an algorithm assembling the product. An algorithm to service a product usually combines instructions disassembling and re-assembling subsets of a product. For simplification, but without loss of generality, in the rest of this paper we will only consider algorithms to assemble products, with initial arrangements where all the parts are separated. Such an algorithm is *correct* if all its instructions specify collision-free motions of the parts (contacts between parts are allowed, however) and the final relative positions of the parts are those specified in the geometric description of the product. An assembly algorithm of this form is often called an *assembly sequence*.

4. Blocking Structure of an Assembly

In assembly planning the goal state (the assembled product) is considerably more constrained than the initial state (the separated parts). This suggests that planning

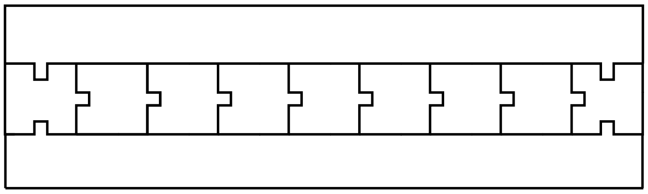

Figure 2: An assembly with two feasible decompositions

should proceed backward from the goal state, i.e., by disassembling the product. Indeed, the contacts among the parts in the assembled product can be used to quickly filter out many impossible motions.

Contact analysis is the basis of much previous work in assembly sequencing (e.g., [20]), where it was part of an overall generate-and-test planning approach. This approach recursively partitions an assembly into subassemblies and uses contact analysis to check the feasibility of each decomposition. However, the number of candidate decompositions is exponential in the number of parts, even though there may exist very few feasible ones. For example, in the assembly of Fig. 2, which consists of $n - 2$ interlocking parts sandwiched between two plates, only two decompositions are feasible. The inherent inefficiency of generate-and-test led many authors to restrict their planners to assembly sequences in which a product is built by adding a single part at a time. With this restriction, generate-and-test has a lower-order exponential dependence on the number of parts. But it is often preferable to build subassemblies that are later merged into larger ones.

The inefficiency of generate-and-test derives from the fact that essentially the same contact analysis is done many times. This repetition can be avoided by performing a complete contact analysis first. The results, recorded in a compact data structure, can then be used to directly determine which assembly decompositions are feasible. This approach yields the concept of an NDBG. For clarity, we introduce a first simple type of NDBG below (2D with infinitesimal translations). More sophisticated NDBGs will be presented in Section 5. In particular, we will show that NDBGs are not limited to contact analysis.

4.1. DIRECTIONAL BLOCKING GRAPH

Consider a planar assembly A made of n polygonal parts $P_1, \ldots, P_n$. The interiors of any two parts in A are disjoint. If the boundaries of two parts intersect, the two parts are said to be in contact.

Suppose that we wish to remove one part, say P_i, by translating it along a direction defined by the unit vector d. We say that a part P_j of A ($j \neq i$) *blocks* the

Figure 3: A simple assembly and two DBGs

translation of P_i along d if an arbitrarily small translation of P_i along d leads the interiors of P_i and P_j to intersect. Hence, if P_j blocks the translation of P_i, the two parts are necessarily in contact. A subassembly S of A is *locally free* to translate in direction d iff no part in $A \setminus S$ blocks the translation of any part of S along d.

The *directional blocking graph*, or DBG, $G(d, A)$ of A for an *infinitesimal translation* along d is a directed graph with nodes representing the parts of A. An arc connects P_i to P_j iff P_j blocks the translation of P_i along d. Fig. 3 shows a simple assembly made of 4 polygonal parts and its DBGs for infinitesimal translations along d_1 and d_2.

A subassembly S of A is locally free to translate in direction d iff no arcs in $G(d, A)$ connect parts in S to parts in $A \setminus S$. If $G(d, A)$ is strongly connected,[1] no such subassembly exists. Otherwise, at least one strong component of $G(d, A)$ must have no outgoing arcs. For example, in Fig. 3 the subassemblies $\{P_1, P_2\}$ and $\{P_1, P_2, P_3\}$ are locally free to translate in direction d_1; only the subassembly $\{P_3, P_4\}$ is locally free in direction d_2.

For a subassembly S to be assemblable with $A \setminus S$ by a translation along $-d$, it must be locally free to translate in direction d. This condition is necessary but not sufficient, since global accessibility is not considered. Also, merging S and $A \setminus S$ may require rotation. For those reasons, in Section 5 we will extend the above notions to non-infinitesimal translations and motions with rotation.

4.2. NON-DIRECTIONAL BLOCKING GRAPH

We represent the set of all translation directions by the unit circle $\mathcal{S}^1$. This circle is the locus of the extremity of the vector d when its origin is fixed.

Let P_i and P_j be two parts in contact. The set of directions in which P_i is locally

---

[1] A *strongly connected component* (or *strong component*) of a directed graph is a maximal subset of nodes such that for any pair of nodes (X_1, X_2) in this subset, a path connects X_1 to X_2. A graph is strongly connected if it has only one strong component.

Figure 4: Part of the NDBG of the assembly of Fig. 3

free to translate relative to P_j is a closed cone (possibly a half-space or a single ray) [20, 48]. For every pair of parts P_i and P_j in contact in A, we draw the diameters of $\mathcal{S}^1$ parallel to the two sides of the cone characterizing the local freedom of P_i relative to P_j. The drawn diameters partition $\mathcal{S}^1$ into an arrangement of regions, the endpoints of the diameters and the open circular arcs between them. Every such region is *regular* in the sense that the DBG $G(d, A)$ remains constant when d varies over it. We denote the DBG of A for any direction in a regular region R by $G(R, A)$.

The arrangement of points and intervals on $\mathcal{S}^1$ and the associated DBGs form the *non-directional blocking graph* $\Gamma(A)$ of A for infinitesimal translations. It represents the blocking structure of A for infinitesimal translations in all directions.

Part of the NDBG of the assembly of Fig. 3 is shown in Fig. 4. The partition of $\mathcal{S}^1$ in this example consists of 20 regular regions.

4.3. COMPUTATION

Let us assume for simplification that the contacts between any two parts in A are edge-edge contacts (hence, there are no isolated contact points). Assume also that if a concave vertex of a part coincides with a convex vertex of another part, then the edges abutting these vertices are in pairwise contact. These assumptions eliminate cases requiring specific attention; however, these cases can be easily treated within the complexity bounds given below.

