T U M

I N S T I T U T F Ü R I N F O R M A T I K

Efficiently Testing for Unboundedness and m-handed Assembly

Fabian Schwarzer, Florian Bieberbach, Leo Joskowicz, Achim Schweikard

TUM-I9750
Dezember 97

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-12-I9750-50/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Efficiently Testing for Unboundedness and m-handed Assembly

Fabian Schwarzer[1], Florian Bieberbach[2]
Leo Joskowicz[3], Achim Schweikard[1]

[1]Institut für Informatik
[2]Lehrstuhl für Allgemeine und Industrielle Betriebswirtschaftslehre
Technische Universität München
80290 München, Germany

[3] Institute of Computer Science
The Hebrew University of Jerusalem
91904 Jerusalem, Israel

December 15, 1997

Abstract

We address the problem of efficiently determining whether the intersection of a given set of d-dimensional halfspaces is unbounded. It is shown that detecting unboundedness can be reduced to a single linear range computation followed by a single linear feasibility test. In contrast, detecting unboundedness is at least as hard as linear feasibility testing and maximization. Our analysis suggests that algorithms for establishing linear unboundedness can be used as a basis of simple and practical algorithms in motion planning, insertability analysis and assembly planning. We show that m-handed assembly planning can be reduced to testing for unboundedness. A valid motion sequence can be computed in polynomial time, if the parts are not already separated in their initital placement. No polynomial algorithms were previously known for this problem. We present experimental results obtained with an implementation of our algorithms.

Keywords: Assembly Planning, Linear Programming, Unboundedness.

1 Introduction

We address the problem of determining whether a given d-dimensional point set, described as an intersection of n halfspaces in d dimensions is unbounded. Practical methods for performing this test have applications in geometric reasoning [3], insertability analysis [4] and assembly sequence planning [6, 7]. We discuss the role of testing for unboundedness in assembly planning, and describe experimental results obtained with an implementation of our methods. It is shown that detecting unboundedness can be reduced to a single linear range computation followed by a single linear feasibility test. In contrast, detecting unboundedness is at least as hard as linear feasibility testing and maximization. We will show that m-handed assembly planning can be reduced to testing for unboundedness. A valid motion sequence can be computed in polynomial time, if the parts are not already separated in their initital placement.

1.1 Problem statement

We consider a set of hyperplanes in d dimensions, defined by the equations

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

These hyperplanes define an arrangement in d dimensions, i.e. they partition $d-$space into convex regions, called *cells*. Let $\mathbf{p} = (p_1, \ldots, p_d)$ be a point which is not on any of the hyperplanes. Then $\mathbf{p}$ determines a cell c in this arrangement, i.e., c is the cell containing $\mathbf{p}$. c can be represented as an intersection of halfspaces containing $\mathbf{p}$.

Each such halfspace is given as an inequality of the form $\mathbf{a}_i\mathbf{x} \leq b_i$ or of the form $\mathbf{a}_i\mathbf{x} \geq b_i$, where $\mathbf{a_i}$ is the normal vector of one of the above hyperplanes, i.e., $\mathbf{a}_i = (a_{i1}, \ldots, a_{id})$.

A cell is called unbounded, if it contains a ray in its interior. The problem of testing for linear unboundedness is stated as follows:

Given hyperplanes in d dimensions and a point $\mathbf{p} = (p_1, \ldots, p_d)$, decide whether there is a ray $r = \{\mathbf{p} + t\mathbf{u} \mid t \geq 0\}$ starting at $\mathbf{p}$, such that r does not cross any of the hyperplanes.

Notice that we do not require the coordinates u_i of the direction vector $\mathbf{u}$ be positive.

1.2 Unboundedness in linear programming

Unboundedness is usually regarded as an error condition in linear programming. However, unboundedness in linear programming is *directional*. I.e., a linear maximization in a *given* direction will fail, if the feasible region is unbounded in this given direction.

Based on simplex linear programming, we can obtain a direct solution to the above problem. Consider an arbitrary linear base $\mathbf{s}_1, \ldots, \mathbf{s}_d$ of $d-$space. Perform a maximization for each of $+\mathbf{s}_i$ and $-\mathbf{s}_i$, subject to the halfspace inequalities defining the given cell. This cell is unbounded, if one of these maximizations reports unboundedness.

It is easy to see that this test is complete for the undirected case, i.e., the result 'unbounded' will be returned, if the given cell is indeed unbounded in *any* direction. This simple test requires $2d$ linear maximization steps. Can we detect unboundedness with a faster method?

