Detecting geometric infeasibility

Achim Schweikard *, Fabian Schwarzer

Technische Universität München, Informatik, 80290 München, Germany

Received 22 May 1997; received in revised form 19 March 1998

Abstract

The problem of deciding whether one or more objects can be removed from a set of other planar or spatial objects arises in assembly planning, computer-aided design, robotics and pharmaceutical drug design. In this context, it will be shown that certain D-dimensional arrangements of hyperplanes can be analyzed in the following way: only a single connected component is traversed, and the arrangement is analyzed as an arrangement of surface patches rather than full hyperplanes. In special cases, this reduction allows for polynomial time bounds, even if the boundary of the set of reachable placements has exponential complexity. The described techniques provide the basis for an exact method for translational assembly planning with many degrees of freedom. Experiments obtained with an implementation suggest that problems with random planning methods, which are related to the choice of internal parameters can be avoided with this exact method. In addition, unsolvability can be established and the program can be applied to the verification of symbolic rules describing the geometry. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Geometric reasoning; Assembly planning; Motion planning; Complete algorithms; Arrangement computation in D dimensions

1. Introduction

We conjecture that certain human abilities in geometric reasoning and motion planning can be represented by a small number of basic principles. Consider the example in Fig. 1. Parts $P$ and $Q$ are movable; the container is fixed. The goal is to decide whether or not part $P$ is removable by a sequence of translational motions of both $P$ and $Q$. To explain how $P$ can be removed, one would examine a series of critical intermediate placements of $P$ and $Q$, and test for removability in each placement.

* Corresponding author. Email: schweika@informatik.tu-muenchen.de.

Email: schwarzf@in.tum.de.
Fig. 1. Detecting whether a part can be removed from other parts. Parts $P$ and $Q$ are movable; the container is fixed.

Fig. 2. To determine whether or not the bolt can be removed, reachable placements of the lock must be examined.

Enumerating such intermediate placements is difficult, if the number of parts is larger than in this simple example. Specifically, the number of possible distinct contact placements between parts expands very rapidly with the number of parts. Therefore it is unlikely that all combinations of contact placements can be examined by the human. However, human geometric reasoning is capable not only of finding a solution to a given motion planning problem rapidly, but also of quickly recognizing that no feasible solution exists. It would seem that recognizing infeasibility is more difficult than finding one motion, if one exists. Establishing that a given problem is infeasible requires a proof.

In the second example (Fig. 2), the human is able to recognize that relatively few intermediate positions of the lock need to be tested to prove that the bolt is not removable. The main problem with reproducing such an intuitive approach in an algorithm is the decision about which of the possible relative placements are indeed critical. The complete enumeration of all distinct contact placements of parts quickly becomes impossible even for simple cases.
We are thus interested in methods for reducing the set of critical relative placements of parts, while retaining completeness. To obtain a practical method, it is necessary to recognize the essential contacts between parts. But notice that our detection of infeasibility would be in error if the set of examined contacts was too small.

In this context we will address the following problem:

*Given an assembly of polygonal or polyhedral parts, decide whether one or more parts can be removed from the remaining set of parts by an arbitrary sequence of translations.*

Notice that we will restrict our attention to translational motion. This includes lock-and-key configurations, where several groups of parts must move simultaneously into distinct directions, and/or change direction during motion. The number of translations in such a sequence is not limited.

2. Related work

Toussaint [12] surveys earlier methods for separating sets in two and three dimensions. Applications in computer-aided design and medicine are considered in [5,6]. Agarwal, de Berg, Halperin and Sharir [1] consider sequences of translations for separating polyhedra. In [1] the set of allowed motion directions is assumed to be given in advance. In [15] the concept of blocking graphs is introduced. This concept allows for deciding whether there is a subassembly $S$ of a given assembly, such that $S$ is removable by a single translation. The output of the corresponding algorithm consists of $S$ and a translational direction $d$ for removing $S$, if there is such a subassembly $S$. This computation is possible in polynomial time, even though the number of removable subassemblies is exponential in general. The algorithm in [15] will thus compute a valid subassembly and a removal direction if there is such a subassembly, but avoids enumerating the entire set of possible subassemblies.

Specifically, the algorithm in [15] uses the following strategy. We pull on each of the parts in turn with a series of given translational directions. For each direction in this set, we compute the subset of parts which must follow the given pulling motion. Blocking graphs are used to compute this subset. The overall computing time is polynomial, since each translational direction can be analyzed in polynomial time, and the number of directions which must be analyzed is polynomial. This simple approach allows for several extensions (see, e.g., [4]), but inherently requires that all moving parts perform the same motion, and cannot be generalized to the case of multiple subassemblies moving independently.

Guibas and Halperin et al. [4] consider infinitesimal translations and rotations for partitioning three-dimensional sets. Such infinitesimal motions can indicate directions for removing parts in a single step. However, it cannot be guaranteed that the corresponding extended motion will be collision-free.

an exponential lower bound for the number of translations necessary to separate objects (see also [111]). This lower bound already holds for a restricted class of polygons in the plane. Interestingly, in special cases our technique allows for establishing infeasibility in polynomial time, even if the boundary of the set of reachable placements has exponential complexity.