Let the input to the computation consist of the geometric model of the parts in the assembly and the specification of the contacting edges between every two parts. If these contacts are not given explicitly they can be easily inferred from the spatial relations among the parts in quadratic time in the total number of vertices of the parts [48]. Let n be the number of parts in A and c the total number of edge contacts. We represent each DBG as an $n \times n$ adjacency matrix.

Under the above assumptions, the partition of $\mathcal{S}^1$ is obtained by drawing the diameters parallel to all edge contacts. It contains $O(c)$ regular regions and is computed in $O(c \log c)$ time.

For each regular region R, we select an arbitrary direction d in R and compute the DBG $G(d, A)$. After clearing the adjacency matrix for the DBG $G(d, A)$, each edge contact is considered separately. For every edge contact between parts P_i and P_j, the inner product of d with the outer normal to P_i in the contacting edge with P_j is computed. If the inner product is strictly positive, an arc from P_i to P_j is added to $G(d, A)$ (if it does not already exist). The computation of $G(d, A)$ takes $O(n^2 + c)$ time. We can avoid the initial clearing of the adjacency matrix by keeping track of the updated cells in a stack of pointers (see [2], page 71). This modification reduces the computing complexity of $G(d, A)$ to $O(c)$. By repeating the computation for all regular regions, the NDBG $\Gamma(A)$ is constructed in $O(c^2)$ time. The size of $\Gamma(A)$ is $O(cn^2)$.

The above computation can be refined as follows. For any pair of parts in contact, if there are more than two edge contacts, we only retain the two diameters of $\mathcal{S}^1$ which bound the cone of directions in which one part is free to translate relative to the other. This yields a total of $O(r)$ diameters, where r is the number of pairs of parts in contact in A. The arrangement of $\mathcal{S}^1$ is computed in $O(c + r \log r)$ time. The resulting NDBG is computed in $O(c + r^2)$ time and has $O(rn^2)$ size. We always have $r \leq c$ and for many assemblies $r \ll c$. While c depends on both the number and the complexity of the parts of A, r only depends on their number. Although in the worst case $r \in O(n^2)$, it is often smaller.

The NDBG can also be incrementally updated as the product is being designed. Each modification can be treated as a combination of part deletions and part additions. The deletion of a part P is obtained by removing the node corresponding to P and all its adjacent arcs from every DBG, and merging regular regions that were previously separated by contacts involving P. It can be done in $O(r)$ time. Adding a part contacting r' parts already in the assembly can be done in $O(r'r)$ time.

4.4. GENERATING CANDIDATE ALGORITHMS

The above NDBG $\Gamma(A)$ is an implicit representation of a set of assembly algorithms in which every instruction merges exactly two subassemblies without rotation. This set contains all correct assembly algorithms; it may also contain incorrect algorithms, since global accessibility constraints were not considered to compute the NDBG. Hence the assembly algorithms given by $\Gamma(A)$ must be validated by further tests (not described here; see [48]). These tests operate on assembly operations; therefore each instruction in an assembly algorithm should be validated before constructing the rest of the algorithm. Since $\Gamma(A)$ allows the direct generation of instructions satisfying local freedom constraints, the number of additional tests required is usually relatively small.

Let a candidate algorithm be any algorithm contained in $\Gamma(A)$, and a candidate partitioning of A be two subassemblies, S and $A \setminus S$, such that one is locally free to translate in some direction d. Each DBG contains n nodes and at most $2r$ arcs. Hence, finding the strong components of any DBG takes $O(r)$ time [3] and generating a candidate partitioning of A, given $\Gamma(A)$, takes $O(r^2)$ time. Furthermore, if a candidate algorithm exists, one candidate algorithm also exists for each of the two subassemblies in any candidate partitioning. Thus, generating a candidate algorithm takes $O(r^2 n)$ time. Let u be the total number of candidate partitionings of A; $u \in O(2^n)$, but for most assemblies it is much smaller. The set of all candidate partitionings of A for some direction d is computed in $O(ru)$ output-sensitive time by reducing $G(d, A)$ to the acyclic graph of its strong components (see [48]). The set of all candidate partitionings of A is computed in $O(r^2 u)$ time.

4.5. COMPUTING VARIANT

One can notice that there is little or no change between the DBGs of two adjacent regular regions. Thus, once we have computed the DBG for one region, call it R_1, we can incrementally modify this graph to get the DBG for the next region in the NDBG list, instead of computing this DBG from scratch. We can proceed in this same way until all the regions have been considered.

To do this, we slightly modify the DBG of a region by attaching a *weight* to each arc of the graph. In $G(R_1, A)$, this weight is the number of inner products that were strictly positive in the above computation. The absence of an arc from P_i to P_j is treated as an arc of weight 0, and conversely. Let R_1 be a circular arc. The next region R_2 in the NDBG is necessarily a singleton. Let D be the diameter of $\mathcal{S}^1$ that ends at R_2, and $\{E_1, \ldots, E_s\}$, $s \geq 1$, be the set of all contact edges in A parallel to D. $G(R_2, A)$ can be derived from $G(R_1, A)$ by applying the following *crossing rule*:

> Initialize G to $G(R_1, A)$. For every contact edge E_k ($k = 1$ to s), let P_i and P_j be the two parts sharing this edge. If the inner product of any direction in R_1 and the outgoing normal to P_i in E_k is strictly positive, then retract 1 from the weight of the arc connecting P_i to P_j in G.

The graph G obtained at the end of the loop is $G(R_2, A)$. (Again, every arc weighted 0 is interpreted as no arc.)

If R_1 is a singleton and R_2 a circular arc, the crossing rule is similar:

> Initialize G to $G(R_1, A)$. For every contact edge E_k ($k = 1$ to s), let P_i and P_j be the two parts sharing this edge. If the inner product of any direction in R_2 and the outgoing normal to P_i in E_k is strictly positive, then add 1 to the weight of the arc connecting P_i to P_j in G.

Using these crossing rules and representing only one DBG at any one time allow us to successively compute all the DBGs in $O(r)$ amortized time after the $O(c + r\log r)$ computation of the partition of $\mathcal{S}^1$. Indeed, the cost of computing a DBG by applying the crossing rule is proportional to the number s of contact edges involved in the computation. This number is in $O(r)$, but throughout the computation of the entire NDBG, each edge is considered only twice. Hence, the time complexity of the construction of *all* the remaining DBGs is only $O(r)$.

