Fabian Schwarzer
Achim Schweikard

Institut für Informatik
Technische Universität München
D-80290 München, Germany

Leo Joskowicz

School of Computer Science and Engineering
The Hebrew University
Jerusalem 91904, Israel

Efficient Linear Unboundedness Testing: Algorithm and Applications to Translational Assembly Planning

Abstract

We address the problem of efficiently testing for linear unboundedness and its applications to translational assembly planning. We describe a new algorithm that performs the test by solving a single homogeneous system of equations followed by a single linear feasibility test. We show that testing for unboundedness is computationally at least as hard as these two subproblems. The new algorithm is the fastest known algorithm and is practical. We then present a framework for general translational assembly planning based on linear constraints. We show the relation of m*-handed assembly planning to unboundedness testing and present a polynomial-time algorithm for* m*-handed assembly of polygonal part assemblies with no initially separated pairs of parts. For the general translational assembly–planning problem, we present a new algorithm that uses unboundedness testing and a cell reduction technique to significantly increase the search efficiency. Experimental results of our implementation on a variety of planar and spatial assemblies demonstrate the practicality of the algorithms.*

KEY WORDS—linear programming, unboundedness testing, translational assembly planning, motion planning

1. Introduction

This paper addresses the problem of efficiently testing for linear unboundedness and its applications to translational assembly planning. Assembly planning consists of finding collision-free sequences of motions that assemble the parts of a product. Because the general problem is known to be computationally hard, finding efficient algorithms for restricted classes of problems is of great theoretical and practical importance in applications such as computer-aided mechanical design, computer-aided manufacturing, and biomedical engineering (Joskowicz and Taylor 1996).

Much research has recently focused on developing algorithms for various classes of assembly-planning problems. The main parameters are (1) the dimensionality of the space, e.g., planar versus spatial; (2) the geometry of the parts, e.g., polygonal, curved, or polyhedral parts; (3) the part topology, e.g., convex versus concave parts; (4) the type of motions required to bring the parts together, e.g., translations, rotations, or coupled motions; (5) the number of steps in each assembly sequence, e.g., a single translation versus a sequence of rotations and translations; and (6) the number of separable subassemblies (or hands required) that must be moved simultaneously during assembly, e.g., one-hand assemblies versus m-hand assemblies. Since assembly motions are equivalent to reversed disassembly motions for rigid parts, many algorithms perform assembly-by-disassembly, which computes a disassembly sequence by partitioning the assembly into subassemblies and recursively generating disassembly sequences for them.

We focus on the problem of translational assembly planning of polygonal and polyhedral parts. Parts can have any topology and can only be assembled by a sequence of part translations, possibly simultaneous. Two examples of this class are shown in Figure 1: a puzzle and a book case. Many puzzle-like problems, and some real-life problems, belong to or can be adequately approximated by this class. Their key property is that contact relations between pairs of parts can be expressed as linear constraints and can be represented as hyperplanes embedded in a higher dimensional space, called the assembly *configuration space* (Schweikard and Schwarzer 1998). The configuration space partitions into an arrangement of cells that can be searched for an assembly path.

Techniques from computational geometry, linear algebra, and linear programming, such as Minkowski sums, linear optimization, and feasibility testing, can be used in assembly-planning algorithms.

We propose a general framework for translational assembly planning based on efficient solution of linear constraints. Within this framework, we present new algorithms for three problems: (1) efficient testing of linear unboundedness, (2) m-handed assembly, and (3) general translational assembly planning.

Linear unboundedness testing determines if the intersection of a set of half-spaces is infinite. When the set is infinite, it contains a ray in an unbounded direction, which represents a single simultaneous collision-free disassembly translation for all parts. Efficient testing of configuration space cells for unboundedness is key for translational motion planning. We present a new algorithm that tests for unboundedness by first solving a single homogeneous system of equations and then performing a single linear feasibility test. The new algorithm is the fastest known algorithm for unboundedness testing and is practical. The algorithm is general and can be used in applications other than motion planning.

M-handed assembly consists of deciding if an assembly can be partitioned into at least two subassemblies by a single simultaneous translational motion. Each part may assume a different direction and velocity of motion but may be moved at most once. No polynomial-time algorithms are known for this problem. We have developed the first polynomial-time algorithm for m-handed assembly under the condition that no pair of parts is separated in the initial placement (the puzzle in Fig. 1a is an example of such an assembly).

General translational assembly planning specifies a sequence of coordinated part motions when changing directions is necessary. Parts may have to be brought to intermediate placements to clear the way for other parts (Fig. 1b is an example of such an assembly). The problem has been shown to be *NP*-hard. We present the first method for exact computation of assembly configuration spaces that has been successfully applied to practical examples. The output-sensitive algorithm derives linear motion constraints for all pairs of parts and incrementally forms configuration space cells by embedding and intersecting the pairwise constraint sets. We develop new techniques for reducing redundancies, which in some practical cases yield exponential speedup.

Experimental results show that our algorithms are practical for both m-handed assembly and general translational assembly planning.

The rest of this paper is organized as follows. Section 2 briefly surveys previous work. Section 3 presents the new algorithm for unboundedness testing and shows that this problem is computationally at least as difficult as solving a single homogeneous system of equations and then performing a single linear feasibility test. Section 4 presents a linear constraint framework and new algorithms for translational assembly planning. First, we show that m-handed assembly planning can be based on testing for unboundedness. We show that for assemblies of polygons in which no pair of parts is separated in their initial placement, a disassembly motion can be computed in polynomial time by solving a single linear unboundedness problem. For assemblies in which all pairs of parts are initially separated, we present a simple linear time algorithm to compute the disassembly motions. For the case with separated and nonseparated pairs, we describe effective heuristics. Second, we present an algorithm based on our previous work (Schweikard and Schwarzer 1998) for general translational assembly planning. The algorithm uses a reduction technique that significantly increases the search efficiency and relies on fast unboundedness testing to search many cells. Section 5 presents experimental results of our implementations on planar and spatial examples, demonstrating the practicality of our algorithms. Section 6 concludes with open questions and directions for future work.

2. Previous Work

We briefly survey previous work on translational assembly planning. For general assembly-planning surveys, see Halperin, Latombe, and Wilson (1998); Homem de Mello and Lee (1991); and Toussaint (1985).

Chazelle et al. (1984) prove that multistep assembly planning is *NP*-hard for polygons by a reduction of the *PARTITION* problem for integers. They provide an exponential lower bound using a construction that simulates the well-known "Towers of Hanoi" problem, for which the minimum number of moves required to remove a part is exponential in the number of parts. The results hold for very simple parts, since in both reductions, each polygon has a constant number of vertices and all segments are either vertical or horizontal.

Other authors have developed polynomial time algorithms by restricting the class of allowed motions. Agarwal et al. (1996) describe an efficient algorithm for a constant set of given disassembly directions. Schweikard and Wilson (1995) show that the single-handed assembly problem, which consists of finding all translational directions that partition a given assembly into two subsets, can be solved in polynomial time. Snoeyink and Stolfi (1994) show that assemblies of convex polyhedral parts may require more than two hands to disassemble.

Existing translational assembly-planning methods are limited in the type of motions they can handle (Schweikard and Wilson 1995), have not been proven to be practical (Halperin and Wilson 1997; Halperin, Latombe, and Wilson 1998), or rely on heuristics based on special part features (Hoffmann 1991).

Linear constraints and linear programming techniques have been proposed for stability analysis (Wolter and Trinkle 1994; Baraff, Mattikalli, and Khosla 1997): given an assembly of parts, determine from the part placements and

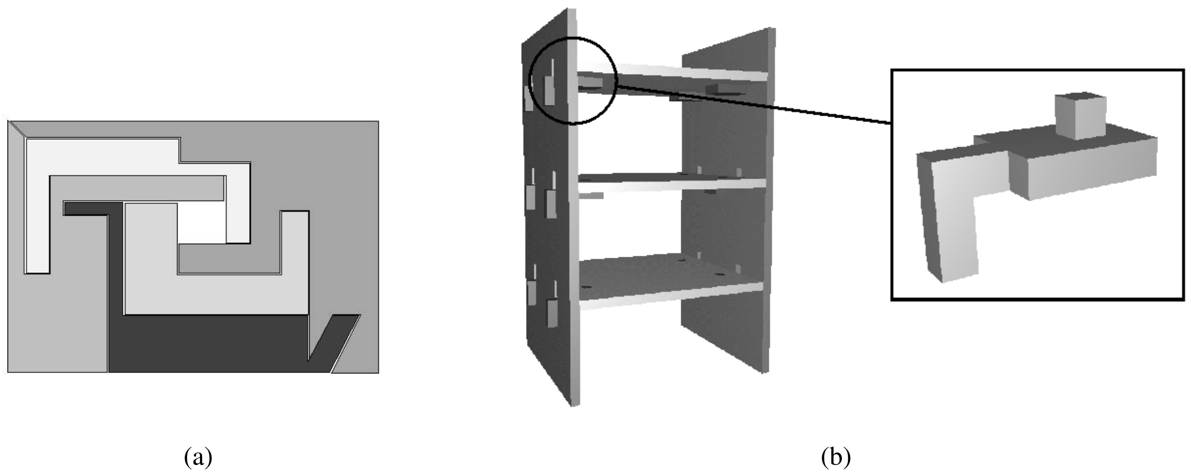

Fig. 1. Two examples of assemblies: (a) a planar puzzle, which requires four hands to simultaneously move parts in different directions to disassemble it, and (b) a spatial book case, which requires several sequential motion direction changes to remove the shelves.

contacts if it is stable under gravity and other forces. Although the techniques for stability analysis bear some resemblance to our work, there are several important differences. Stability constraints are derived from local part contacts, whereas for the m-handed assembly problem we may have to take constraints for all pairs of parts into account since we consider extended translations. In stability analysis for polygonal parts, nonconvex constraints arise only when two vertices come into contact. The problem can be solved by introducing a round vertex model (Wolter and Trinkle 1994). In our case, nonconvex constraints cannot be discarded in general by modifying the part models. Using mixed-integer programming techniques, as proposed in Wolter and Trinkle (1994), has a much higher computational cost. The algorithm for solving the system of constraints in Baraff, Mattikalli, and Khosla (1997) requires all variables to be strictly positive. This is overly restrictive for m-handed assembly since some variables may have to be negative or zero to model parts at rest.

We present the first method for exact computation of assembly configuration spaces that has been successfully applied to practical examples. Despite the theoretical complexity of the general problem, most practical cases are much simpler and can be solved with our output-sensitive approach. Parts can be general (nonconvex) polygons or polyhedra. We use the fact that for pairs of polygons or polyhedra, linear constraints that describe valid relative placements can be directly derived. Techniques for computation of configuration spaces of pairs of parts are well-known (Latombe 1991) and will not be discussed here.

Within the assembly configuration space framework, direct computation of hyperplane arrangements is impractical due to their number and high dimension. For assemblies with more than two parts, we therefore derive the set of valid placements incrementally by embedding and intersecting the sets of valid relative placements of all pairs in the composite configuration space of all parts.

3. Linear Unboundedness Testing

In this section, we first define formally the linear unboundedness testing problem and describe a simple algorithm for solving it based on $2d$ linear programming problems in d dimensions. We then present a new algorithm that answers the query by solving a single system of homogeneous linear equations followed by a single linear feasibility test. By reducing these two subproblems to unboundedness testing, we show that testing for unboundedness is computationally at least as hard as the two dominating steps of our new algorithm. Using the best known algorithms for solving homogeneous equations and linear feasibility testing, the new algorithm is faster than the simple algorithm by a factor of d.