The next section gives an informal description of the basic principles in this context. The main idea of the present approach is described in Section 4. Indeed, there is a remarkably simple way to compute certain multi-dimensional arrangements as arrangements of surface patches, while avoiding the computation of the entire underlying arrangement of hyperplanes. However, this is only possible for a particular class of arrangements. The arrangements to be considered here all belong to this class. Furthermore, we can avoid computing the boundaries of connected components in the underlying arrangement. As noted above, in special cases this allows for a complete search in polynomial time. Section 5 derives an algorithm from these principles and proves completeness. Section 6 describes a fast test for establishing whether a cell in a $D$-dimensional arrangement is bounded. Section 7 gives the analysis of the mentioned reductions. Section 8 describes experimental results. The evaluation compares the derived methods with random motion planners. The experiments suggest that it is possible to decide about infeasibility in a complete way. It is shown that the above principles lead to an exact algorithm of surprising performance in experiments.

3. Basic concepts

Let $P_1, \ldots, P_k$ be an assembly of three-dimensional parts. Thus $P_1, \ldots, P_k$ are nonintersecting polyhedra in given spatial placement. We allow for all parts to translate in space independently. The position of each part is given by three parameters. A space $E^D$ of dimension $D$ describes all simultaneous placements of all parts. Here $D = 3k$. A point in this space is called forbidden if two or more parts intersect in the corresponding placement. The origin in $E^D$ represents the initial placement of all parts and is not forbidden.

We consider pairwise Minkowski-differences ($C$-obstacles, [8]) for each pair of parts $P_i, P_j$. The $C$-obstacle for $P_i$ and $P_j$ is a three-dimensional region, defined by $C(P_i, P_j) = \{p_j - p_i \mid p_i \in P_i, p_j \in P_j\}$. The $C$-obstacles determine a set of halfspaces $H_1, \ldots, H_5$ in $E^D$, such that the forbidden regions are bounded by the corresponding hyperplanes in the following sense. $H_1, \ldots, H_5$ determine an arrangement $A_D$ of halfspaces in $E^D$, and partition $E^D$ into cells. A cell is a maximal connected region not containing any points on any of the hyperplanes bounding $H_1, \ldots, H_5$. Cells are thus open $D$-dimensional sets. Cells are regular, i.e., for each cell $c$ either all or no points in $c$ are forbidden. The partitioning given by $H_1, \ldots, H_5$ also defines cells of dimension lower than $D$. Lower-dimensional cells are regular as well. (The construction of $H_1, \ldots, H_5$ from the part set will be considered in more detail below.) Notice that $A_D$ contains unbounded cells. An unbounded cell is a cell entirely containing a ray in its interior. A valid removal motion for one or more parts is a sequence of nonforbidden cells connecting the origin to one of the unbounded cells.
A direct way to obtain an answer to the problem stated above is the following: we compute a graph representation of the arrangement $A_D$ [2]. Graph nodes for unbounded cells are labeled. Searching this graph will give an exact solution to the above problem. However, this direct method cannot be used in practice, because the graph representation of $A_D$ has an exponential number of nodes, and it is not possible to store this graph even for a small number of simple polyhedra.

To see whether parts are removable, it suffices to search a single connected component of the arrangement, namely the component containing the origin. It is obvious that searching only a single component will often reduce storage requirements to some extent, but this reduction is insufficient in practice. However, this simple idea provides a basis for a more effective reduction of search effort and storage requirements, described in the next section.

4. Floorgraphs

In the following we assume that all parts $P_1, \ldots, P_k$ are open sets. Thus placements in which two parts are in contact, but do not overlap, are not regarded as intersections. Motions can consist of segments where two or more parts slide along each other. We also assume that all parts are bounded sets.

Constraints for pairs of parts. For two parts $P_i$ and $P_j$, the bounding planes of $C(P_i, P_j)$ define an arrangement of planes $A(P_i, P_j)$ in three-dimensional space. We partition the complement of $C(P_i, P_j)$ into convex regions. Each convex region thus obtained is given as the intersection of halfspaces. Let $H_i^{ij}, \ldots, H_j^{ij}$ be the halfspaces stemming from $P_i$ and $P_j$. These halfspaces are given as inequalities in the three parameters $x, y$ and $z$.

Let $R_1$ and $R_2$ be two adjacent convex regions in the partitioning of the exterior of $C(P_i, P_j)$. Then there is a cell $c$ (of lower dimension) in the three-dimensional arrangement $A(P_i, P_j)$ such that all points in $R_1$ are visible from any point in $c$, and all points in $R_2$ are visible from any point in $c$ (Fig. 3). We call $c$ a passage or door cell.

A floorgraph is defined in the following way. Each convex region in the above partitioning of the exterior of $C(P_i, P_j)$ corresponds to one node in the floorgraph. A door cell $c$ shared by two regions $R_1$ and $R_2$ defines an edge in the floorgraph. To each node/edge we assign a set of defining inequalities and equations.

![Fig. 3 Door cell c. All points in both $R_1$, $R_2$ are visible from points in c.](image-url)
Simultaneous motions of all parts. The $D$-dimensional arrangement $A_D$ is constructed from the halfspace inequalities $H_1, \ldots, H_s$ in the following way. The position of each part $P_i$ is given by three parameters $p_x^{(i)}, p_y^{(i)}, p_z^{(i)}$. These parameters describe the placement of $P_i$ with respect to the initial placement of $P_i$. Thus a point $(p_x^{(1)}, p_y^{(1)}, p_z^{(1)}, \ldots, p_x^{(k)}, p_y^{(k)}, p_z^{(k)})$ describes a simultaneous placement of all parts $P_1, \ldots, P_k$.