A candidate partitioning can be computed in $O(r^2)$ time, and a candidate algorithm in $O(r^2 n)$ time by constructing the DBG of every generated subassembly. Compared to the algorithm of the previous two subsections, this computing variant allows substantial space saving, since it does not require representing all $O(r)$ DBGs at any given time. Moreover, if a computation (for example, evaluating a complexity measure) uses the DBGs, one at a time, in the same sequence as they are generated, this incremental technique also has greater time efficiency.

5. OTHER BLOCKING GRAPHS

The notion of an NDBG introduced in the previous section admits several extensions and variants. We present some of them below. See [48] for more detail.

5.1. 3D ASSEMBLIES

The NDBG for infinitesimal translations can be easily extended to 3D assemblies made of polyhedral parts. As in the 2D case, we simplify our analysis and eliminate cases requiring specific, but straightforward attention, by making the following assumptions: (1) the contacts between any two parts in A are face-face contacts; (2) if a concave edge of a part coincides with a convex edge of another part, then the faces bounded by these edges are in pairwise contact; (3) if a non-convex vertex of a part coincides with a vertex of another part, then the edges abutting these vertices are in pairwise contact. Let c be the total number of face-face contacts between parts of A.

Figure 5: Need for rotation

The set of translation directions in 3D is represented by the unit sphere $\mathcal{S}^2$. The c plane-plane contacts induce arcs of great circles partitioning $\mathcal{S}^2$ into an arrangement of $O(c^2)$ regular regions of dimensions 2, 1 and 0 (faces, edges, and vertices, respectively). This arrangement is computed in $O(c^2)$ time using a topological sweep [12]. The NDBG is computed in $O(c^3)$ time and has size $O(c^2n^2)$. If only one DBG is represented at any one time, we can use crossing rules between regions and successively compute all the DBGs in $O(c^2)$ time. Generating a candidate partitioning of A into two subassemblies, given the NDBG, takes $O(rc^2)$ time, where r is the number of pairs of parts in contact. Generating a candidate assembly algorithm takes $O(rc^2n)$ time.

As in the 2D case, slightly lower complexity bounds can be obtained by considering each of the r pairs of parts in contact in sequence. For each pair, the set of directions in which one part is locally free to translate relative to the other is a convex cone. The intersection of this cone with the sphere $\mathcal{S}^2$ is a "polygon" bounded by arcs of great circles. Only the arrangement of regions created by these arcs need be used to compute the NDBG.

5.2. Infinitesimal Generalized Motions

One important extension is to allow motions in rotation. Fig. 5 shows a simple case where a part blocks another part for any infinitesimal translation, while a rotation is feasible.

Let us consider the 3D case only (the 2D case is just simpler), with polyhedral parts. The direction of an infinitesimal generalized motion (combining translation and rotation) is given by a unit vector in 6D. Hence, the set of all possible directions of motion make up the unit 5D sphere $\mathcal{S}^5$.

A Cartesian frame is attached to each part in A. For any two parts P_i and P_j,

Figure 6: Plane-plane contact expressed as point-plane contacts

the configuration (position and orientation) of P_i relative to P_j is defined as the position and orientation of the frame attached to P_i relative to the frame attached to P_j. An infinitesimal generalized motion of P_i with respect to P_j is described by a 6D vector $dX = (dx, dy, dz, d\alpha, d\beta, d\gamma)$. The components dx, dy, and dz are those of the translation of the origin of the frame of P_i along the axes of the frame of P_j. 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 the frame of P_j.

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 the 3×6 Jacobian matrix of the transform that gives the coordinates of V in the frame of P_j as a function of the configuration of P_i relative to P_j. 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 *allowed* by the contact between V and F are those satisfying $n_F J_V dX \geq 0$.

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 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 [18, 49]. For example, in Fig. 6, the vertices V_k are circled.

We make the same simplifying assumptions about contacts as in Subsection 5.1. 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 5D hyperplane in the 6D space of infinitesimal generalized 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 regions of dimensions $5, \ldots, 1, 0$ on $\mathcal{S}^5$. It is plain to see that the DBG is constant over each such region; hence, the regions of the arrangement are regular.

The intersection of two non-convex 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 [40] 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 by a multi-dimensional topological sweep [12]. Constructing a DBG in any region is done in time $O(k)$. A crossing rule similar to the pure translational case can be established, yielding an $O(rk^5)$ time algorithm to build the NDBG for a 3D assembly, where $r \in O(n^2)$ is the number of pairs of parts in contact.

A necessary condition for a subassembly S to be assemblable with $A \setminus S$ is that there exists a DBG G such that no arcs in G connect parts in S to parts in $A \setminus S$. However, this condition is not sufficient. The above NDBG is thus an implicit representation of a set of assembly algorithms in which every instruction merges exactly two subassemblies. This set contains all correct assembly algorithms; it may also contain incorrect algorithms due to the fact that global accessibility constraints have not been considered.

Given the above NDBG, computing a candidate partitioning of A into two subassemblies takes $O(rk^5)$ time. Generating a candidate assembly algorithm takes $O(rk^5 n)$ time.

5.3. INFINITE TRANSLATIONS

To address global accessibility, we now present a variant of the NDBG that derives from the analysis of the interferences among parts for a family of non-infinitesimal motions: infinite translations. Given any two parts P_i and P_j in A (these two parts might not be in contact), we say that P_j blocks the infinite translation of P_i along d if the volume that P_i sweeps out when it translates along d from its initial position in A to infinity intersects P_j. The notions of a DBG and an NDBG for infinite translations follow from this blocking relation.

Let P_i and P_j be any two parts in the assembly A. The set $\mathcal{B}_{ij}$ of directions d along which P_j blocks P_i is identical to the set of directions along which the "grown" object

$$P_j \ominus P_i = \{a_j - b_i \mid a_j \in P_j, b_i \in P_i\},$$

i.e., the Minkowski difference of the two sets of points P_j and P_i at their positions in A, blocks the infinite translation of the coordinate origin O. If P_i and P_j are polygons in 2D (resp. polyhedra in 3D), then $P_j \ominus P_i$ is also a polygon (resp. a polyhedron) [27, 32].

Again, let us consider the 3D case with polyhedral parts. $\mathcal{B}_{ij}$ is the intersection of $\mathcal{S}^2$ and the polygonal cone of all rays erected from O and intersecting $P_j \ominus P_i$. This intersection is a region of $\mathcal{S}^2$ bounded by segments of great circles. The great circles supporting these segments create an arrangement of regular regions overwhich the DBG for infinite translations remains constant. This arrangement and the associated DBGs form the NDBG of the assembly for infinite translations.