3.1. Problem Statement

Consider a set of half-spaces in d dimensions, defined by the inequalities

$$\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1d}x_d \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2d}x_d \geq b_2 \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nd}x_d \geq b_n, \end{array}$$

or, using matrix notation,

$$\mathbf{Ax} \geq \mathbf{b}.$$

Each of the half-spaces is bounded by a hyperplane of the form $\mathbf{a}_i\mathbf{x} = b_i$, where $\mathbf{x} = (x_1, \dots, x_d)$, and $\mathbf{a}_i = (a_{i1}, \dots, a_{id})$

are the hyperplane normal vectors. We assume the intersection of the half-spaces is nonempty.

The intersection of these half-spaces defines a *convex cell* in d-dimensional space whose dimension can be lower than d. A cell is said to be unbounded if it entirely contains a ray.

The problem of testing for linear unboundedness is stated as follows:

> Given half-spaces in d dimensions with nonempty intersection, decide whether there is a ray $r = \{\mathbf{p} + t\mathbf{u} \mid t \geq 0\}$ that is contained in all half-spaces. If so, return a valid direction vector $\mathbf{u} \neq 0$.

Since the cell is convex, the choice of the direction vector $\mathbf{u}$ is independent of $\mathbf{p}$. Thus, any combination of a valid direction vector and a valid point will yield a ray that is contained in the cell. Note that the point $\mathbf{p}$ can be determined by a linear feasibility test and that we do not require, as is customary, that the coordinates of $\mathbf{u}$ be nonnegative (otherwise, the problem can be directly solved with a single standard linear maximization).

3.2. A Simple Unboundedness Test

There are two difficulties in solving the unboundedness testing problem with standard tools for linear programming. First, although it can be formulated as a linear optimization problem, the actual objective function, i.e., the direction in which the maximum is to be found, is not known in advance. Second, even if a direction is known, unboundedness is usually regarded as an error condition in linear programming: the maximization will fail if the given objective function has no upper bound inside the cell.

We observe that any base of the d-dimensional space provides a sufficient set of directions to test. Let $\mathbf{s}_1, \ldots, \mathbf{s}_d$ be a linear base of $d-$space. We can perform a maximization for each of $+\mathbf{s}_i$ and $-\mathbf{s}_i$, subject to the half-space inequalities defining the given cell. If one of these maximizations fails, the cell is unbounded. Notice, however, that a direction $\pm\mathbf{s}_i$ that causes the maximization to fail is not necessarily a valid direction vector for a ray inside the given cell. For example, consider the half-plane defined by the constraint $y - x \geq 1$ in E^2. A maximization along $\mathbf{e}_1 = (1, 0)$ will report unboundedness, since x can become arbitrarily large, but no ray with direction vector $\mathbf{e}_1$ is contained in the half-plane.

To compute a valid direction vector, we first transform the cell S by shifting the bounding hyperplanes to the origin, that is, we replace the defining half-spaces $\mathbf{a}_i\mathbf{x} \geq b_i$ by the corresponding homogeneous half-spaces $\mathbf{a}_i\mathbf{x} \geq 0$. We denote the transformed cell by S_0, since it always contains the origin. The following lemma and corollary show that this transformation does not affect the set of unbounded directions: S is unbounded in a direction $\mathbf{u}$ if and only if S_0 is unbounded in direction $\mathbf{u}$.

LEMMA 1. S_0 contains a point $\mathbf{u} \neq 0$ if and only if S is unbounded in direction $\mathbf{u}$.

Proof. Let $\mathbf{u} \neq 0$ be a vector in S_0 ($\mathbf{Au} \geq 0$) and $\mathbf{p}$ be a point in S ($\mathbf{Ap} \geq \mathbf{b}$). Thus, for $t \geq 0$, $\mathbf{A}(\mathbf{p} + t\mathbf{u}) \geq \mathbf{b}$ and S is unbounded in direction $\mathbf{u}$. Now assume that S contains a ray $\{\mathbf{p} + t\mathbf{u} \mid t \geq 0\}$. Thus, $\mathbf{A}(\mathbf{p} + t\mathbf{u}) \geq \mathbf{b}$ and, equivalently, $t(\mathbf{Au}) \geq \mathbf{b} - \mathbf{Ap}$ for any $t \geq 0$. This implies $\mathbf{Au} \geq 0$. Thus, $\mathbf{u}$ is in S_0 and $\mathbf{u} \neq 0$ by definition of the ray. □

COROLLARY 1. S_0 contains a point $\mathbf{u} \neq 0$ if and only if S_0 is unbounded in direction $\mathbf{u}$.

Now, testing for unboundedness can be reduced to finding a point $\mathbf{u} \neq 0$ within the transformed cell. If successful, the location vector of this point specifies a direction in which the original cell is unbounded. We can determine $\mathbf{u}$ by, e.g., intersecting the transformed cell S_0 with the bounding hyperplanes of a d-dimensional box centered at the origin. Instead of maximizing in the directions $+\mathbf{e}_i$ and $-\mathbf{e}_i$ for each unit vector $\mathbf{e}_i$, we perform feasibility tests on the constraint sets $\{\mathbf{Ax} \geq 0, \mathbf{e}_i\mathbf{x} = 1\}$ and $\{\mathbf{Ax} \geq 0, \mathbf{e}_i\mathbf{x} = -1\}$. Any solution is then a nontrivial point (not the origin) in S_0 and thus a valid direction vector $\mathbf{u}$.

This simple algorithm requires $2d$ linear feasibility tests in the worst case to decide if the set is unbounded. Is this the most efficient way of testing? Is it optimal? We address these questions next.

3.3. The New Unboundedness Testing Algorithm

Table 1 presents the new algorithm. We begin with an informal description and then formally prove its correctness.

Lemma 1 shows that a cell is unbounded if and only if it contains a point other than the origin after the transformation. We can therefore work directly in the homogeneous cell S_0 and test if it contains a point other than the origin. First, we test if the corresponding homogeneous system of equations is underconstrained (step 1). If it is, the cell is unbounded since the solution space is at least of dimension one, and we can compute nontrivial solutions directly. If the homogeneous system of equations is *not* underconstrained, we construct a combined vector as the sum of the hyperplane normal vectors (step 2). The hyperplane defined by this vector at a positive distance from the origin will intersect any ray emanating from the origin in the cell due to the full rank of the matrix. If the combined vector is zero, no such plane exists and thus the cell is bounded. Otherwise, we construct a new set of constraints consisting of the homogeneous inequalities and the new hyperplane (step 3). Now unboundedness can be detected with a single linear feasibility test on the new constraint set.

This algorithm reduces the number of hyperplanes to be tested from $2d$ hyperplanes (the previous simple test) to a single hyperplane with combined normal $\mathbf{a}_\Sigma$. This reduction is only possible after we have established the range of the

Table 1. New Linear Unboundedness Testing Algorithm

Input: Half-spaces $H_1^+, \dots, H_n^+$ defined by inequalities $H_i^+ : \mathbf{a}_i\mathbf{x} \geq b_i$.

Output: Unbounded direction $\mathbf{u}$ when the cell defined by $H_1^+ \cap \dots \cap H_n^+$ is *unbounded* or *bounded* otherwise.

1. If the homogeneous system of equations $\mathbf{Ax} = 0$ has a nontrivial solution $\mathbf{u} \neq 0$, return *unbounded*, the vector $\mathbf{u}$, and exit.
2. Construct a combined direction $\mathbf{a}_\Sigma := \sum_{i=1}^n \mathbf{a}_i$. If $\mathbf{a}_\Sigma = 0$, return *bounded* and exit.
3. Construct the constraint set C as

   $\mathbf{ax} \geq 0, \quad 1 \leq i \leq n$
   $\mathbf{a}_\Sigma\mathbf{x} = 1.$

   If C is feasible, return *unbounded*. The solution of the feasibility test is the direction $\mathbf{u}$.
   Else return *bounded* and exit.

underlying constraint set, which means that step 1 is not redundant, as illustrated with the following example.

EXAMPLE. Let $\mathbf{a}_1 = (-1, 0), \mathbf{a}_2 = (2, 0), b_1 = -2, b_2 = 2$. After transformation, the cell containing the origin, i.e., the cell defined by $\mathbf{a}_1\mathbf{x} \geq 0$ and $\mathbf{a}_2\mathbf{x} \geq 0$, is unbounded, but $\mathbf{a}_\Sigma = (1, 0)$, so that C is infeasible.

We now prove the correctness of the algorithm.

Proof. The correctness of step 1 follows from Lemma 1 and the fact that the set of solutions of $\mathbf{Ax} = 0$ is a subset of S_0. Notice that step 1 fails if and only if $dim\langle \mathbf{a}_1, \dots, \mathbf{a}_n \rangle = d$. □

To prove steps 2 and 3, we state Lemma 2.

LEMMA 2. (variant of Farkas' lemma). Let $dim\langle \mathbf{a}_1, \dots, \mathbf{a}_n \rangle = d$. Then S_0 contains a point $\mathbf{u} \neq \mathbf{0}$ if and only if C is feasible.

Proof. Let $\mathbf{u} \neq \mathbf{0}$ be a point in S_0. We must show that C is feasible. Assume $\mathbf{a}_\Sigma\mathbf{u} = 0$. Since $\mathbf{u}$ is in S_0 ($\mathbf{a}_i\mathbf{u} \geq 0$), we have $\mathbf{a}_i\mathbf{u} = 0$ for each $i \leq n$, so $\mathbf{u}$ is orthogonal to each $\mathbf{a}_i$. But this would imply that $dim\langle \mathbf{a}_1, \dots, \mathbf{a}_n, \mathbf{u} \rangle$ is strictly greater than d. Thus $\mathbf{a}_\Sigma\mathbf{u} \neq 0$. The same argument shows that $\mathbf{a}_\Sigma \neq \mathbf{0}$ if the space spanned by $\mathbf{a}_1, \dots, \mathbf{a}_n$ has full dimension and S_0 contains a point $\mathbf{u} \neq 0$. For each scalar $t \geq 0$, the point $t\mathbf{u}$ is in S_0. Thus after appropriate scaling, we can assume $\mathbf{a}_\Sigma\mathbf{u} = 1$, so that $\mathbf{u}$ satisfies C. Conversely, a point satisfying C is clearly a non-zero point in S_0. □

3.4. Complexity Analysis

Is this test the fastest possible test? Our algorithm relies on finding a nontrivial solution for a homogeneous system of linear equations and linear feasibility testing. Is there a test that does not require one or both of these two steps? The following constructions show that the problem of testing for unboundedness and finding an unbounded direction is at least as hard as these two steps. Based on these reductions, the new algorithm is optimal when optimal algorithms for solving the homogeneous equations and for feasibility testing are used. We show that with known algorithms for these two problems, the new unboundedness testing algorithm is faster by a factor of d over the simple approach in Section 3.2.

LEMMA 3. Testing for linear unboundedness is at least as hard as finding a nontrivial solution for a homogeneous system of linear equations.