Each halfspace inequality $H_i$ contains the three variables $x, y$ and $z$. For an inequality $H$ stemming from a pair $(P_i, P_j)$, we replace the variable $x$ by the expression $p_x^{(j)} - p_x^{(i)}$. Similarly, we substitute $p_y^{(j)} - p_y^{(i)}$ for $y$ and $p_z^{(j)} - p_z^{(i)}$ for $z$. After this substitution, a pair $(P_j, P_i)$ gives the same inequalities as the pair $(P_i, P_j)$. (This follows from the fact that $C(P_j, P_i) = -C(P_i, P_j)$ by definition.) Thus it is sufficient to consider each pair $(P_i, P_j)$, where $i < j$.

The inequalities are $\leq$-inequalities and not $<$-inequalities, because parts are open sets. We regard all inequalities thus obtained as $D$-dimensional inequalities, where at most six variables have nonzero coefficients.

The position of one (arbitrary) part must be fixed, since we are testing for removability. Otherwise all ($D$-dimensional) cells in $A_D$ would be unbounded. We thus set the parameters $p_x^{(k)}, p_y^{(k)}$ and $p_z^{(k)}$ of part $P_k$ to zero.

**Example 1.** A simple assembly illustrates the above definitions (Fig. 4). In this example, we consider the two-dimensional case to simplify the description. The necessary modifications of the definitions are straightforward. The assembly in Fig. 4 contains three parts $P, Q, R$. $R$ is the (fixed) container. $P$ and $Q$ are rectangular. Here $Q$ can only be moved vertically. $P$ can be translated arbitrarily. $C(P, Q)$ is a rectangle (Fig. 5(a)). The exterior

![Fig. 4. Deciding whether a part can be removed from other parts. Part $Q$ is constrained to move vertically only.](image1)

![Fig. 5. (a) Hyperplanes $H_1, \ldots, H_4$ bounding $C(P, Q)$. (b) Convex partitioning of the exterior of $C(P, Q)$ into regions $R_i$. Door cell $r_{12}$. (c) Floorgraph for pair $(P, Q)$.](image2)
Fig. 6. (a) C-obstacle \(C(P, R)\) for parts \(P, R\) in Example 1. (b) Floorgraph for pair \((P, R)\).

Fig. 7. An arrangement of surface patches (a) has fewer cells than the corresponding arrangement of (extended) hyperplanes (b) (c is a door cell).

of \(C(P, Q)\) is a union of halfspaces bounded by hyperplanes \(H_1, \ldots, H_4\). The orientations of \(H_1, \ldots, H_4\) are indicated by arrows in the figure.

By convention, the halfspace above \(H_i\) is denoted by \(H_i^+\). \(H_i^-\) is the halfspace below \(H_i\). The exterior of \(C(P, Q)\) is thus partitioned into four regions \(R_1, \ldots, R_4\), where \(R_1, \ldots, R_4\) are given by:

\[
R_1: H_1^+ \cap H_2^- \quad R_2: H_2^+ \cap H_3^- \quad R_3: H_3^+ \cap H_4^- \quad R_4: H_4^+ \cap H_1^-.
\]

The floorgraph for \(C(P, Q)\) is shown in Fig. 5(c). The partitioning gives four door cells:

\[
r_{12}: H_1^+ \cap H_2, \quad r_{23}: H_2^+ \cap H_3, \quad r_{34}: H_3^+ \cap H_4, \quad r_{41}: H_4^+ \cap H_1.
\]

The floorgraph for \(C(P, R)\) is shown in Fig. 6(b). Since we constrain \(Q\) to move vertically only, the floorgraph for \((Q, R)\) can be represented by a single node \(T_1\) (not shown in the figure).

In the next section it will be shown that floorgraphs allow for searching \(A_D\) in the following way:
• Only a single connected component within $A_D$ is traversed. We do not need to enumerate all free cells in $A_D$.
• $A_D$ is searched as an arrangement of surface patches (Fig. 7), rather than an arrangement of full hyperplanes.
• The search avoids computing the boundaries of free connected components in $A_D$.

5. Searching floorgraphs

In the above construction, one floorgraph corresponds to each pair of parts $P_i, P_j$ where $i < j$.

Let $G_1, \ldots, G_f$ be the floorgraphs thus obtained. Let $n_1, \ldots, n_f$ be nodes in the floorgraphs, where $n_i$ is a node in $G_i$ for each $i$. The tuple $S = (n_1, \ldots, n_f)$ will be called a $D$-node. Thus, a $D$-node contains exactly one node of each floorgraph. For a node $n$ in one floorgraph, let $C(n)$ be the set of defining constraint inequalities. Similarly $C(e)$ gives the defining constraints for an edge $e$ in one floorgraph. If $x \in E^D$ satisfies the constraints $C(n_1), \ldots, C(n_f)$ then $S$ is called a feasible $D$-node.

The $D$-nodes cover free space in $E^D$, i.e., for each nonforbidden point $a$, there is a $D$-node $S = (n_1, \ldots, n_f)$ such that $a$ satisfies $C(n_1), \ldots, C(n_f)$. In particular, we can find a $D$-node for the origin in $E^D$.

We define a successor of a $D$-node in the following way. Let $S = (n_1, \ldots, n_{i-1}, n_i, n_{i+1}, \ldots, n_f)$ be a $D$-node. Then $S' = (n_1, \ldots, n_{i-1}, n_i', n_{i+1}, \ldots, n_f)$ is a successor of $S$, if $n_i'$ is a successor of $n_i$ in the floorgraph $G_i$.

Similarly, we define a successor-edge of a $D$-node:

$S' = (n_1, \ldots, n_{i-1}, e, n_{i+1}, \ldots, n_f)$

is a successor edge of

$S = (n_1, \ldots, n_{i-1}, n_i, n_{i+1}, \ldots, n_f)$

if $e$ is an edge emerging from $n_i$ in the floorgraph $G_i$.