A sufficient condition for a subassembly S to be directly assemblable with $A \setminus S$ is that there exists a DBG G in the above NDBG such that no arcs in G connect parts in S to parts in $A \setminus S$. This condition is not necessary since, for instance, a path with multiple extended translations may be required. The above NDBG is an implicit representation of all correct assembly algorithms in which every instruction merges two subassemblies by a single translation.

Let v be the total number of vertices in the parts of A. The arrangement on $\mathcal{S}^2$ has size $O(v^4)$ and can be computed in $O(v^4)$ time. The NDBG is computed in $O(n^2v^4)$ time. Each DBG has $O(n^2)$ arcs, so that finding its strong components takes $O(n^2)$ time. Hence, deciding whether A can be broken down into two subassemblies that can be merged by a single translation (and constructing one such partitioning, if one exists) takes $O(n^2v^4)$ time. Generating an assembly algorithm takes $O(n^3v^4)$ time.

5.4. DISCUSSION

One can easily imagine other NDBGs. For example, one may consider the case where an assembly is constructed by merging more than two, say $m + 1$, subassemblies at a time. The corresponding NDBG for infinitesimal translations in 3D would be obtained by partitioning the $(3m-1)$-dimensional sphere $\mathcal{S}^{3m-1}$, representing m simultaneous translation directions, into an arrangement of regular regions.

NDBGs for infinite generalized motions and, more generally, sequences of extended motions along different directions could be investigated as well. The investigation for sequences of extended translations is under way in [33]. It leads to partitioning the composite configuration space $\mathcal{C}_A$ of the parts in A into regions over which all the relative placements of the parts induce the same NDBGs for infinitesimal translations. This construct yields a collection of NDBGs distributed over regions of $\mathcal{C}_A$. A careful representation of $\mathcal{C}_A$ yields a polynomial collection of NDBGs. From there, the generation of a partitioning of A for a bounded-length sequence of extended translations takes polynomial time. The construct can be generalized to sequences of extended generalized motions.

Each NDBG is defined for a limited family of motions and describes only the class of assembly algorithms for these motions. NDBGs for infinitesimal motions are underconstrained and may contain incorrect algorithms. NDBGs for infinite motions or bounded-length sequences of extended motions are overconstrained and may not include all possible algorithms. Trying to construct and exploit a unique NDBG or collection of NDBGs covering all possible motions would bring us back to the general planning problem of finding a coordinated path for a set of parts. This problem is known to be PSPACE-hard [23, 37] and is strongly believed to require exponential time in the number of parts.

Polynomial NDBGs such as those described above should be regarded as efficient filters to quickly identify feasible assembly operations and algorithms (for instance, using the NDBG for infinite translations) and eliminate infeasible ones (with the NDBG for infinitesimal rigid motions). When required, the candidate algorithms generated by the latter can be analyzed further using more general, but also more expensive, motion planning techniques.

Allowing more than two subassemblies to be merged simultaneoulsy and/or accepting sequences of extended motions yield NDBGs that are considerably more expensive to compute. An interesting development would therefore be to first construct the simplest NDBGs and then extend them locally (that is, for the subsets of parts that require the extension) when they do not allow the generation of assembly algorithms.

6. IMPLEMENTATION

We have implemented the algorithms constructing the NDBGs for infinitesimal and infinite translations both in 2D and 3D [48]. In 3D our implementation allows parts with planar, cylindrical, and some helicoidal faces. For infinitesimal generalized motions, we have implemented a hybrid algorithm that has the same time complexity as the translational version. This algorithm considers all pure translations, plus some "suggested generalized motions" inferred from nonplanar contacts in the assembly. For example, a cylindrical contact suggests pure rotation about its axis; a threaded contact between two helicoidal surfaces suggests a screwing motion; etc. The set of suggested generalized motions is incomplete but accounts for most motions required

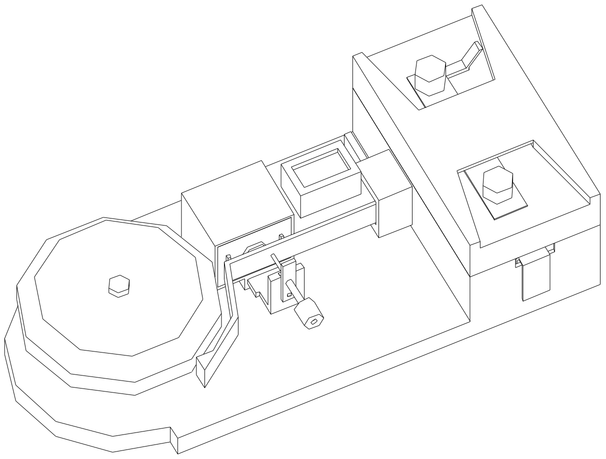

Figure 7: Electric bell

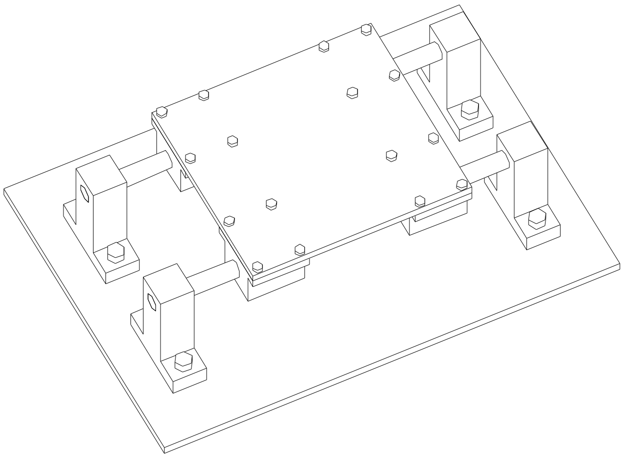

Figure 8: Friction-testing machine

by actual assembly products.