Proof. We reduce the problem of finding a nontrivial solution of a homogeneous linear system to unboundedness testing. Given a homogeneous system (E) of n linear equations $\mathbf{Ax} = 0$, we construct an equivalent homogeneous system (U) of $2n$ linear inequalities consisting of $\mathbf{Ax} \geq 0$ and $\mathbf{Ax} \leq 0$. We determine a nontrivial solution $\mathbf{u} \neq 0$ for (E) by finding an unbounded direction for (U). From Corollary 1, (E) has a trivial solution if and only if (U) is bounded. Even though (U) has twice as many rows as (E), this increase does not affect the asymptotic computing time (assuming that testing for unboundedness is polynomial in n). Also, the described transformation itself is dominated by the unboundedness test and thus does not increase the complexity. □

LEMMA 4. Testing for linear unboundedness is at least as hard as linear feasibility testing for the constraint set C.

Proof. We give a simple reduction of feasibility testing for C to unboundedness testing. Consider the linear feasibility problem (C)

$$\begin{aligned} \mathbf{Ax} &\geq 0 \\ \mathbf{a}_\Sigma\mathbf{x} &= 1, \end{aligned}$$

with $dim\langle \mathbf{a}_1, \dots, \mathbf{a}_n \rangle = d$, i.e., the rank of the matrix $\mathbf{A}$ is full. We can transform this problem to a (homogeneous) test for unboundedness by setting (U) to

$$\begin{aligned} \mathbf{Ax} &\geq 0 \\ \mathbf{a}_\Sigma \mathbf{x} - z &= 0 \\ z &\geq 0. \end{aligned}$$

We must show that (C) is feasible if and only if the cell defined by the system (U) is unbounded. First, assume that (C) is feasible, i.e., $\mathbf{Ax}_0 \geq 0$ and $\mathbf{a}_\Sigma \mathbf{x}_0 = 1$ for some $\mathbf{x}_0 \in E^d$. Thus, $z(\mathbf{Ax}_0) \geq 0$ and $z(\mathbf{a}_\Sigma \mathbf{x}_0 - 1) = 0$ for any $z \geq 0$ and (U) is unbounded in direction $(\mathbf{x}_0, 1)$. Now assume that (U) is unbounded, i.e., there is a vector $\mathbf{u} = (\mathbf{x}_0, z) \neq 0$ in (U). We first show that $z > 0$. Assume that $z = 0$ (and thus $\mathbf{x}_0 \neq 0$). Since $\mathbf{u}$ is in (U), we would then have $\mathbf{a}_\Sigma \mathbf{x}_0 = 0$ and $\mathbf{Ax}_0 \geq 0$. But this implies that $\mathbf{a}_i \mathbf{x}_0 = 0$ for all rows of $\mathbf{A}$ and we have a nonzero vector $\mathbf{x}_0$ that is orthogonal to all $\mathbf{a}_i$, in contradiction to $dim\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle = d$. Thus, after appropriate scaling of $\mathbf{u}$, we can assume that $z = 1$, so that $\mathbf{u}' = (\mathbf{x}_0', 1)$ is in (U). Thus, $\mathbf{x}_0'$ is in (C).

The transformation from (C) to (U) has increased the dimension d and the number n of constraints of the problem by one, since a new variable z has been added. Furthermore, the transformation increases the number of constraints by one. Assuming the number of steps required for a linear feasibility test (with n constraints in d dimensions) is polynomial in n and in d, this increase does not affect the asymptotic computing time. □

The new algorithm does not specify how solving the homogeneous equations and linear feasibility testing are to be performed. Based on known algorithms for these two subproblems, we obtain the following time bound for our test.

LEMMA 5. The complexity of the algorithm in Table 1 is $O(nd^{4.5})$ where n is the number of constraints and d is the number of variables.

Proof. Based on Karmarkar's (1984) method, linear maximization and feasibility testing take at most $O(Ld^{3.5})$ steps, where L is the accumulated length of the input coefficients (Wright 1992). If the length of each individual coefficient is fixed and constant, i.e., the number of bits necessary to represent each number is at most l, then $L = O(nd)$, so that the described feasibility test takes at most $O(nd^{4.5})$ steps. Step 1 can be performed in $O(nd^2)$ steps (for n constraints and d variables, [Stoer 1976]). Thus, the complexity of the algorithm in Table 1 is $O(nd^{4.5})$ versus $O(nd^{5.5})$ for the simple method in Section 3.2. Thus, our algorithm allows for a reduction of time bounds by a factor of d when compared to the simple method. □

4. A Linear Constraint Framework for Translational Assembly Planning

This section introduces the linear constraint framework for translational assembly planning and presents algorithms for m-handed assembly and general translational assembly.

4.1. Basic Concepts

For simplicity, we describe the basic concepts for the case of polygonal parts. The formulation extends directly to polyhedra.

Let $P_1, \ldots, P_k$ be polygons in a given initial placement, which is, by definition, overlap-free. Since we consider only translational motions, the position of each movable part P_i is given by a vector $\mathbf{p}_i = (x_i, y_i)$. We assume $\mathbf{p}_i = (0, 0)$ for the initial placement, so $\mathbf{p}_i \neq (0, 0)$ describes the location of part P_i after a translational displacement. A *simultaneous placement* of all parts is represented by a vector $\mathbf{u} = (x_1, y_1, \ldots, x_k, y_k)$ in E^{2k}. The origin in this space describes the initial placement of all parts. For example, the vector $(1, 0, \ldots, 0)$ is the placement, possibly overlapping, obtained by translating the first part along the positive x-axis by one length unit.

The configuration space obstacle $CO(P_i, P_j)$ of two parts describes the set of relative placements of P_i with respect to P_j for which P_i and P_j will overlap (Latombe 1991). We regard all parts as open sets. The complement of $interior(CO(P_i, P_j))$ represents relative placements of polygons P_i and P_j in which their interiors do not intersect. This complement is a union of closed 0-, 1-, and 2-dimensional convex cells. Each cell in this union can be represented by an intersection of half-planes $H_1 \cap \ldots \cap H_s$. A single half-plane constraint has the form

$$\mathbf{ax} \geq b.$$

Here, $\mathbf{x} = \mathbf{p}_i - \mathbf{p}_j$ denotes the coordinates of P_i relative to P_j. Thus, each half-plane constraint induces a four-dimensional constraint on the coordinates of P_i and P_j

$$\mathbf{a}(\mathbf{p}_i - \mathbf{p}_j) \geq b.$$

Note that since $CO(P_i, P_j) = -CO(P_j, P_i)$, these four-dimensional constraints do not depend on the order of the parts. It is thus sufficient to consider only pairs (P_i, P_j) where $i < j$.

Once we have derived cells for all pairs of parts, we embed them in the compound assembly configuration space by regarding their constraints as d-dimensional constraints with only four nonzero coefficients.

4.2. M*-Handed Assembly Planning*

The m-handed assembly problem is stated as follows:

> Given an initial, overlap-free placement of parts, decide whether at least one part can be removed without collisions by a single simultaneous ("one-shot") translation. Each moving part may translate in a different direction with its own constant velocity.

We first introduce the concept of separated pairs of parts. Two sets are said to be *separated* if there exists a hyperplane $\mathbf{ax} = b$ such that one set is entirely contained in the half-space $\mathbf{ax} \geq b$ while the other set is entirely contained in the (open) half-space $\mathbf{ax} < b$. For example, the polygons in Figure 2a are not separated, while those in Figure 2b are.

We distinguish among three types of assemblies according to the initial part placement: (1) assemblies in which no pairs of parts are separated, (2) assemblies in which all pairs of parts are separated, and (3) assemblies with separated and nonseparated pairs. We provide polynomial time algorithms for the first two cases and heuristics for the third. For planar assemblies in which no pair of parts is separated in the initial placement, a motion can be computed in polynomial time with a single unboundedness test. No polynomial-time algorithms were previously known for this problem. When all pairs of parts are initially separated, we show that collision-free outward motions can always be computed for both planar and spatial assemblies. In the general case with both separated and nonseparated pairs of polygons, an exponential number of unboundedness tests may be necessary. It is currently an open question whether the general m-handed assembly problem is *NP*-hard.

4.2.1. Constraints and Unboundedness Testing

We first derive linear constraints for pairs of polygons. For each pair of parts (P_i, P_j) in the initial placement, where $1 \leq i < j \leq k$, we consider the set $separate(P_i, P_j)$ of one-shot translations separating P_i from P_j without collisions when P_j is fixed (motions of P_i relative to P_j). This set of translations is represented by a cone of rays emanating from the origin (in the plane). Any ray that intersects $interior(CO(P_i, P_j))$ represents an invalid translation of P_i during which the interiors of the parts will intersect.

All pairs of parts initially nonseparated. We first consider the special case in which no pair of polygons is separated in the initial placement. Figure 1a and Figure 3 show examples in which no pair of parts is initially separated.

Under this restriction, each of the sets $separate(P_i, P_j)$ forms a convex sector (or half-cone) in the plane, which is the intersection of two half-planes (Fig. 2c). A conjunction of two linear constraints describes this convex sector:

$$\begin{aligned} \mathbf{a}_1^{(ij)}\mathbf{x} &\geq 0 \\ \mathbf{a}_2^{(ij)}\mathbf{x} &\geq 0, \end{aligned}$$

where $\mathbf{x}$ denotes the coordinates of P_i relative to P_j. The substitution $\mathbf{x} = \mathbf{p}_i - \mathbf{p}_j$ yields four-dimensional linear constraints on the absolute part positions $\mathbf{p}_i$ and $\mathbf{p}_j$:

$$\begin{aligned} C_1^{(ij)} : \mathbf{a}_1^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) &\geq 0 \\ C_2^{(ij)} : \mathbf{a}_2^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) &\geq 0. \end{aligned}$$

Each ray in this set is a direction of simultaneous translation for both P_i and P_j, during which P_i and P_j will not collide.

For the entire assembly, we embed the sets of separating directions from all pairs of parts in the compound assembly configuration space by considering the four-dimensional inequalities $C_1^{(ij)}$ and $C_2^{(ij)}$ as d-dimensional inequalities. These inequalities describe a single convex cell of separating directions for the whole assembly.

Without loss of generality, we fix the placement of one part (the basis) and compute assembly motions relative to it. This is possible since if the given parts can be separated at all, then there is also a separating motion that will leave one of the parts fixed. If the resulting cell is bounded, then the parts interlock. Otherwise, at least one part can be removed. Since the cell is convex, a single unboundedness test is sufficient and thus the running time is polynomial. Here, the first step in our new algorithm (transformation of the cell) is not necessary, since all constants on the right sides of the inequalities are already equal to zero.

Notice that at least one part must be fixed. Otherwise, the resulting cell is always unbounded, at least in all directions $(v_x, v_y, v_x, v_y, \ldots, v_x, v_y)$, since the assembly can always be moved as a whole.

Note also that there is no need to explicitly compute the entire assembly configuration space nor any configuration space obstacles for pairs of polygons. Instead, it is sufficient to compute and test the cell defined by two tangent constraints per pair of parts. Thus, even curved planar parts can be taken into consideration as long as tangent rays can be computed for all pairs.

All pairs of parts initially separated. As shown in Dawson (1984), any collection of k star-shaped objects with disjoint interiors can be separated with $k - 1$ hands using a simultaneous one-shot translation. It follows that if all pairs of parts are initially separated, there is always a valid m-handed removal motion. Note that this is true for both planar and spatial assemblies. In our case, parts need not be star shaped, since all pairs of parts are initially separated.