In general, a single $D$-node has several successors. Thus the successor relation on $D$-nodes defines a graph. We search this graph in depth-first order, where all $D$-nodes previously visited are marked and not visited again:

(1) Compute all floorgraphs for pairs $P_i, P_j$ with $i < j$.
(2) Compute a $D$-node $S$ for the origin in $E^D$.
(3) Set $L = \{S\}$.
(4) While $L \neq \emptyset$ empty
   (a) Set $P = \text{first node in } L$ and remove $P$ from $L$. If $P$ is unbounded, return the path from $O$ to $P$ and stop.
      Set $L' = \text{successor edges of } P$.
   (b) For each entry $S$ in $L'$, test whether $S$ is feasible. If so, store a point $x$ satisfying the constraints in $S$, and a pointer from $S$ to $x$. Otherwise, remove $S$ from $L'$.
(c) Replace each successor edge in $L'$ by the corresponding successor node. Remove all previously visited nodes from $L'$. Insert all remaining entries of $L'$ at the front of $L$.

(5) If $L = \emptyset$, return result 'infeasible', and stop.

Notice that each intermediate point $x$ stored with a $D$-node $S$ (except the origin) is a point in a door cell. Thus each point $x$ computed in step 4(b) satisfies the constraints in one floorgraph edge and $f - 1$ floorgraph nodes. In step 4(c), the constraints for this edge are replaced by the constraints for the successor node in the corresponding floorgraph. New nodes generated at the initialization of the list $L'$ in step 4 again have constraints for $f - 1$ floorgraph nodes and one floorgraph edge. Segments of the paths can be contained entirely in cells of dimension less than $D$.

Let $S = (n_1, \ldots, n_f)$ be a $D$-node. In the remainder of this section we will regard $S$ as a point set in $E^D$, i.e., $S$ is identified with the set of points satisfying the constraints in $C(n_1), \ldots, C(n_f)$. Similarly, to simplify the notation, we will associate nodes and edges in floorgraphs with the corresponding point sets in $E^3$.

We must show that the sequence of points thus computed indeed yields a feasible path.

**Lemma 1.** Let $S_1$ be a feasible $D$-node and $S_2$ be a feasible successor edge of $S_1$. Let $x_1$ be a point in $S_1$, and $x_2$ be a point in $S_2$. Then all points on the line segment connecting $x_1$ to $x_2$ in $E^D$ are in free space.

**Proof.** $x_1$ is in a region defined by nodes $n_1, \ldots, n_f$. $x_2$ satisfies the constraints

$$C(n_1) \cup \cdots \cup C(n_{i-1}) \cup C(e) \cup \cdots \cup C(n_{i+1}) \cup \cdots \cup C(n_f),$$

where $e$ is an edge emerging from $n_i$ in the floorgraph $G_i$.

Points satisfying $C(e)$ also satisfy $C(n_i)$. Thus $x_2$ satisfies

$$C(n_1) \cup \cdots \cup C(n_i) \cup \cdots \cup C(n_f),$$

namely the constraints for $x_1$. Each constraint set $C(n_i)$ defines a convex set in $E^D$, and their intersection is convex. Thus the entire line segment joining $x_1$ and $x_2$ is in free space. $\square$

Two $D$-nodes $S$ and $S'$ will be called adjacent, if their intersection is nonempty.

**Lemma 2.** Let $S = (n_1, \ldots, n_f)$ and $S' = (n'_1, \ldots, n'_f)$ be two adjacent $D$-nodes. Then for each $i$ with $n_i \neq n'_i$, the nodes $n_i$ and $n'_i$ are connected by an edge $e$ in the floorgraph $G_i$. Both $C(n_1), \ldots, C(n_{i-1}), C(e), C(n_{i+1}), \ldots, C(n_f)$ and $C(n'_1), \ldots, C(n'_{i-1}), C(e), C(n'_{i+1}), \ldots, C(n'_f)$ are feasible.

**Proof.** We require that the (3D) partitioning of the exterior of each $C$-obstacle be such that two distinct regions have disjoint interiors. Thus all points in the intersection of two floorgraph nodes are points in door cells. Assume, $n_i \neq n'_i$. If $n_i \cap n'_i$ is empty, then $S \cap S'$ is also empty, contradicting the adjacency of $S$ and $S'$. Thus $n_i$ and $n'_i$ are connected by an edge $e$ in $G_i$. Let $x$ be in $S \cap S'$. Then $x$ satisfies both $C(n_1), \ldots, C(n_f)$ and $C(n'_1), \ldots, C(n'_f)$. All points satisfying both $C(n_i)$
and \( C(n_i') \) also satisfy \( C(e) \), due to the above property of the three-dimensional partitioning. Thus \( x \) satisfies \( C(n_1), \ldots, C(n_{i-1}), C(e), C(n_{i+1}), \ldots, C(n_f) \) as well as \( C(n_1'), \ldots, C(n_{i-1}'), C(e), C(n_{i+1}'), \ldots, C(n_f') \). □

The completeness of the above method follows from Lemma 3:

**Lemma 3.** Let \( u \) be a continuous path connecting the origin to an unbounded cell, such that all points on \( u \) are in free space. Then there is a sequence \( T_1, \ldots, T_r \) of \( D \)-nodes, which covers \( u \), such that \( T_{j+1} \) is a successor of \( T_j \) in the search graph.

**Proof.** The \( D \)-nodes cover free space in \( E^D \). Thus \( u \) traverses a sequence \( S_1, \ldots, S_r \) of \( D \)-nodes, covering \( u \). Let \( j < r \). The \( D \)-nodes are closed sets so that, due to the continuity of the path, \( S_j \cap S_{j+1} \) is nonempty, i.e., \( S_j \) and \( S_{j+1} \) are adjacent.