Our programs are written in CommonLisp as part of a larger assembly sequencing system [48] and run on a DEC 5000 workstation. We have used the NDBGs to compute candidate partitionings of assemblies and construct candidate assembly algorithms. We ran the programs on a variety of assemblies including a 22-part electric bell (Fig. 7), a friction-testing machine with 36 parts (Fig. 8), and the 42-part model-aircraft combustion engine (Fig. 9). Table 1 gives the running times of both the generate-and-test approach and the NDBG-based methods to identify all parti-

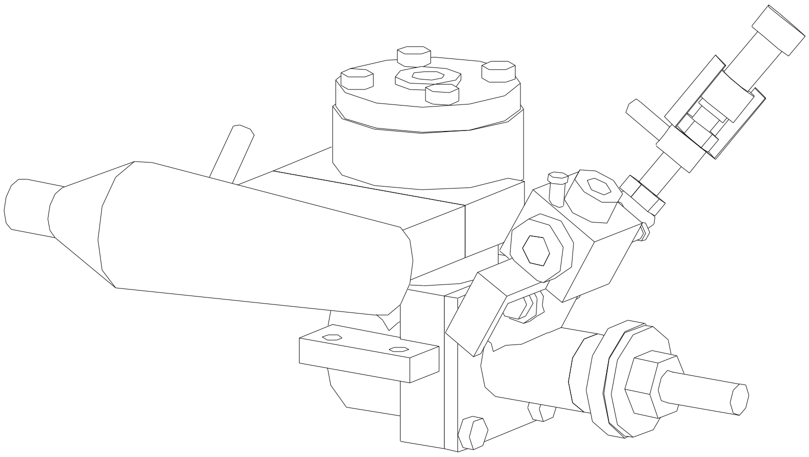

Figure 9: Model-aircraft engine

Assembly	Bell	Friction	Engine
Parts	22	36	42
Gen-and-Test	4.7	—	74
NDBG	1.4	3.4	8.5

Table 1: Times to identify candidate assembly instructions, in seconds

tionings (hence, final assembly instructions) that satisfy local freedom constraints. The generate-and-test algorithm was stopped after failing to find any candidate instructions for the friction-testing machine in two days.

Assembly sequences generated for the electric bell were converted into robot programs executed by a RobotWorld system [43].

7. COMPLEXITY OF AN ASSEMBLY

The outcome of assembly planning, i.e., assembly algorithms, can be used to specify, select, design, and/or program the manipulation systems that will execute these algorithms. Another goal which currently attracts increasing interest is to use this outcome to help designers create products that are easier to manufacture and service. The problem, however, is that there may be many possible assembly algorithms, especially when the manipulation system is loosely defined. Showing the most promising algorithms (for some criterion), e.g., by displaying a graphic simulation of their execution, would certainly be useful. But it would have to be limited to a few milestones of the design process in order to avoid taking too much of the designers' time.

Instead, we propose to measure the inherent complexity of a product (at its current stage of design) as the lower-bound complexity of all the assembly algorithms that are feasible for this product. This idea is analogous to the computational complexity of a problem, which is defined as the lower-bound complexity (in terms of time and space) of all the computer algorithms that solve the problem [16]. The estimated complexity measures for a product can be used to provide concise, pertinent information to the designers.

To be useful, however, the complexity of an assembly product must be measured along many more axes than that of a computational problem. We propose several such axes below, but by no means is this list exhaustive. Moreover, assembly algorithms apply to a number of parts that is small relative to the amount of data processed by typical computer algorithms. As a result, a much tighter evaluation of assembly complexity is required than the classical asymptotic analysis applied to computer algorithms. We will show how the NDBGs make it possible to perform this analysis in many cases.

7.1. COMPLEXITY MEASURES

Let an assembly product be *admissible* if it admits at least one correct assembly algorithm. In the following discussion we consider only admissible assemblies.

An interesting measure of the complexity of an assembly algorithm is the number of hands it requires. Let us say that an assembly instruction is ℓ-handed if it has $\ell + 1$ moved sets ($\ell \geq 1$). An assembly algorithm is m-handed if all its assembly instructions are ℓ-handed, with $\ell \leq m$, and at least one is m-handed. A product is *p-handed* if all correct assembly algorithms for this product are m-handed, with $m \geq p$, and at least one is p-handed. A p-handed product requires p moving "hands" to be (dis)assembled, in addition to a fixed one (e.g., a vise). It is shown in [37] that an assembly made of n parts may require up to $n - 1$ moving hands to be assembled (i.e., the assembly can only be built by moving every part relative to all the other parts simultaneously). However, it is in general much more cost-effective to manufacture and service a 1-handed product than a multi-handed one. The class of 1-handed products is thus an important one.

Let a *subassembly* be any subset of parts in their final relative positions. An assembly algorithm is *monotonic* if each of its instructions merges the corresponding moved sets into a subassembly [48, 51]. A product is *p-handed monotonic* if it admits a p-handed monotonic assembly algorithm. For example, the 3-part latch assembly shown in Fig. 10 is not 1-handed monotonic, but is 2-handed monotonic. It is clear that any admissible n-part assembly is $(n - 1)$-handed monotonic; however,

Figure 10: Three-part latch assembly

there exist p-handed n-part products, with $p < n - 1$, that are not $(p + i)$-handed monotonic, for any $i \in [0, n - p - 2]$. A non-monotonic assembly (for some given number of hands) requires bringing parts into intermediate relative positions (where they are ungrasped) that have to be changed later (which requires regrasping parts). It may require many more motions than a monotonic assembly with the same number of parts. So, the class of 1-handed monotonic products is an important subclass of the 1-handed products.

The nature and the number of degrees of freedom required to perform the motions specified by the assembly instructions are another key measure of product complexity. Let an assembly instruction be *m-prismatic* if the motion of every moved set is a sequence of at most m extended translations. Let a product be p-prismatic if it admits an assembly algorithm in which all instructions are m-prismatic, with $m \leq p$, for some initial arrangement of its parts in which every two parts are separated, and one instruction is p-prismatic. Since single translations are much more cost-effective to generate than multiple translations and generalized motions, the class of 1-prismatic products is another important one. One can further characterize the complexity of a product within this class by the minimal number of translation directions that are needed. A *stack* product is a 1-prismatic 1-handed monotonic product that admits an algorithm in which all instructions specify translations along the same direction. Large subassemblies of many small consumer electronic products are stack products. The concepts of prismatic and stack products can be easily extended to allow for screws in the products.

The length of the longest sequence of instructions in an assembly algorithm, the *length* of the algorithm, is another important attribute of this algorithm. Indeed, assuming that the time taken by any assembly instruction is lower and upper bounded regardless of the moved sets, it directly impacts the total execution time. The length of the shortest correct algorithm for a product is therefore a pertinent measure of the complexity of this product. It characterizes the extent to which the product can

Figure 11: Nonlinearizable 1-handed assembly

be broken down into subassemblies that can be manufactured concurrently.