General case: initially separated and nonseparated pairs of parts. Let $\mathcal{N}$ be the set of initially nonseparated pairs and $\mathcal{S}$ the set of initially separated pairs. For nonseparated pairs, the set of separating directions is convex and can be represented by a conjunction of two half-plane constraints $C_1^{(ij)} \wedge C_2^{(ij)}$. For an initially separated pair (Fig. 2b), the corresponding set is nonconvex (Fig. 2d) and must be split into a disjunction of two half-plane constraints $C_1^{(kl)} \vee C_2^{(kl)}$.

The set of all embedded constraints can then be expressed by the conjunction

$$C_0 \quad \wedge \bigwedge_{(i,j)\in\mathcal{N}} (C_1^{(ij)} \wedge C_2^{(ij)}) \quad \wedge \bigwedge_{(k,l)\in\mathcal{S}} (C_1^{(kl)} \vee C_2^{(kl)}), \qquad (1)$$

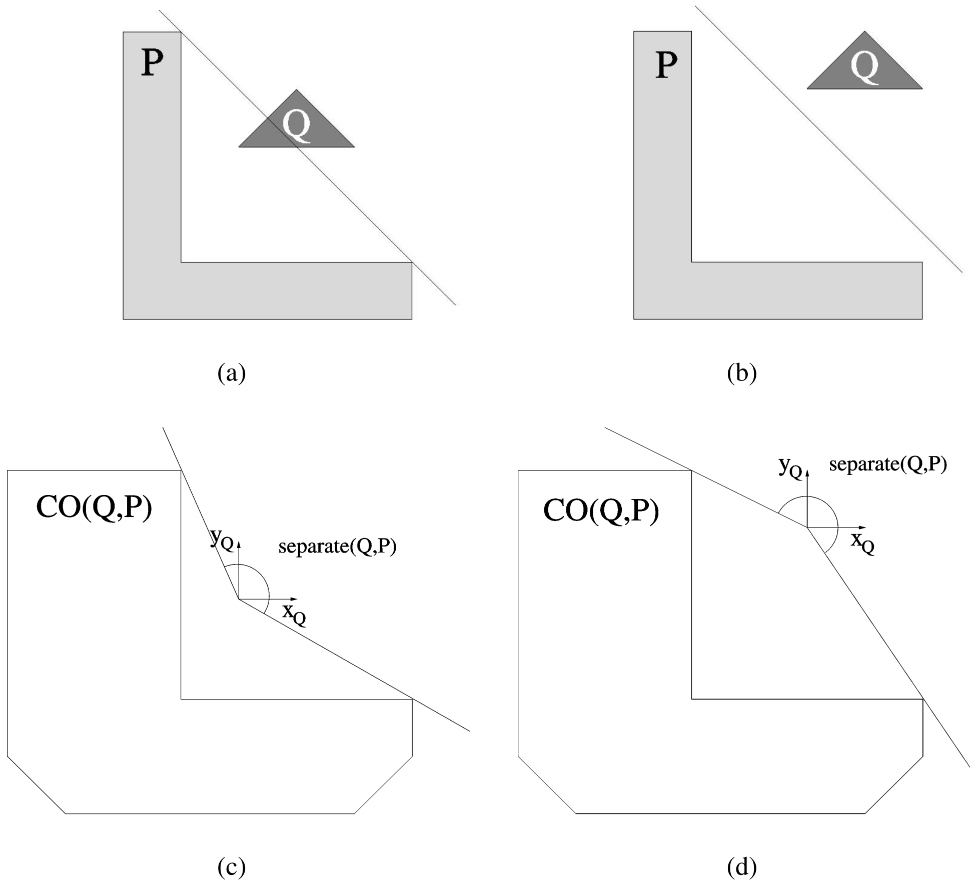

Fig. 2. (a) Nonseparated and (b) separated parts and their respective configuration spaces (c,d).

where C_0 denotes constraints that fix an arbitrary part. To apply the unboundedness test, we transform this expression into disjunctive normal form. Each conjunctive clause in the normal form expression represents a d-dimensional convex cell and can thus be tested for unboundedness. We stop once an unbounded cell is found. In the worst case, 2^s cells must be examined, where $s = |\mathcal{S}|$ is the number of initially separated pairs of parts.

When the number of initially separated pairs is not constant, the above procedure is exponential in the worst case. It is an open question whether the general m-handed assembly problem is *NP*-hard.

We can heuristically reduce the number of nonconvex constraints in (1) by analyzing the influence of the constraints from all initially nonseparated pairs of parts. These constraints define a single convex cell $\mathbf{Ax} \geq 0$, which can be projected into the configuration spaces of all pairs of parts (Huyn et al. 1991). Each such projection is a single two-dimensional cone. We can then use this projection to determine, in each pairwise configuration space, how the allowed motion directions for the considered pair are further restricted by other parts. Specifically, separating directions for initially separated pairs may be restricted to a convex set. We provide examples for this heuristics in Section 5.

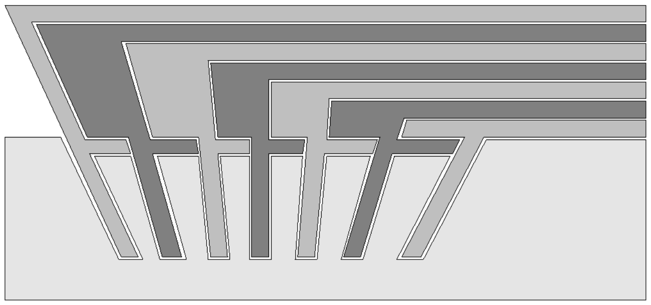

Fig. 3. Assembly with no separated pair of parts.

4.2.2. *Polyhedral Assemblies*

The linear constraint framework can be extended directly to the spatial case by introducing a z-coordinate for each part. However, the actual construction of the sets of directions separating each pair of parts, *separate*(P_i, P_j), is more difficult than in the polygonal case. In Schweikard and Wilson (1995), pairwise separating directions are computed using a central projection from the origin onto the unit sphere or two parallel planes ($z = -1$, $z = 1$). Our constraints can then be derived from a planar arrangement since all bounding planes of the cones of directions must contain the origin.

In the polygonal case, we were able to develop efficient algorithms for assemblies with no initially separated pairs of parts because the pairwise constraints for nonseparated pairs were convex. However, this does not hold for spatial pairs: the set of separating directions can partition into several convex sets even for nonseparated polyhedra (e.g., a part with several connected cavities that contain another part).

Another interesting difference between polygonal and polyhedral assemblies is that, unlike assemblies of convex polygons, assemblies of convex polyhedra cannot always be assembled with a single hand (Snoeyink and Stolfi 1994). Therefore, for polyhedral assemblies, it is not sufficient that all pairs of parts be separated to allow for single-handed assembly. In contrast, as shown above, m-handed assembly is always possible for pairwise separated parts in any dimension.

4.3. General Translational Assembly Planning

The general translational assembly-planning problem is stated as follows:

> Given an assembly of polygonal or polyhedral parts, decide whether one or more parts can be removed without collisions from the remaining set of parts by an arbitrary sequence of translations.

General translational assembly planning includes nonmonotone assembly sequences, in which one or more subassemblies have to be brought in intermediate placements. For example, the wooden cube puzzle in Figure 4 has 12 pieces. To disassemble it, several parts must be shifted back and forth to remove other parts. Known algorithms cannot handle such multistep translations in a uniform and complete way.

We now show how our linear constraints framework can be used in an algorithm for general translational assembly planning. The algorithm incrementally computes a d-dimensional arrangement of hyperplanes representing simultaneous placements of all parts. Valid and forbidden placements correspond to cells in the d-arrangement. A sequence of valid cells connecting the initial cell to an unbounded cell directly yields a feasible disassembly plan. The algorithm is opportunistic because it computes only certain required parts of the reachable free configuration space regions. We first summarize the basic concepts from Schweikard and Schwarzer (1998), and then present a new method, called *D–node reduction*, that can significantly speed up the search by reducing redundancies in the arrangement. We use our new unboundedness test to efficiently test cells during the search.

4.3.1. Constraints for Pairs of Parts

The configuration space obstacle of two translating polyhedral parts in space, P_i and P_j, is also a polyhedron (Latombe 1991). The bounding planes of $CO(P_i, P_j)$ define an arrangement of planes $A(P_i, P_j)$ in three-dimensional space. $A(P_i, P_j)$ decomposes the complement of *interior*$(CO(P_i, P_j))$ into closed convex cells, as illustrated in Figure 5a in two dimensions. We can embed and intersect these free cells from all pairs of parts in the composite assembly configuration space, yielding a d-dimensional arrangement A^d as described above. Computing this arrangement yields cell fragmentation and is impractical due to the high dimension of the space. To remedy this, we introduce two new data structures, floorgraphs and D–graphs, for implicit representation and efficient storage of free space.

4.3.2. Floorgraphs

A *floorgraph* for a pair of parts (P_i, P_j) is a graph obtained by decomposing the complement of *interior*$(CO(P_i, P_j))$ into overlapping closed convex cells defined by oriented plane patches from $CO(P_i, P_j)$, instead of full bounding planes. Each cell corresponds to one node in the floorgraph and is defined by linear equalities and inequalities. Two nodes are connected by an undirected edge if their associated cells intersect. Edges are associated with constraints describing the intersection of the two adjacent nodes.

Figure 5b shows an example of a decomposition. Note that it has fewer cells than the decomposition in Figure 5a. Figure 5c shows the corresponding floorgraph. The constraints associated with node 5 are $\{x_P - x_Q \geq a, y_P - y_Q \leq b\}$, where a and b are constants. The constraints of the edge connecting nodes 4 and 5 are $\{x_P - x_Q \geq a, y_P - y_Q = b\}$.

The advantage of the floorgraph representation is that cells are bounded by surface patches rather than full planes. As a consequence, free space is decomposed into considerably fewer cells.

4.3.3. D–Graphs

To obtain a composite configuration space, we combine the floorgraphs from all pairs of parts into a new data structure, called *D–graph*. A D–graph is an exact and compact representation for cells and their adjacency relations in the d-dimensional assembly configuration space.

A *D–node* is a tuple $S = (n_1, \ldots, n_f)$ of floorgraph nodes, where a tuple entry n_i represents the current floorgraph node of the ith pair of parts. The length of the tuple is $f = k(k-1)/2$, where k is the number of parts in the assembly. Associated with each D–node is the set of all embedded constraints stemming from the current floorgraph nodes. Note that most D–nodes represent empty cells, as most combinations of constraints stemming from the pairwise floorgraph cells are infeasible. The D–node containing a given valid assembly configuration is obtained by projecting the configuration in each pairwise configuration space and searching the corresponding floorgraphs for the cell containing the projection. A *D–edge* is an edge between two nonempty neighboring D–nodes whose associated cells intersect.