We must show that \( S_{j+1} \) will be expanded, once \( S_j \) has been expanded. Thus it remains to be shown that \( S_{j+1} \) is either a direct successor of \( S_j \) in the search graph, or \( S_j \) and \( S_{j+1} \) are connected by a chain of feasible \( D \)-nodes in this graph.

Let \( S_j = (n_1, \ldots, n_f) \) and \( S_{j+1} = (n_1', \ldots, n_f') \). Assume \( n_1 \neq n_1' \). By Lemma 2, \( n_1 \) and \( n_1' \) are connected by an edge \( e \), and \( C(e), C(n_2), \ldots, C(n_f) \) is feasible. Thus the \( D \)-node \( S'(1) \) defined by \( S'(1) = (n_1', n_2, \ldots, n_f) \) is feasible. \( S'(1) \) is a successor of \( S_j \), since it is connected to \( S_j \) by the (feasible) edge \( e \).

Since \( S_j \) to \( S_{j+1} \) are adjacent, there is a point \( x \) satisfying both \( C(n_1), \ldots, C(n_f) \) and \( C(n_1'), \ldots, C(n_f') \). Thus \( x \) satisfies \( C(n_1'), C(n_2), \ldots, C(n_f), \) i.e., the constraints for \( S'(1) \). But then \( x \) is a common point of \( S'(1) \) and \( S_{j+1} \), so that \( S'(1) \) is adjacent to \( S_{j+1} \). Thus we can apply the same argument to the pair \( n_2, n_2' \). If \( n_2, n_2' \) are distinct, then the \( D \)-node \( S''(2) = (n_1', n_2', n_3, \ldots, n_f) \) is feasible and adjacent to \( S_{j+1} \). Furthermore \( S''(2) \) is a successor of \( S'(1) \). Repeating this replacement step at most \( f \) times, we obtain a sequence of \( D \)-nodes connecting \( S_j \) to \( S_{j+1} \) in the search graph via a chain of feasible edges. □

**Remark.** Let \( n \) and \( n' \) be two nonadjacent nodes in one floorgraph \( G \). The constraint hyperplanes in \( n \) do not partition the region corresponding to \( n' \) and vice versa. Similarly, a door cell between two regions is a patch, not intersecting regions determined by other nodes in \( G \). In this sense, the regions in \( A_D \) corresponding to floorgraph nodes are bounded by surface patches rather than full hyperplanes.

### 6. Implementation

To obtain a practical algorithm, we must implement the test for feasibility in step 4(b) of the above algorithm. This test is implemented as a linear feasibility test [10]. If positive, the test returns a point \( x \) satisfying the given constraints. In the feasibility test, we must ensure that variables may become negative. Here standard methods apply.

Furthermore, we must implement the test for boundedness in step 4(a) to decide whether the current \( D \) node contains a ray \( u \). Let \( S \) be this \( D \)-node. \( u \) may have points in the
boundary of $S$, i.e., on the defining hyperplanes of $S$, but must not cross these hyperplanes. Let

$$H_1: a_1x - d_1 = 0, \ldots, H_r: a_rx - d_r = 0$$

be the equations of the hyperplanes defining $S$. $a_1, \ldots, a_r$ are the normal vectors of the hyperplanes $H_1, \ldots, H_r$. Let $x_0$ be the point in $S$ computed in step 4(b). Assume these hyperplane equations are oriented such that $x_0$ is above (or in) all hyperplanes $H_1, \ldots, H_r$, i.e., $a_1x_0 - d_1 \geq 0, \ldots, a_rx_0 - d_r \geq 0$. To decide whether there is a vector $u$ such that $x_0 + tu$ does not cross one of $H_1, \ldots, H_r$ for any $t > 0$ it suffices to test whether there is a nonzero vector $u$ with $a_1u \geq 0, \ldots, a_ru \geq 0$.

Let $d \neq 0$ and $H: ax - d = 0$ be an arbitrary hyperplane. Then $H': ax + d = 0$ is parallel to $H$. Since $d \neq 0$, both $H$ and $H'$ do not contain the origin. To find $u$ we use two linear feasibility tests. The first test determines whether there is a $u$ with $au - d = 0$, $a_1u \geq 0, \ldots, a_ru \geq 0$. The second test determines whether there is a $u$ with $au + d = 0$, $a_1u \geq 0, \ldots, a_ru \geq 0$. As above, in both tests we must allow for coordinates of $u$ to become negative. $S$ is unbounded if one of the tests succeeds. However, the test may fail to detect unboundedness. This case occurs if the set of all vectors $u$ representing valid removal directions is entirely contained in the hyperplane $ax = 0$. We can ignore this case, for the following reason: all parts are bounded. Thus free space in $E^D$ contains a single unbounded component. The unbounded component contains at least one $D$-node with full dimension $D$.

This simple test assumes bounded parts. A more detailed analysis shows that the latter restriction can be removed. Generally, a single linear range computation followed by a single linear feasibility test is sufficient for establishing whether a given cell is unbounded [14].

7. Analysis

The above algorithm outputs a path—if one exists—as a sequence of line segments in $E^D$. Each segment represents a simultaneous translation of one or more parts, possibly in distinct directions. To find such a path, a tree of cells is expanded.