However, some mass-produced assemblies are more cost-effective to manufacture by bringing one part at a time. This concept can be captured as follows. Let us say that an instruction is *linear* if it is 1-handed and one of its moved sets is a single part. A linear algorithm consists of linear instructions only, hence is a total ordering. A product that admits a linear algorithm is said to be *linearizable*. Many 1-handed products are not linearizable (see one example in Fig. 11) and are in general less amenable to mass-production assembly lines. A refinement of this concept is to measure the minimal number p of nonlinear instructions over all possible 1-handed assembly algorithms; the minimal number of separate assembly lines needed for the product is $p + 1$.

Moved sets need to be grasped or fixtured in a way that parts in the same set cannot move relative to each other under a variety of forces (gravity, acceleration, contact). This is achieved by positioning fingers or fixture elements on their boundary. A grasp of a moved set achieves *form closure* if, when the fingers are locked relative to each other, no part in the moved set can move relative to any other and the fingers [26, 27]. (Another concept, involving friction and forces, is that of force closure; we will not discuss it here, though it is perhaps more practical than form closure.) Given some abstract dimensionless model of a finger, e.g., a point-plane contact, a measure of the complexity of an assembly instruction is the number of fingers required to achieve a form-closure grasp of all its moved sets. Weaker notions can also be considered. For instance, a grasp achieves *prismatic form-closure* if it prevents every part to move in translation relative to any other. A grasp achieves *form closure of order q* if it prevents up to $q + 1$ subassemblies to move relative to each other. Thus, a form-closure grasp of order 0 assumes that no internal motion in the moved set is possible; a form-closure grasp of order 1 assumes that a single internal motion (along any direction) is possible; etc. The higher the degree of form-closure, the safer the grasp. The number of fingers to achieve form-closure grasps for a *single* rigid part (equivalently, form-closure of order 0 for a set of parts) is investigated in several papers, including [34, 36].

There exist other pertinent measures of complexity. For example, uncertainties may also play an important role in assembly instructions, requiring sensors to be used. The number of elementary sensors (e.g., plane probe, diameter sensors) may also be used as a measure of product complexity. However, since we have ignored physical and uncertainty issues in this paper, we will not consider such a measure further.

7.2. EVALUATION OF COMPLEXITY MEASURES

We now show how NDBGs can be used to estimate complexity measures of a product. We mainly focus on the NDBG for infinitesimal translations in 3D, but a similar development can be done with other NDBGs. We evaluate the time complexity for evaluating some complexity measures assuming the NDBG as the input to this computation. However, several measures can be computed while the NDBG is constructed, within the time complexity required by this construction; some can even be more efficiently obtained without computing the NDBGs.

Let us say that a 1-handed assembly algorithm is *correct for infinitesimal translations* if, for every instruction, one moved set is locally free to translate in some direction d in the final arrangement specified by the instruction. The set of assembly algorithms that can be extracted from the NDBG $\Gamma(A)$ of an assembly A for infinitesimal translations includes all correct 1-handed monotonic prismatic assembly algorithms, the importance of which was discussed above. If no assembly algorithm can be extracted from $\Gamma(A)$, it is guaranteed that the assembly is not 1-handed monotonic prismatic.

If a candidate algorithm can be extracted from $\Gamma(A)$, one may wish to know if the assembly is linearizable for translations. Checking the existence of a candidate partitioning of A into S and $A \setminus S$, where S is a singleton, and finding one such partitioning, if one exists, take the same time, $O(rc^2)$, as checking the existence of any candidate partitioning (see Subsection 5.1). In addition, if an assembly is linearizable, all its subassemblies are also linearizable. Hence, checking that a candidate linear algorithm exists takes $O(rc^2n)$ time. If none exists, the product is not linearizable for translations. (This check is a case where using the NDBG is not very pertinent. Indeed, a candidate linear algorithm can be directly computed from the description of the c contacts in only $O(nc \log c)$ time.)

The minimal length of all the candidate algorithms for a product is a lower bound on the minimal length of all correct one-handed monotonic prismatic assembly algorithms. The set of all candidate algorithms can be represented as an AND/OR graph [21] and searched for the shortest one using alpha-beta pruning [38]. But in general, the size of this graph is exponential in the number n of parts. Efficiently computing the minimal length of candidate algorithms, or a bounded approximation of this length, is still an open problem.

The NDBG $\Gamma(S)$ of any subassembly $S \subseteq A$ for infinitesimal translations can also be used to generate a prismatic form-closure grasp of order 1 for S. Let each finger be modeled by a point-plane contact. We scan all the DBGs in $\Gamma(S)$. For every DBG, we consider the reduced graph of its strong components; for every component C with no outgoing arc (i.e., not blocked by any other component), we place a dimensionless finger f in a face F of C whose external normal vector has positive inner product with d and we update $\Gamma(S)$ by adding f to the parts of S, that is, by adding the plane parallel to F to the partition of $\mathcal{S}^2$. At the end, the arrangement in $\mathcal{S}^2$ must be such that, in every DBG, each strongly connected component with no outgoing arc contains one finger. In general, the number of fingers in the generated grasp is not minimal to achieve prismatic form-closure of order 1; it nevertheless gives a good indication of the cost of grasping or fixturing S.

The set of assembly algorithms that one can extract from the NDBG of A for infinitesimal generalized motions includes all correct 1-handed monotonic assembly algorithms. Various complexity measures can be extracted as above.

The set of assembly algorithms that can be extracted from the NDBG of A for infinite translations is the set of all the 1-handed monotonic 1-prismatic assembly algorithms for A. This NDBG allows the computation of various complexity measures for this family of algorithms. In particular, A is a stack product exactly when an algorithm can be extracted from a single DBG. Such a DBG must be an acyclic graph (this is a necessary and sufficient condition), which can be checked in $O(n^2)$ time. Hence, determining whether A is a stack product takes $O(n^2 v^4)$ time.

Notice that no combination of the previous NDBGs allows one to decide in all cases whether a product is admissible.