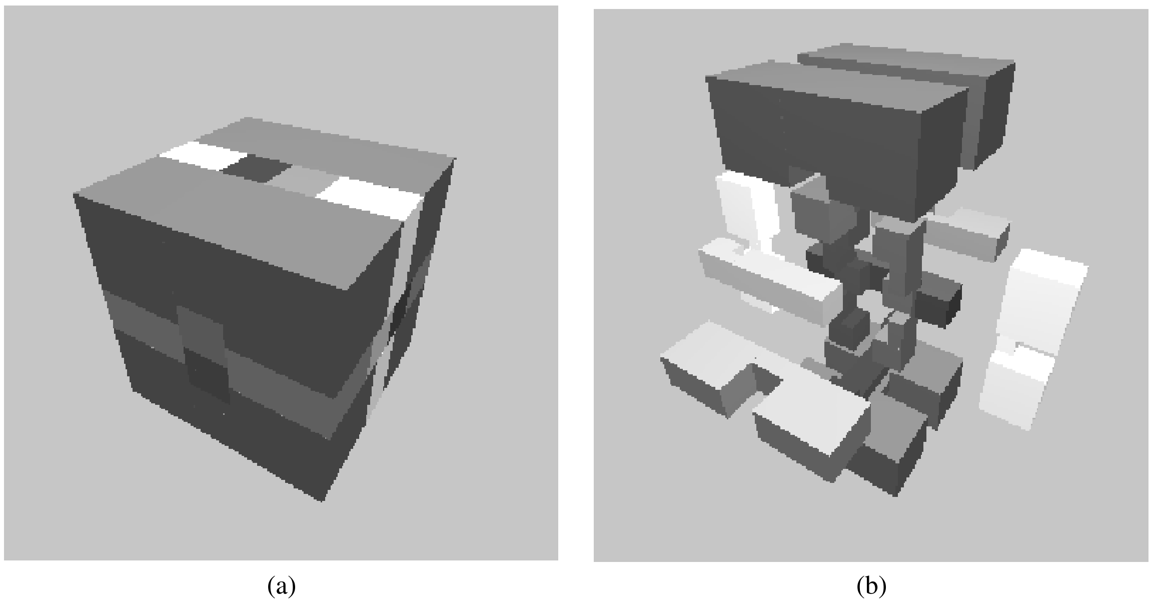

Fig. 4. (a) Wooden cube puzzle with 12 parts requiring a nonmonotone assembly sequence. (b) Exploded view showing the individual parts.

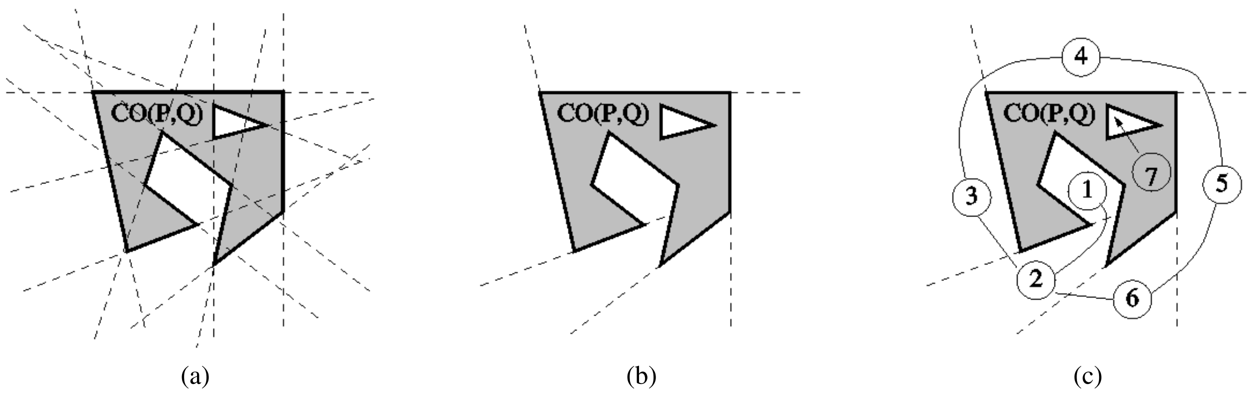

Fig. 5. (a) Decomposition of two-dimensional configuration space obstacles into an arrangement of half-planes; (b) decomposition of the same obstacle into overlapping cells defined by oriented bounded half-planes; and (c) its corresponding floorgraph.

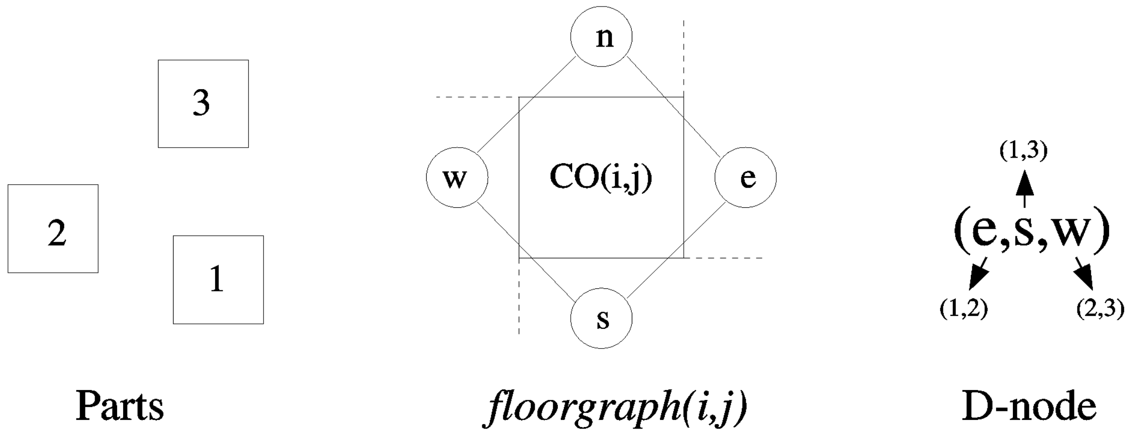

Fig. 6. Example of D–node construction from pairwise floorgraphs.

The example in Figure 6 illustrates D–node construction from pairwise floorgraphs. It shows a configuration of three boxes on the left, the generic floorgraph for a pair of boxes in the middle, and on the right, the D–node containing the given configuration. For clarity, the pairs corresponding to the D–node entries are referenced by arrows in the right picture. The first entry e stems from the pair $(1, 2)$ and indicates that the relative position of part 1 with respect to part 2 is contained in cell e in all configurations contained in this D–node.

4.3.4. Incremental Search Algorithm

A valid disassembly path is obtained by searching the D–graph. We begin by constructing the initial D–node corresponding to the cell containing the assembled parts placement and explore neighboring D–nodes until we encounter an unbounded D–node. The disassembly motions are directly obtained from D–nodes in the path from the initial D–node to the unbounded one.

Precomputing the D–graph and then searching it for a path is both impractical and unnecessary: instead, we can start from the initial D–node and incrementally construct and test reachable D–nodes for unboundedness. Visited D–nodes are stored and marked, so that they are not visited again later. To generate successor D–nodes, it is not necessary to compute the complete boundaries of cells in higher-dimensional space: it is sufficient to test so-called door cells. These door cells are convex patches on the cell boundaries and will be defined next.

Formally, D–node $S' = (n'_1, \ldots, n'_f)$ is said to be a *successor* of a nonempty D–node $S = (n_1, \ldots, n_f)$ if (1) n_i and n'_i are neighbors in floorgraph i, (2) $n_j = n'_j$ for all $j \neq i$, and (3) the combined constraints from (S, n'_i) define a nonempty cell, called door cell. For a given valid D–node, we generate all successor D–nodes by successively switching all entries, one at each step, and verifying that condition (3) holds with a linear feasibility test. In Schweikard and Schwarzer (1998), we prove that all neighbors of a feasible D–node can be reached via one or more successor D–nodes each.

In most cases, only a fraction of the entire D–graph will be searched, resulting in a practical algorithm.

4.3.5. Searching with Reduced D–Nodes

Experimentation with practical examples shows that the incremental algorithm is still inefficient in certain cases because it generates many redundant D–nodes. Some of them can be avoided by dynamically eliminating unnecessary pairwise constraints based on the search context. This technique, which we call *D–node reduction*, can speed up the search significantly. We motivate the problem and briefly describe its solution with a representative example next.

Consider the example in Figure 7a. It consists of a container X with four separate slots, numbered 1 to 4, and four boxes, named A, B, C, and D. The four boxes cannot interact, as there is no passage between them. The pairwise floorgraphs of the container and the boxes are topologically equivalent to the container: they consist of four disconnected cells, which we also number 1 to 4 to correspond with the slot number. The pairwise floorgraphs of box pairs are all similar to the floorgraph in Figure 7b: they consist of a single component of four connected cells.

The initial D–node is formed by taking one floorgraph cell for each pair of parts in their initial configuration. We denote the initial D–node by

$$(1_{AX}, 2_{BX}, 3_{CX}, 4_{DX}, 1_{BA}, 1_{CA}, 1_{DA}, 1_{CB}, 1_{DB}, 2_{DC}),$$

where 1_{AX} denotes cell 1 in the configuration space of A and X, and 1_{BA} denotes that the relative position of part B with respect to part A is contained in cell 1 in the floorgraph of this pair of parts. The incremental algorithm will generate and test all combinations of vertical orderings of the boxes (a box is either above or below the other), which is exponential in the number of boxes.

We can reduce this D–node to $(1_{AX}, 2_{BX}, 3_{CX}, 4_{DX})$ without loss of information. Since the boxes cannot interact with each other, the constraints of the reduced node are sufficient to prevent all boxes from intersecting. This can be verified by projecting the cell to all pairwise configuration spaces of the eliminated pairs. The projections will not intersect the corresponding pairwise configuration space obstacles. In this example, the reduction results in a single D–node without any successor nodes, thus achieving an exponential speedup.

More formally, we can interpret a D–node itself as a graph. Nodes in this graph correspond to parts in the assembly. An edge between two parts will be labeled with the currently active floorgraph node from the two adjacent parts. Consequently, all D–nodes were complete graphs (cliques). A reduced D–node will contain only those edges (pairwise constraints) that are currently required to prevent intersections. Figure 7c shows the reduced D–node graph for the assembly configuration in Figure 7a. Note that only constraints for the pairs AX, BX, CX, and DX are necessary since they prevent all pairs of boxes from interacting. The edges are labeled with the corresponding floorgraph cells.

By using reduced D–nodes instead of full nodes, the incremental search algorithm gains in efficiency because

- fewer successor D–nodes have to be generated and tested,
- fewer constraints contribute to each cell in d-dimensional space, and
- the cell decomposition becomes coarser, thereby reducing the size of the graph.

The extra work is maintaining the D-nodes during search. When moving to a successor D–node, a validation of the reduced D–node must be performed. As a consequence, edges

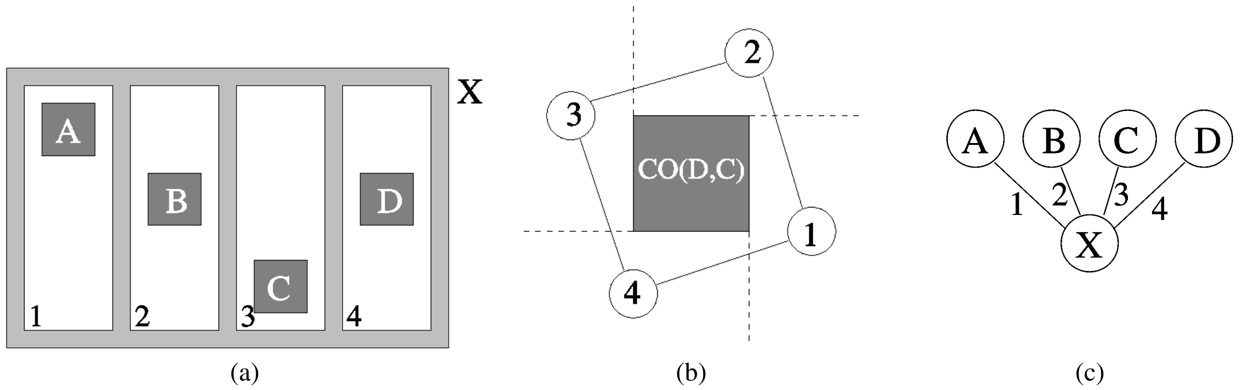

Fig. 7. (a) Example for D–node reduction, (b) reduced D–node graph for the configuration in (a), and (c) representative floorgraph for two boxes.

may have to be added or can be deleted from the D–node graph.