We will first consider the number of node expansion steps for generating this tree. We decompose the faces of each part into triangles. Let $n$ be the maximum number of triangles in each of the $k$ parts. Then $n$ is a bound for the number of vertices of each part. To compute the pairwise Minkowski-differences, it suffices to compute the Minkowski-differences for pairs of triangles on faces, and we then obtain $O(n^2)$ inequalities for each pair of parts. The number of such pairs is bounded by $k^2$, so that we obtain a total of $O(k^2n^2)$ halfspaces in $A_D$. Each step will reach a new cell in $A_D$, since each step reaches a yet unvisited $D$-node, and crosses at least one hyperplane at the same time.

It is well known that an arrangement of $u$ hyperplanes in $D$ dimensions has at most $O(u^D)$ cells, including cells of lower dimension (see, e.g., [2]). Thus $A_D$ has at most $O((k^2n^2)^D)$ cells. Here $D = 3k - 3$ for polyhedral assemblies and $D = 2k - 2$ in the planar case.
Lemma 4. The number of node expansions in the above algorithm is bounded by $O(k^2 m^D)$, where $m$ is the maximum number of nodes in a connected component of a floorgraph and $k$ is the number of parts.

Proof. Each node in each floorgraph represents a convex (generalized) cylinder in $E^D$. (The cylinders are called generalized cylinders because their bounding surfaces are linear, not curved, so that the cylinders are prisms with convex base.) The set of feasible points for a $D$-node is the intersection of $O(k^2)$ such cylinders. At each step, we proceed from one convex cell of this cylinder arrangement to the next convex cell. Notice that the cylinder arrangement contains non-convex cells as well. At most one unbounded $D$-node will be examined, so we must only account for bounded nodes. There is a vector $v$ in $E^D$ such that each bounded convex cell has exactly one extremal vertex in direction $v$. Each vertex is the intersection of exactly $D$ hyperplanes, except in cases where hyperplanes are not in general position. After a sufficiently small displacement of all hyperplanes, we can reach a placement in which each $v$-extremal vertex is the intersection of exactly $D$ distinct hyperplanes. This displacement of each hyperplane can be made sufficiently small so that none of the convex cells of the cylinder arrangement will vanish. Therefore we can assume that the number of convex cells will not be decreased by the displacement. Of course, the displacement is only done for accounting purposes, not by the program.

Each $v$-extremal vertex of each convex cell is the intersection of, at most, $D$ cylinders. There are $O(k^2 m^D)$ cylinders. We consider the number of subsets with, at most, $D$ elements in this set of size $O(k^2 m)$. This number is bounded by

$$\left(\begin{array}{c}k^2 m \\1\end{array}\right) + \cdots + \left(\begin{array}{c}k^2 m \\D\end{array}\right) \leq \sum_{i=1}^{D} \frac{(k^2 m)^i}{i!} \leq (k^2 m)^D \sum_{i=1}^{D} \frac{1}{i!} \leq (k^2 m)^D e,$$

where $e$ is the Euler constant. The first step in this chain of inequalities is the least obvious, but becomes clear if we observe that for any $a \geq i$

$$\binom{a}{i} = \frac{a!}{i! \cdot (1 \cdots (a-i))} \leq \frac{a!}{i!}.$$

Thus there are $O((k^2 m)^D)$ extremal vertices of convex cells in the cylinder arrangement. Each such convex cell has one $v$-extremal vertex, i.e., there are $O((k^2 m)^D)$ convex cells in the cylinder arrangement.

Since $k$ and $D$ are related, the above bound on the number of node expansions can be reduced: in the following we consider only the case of three-dimensional assemblies.

Lemma 5. There is a constant $D_0$ such that the number of node expansions in the above algorithm is bounded by $O(k^f(D) m^g(D))$, where $f$ and $g$ are functions with $f(D) < D$ and $g(D) < D$ for any $D$ larger than $D_0$.

Proof. From the proof of the previous lemma, the expression

$$\left(\begin{array}{c}k^2 m \\1\end{array}\right) + \cdots + \left(\begin{array}{c}k^2 m \\D\end{array}\right)$$
is a bound for the number of node expansion steps. As noted in Section 4, the pair \((P_i, P_j)\)

of parts will give rise to the same constraints as the pair \((P_j, P_i)\). Therefore we must only

consider ordered pairs of parts \((P_i, P_j)\) where \(i < j\), i.e., the total number of floorgraphs

is bounded by \(k^2/2\). Then the value

\[ q = \binom{k^2m/2}{1} + \ldots + \binom{k^2m/2}{D} \]

equally bounds the number of node expansions. If \(k\) is sufficiently large, we have \(2D \leq k^2m/2\), since \(D = 3k - 3\).

By an elementary property of binomial coefficients, the values in a sequence of the form
\(\binom{n}{0}, \ldots, \binom{n}{j}\) grow monotonically, as long as \(2j \leq n\).

Therefore

\[ \binom{k^2m/2}{i} \leq \binom{k^2m/2}{D} \]

for each \(i = 1, \ldots, D\), if \(k\) is sufficiently large. Then

\[ q \leq D \binom{k^2m/2}{D} \]

and by Stirling's formula,

\[ q \leq D \frac{m^D}{2^D \sqrt{DDD} e^{-D}}. \]

Rearranging gives

\[ q = O\left( \sqrt{D} \frac{m^D}{2^D} k^{2D-D(\log D-1)/\log k} \right). \]

With

\[ f(D) = 2D - D \frac{\log D - 1}{\log k} + \frac{1}{2} \quad \text{and} \quad g(D) = D \left( 1 - \frac{\log 2}{\log m} \right) \]

we can finally write \(q = O(k^{f(D)} m^{g(D)})\).