We will show that testing for unboundedness is at least as hard as linear feasibility testing and maximization. It will then be shown that testing for unboundedness can be performed by a single linear range computation followed by a single feasibility test.

1.3 Assembly Sequences

Our test is applied to the problem of computing assembly sequences: Given a set of objects in the plane or in space, decide whether these objects can be assembled (or, *dis*assembled) without collision. This problem has been studied in a variety of contributions [1, 2, 6, 8]. Fig. 1 shows an example.

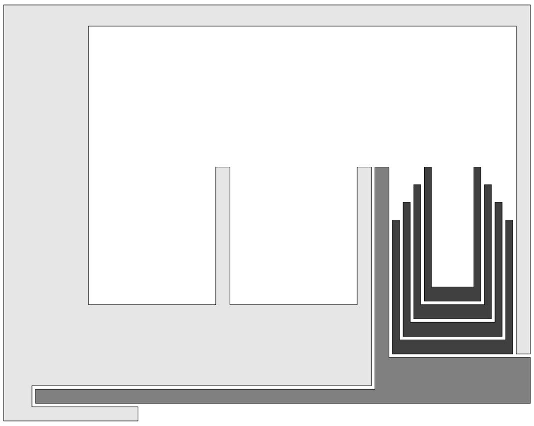

Figure 1: Geometric representation of *The Towers of Hanoi* [2]

The problem of finding a sequence of *translational* motions for partitioning a given assembly has *exponential lower bounds* [2]. This is illustrated in fig. 1: For a sufficiently large number k of u-shaped parts the rules of the *Towers of Hanoi* problem must be obeyed, in order to remove one of the parts. The number of distinct translational motions (i.e. motions with distinct directions or distinct parts sets) for removing a part is exponential in k.

By further restricting the class of allowed motions, polynomial time algorithms can be devised. Specifically, the problem of $single-handed\ assembly$ is to decide whether there is a translational direction which partitions a given assembly into two (arbitrary) subsets. Interestingly, despite the fact that we must compute a removable *subset*, this decision can be made in polynomial time [6].

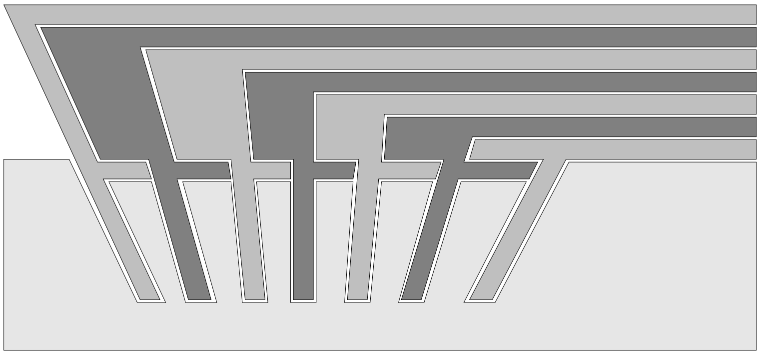

Figure 2: An assembly requiring m hands to disassemble

In between single-handed assembly and general translational assembly is the problem of m *− handed assembly*: Decide whether given parts can be separated by a simultaneous translational motion, where each part may assume a different direction and velocity of motion, but may be moved at most once (fig. 2).

We will show that m-handed assembly motions can be computed in polynomial time, as long as no pair of parts is already separated in the initial placement. Indeed, after (polynomial-time) preprocessing, a single test for unboundedness is sufficient.

2 Algorithm for Determining Unboundedness

We consider the following algorithm:

Input: Hyperplanes $H_1, \ldots, H_n$ with equations $H_i : \mathbf{a}_i\mathbf{x} = b_i$ and a point $\mathbf{p}$.

Output: Decision whether the cell containing $\mathbf{p}$ is *bounded* or *unbounded.*

1. Determine, whether the homogeneous system of equations $\mathbf{a}_i\mathbf{x} = 0$ $(1 \leq i \leq n)$ has a non-trivial solution $\mathbf{u} \neq 0$. If so, return *unbounded*

and exit.

2. Select the orientations of the normal vectors $\mathbf{a}_i$ for H_i such that $\mathbf{a}_i\mathbf{p} \geq b_i$ for each i.