Note that the eliminated constraints are *not* necessarily redundant in the classical sense of linear programming, since they indeed may partition the original cell (as in the above example). If this is the case, by discarding pairwise constraints the cell will be augmented and the compound configuration space decomposition becomes coarser. We can therefore also interpret the impact of D–node reduction as *cell augmentation*.

5. Experimental Results

To demonstrate the practical value of our algorithms and quantify their performance, the unboundedness testing algorithms (Sections 3.2 and 3.3) and the assembly-planning algorithms (Section 4) were implemented in C++ and run on a SUN Ultra-60 workstation with 256 MB of RAM under Unix. We constructed representative planar and spatial examples to test the different properties of the algorithms and to evaluate the heuristics. Singular value decomposition was used for solving homogeneous equations, and the simplex method was used for feasibility testing.

5.1. Unboundedness Testing

To quantify the performance of the new unboundedness testing algorithm with respect to the simple one, we created a series of problems consisting of randomly generated homogeneous cells.

Table 2 summarizes the results. The first 5 columns (cell characteristics) describe the properties of the randomly generated cells: # the name, d the number of variables (problem dimensionality), $n^{\geq}$ the number of random half-space constraints of the form $\mathbf{a}_i\mathbf{x} \geq 0$, $n^{=}$ the number of random hyperplane constraints $\mathbf{a}_i\mathbf{x} = 0$, and unb whether the cell is unbounded or not. The next two columns (simple algorithm) describe the running time t (in seconds) that was measured for f actually performed feasibility tests. For all bounded cells except (b), we did not run the simple algorithm to completion, since this requires exactly $2d$ feasibility tests (all failing). For these cases, the following column specifies an estimate $T = 2dt/f$ for the complete running time of all $2d$ tests. The last four columns (new algorithm) describe the running times t_h and t_f of solving the homogeneous equations and performing the feasibility test and whether the cell was found to be unbounded by the respective step (unb_h and unb_f).

The results show that the new algorithm can handle large, higher-dimensional problems with many constraints. It is significantly faster on large bounded cells for which the simple algorithm leads to impractical running times (column T in cases d, f, and h). This is of special importance for general translational assembly applications where a large number of bounded cells may have to be tested before an unbounded cell is found. The simple test has better performance only when the set of unbounded directions is a subset of the solutions of the homogeneous equations, and where the number of necessary feasibility tests is small (cases a, c, and g). In the new algorithm, the better performance of the feasibility test indicates that it can be advantageous to first perform this test before solving the homogeneous equations.

5.2. M*-Handed Assembly*

We tested the m-handed translational disassembly algorithm in Section 4.2 on a variety of representative planar examples. The program uses the new unboundedness testing algorithm and the LEDA library (Mehlhorn and Naeher 1995) for the geometric computations.

The example in Figure 8 shows a planar assembly with no separated pair of parts and snapshots of the computed m-handed motion that separates the parts. The program establishes that four of the five parts must be translated simultaneously, with distinct velocities and directions. Unboundedness testing takes only 0.02 s. By examining subassemblies, the program also determined that no m-handed disassembly motion with fewer than four hands is possible. An extension of the example in Figure 3 from Section 4.2.1 to 16 moving

Table 2. Results of Running the Simple and New Linear Unboundedness Testing Algorithm on Randomly Generated Homogeneous Cells Defined by d Variables, $n^{\geq}$ Half-Space Constraints, and $n^{=}$ Hyperplane Constraints

Cell Characteristics					Simple Algorithm			New Algorithm			
#	d	$n^{\geq}$	$n^{=}$	unb	t	f	T	t_h	unb_h	t_f	unb_f
a	40	0	39	yes	0.1	1	0.1	0.2	yes	0.1	no
b	40	0	40	no	7.8	80	7.8	0.2	no	0.1	no
c	200	0	199	yes	10	1	10	20	yes	7.6	no
d	200	0	200	no	60	8	3,000	20	no	8.1	no
e	200	100	0	yes	1.1	1	1.1	6.6	yes	0.8	yes
f	200	500	0	no	976	8	48,800	54	no	119	no
g	500	0	499	yes	171	1	171	399	yes	135	no
h	500	200	400	no	1233	4	308,250	841	no	275	no

parts also required only a small fraction of a second to find a disassembly motion.

Figure 9 shows two nearly identical eight-part assemblies with 14 separated pairs and 14 nonseparated pairs of parts, respectively. In both cases, the nonconvex constraints from all separated pairs of parts can be discarded since the constraints from the nonseparated pairs are restrictive enough to prevent collisions. This is detected in a preprocessing step by projecting the constraints from all nonseparated pairs into the configuration spaces of all separated pairs of parts. The remaining test for unboundedness determines that the assembly in Figure 9a can be disassembled, but not the one in Figure 9b. The total running time is about 0.55 s, with 0.5 s for the projections and 0.05 s for unboundedness testing. Without the reductions, 2^{14} cells would have to be tested for the example in Figure 9b, with a running time of about 800 s. Heuristic preprocessing eliminates all nonconvex constraints and leaves a single d-dimensional cell to be tested for unboundedness. Since the number of initially separated pairs grows quadratically with the number of parts, the speedup obtained by the heuristic is exponential.

We tested a large number of examples with both initially separated and nonseparated pairs of parts and found that the described reduction of nonconvex constraints works very effectively on them.

5.3. General Translational Assembly

We ran the algorithm for general translational assembly planning with D–node reduction from Section 4.3 on several realistic spatial examples. For comparison, we incorporated both the simple (Section 3.2) and the new (Section 3.3) unboundedness testing algorithm. Running times were recorded for the following two tasks:

- **Search for the first unbounded cell.** The search stops when the first unbounded cell is found, i.e., at least one part can be removed from the remaining parts. Table 3 summarizes the results. The columns specify the problem name, the number of parts, the number of cells tested for unboundedness during the search, the times of the simple and new unboundedness test for all cells, and the total search time required.
- **Complete disassembly.** Having found an unbounded cell, the problem is split into separable subsets and the subproblems are processed recursively until they consist of single parts or cannot be disassembled any further. Table 4 summarizes the cumulative times for complete disassembly.

We now discuss the results in more detail.

Cube. For the 12-part wooden cube in Figure 4 from Section 4.3, 38 unboundedness tests were necessary to find a sequence for removing the first subset of parts (Table 3). Our new unboundedness test is almost 6 times faster than the simple algorithm. For complete disassembly, the required nonmonotone sequence of translations was found by our program in about 13 s, of which about 2 s are for the new unboundedness test (instead of more than 12 s for the simple test, Table 4). The first row in Table 5 shows the running times of the algorithm without D–node reduction (for comparison, the numbers in parentheses from Table 4 are with D–node reduction). The speedup factor of about three shows that D–node reduction is very effective even for tightly packed assemblies with many interlaced parts.

Bookcase. For the 17-part bookcase in Figure 1b from Section 1, constraints from many pairs of parts can be eliminated due to redundancy. This is illustrated in Figure 10, which shows an exploded side view of one half of the book case (11 parts instead of 17) and the reduced D–node for the initial mounted configuration. Only 22 out of 55 pairwise constraints (edges) are necessary for the 11 parts. For the entire bookcase assembly, the initial reduced D–node graph contains 50 out of 136 edges. The disassembly plan was computed in about 16 s for the search and 10 s for the new unboundedness test. A total of 277 cells were tested for unboundedness. In contrast, the simple algorithm took about 5 times longer (56 s). Table 5 shows that D–node reduction with the new unboundedness test speeds up the computation by a factor of about 3.5.

Desk. Figure 11a shows a desk with eight drawers. The container of the drawers is a single part. Each of the draw-

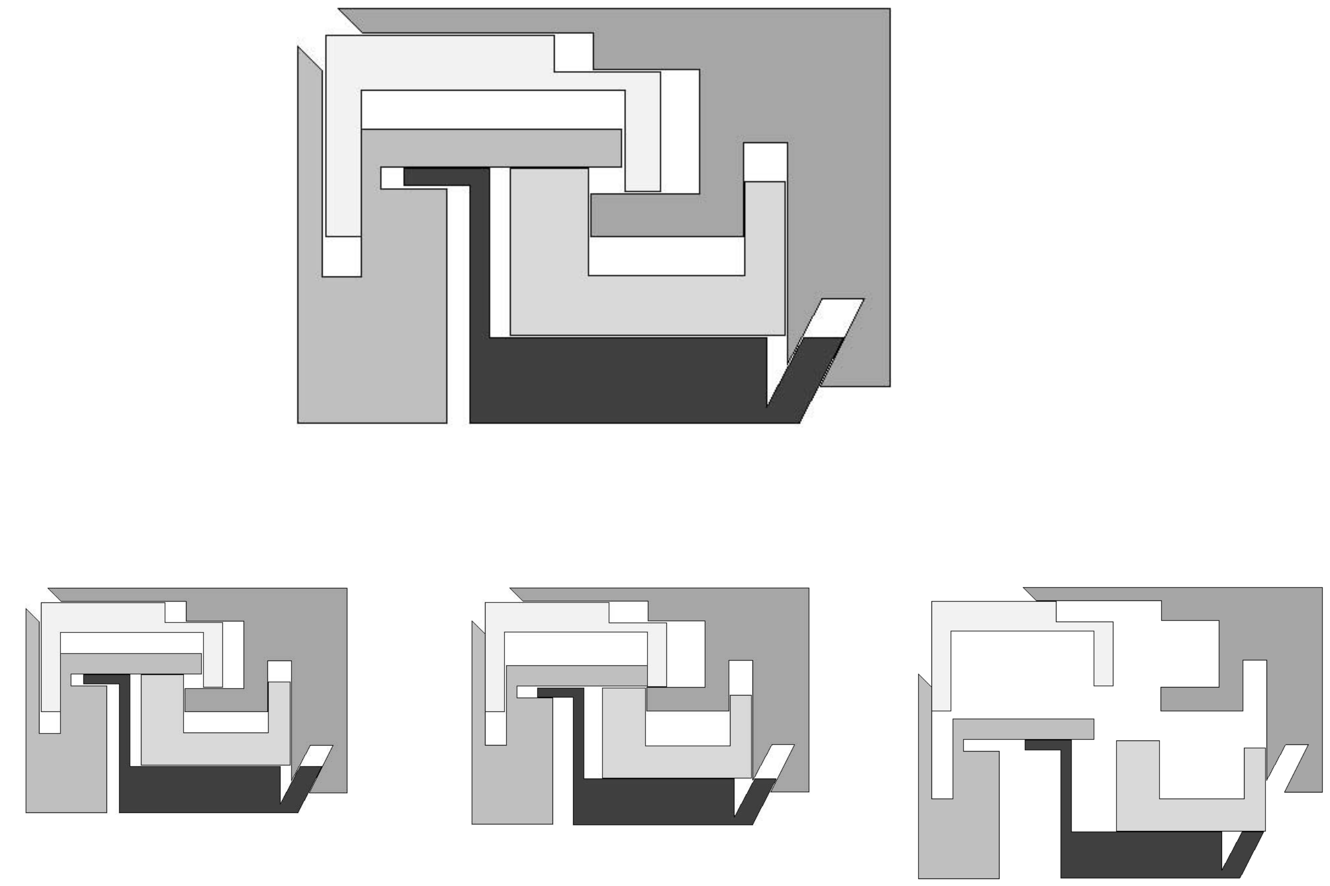

Fig. 8. Top: parts of an assembly with no initially separated pair of parts can be separated by the motion indicated by arrows, whose length represents relative velocities. Bottom: snapshots of separating motion.