7.3. EXAMPLE

We have implemented several algorithms to evaluate complexity measures of a product from its NDBG for infinitesimal translations and suggested rotations (see Section 6). We applied these algorithms to a 23-part simplified version of the engine shown in Fig. 9. The results are as follows:

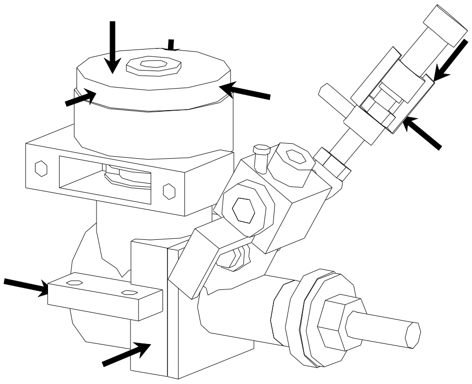

Figure 12: Form-closure grasp of order 1 of the engine

Number of hands and monotonicity. The NDBG allows the generation of candidate algorithms. Hence, the engine admits 1-handed monotonic algorithms that are correct for infinitesimal motions.

Is it prismatic? The engine contains a number of threaded contacts, so it is not prismatic. If these threaded contacts are replaced by cylindrical ones, the existence of candidate algorithms in the modified NDBG suggests that there may exist 1-handed monotonic prismatic algorithms.

Is it linearizable? As far as infinitesimal motions are concerned, the product is linearizable. This computation was performed with the faster method mentioned above, not directly using the NDBG.

Minimal length. The minimum-length algorithm contained in the NDBG has depth 6. This was computed by searching the AND/OR graph representing all 1-handed monotonic assembly algorithms for the engine. The graph has 1027 nodes (each distinct subassembly is represented by a single node) and required 14 minutes to build and search. Better algorithms are needed for products having many more parts.

Is it a stack product? The engine requires infinitesimal translation in several directions to assemble, so it cannot be a stack product.

Number of fingers. Fig. 12 shows a prismatic form-closure grasp of order 1 computed for the simplified engine. The grasp consists of 10 fingers. The contact points of 8 fingers are shown as arrows; two additional fingers are required to hold a loose part inside the engine. It may seem that an additional finger is needed to support the assembly from below. In fact, the finger pointing up and left at the top-right achieves that function.

The implemented algorithms illustrate the kind of information which one may extract from NDBGs. Some of them, however, are based on inefficient brute-force methods that are not practical for complex products. Furthermore, no algorithms have been implemented yet to exploit NDBGs for infinite translation or any other family of extended motions. It is not obvious that polynomial NDBGs will allow all interesting complexity measures to be computed in polynomial time. Very likely, for some measures, we will have to content ourselves with approximate algorithms. Incremental algorithms operating in marginal time while the product is being designed will also be useful.

8. CONCLUSION

This paper presented techniques to reason about mechanical assembly. We investigated the following two problems: (1) the automatic generation of assembly algorithms from CAD data, and (2) the characterization of the complexity of assembly designs. Our main goal was to provide efficient computational support to help designers create products that are easier to manufacture and service. The proposed techniques nevertheless have broader application. They could also be used, for example, to plan the actions of autonomous maintenance robots.

One contribution of this paper consists of the notion of an NDBG and the algorithms to construct it. The construction of an NDBG is based on an analysis of the interferences (blocking relations) among the parts in an assembly. Several variants were presented for different families of motion inducing different blocking relations. Unlike previous methods, the NDBG allows assembly sequences to be computed in time polynomial in the complexity of the parts of the assembly.

Another contribution of the paper is the notion of the algorithmic complexity of an assembly product, the set of complexity measures proposed, and the techniques to estimate them from precomputed NDBGs. Our goal here was to concisely assess the difficulty, hence the cost, of (dis)assembling a product.

Computational complexity has been instrumental to identify the strengths and weaknesses of particular computer algorithms, and classify problems. It has contributed enormously to the progress of computer technology. We think that algorithmic complexity measures for assembly products could have a similar impact on assembly products and processes, bringing improvements that would outpace any advances in the underlying manufacturing technologies themselves. We also expect that it will contribute to the emergence of a completely new type of manufacturing technology where a large number of very simple, mass-produced actuators and sensors are combined in highly modular manufacturing systems. The RISC approach to robotics recently proposed in [8] takes a similar view.

One of our current research goals is to construct polynomial NDBGs for more complex families of motion and investigate the effect of additional constraints (e.g., uncertainty, grasp accessibility) on the construction of NDBGs. Another goal is to investigate further the exploitation of NDBGs and design a more comprehensive set of efficient complexity evaluation methods. On the experimental side, we are currently integrating our algorithms into a highly interactive design-assistant system.

References

[1] —– 1990. Special Issue on Robotic Assembly and Task Planning. *AI Magazine*, AAAI Press, 11(1).

[2] Aho, A.V., Hopcroft, J.E., and Ullman J.D. 1974. The Design and Analysis of Computer Algorithms. Reading, MA: Addison-Wesley.

[3] Aho, A.V., Hopcroft, J.E., and Ullman J.D. 1983. Data Structures and Algorithms. Reading, MA: Addison-Wesley.

[4] Arkin, E.M., Connelly, R., and Mitchell, J.S. 1989. On Monotone Paths among Obstacles with Applications to Planning Assemblies. *ACM Symp. on Computational Geometry*, 334–343.

[5] Baldwin, D.F. 1990. Algorithmic Methods and Software Tools for the Generation of Mechanical Assembly Sequences, Master's thesis, MIT.

[6] Boothroyd, G. 1991. Assembly Automation and Product Design. Marcel Dekker, Inc., New York, NY.

[7] Bourjault, A. 1984. Contribution à une Approche Méthodologique de l'Assemblage Automatisé: Elaboration Automatique des Séquences Opératoires. PhD thesis, Faculté des Sciences et des Techniques de l'Université de Franche-Comté, France.

[8] Canny, J.F. and Goldberg, K.Y. 1992. RISC Robotics. in preparation.

[9] Cutkosky, M.R. and Tenenbaum, J.M. 1990. A Methodology and Computational Framework for Concurrent Product and Process Design. *ASME J. of Mechanism and Machine Theory*, 25(3):365–381.

[10] De Fazio, T.L. and Whitney, D.E. 1987. Simplified Generation of All Mechanical Assembly Sequences. *IEEE J. of Robotics and Automation*, RA-3(6):640–658. Errata in RA-4(6):705-708.

[11] de Kleer, J. and Williams, B.C. (guest editors). 1991. Special Volume, Qualitative Reasoning about Physical Systems II. *Artificial Intelligence*, 51(1–3).

[12] Edelsbrunner, H. and Guibas, L. 1989. Topologically Sweeping an Arrangement. *J. of Computer and System Sciences*, 38:165–194.

[13] Fahlman, S.E. 1974. A Planning System for Robot Construction Tasks. *Artificial Intelligence*, 5(1):1–49.