3. Set $\mathbf{a}_\Sigma := \sum_{i=1}^{n} \mathbf{a}_i$. If $\mathbf{a}_\Sigma = 0$ return *bounded* and exit. Define a constraint set C by setting $C =$

$$\begin{aligned} \mathbf{a}_i \cdot \mathbf{x} &\geq 0, \quad 1 \leq i \leq n \\ \mathbf{a}_\Sigma \cdot \mathbf{x} &= 1. \end{aligned}$$

4. Test whether the constraint set C is feasible. If so, return *unbounded*, otherwise return *bounded* and exit.

Notice that the coordinates x_i of $\mathbf{x}$ in the feasibility test (steps 3, 4) are *not* restricted to be positive.

The following observations show that the above algorithm is complete. We will assume that the orientation of each hyperplane is chosen such that $\mathbf{a}_i\mathbf{p} \geq b_i$ for $1 \leq i \leq n$. Let S denote the intersection of the corresponding halfspaces, in this orientation. Thus S is the cell containing $\mathbf{p}$. By S_0 we denote the intersection of the corresponding *homogeneous* halfspaces, i.e. S_0 is the intersection of the halfspaces $\mathbf{a}_i\mathbf{x} \geq 0$.

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

Lemma 2 *If $dim\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle \neq d$ then S_0 is unbounded.*

The proofs of lemma 1 and lemma 2 directly follow from elementary properties of linear vector spaces, see e.g. [9]. To state lemma 3, we recall the definition of the constraint set C in step 3 of the above algorithm.

Lemma 3 *Let $dim\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle = d$. 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 $dim\langle \mathbf{a}_1, \ldots, \mathbf{a}_n, \mathbf{u} \rangle$ is strictly greater than d. Thus $\mathbf{a}_\Sigma \mathbf{u} \neq 0$. The same argument shows $\mathbf{a}_\Sigma \neq \mathbf{0}$ if the space spanned by $\mathbf{a}_1, \ldots, \mathbf{a}_n$ 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. $\square$

Computing an unbounded direction. The next section describes the application of the above algorithm in assembly planning. It is useful to return a ray r pointing into an unbounded direction, if there is such a ray. This is done by modifying steps 1 and 4 as follows:

Step $1'$: Determine, whether the homogeneous system of equations $\mathbf{a}_i \mathbf{x} = 0$ ($1 \leq i \leq n$) has a non-trivial solution $\mathbf{u} \neq 0$. If so, return *unbounded*, the vector $\mathbf{u}$, and exit.

Step $4'$: Test whether the constraint set C is feasible. If so, return *unbounded* and a point $\mathbf{u}$ satisfying C, otherwise return *bounded* and exit.

3 m-handed Assembly Sequences

Given a set of polygonal (or polyhedral) objects, decide whether one or more of these objects can be removed without collision.

Two parts P, Q will be called *separated*, if P is entirely contained in a halfspace not intersected by Q.

Let $P_1, \ldots, P_k$ be an assembly of polygonal parts. We assume that in the given initial placement, no pair of parts is separated in the above sense. I.e., for $i, j \leq k$, there is no halfplane containing all of P_i but none of P_j.

When determining whether parts are removable, we can fix the placement of one (arbitrary) part, I.e., we can assume that the part P_k will never be moved. This is possible for the following reason: if the given parts can be separated at all, then there is also a separating motion which will leave one of the parts fixed. Note that the choice of the fixed part is arbitrary.

Since we only consider translational motion, the placement of each movable part P_i is given by a vector $\mathbf{p}^{(i)} = (p_x^{(i)}, p_y^{(i)})$. Thus $\mathbf{p}^{(i)}$ describes the location of part P_i after a translational displacement. A *simultaneous* placement of all parts is then represented by a vector $\mathbf{u} = (p_x^{(1)}, p_y^{(1)}, \ldots, p_x^{(k)}, p_y^{(k)})$. $\mathbf{u}$ is a vector in E^{2k}. The origin in this space describes the initial placement of all parts. Similarly, the vector $(1, 0, \ldots, 0)$ is the placement obtained by translating the first part along the positive $x-$axis by one length unit. Notice that this placement may not be free of overlap between parts.

For each pair of parts P_i, P_j, where $i, j < k$, we consider the set of directions separating P_i from P_j. This set of directions is represented by a set of rays starting at the origin (in the plane). Since we assume that parts are not separated in the initial placement, the directions separating P_i from P_j form a convex sector (or half-cone) in the plane. This sector is an intersection of two halfplanes.