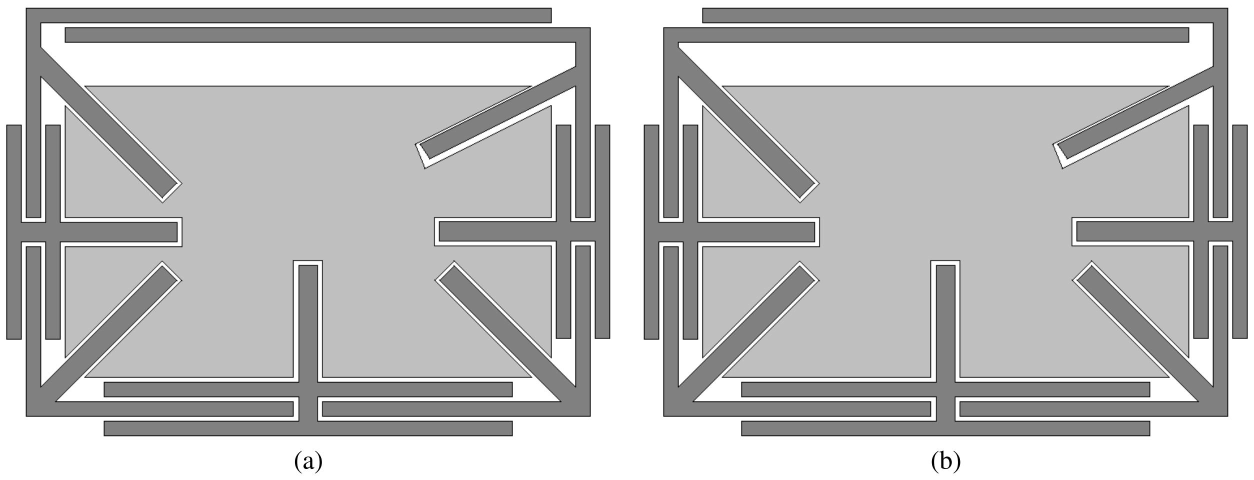

Fig. 9. Two assemblies with several separated pairs of parts. Parts in (a) are separable, whereas the parts in (b) are blocked.

Table 3. Searching for First Unbounded Cell. Total Running Times (in seconds) Are the Sum of the Numbers in Columns Simple and Search or New and Search

Example	Parts	U-tests	Simple	New	Search
Cube	12	38	5.91	1.00	7.24
Book case	17	147	50.71	9.82	14.83
Desk1, can be disassembled					
(not decoupled)	9	504	25.17	2.61	7.64
(decoupled)	2 × 5	25	0.37	0.03	0.18
Desk2, cannot be disassembled					
(not decoupled)	9	256	11.14	1.23	13.09
(decoupled)	2 × 5	32	0.50	0.08	0.52

Table 4. Complete Disassembly. Total Running Times (in seconds) Are the Sum of the Numbers in Columns Simple and Search or New and Search

Example	Parts	U-tests	Simple	New	Search
Cube	12	132	12.42	2.05	10.94
Book case	17	277	55.82	10.42	16.23
Desk1, can be disassembled					
(not decoupled)	9	960	37.10	3.86	10.35
(decoupled)	2 × 5	118	1.24	0.10	0.54

Table 5. Running Times for Complete Disassembly without D–Node Reduction. Times Using D–node Reduction from Table 4 Are Shown in Parentheses

Example	Simple		New		Search	
Cube	22.59	(12.42)	3.34	(2.05)	33.85	(10.94)
Book case	115.31	(55.82)	16.62	(10.42)	77.90	(16.23)
Desk2, cannot be disassembled (not decoupled)	195.93	(11.14)	22.7	(1.23)	1635.18	(13.09)

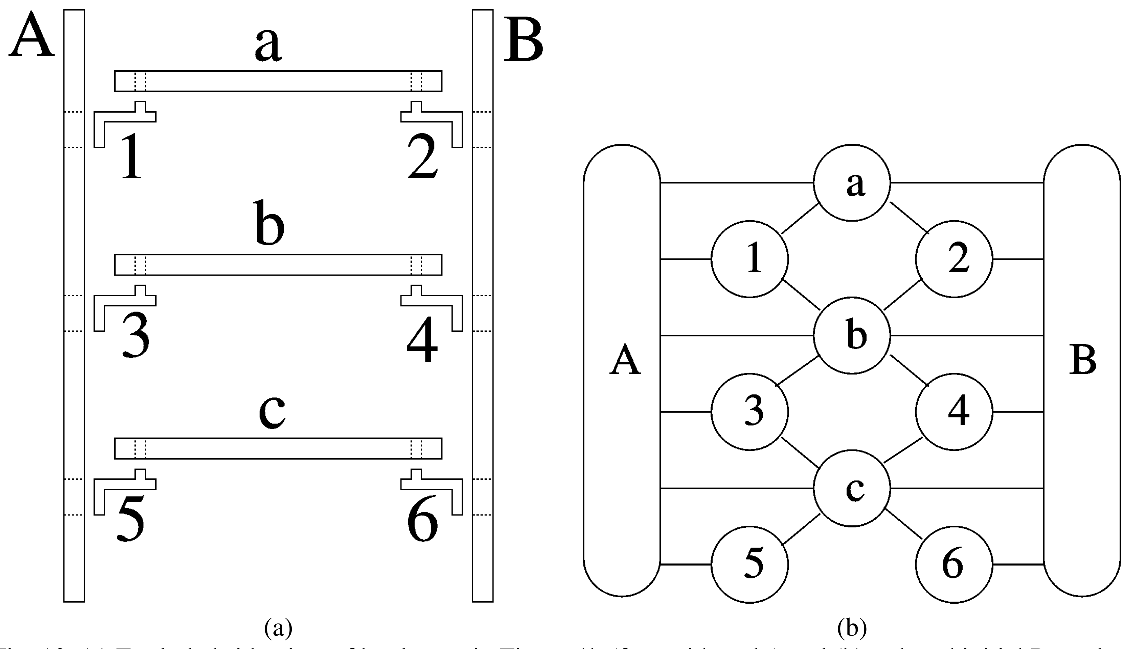

Fig. 10. (a) Exploded side view of book case in Figure 1b (front side only) and (b) reduced initial D–node graph.

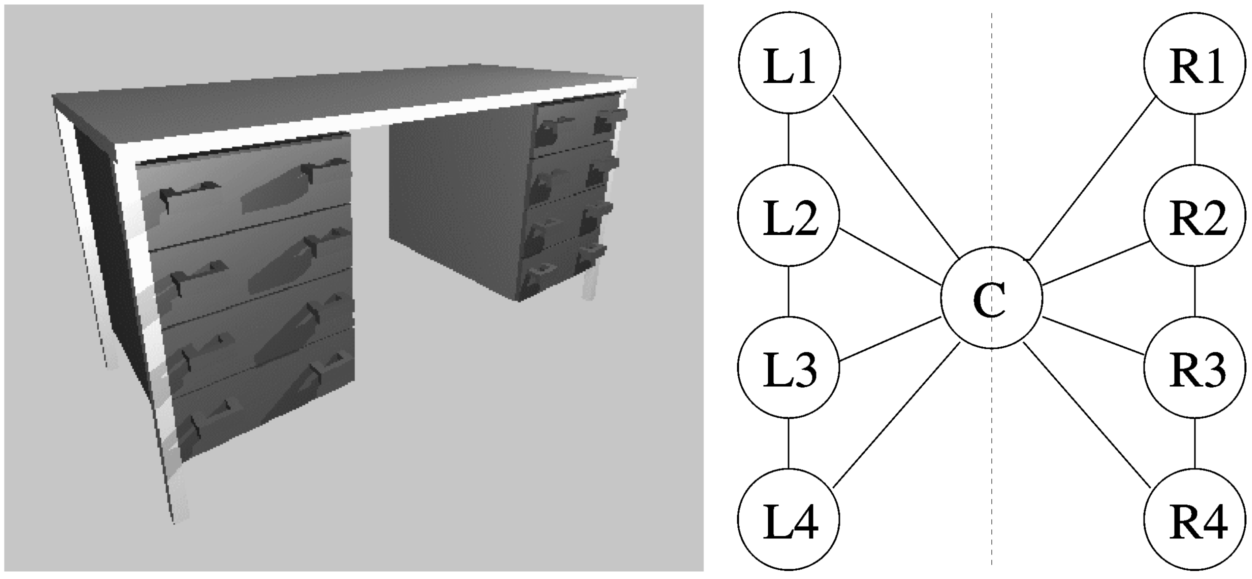

Fig. 11. (a) Desk with eight drawers and (b) initial D–node allowing for decoupling into two subproblems.

ers slides along two splints, secured against falling out. The drawers must be pulled out, lifted, and then pulled further out to be removed. Before a single drawer can be lifted enough for removal, the one above it must have been removed. We created two versions of the desk, one demountable and the other one not demountable. The second model is nearly identical to the first, but the back panels of the topmost drawers were heightened so that these drawers could no longer be lifted enough for removal. Consequently, none of the drawers can be removed, although they can all be moved back and forth independently.

The running times for finding the first unbounded cell and complete disassembly are shown in Tables 3 and 4 (rows indicated by "not decoupled"). For comparison, Table 5 shows that the total running time (using our new unboundedness test) is more than 100 times slower when D–node reduction is turned off.

With D–node reduction, 2^8 cells are examined since the two cabinets are not checked independently (each of the eight drawers is checked in two positions: pushed in and pulled out). However, the drawers in the left cabinet cannot interact with the drawers in the right cabinet. Figure 11b shows the initial reduced D–node. The container (C) separates the left (L1-L4) and right (R1-R4) drawers. Thus, no edges between any L- and R-node are present and the problem can be split in two simpler decoupled subproblems (left and right cabinet). A simple depth-first search (DFS) on the graph of the current D–node can be used to verify a given part separates the assembly in two unrelated subassemblies. We start the search at the node of the part in question. If the root node of the resulting search tree has two or more children, then their subtrees comprise independent subassemblies that can be decoupled. If the root node has only a single child, the assembly cannot be decoupled at the root node. To find a part that allows for decoupling the assembly in this sense, we successively perform DFS starting at all parts of the assembly in overall $O(k^2)$ time.