In the three-dimensional case \(D = 3k - 3\). Therefore \((\log D - 1)/\log k > 1\) for sufficiently large \(k\) and \(D\), so that \(f(D) < D\) and \(g(D) < D\). \(\square\)

From the number of expansion steps, one can obtain a bound for the running time

of the algorithm; let \(LP(s, D)\) be the number of steps required for solving a linear

program with \(s\) constraints and \(D\) variables. Then each node expansion requires at

most \(O((k^2r)LP(k^2r, D))\) steps, where \(r\) is the maximum number of constraints in each

floorgraph. The total running time is thus bounded by \(O((k^2m)^D k^{2r}LP(k^2r, D))\).

In this bound we have not accounted for the precomputation of pairwise floorgraphs. The

analysis of this preprocessing step is straightforward and follows methods in [11] and [15].

**Remark 1.** The parts in Fig. 8 interlock. A series of similar assemblies with growing

number \(k\) of parts is shown in Fig. 9. Here \(m > 1\). However, the total number of examined
Fig. 8. An assembly with interlocking parts.

Fig. 9. Recursive continuation of the example in Fig. 8. Successively replacing the rectangular center part by a copy of the original assembly gives a series of interlocking assemblies (see also Table 2).

Fig. 10. Planar assembly from [16]. The assembly in (b) is a continuation of the example in (a) with more parts.

D-nodes is 1, since all successors of nodes in floorgraphs are unreachable. The total number of arcs in all floorgraphs is polynomial in $k$. Assuming $LP(s, D)$ is polynomial, we obtain a polynomial time bound for performing a complete search. Interestingly, in this case the boundary of the free ($D$-dimensional) component containing the origin has exponential complexity: let $C$ be this connected component. The tolerances allow for placing each inner part into two distinct corners of the next larger part. Selecting one such corner for each of the inner parts corresponds to one $D$-dimensional vertex on the boundary of $C$. This gives rise to $2^{k-1}$ distinct vertices reachable from the origin. (The vertices constructed in this way are only a subset of the total vertex set of $C$.) Thus it is not practical to compute the entire boundary of the connected component $C$. Experiments with
the examples in Fig. 9 illustrate the growth of the computing time with growing number \( k \) of parts (see Table 2 in the next section).

The above remark does not only apply to infeasible assemblies. An example of a feasible assembly is shown in Fig. 10. In this case, the complexity of the cell containing the origin is exponential as well (the boundary of this cell has an exponential number of vertices). However, the number of node expansions is only linear in the number of parts which results in a polynomial overall running time.

8. Experiments

To find practical limitations, the above algorithm was implemented for the planar case on a Unix-workstation HP 700 in C++ based on the LEDA-library [9]. Integer arithmetic was used in the preprocessing steps, and simplex linear programming was used for the feasibility tests.

8.1. Simple examples

Figs. 11(a)–(c) show three simple examples (see also Table 1). In Fig. 11(b), a sliding motion is required for separating the parts. Specifically, the two center parts have total length equal to the width of the container opening. (All coordinates are integers.) Therefore the system must find and traverse a channel of width zero in the set of feasible placements to remove a part in case (b).

Fig. 11. Simple examples.
Table 1
Computing times for the examples in Figs. 11 and 14 (in seconds CPU-time)

<table>
<thead>
<tr>
<th>Case</th>
<th>Fig. 11(a)</th>
<th>Fig. 11(b)</th>
<th>Fig. 11(c)</th>
<th>Fig. 11(d)</th>
<th>Fig. 14(a)</th>
<th>Fig. 14(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>prep</td>
<td>0.3</td>
<td>0.4</td>
<td>1.6</td>
<td>1.3</td>
<td>6.1</td>
<td>6.1</td>
</tr>
<tr>
<td>search</td>
<td>0.1</td>
<td>1.4</td>
<td>1.5</td>
<td>5.2</td>
<td>11.1</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Table 2
Computing times for establishing infeasibility in the example in Fig. 9, with growing number $k$ of parts (in seconds CPU-time)

<table>
<thead>
<tr>
<th>$k$</th>
<th>5</th>
<th>9</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>prep</td>
<td>0.6</td>
<td>2.8</td>
<td>11.6</td>
</tr>
<tr>
<td>search</td>
<td>0.6</td>
<td>3.2</td>
<td>19.3</td>
</tr>
<tr>
<td>$t_{\text{search}}/k^5$</td>
<td>0.00019</td>
<td>0.000054</td>
<td>0.000013</td>
</tr>
</tbody>
</table>

The dimension $D$ in Fig. 11(d) is larger than in the previous examples (a)–(c), i.e., here $D = 12$. All of the examples in Fig. 11 admit a separating motion. Table 1 summarizes the computing times for the examples in Fig. 11.

We next consider the assembly in Fig. 8. We replace the central rectangle in this assembly by an identical, but smaller copy of the original assembly. The new assembly has 9 parts (Fig. 9). Repeating this replacement step several times, we obtain a series of interlocking assemblies, with growing number of parts. Table 2 shows computing times obtained for this case. When moving from left to right in the table, the number of moving parts is doubled at each step, since we can assume one of the parts is fixed. The corresponding growth of search time is shown in the table.

Fig. 10 shows a feasible assembly, for which the total running time is polynomial in the number of parts. The total computing times for the cases (a) and (b) shown in the figure are 6.7 s and 24.8 s, respectively.

8.2. More difficult examples

In Fig. 12(a) the blocks in the center can assume many distinct relative placements, all of which are reachable. To partition the assembly, each block must be placed into the corresponding cavity, before the two outer parts can be separated with a two-step motion.