Now both parts P_i and P_j are movable. Therefore we must not only represent directions for separating P_i from P_j but also the directions separating P_j from P_i. Let

$$\mathbf{a} \cdot \mathbf{x} \geq 0$$

$$\mathbf{a}' \cdot \mathbf{x} \geq 0$$

be the directions separating P_i from P_j. Thus $\mathbf{ax} \geq 0$ and $\mathbf{a}'\mathbf{x} \geq 0$ are two halfplanes. Any point $\mathbf{u}$ with $\mathbf{au} \geq 0$ and $\mathbf{a}'\mathbf{u} \geq 0$ is a valid direction for separating P_i from P_j. I.e., P_i can be translated along $\mathbf{u}$ to infinity without intersecting P_j.

It is easy to see that the rays in the opposite sector, i.e., the sector

$$\mathbf{a} \cdot \mathbf{x} \leq 0$$

$$\mathbf{a}' \cdot \mathbf{x} \leq 0$$

represent translations separating P_j from P_i.

To simultaneously represent both sectors, we consider the 4-dimensional set defined by the constraints

$$\mathbf{a}(\mathbf{p}^{(i)} - \mathbf{p}^{(j)}) \geq 0$$

$$\mathbf{a}'(\mathbf{p}^{(i)} - \mathbf{p}^{(j)}) \geq 0.$$

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

Finally, the set of directions separating one of the movable parts P_i (where $i < k$) from the *fixed* part (here part P_k) is represented in the same way. But the since P_k is fixed, we have $(p_x^{(k)}, p_y^{(k)}) = (0, 0)$, so that this set of directions is given by

$$\mathbf{a}(\mathbf{p}^{(i)}) \geq 0$$

$$\mathbf{a}'(\mathbf{p}^{(i)}) \geq 0.$$

Taking all constraints together (for all pairs of parts), we obtain a convex cell in $d = 2k - 2$ dimensions, which contains the origin. The given parts can be separated by an $m-$handed translation if and only if this cell is unbounded.

For two-dimensional assemblies we have imposed the restriction that no pair of parts is separated in the initial configuration. A requirement which is sufficient in both the two-dimensional and the three-dimensional case is the following: The set of directions separating parts must be convex for each pair of parts.

4 General Translational Assembly Planning

The assembly in fig 1 cannot be partitioned by a single translation. In this case, a valid motion for removing parts consists of a series of translational motions in different directions, each of which may involve more than one part. However, it can be shown that general translational assembly planning can be performed with the same basic methods as above. Indeed, we can compute a $D-$dimenisonal (static) arrangement of hyperplanes representing simultaneous placements of all objects. Valid and forbidden placements correspond to cells in this $D-$arrangement [7]. A simple scheme for deciding whether or not the assembly can be parititioned by arbitrary translational motions proceeds as follows: We traverse the non-forbidden cells of the arrangement, starting with the cell containing origin. If all reachable cells are bounded, then no translational assembly sequence exists. Otherwise, a path connecting the origin to an unbounded cell represents a valid assembly sequence. In this case, the test for unboundedness is called in each cell. A fast test is useful, if the number of cells thus traversed is large.

5 Computing Time

The test for unboundedness in section 2 relies on linear feasibility testing. In contrast, the direct method mentioned in the introduction uses linear maximization. To allow for comparison, note that the test in section 2 can be modified in such a way that maximization is used rather than feasibility testing. In step 4, we must simply maximize in direction $\mathbf{a}_\Sigma$, subject to the constraints $\mathbf{a}_i \cdot \mathbf{x} \geq 0$. Step 1 of the test for unboundedness is a linear range computation. This test can be performed in $O(d^2n)$ steps (for n constraints and d variables, [9]). Known time bounds for linear maximization dominate the latter time bound. Thus the above test allows for a reduction of time bounds by a factor of d when compared to the direct method in the introduction. Specifically, based on Karmarkar's method, linear maximization takes at most $O(Ld^{3.5})$ steps, where L is the accumulated length of the input coefficients [10]. If the length of each individual coefficient is fixed and constant, (i.e., each coefficient has length less than a fixed value l), then $L = O(nd)$, so that the described test for unboundedness takes at most $O(nd^{4.5})$ steps, versus $O(nd^{5.5})$ for the direct method in the introduction.