For the desk example, we can decouple the left and right cabinets as described above. The resulting running times are included in the tables (rows indicated by "decoupled" in the leftmost column). The problem is split in two decoupled problems with two parts each, which results in significant speedup up to a factor of almost 50. The time needed for identifying the separate subproblems by performing a DFS on the D–node graph is negligible.

5.4. Local Computation of Constraints for Pairs of Polyhedra

In the above examples, all parts have relatively few faces. As a consequence, preprocessing times (computation of complete floorgraphs for all pairs of parts) range from a small fraction of a second to a few seconds per pair. For larger models, configuration space computation for pairs of parts can become much slower. Note, however, that since in our application the search proceeds incrementally, we do not need to precompute complete configuration spaces. Instead, it is sufficient to compute a current cell and its neighbors for each pair of parts. During the search, only the required parts of the pairwise configuration spaces will be expanded. Although parts may be quite complex, the portions of the Minkowski sums that actually have to be computed are usually much simpler in assembly-planning applications (e.g., for a situation similar to a peg in a hole, the current cell may be only a ray or a cylinder as illustrated in Fig. 12).

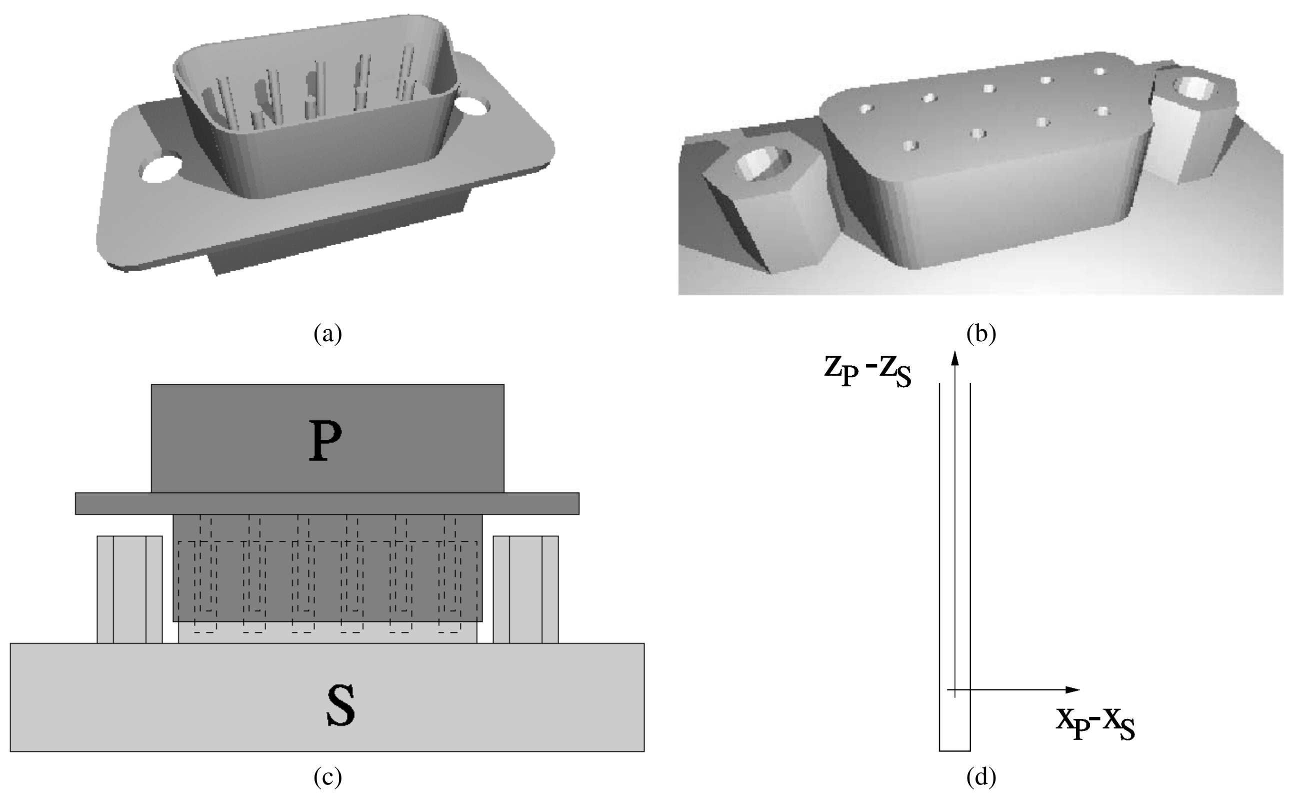

Fig. 12. (a) RS-232 plug P and (b) socket S. (c) Initial configuration (side view). (d) Projection of cell for placement in (c).

The idea of computing configuration spaces locally is not new. Ramkumar (1998) proposes in his conclusions to compute local portions of the convolution from which parts of the Minkowski sum surface can be derived. However, no algorithm or experimental results are given there. Bounding volume hierarchies as described, e.g., in Larsen et al. (1999), are a common approach for computing the distance between nonconvex polyhedra. They are practical for large models and can be easily extended to extract relevant pairs of triangles for incremental configuration space cell computation. To illustrate this, we created two coaxial cylinders with 512 triangles each. The cell for the selected configuration is also a cylinder. Filtering the closest triangle pairs required testing $13 \cdot 10^3$ bounding volume pairs and $2.6 \cdot 10^3$ triangle pairs. The resulting triangle pairs are sufficient to construct the cell. The total running time is only a fraction of a second. In contrast, considering all $2.6 \cdot 10^5$ pairs and computing the complete convolution of the two cylinders would be much slower.

6. Conclusion and Open Problems

Linear unboundedness testing is a classical problem in linear programming and plays an important role in assembly planning. In this paper, we presented a new unboundedness testing algorithm and showed that testing for unboundedness reduces to solving a single homogeneous system of equations and performing a single linear feasibility test. The $O(nd^{4.5})$ algorithm, where n is the number of constraints and d their dimension, is the fastest known algorithm for unboundedness testing and is practical, as our implementation and experiments suggest.

We introduced a framework for translational assembly-planning applications based on linear constraints. We formulated the m-handed assembly problem and showed its relation to unboundedness testing. Polynomial running time can be achieved if the number of nonconvex pairwise constraints is constant or can be reduced to a constant number. For the latter, we proposed an effective heuristic reduction technique that even allows for exponential speedup in special cases.

An interesting open theoretical question is whether the general m-handed assembly-planning problem with initially separated and nonseparated pairs of parts is *NP*-hard. We have not further investigated this question here, but a reduction of an *NP*-hard problem to m-handed assembly seems not trivial for the following reasons:

1. If the considered *NP*-hard problem has a solution, then $\Theta(k)$ parts will have to move simultaneously in the constructed assembly.

2. The motions must be extended translations, which imposes strong restrictions on the possible part shapes and assembly constructions.

Based on the linear constraint framework, we developed practical algorithms for general translational assembly planning of polygonal and polyhedral parts. To decide if the assembly can be disassembled by arbitrary translational motions, we traverse the reachable free cells in a d-dimensional static arrangement of hyperplanes representing simultaneous placements of all parts. We start with the cell containing the origin and test each visited cell for unboundedness. Since the unboundedness test is called at each step of the search, a fast and practically efficient test such as the one presented here is essential. Our experimental results demonstrate that our approach is practical.

Several extensions can be directly integrated into our linear constraint framework. For example, small overlap between parts, e.g., to model deformations, can be allowed by introducing an overlap-parameter $\epsilon^{(ij)} \geq 0$ for each pair of parts. A pairwise constraint will thus simply be rewritten as $\mathbf{a}^{(ij)}\mathbf{x} \geq b - \epsilon^{(ij)}$. Other extensions include linear models for shape and motion uncertainty.

Acknowledgments

We would like to thank one of the reviewers for comments and bringing the work in Baraff, Mattikalli, and Khosla (1997) to our attention. Joskowicz is supported by grant 98/536 from the Israeli Academy of Science and by a grant from the Authority for Research and Development, The Hebrew University of Jerusalem, Israel.

References

Agarwal, P. K., de Berg, M., Halperin, D., and Sharir, M. 1996. Efficient generation of k-directional assembly sequences. *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Atlanta, GA, pp. 122–131.

Baraff, D., Mattikalli, R., and Khosla, P. 1997. Minimal fixturing of frictionless assemblies: Complexity and algorithms. *Algorithmica* 19:4–39.

Chazelle, B., Ottmann, T. A., Soisalon-Soininen, E., and Wood, D. 1984. The complexity and decidability of SEPARATION. *Proc. 11th Int. Colloquium on Autom. Lang. Prog.* LNCS 172, 119–127.

Dawson, R. 1984. On removing a ball without disturbing the others. *Mathematics Magazine* 57(1):27–30.

Halperin, D. Latombe, J. C., and Wilson, R. H. 1998. A general framework for assembly planning: The motion space approach. *Proc. of the 12th ACM Conference on Computational Geometry*, Minneapolis, MN, pp. 9–18.

Halperin, D., and Wilson, R. H. 1997. Assembly partitioning along simple paths: The case of multiple translations. *Advanced Robotics* 11:127–146.

Hoffman, R. L. 1991. A common sense approach to assembly sequence planning. In *Computer-Aided Mechanical Assembly Planning*, ed. L. S. Homem de Mello and S. Lee, 289–314. Boston: Kluwer.

Homem de Mello, L. S., and Lee, S., eds. 1991. *Computer-Aided Mechanical Assembly Planning*. Boston: Kluwer Academic.

Huyn, T., Joskowicz, L., Lassez, C., and Lassez, J. L. 1991. Practical tools for reasoning about linear constraints. *Fundamenta Informaticae* 15:3–4.

Joskowicz, L., and Taylor, R. H. 1996. Interference-free insertion of a solid body into a cavity: An algorithm and a medical application. *International Journal of Robotics Research* 15(3):211–229.

Karmarkar, N. 1984. A new polynomial-time algorithm for linear programming. *STOC*, Washington, DC, pp. 302–311.

Larsen, E., Gottschalk, S., Lin, M. C., and Manocha, D. 1999. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, University of North Carolina, Chapel Hill.

Latombe, J. C. 1991. *Robot Motion Planning*. Boston: Kluwer Academic.

Mehlhorn, K., and Naeher, S. 1995. *LEDA: A Platform for Combinatorial and Geometric Computing*. Saarbrücken: Max-Planck-Institut für Informatik.

Ramkumar, G. D. 1998. *Tracings and Their Convolution: Theory and Applications*. Ph.D. thesis, Department of Computer Science, Stanford University.

Schweikard, A., and Schwarzer, F. 1998. Detecting geometric infeasibility. *Artificial Intelligence* 105(1-2):139–159.

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

Snoeyink, J., and Stolfi, J. 1994. Objects that cannot be taken apart with two hands. *Discrete and Computational Geometry* 12:367–384.

Stoer, J. 1976. *Einführung in die Numerische Mathematik I*. Berlin/New York: Springer-Verlag.

Toussaint, G. T. 1985. Movable separability of sets. In *Computational Geometry*, ed. G. T. Toussaint. New York: Elsevier.

Wolter, J. D., and Trinkle, J. C. 1994. Automatic selection of fixture points for frictionless assemblies. Technical Report CS TR 94-017, Texas A&M University.

Wright, M. H. 1992. Interior methods for constrained optimization. In *Acta Numerica*, ed. A. Iserles. New York: Cambridge University Press.