The example in Fig. 12(b) shows that the problem considered here is NP-hard [11]. The ‘PARTITION’ problem for integers must be solved to find a removal motion. Specifically, the locking bolt can only be removed if the enclosed (horizontal) stack of rectangular parts can be partitioned into two stacks of equal length.

The assembly in Fig. 12(c) is infeasible and a complete search of all reachable $D$-nodes must be performed to establish infeasibility. In this case, 18600 $D$-nodes are examined for a complete search. The computing time is substantially higher than in the case of the simple examples above (see Table 3).

The problem of finding a sequence of translational motions for partitioning a given assembly has exponential lower bounds. This is illustrated in Fig. 12(d): let $k'$ be the
number of \( u \)-shaped parts in the figure. If \( k' \) is sufficiently large, we must follow a variant of the rules for the \textit{Towers of Hanoi} in order to remove a part [3]. The number of distinct translational motions (i.e., motions with distinct directions or distinct part sets) for removing a part is exponential in \( k' \). The computing times for the example in Fig. 12(d) are shown in the last column of Table 3.

Feasible problems can be solved more rapidly if the \( D \)-nodes are not expanded in an uninformed depth-first order. A very simple modification is to use a heuristic search where the node with largest distance to the origin (among new nodes) is always expanded first. The second line of Table 3 shows computing times obtained with this modification. Notice
Fig. 13. Search in narrow channels.

that heuristic search only modifies the ordering in which nodes are expanded. This does not affect completeness and still allows for detecting infeasibility. However, for infeasible cases this modification will not allow improvements (Fig. 12(c) and third column of Table 3).

8.3. Path planning with random planners

Random planning methods have been described for a variety of motion planning applications, including motion planning for jointed mechanisms and cars [7]. Solutions for practical problems with many degrees of freedom can often be found very rapidly with random planners.

The running time of random planners depends upon the width of the smallest passage the planner must find. This is illustrated in Fig. 13. To find a path from start to goal, the planning system must find several narrow channels in the set of reachable placements and traverse these channels in an appropriate order.

Random planners use internal parameters specifying the distributions of random variables, such as mean and standard deviation of the step-length. The choice of such internal parameters can be difficult. For narrow passages the step-length should be small, while small step-lengths substantially increase the running time for finding longer paths. Typically, a random planning scheme will examine intermediate configurations (generated at random) until a predefined running time limit has been exceeded, or a path has been found. In general, it is difficult to establish that two intermediate configurations cannot be connected at all, and that the search should be continued in other feasible regions. If parts must slide along each other, the probability for finding narrow passages in high-dimensional space with random search is very small. The above experiments suggest that such narrow passages can be examined more easily with the described exact method.

For further experiments, a random planner based on the principles in [7] was implemented for the mentioned environment. In Fig. 14(a) parts must move into distinct directions simultaneously. For the random planning system, the goal position shown in Fig. 14(c) was given in advance. With the initial choice of the step-length parameters, no valid placements other than the given initial placement were found. After reduction of the mean step-length, the random planner was able to find new valid placements, but not the solution. By further reducing the mean step-length, more intermediate placements were found, but the random planning system was practically no longer able to advance. In
Figs. 14(a), (b) the running times for the exact method were 17.2 s and 11.1 s, respectively. Notice that the example in Fig. 14(b) is infeasible. The computing time in case (b) is smaller than in case (a), since the set of reachable D-nodes is more restricted than in case (a).

8.4. Further examples

The problem of m-handed assembly planning is stated as follows: decide whether parts in a given assembly can be separated by a single translation, where parts or subassemblies may move into distinct translational directions. Here m denotes the number of subassemblies moving into distinct directions. The assembly in Fig. 11(a) can be partitioned with such a single-step motion.

After small modifications, the program can be used to decide whether a given assembly admits an m-handed assembly sequence. For planar parts it can be shown that this computation can always be performed in polynomial time if no pair of parts is separated by a line in the initial configuration [14].

Interestingly, the assembly in Fig. 15 allows for three-handed assembly which is established by the program in less than 0.1 s. The program also establishes that no two-handed assembly motion exists in this case. For the human it is not obvious that the parts in this figure can be separated by a single three-handed translation. The intuitive approach would be to move one pair of adjacent parts (vertically or horizontally) until a contact with one of the other parts occurs. Then one would select a different pair and remove it with a translation perpendicular to the first, which is possible from the intermediate placement. Finding a single translation (i.e., a three-handed motion) in this case is left to the reader.
9. Conclusions

In special cases, the proposed graph representation allows for a complete search in polynomial time, even if the boundary of the free component has exponential complexity.

A direct way to include heuristics into the search process (while retaining completeness) is to modify the ordering in which D-nodes are expanded. Experiments suggest that appropriate heuristics can reduce search times. However, heuristics can only give improvements for feasible assemblies.

In the experiments, computing times for establishing infeasibility are substantially shorter, if the set of reachable D-nodes is small. This appears to be the case for tight assemblies in particular and suggests that the above approach is more suitable for applications in assembly planning than for general motion planning.

Assembly motions requiring many changes of directions or velocities increase assembly costs, and are often impractical due to fixturing and stability problems. From a practical point of view, finding one assembly motion—if there is such a motion—is the main objective of planning methods. In this context it seems useful to fix an upper bound for the number of allowed changes in direction/velocity of parts. The number of D-nodes along a path bounds the number of direction/velocity changes during the motions. However, the direct computation of paths with minimum number of direction/velocity changes has not yet been explored in the context of the above methods.

Acknowledgements

The authors thank Florian Bieberbach, Akos Czopf, Leo Joskowicz and Randy Wilson for comments and discussions.

References