[14] Faltings, B. 1992. A Symbolic Approach to Qualitative Kinematics. *Artificial Intelligence*, 56(2–3), 139–170.

[15] Fikes, R.E. and Nilsson, N.J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3-4):189–208.

[16] Garey, M.R. and Johnson, D.S. 1979. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. New York, NY:W.H. Freeman and Co.

[17] Goldberg, K.Y. 1990. **Stochastic Plans for Robotic Manipulation**. PhD Thesis, Dept. of Computer Science, Carnegie Mellon University.

[18] Hirukawa, H., Matsui, T., and Takase, K. 1991. A General Algorithm for Derivation and Analysis of Constraint for Motion of Polyhedra in Contact. *IEEE/RSJ Int. Workshop on Intelligent Robots and Systems*, Osaka, 38–43.

[19] Hoffman, R.L. 1991. A Common Sense Approach to Assembly Sequence Planning. In Homem de Mello, L.S. and Lee, S., editors, **Computer-Aided Mechanical Assembly Planning**, Boston, MA: Kluwer Academic Publishers, 289–314.

[20] Homem de Mello, L.S. 1989. **Task Sequence Planning for Robotic Assembly**. PhD Thesis, Carnegie Mellon University.

[21] Homem de Mello, L.S. and Sanderson, A.C. 1991. Representations of Mechanical Assembly Sequences. *IEEE Tr. on Robotics and Automation*, 7(2):211–227.

[22] Homem de Mello, L.S. and Lee, S. (editors) 1991. **Computer-Aided Mechanical Assembly Planning**. Boston, MA: Kluwer Academic Publishers.

[23] Hopcroft, J.E., Schwartz, J.T., and Sharir, M. 1984. On the Complexity of Motion Planning for Multiple Independent Objects: PSPACE-Hardness of the 'Warehouseman's Problem', *The Int. J. of Robotics Research*, 3(4):76–88.

[24] Joskowicz, L. and Sacks, E. 1991. Computational Kinematics. *Artificial Intelligence*, 51(1–3):381–416.

[25] Kriegman, D.J. and Ponce, J. 1990. Computing Exact Aspect Graphs of Curved Objects: Solids of Revolution. *Int. J. of Computer Vision*, 5(2):119–135.

[26] Lakshminarayana, K. 1978. The Mechanics of Form Closure. *ASME Paper*, No. 78-DET-32.

[27] Latombe, J.C. 1991. Robot Motion Planning. Boston, MA: Kluwer Academic Publishers.

[28] Lee, S. and Shin, Y.G. 1990. Assembly Planning Based on Geometric Reasoning, *Computation and Graphics*, 14(2):237–250.

[29] Liebermann, L.I. and Wesley, M.A. 1977. AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly. *IBM J. of Research and Development*, 21(4):321–333.

[30] Liu, Y. 1990. Symmetry Groups in Robotic Assembly Planning. PhD Thesis, COINS Tech. Rep. 90-83, Computer and Information Science Dept., University of Massachusetts, Amherst.

[31] Lozano-Pérez, T. 1976. The Design of a Mechanical Assembly System. Tech. Rep. AI-TR 397, AI Lab., MIT.

[32] Lozano-Pérez, T. 1983. Spatial Planning: A Configuration Space Approach, *IEEE Tr. on Computers*, 32(2):108–120.

[33] Lozano-Pérez, T. and Wilson, R.H. 1992. Assembly Sequencing. In preparation.

[34] Markenscoff, X., Ni, L., and Papadimitriou, C.H. 1990. The Geometry of Grasping. *The Int. J. of Robotics Research*, 9(1):61–74.

[35] McDermott, D.V. 1992. Robot Planning. *AI Magazine*, AAAI Press, 13(2):55–79.

[36] Mishra, B., Schwartz, J.T., and Sharir, M. 1987. On the Existence and Synthesis of Multifinger Positive Grips. *Algorithmica*, 2:541–558.

[37] Natarajan, B.K. 1988. On Planning Assemblies. *ACM Symp. on Computational Geometry*, 299–308.

[38] Nilsson, N.J. 1980. Principles of Artificial Intelligence, Los Altos, CA: Morgan Kaufmann.

[39] Pollack, R., Sharir, M., and Sifrony, S. 1988. Separating Two Simple Polygons by a Sequence of Translations, *Discrete and Computational Geometry*, 3:123–136.

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

[41] Reif, J.H. 1979. Complexity of the Mover's Problem and Generalizations. *20th IEEE Symp. on Foundations of Computer Science*, 421–427.

[42] Sacerdoti, E.D. 1977. A Structure for Plans and Behavior. Amsterdam: Elsevier.

[43] Scheinman, R. 1988. RobotWorld: A Multiple Robot Vision Guided Assembly System. In Bolles, R.C. and Roth, B., editors, Robotics Research, Cambridge, MA: MIT Press, 23–27.

[44] Subramani, A. 1992. Development of a Design for Service Methodology. PhD Thesis, Dept. of Industrial and Manufacturing Engineering, The University of Rhode Island, Kingston, RI.

[45] Taylor, R.H. 1976. Synthesis of Manipulator Control Programs from Task-Level Specifications. PhD Thesis, Dept. of Computer Science, Stanford University.

[46] Toussaint, G.T. 1985. Movable Separability of Sets. In Toussaint, G.T., editor, Computational Geometry. Amsterdam: Elsevier.

[47] Wilson, R.H. and Rit, J.F. 1990. Maintaining Geometric Dependencies in an Assembly Planner. *IEEE Int. Conf. on Robotics and Automation*, Scottsdale, AZ, 890–895.

[48] Wilson, R.H. 1992. On Geometric Assembly Planning. PhD Thesis, Dept. of Computer Science, Stanford University.

[49] Wilson, R.H. and Matsui, T. 1992. Partitioning an Assembly for Infinitesimal Motions in Translation and Rotation. *IEEE Int. Conf. on Intelligent Robots and Systems*, Raleigh, NC, 1311–1318.

[50] Wilson, R.H. and Latombe, J.C. 1992. On the Qualitative Structure of a Mechanical Assembly. *AAAI-92 Nat. Conf. on Artificial Intelligence*, San Jose, CA, 697–702.

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

[52] Wolter, J., Chakrabarty, S., and Tsao, J. 1992. Mating Constraint Languages for Assembly Sequence Planning. *IEEE Int. Conf. on Robotics and Automation*, Nice, France, 2367–2374.