Is this test the fastest possible test? Our test relies on linear feasibility testing in step 4. Is there a test which does not require feasibility testing? The following construction shows that the problem of testing for unboundedness is at least as hard as linear feasibility testing.

Consider the linear feasibility problem (LF):

(LF)

$$\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1d}x_d \geq \ b_1 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nd}x_d \geq b_n. \end{array}$$

We can transform this problem to a (homogeneous) test for unboundedness, by setting

(U)

$$\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1d}x_d - b_1 z \geq 0 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nd}x_d - b_n z \geq 0 \\ \\ z \geq 0. \end{array}$$

Indeed the cell defined by the system (U) is unbounded if and only if the original problem (LF) is feasible.

Notice that the transformation from (LF) to (U) has increased the dimension d of the problem by one, since a new variable z has been added. Furthermore, the transformation increases the number of constraints by one. When solving (U), we increase the number n of constraints by one in step 4. 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. This suggests that there is no faster test, provided an optimal feasibility test is available, and the time required for step 4 dominates the time for the remaining steps.

Example: A simple example shows that the linear range computation in step 1 of the above test for unboundedness is not redundant: Let $\mathbf{a}_1 = (-1, 0)$, $\mathbf{a}_2 = (2, 0)$, $b_1 = 1$, $b_2 = 2$ and $\mathbf{p} = (0, 0)$. The cell defined by $\mathbf{a}_1 \cdot \mathbf{x} \geq b_1$ and $\mathbf{a}_2 \cdot \mathbf{x} \leq b_2$ is unbounded, but $\mathbf{a}_\Sigma = (1, 0)$, so that C is infeasible.

Let P and Q be two polygons or polyhedra. The computation of the set of directions for separating P from Q is straightforward, and can be carried out based on a computation of the Minkowski-difference of P and Q. Specifically, in the three-dimensional case (P and Q are polyhedra), an algorithm

number of parts, k	4	8	16
running time, t	0.08	0.30	1.12

Table 1: Running times for the assembly in fig. 2

for computing all directions for separating objects is given in [6]. This algorithm is optimal, and requires at most $O(n^4)$ steps, where n is the total number of vertices of P and Q. Notice that the latter algorithm is optimal for the general case of two arbitrary polyhedra. Clearly, faster methods for computing the directions separating two polyhedra can be constructed, if this set of directions is convex.

6 Implementation

The algorithm for m-handed assembly in section 3 was implemented in C in an assembly planning system. Our implementation is based on routines in the LEDA-library [5], and uses the above algorithm for determining unboundedness. The program was run under UNIX on an HP 700 computer. Integer arithmetic was used for step 1 of the test for unboundedness, and simplex linear programming was used for feasibility testing. Table 6 shows computing times (in seconds) for the assembly in fig. 2. In the table, k represents the total number of parts for assemblies as in fig. 2. To increase k, assemblies with growing number of $L-$shaped parts were used as input. Thus, as we move from left to right in the table, k doubles at each step. In the experiments we observe a four-fold increase of computing time at each step.

Fig. 3 shows a second example. Our program establishes that the assembly shown allows for m-handed assembly. The computed motions are shown in fig. 4. Here $m = 4$, i.e. four of the five parts must be translated simultaneously, with distinct velocities and directions. The computing time was 0.1 seconds in the above environment. The velocites are indicated by the lengths of the arrows shown. By considering a series of appropriate subassemblies, the program also determined that no m-handed assembly sequence with $m < 4$ exists.

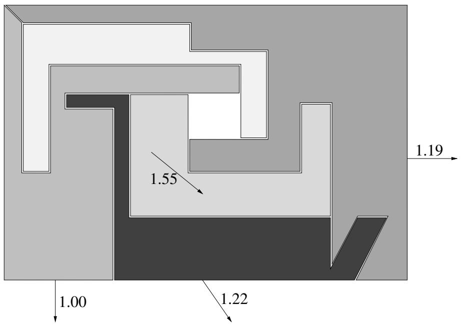

Figure 3: Parts can be separated by the motion indicated by arrows. Lenghts of arrows (values shown with arrows) represent relative velocities.

7 Conclusion

After (polynomial-time) preprocessing, m-handed assembly can be reduced to a single test for unboundedness, under the restriction that no pair of parts be separated in the initial assembly. Thus the assembly must be sufficiently tight. A fairly simple example establishes the exponential lower bound on the problem of computing general translational assembly sequences. It is not known whether the exponential lower bounds can be specialized to $m-$handed assembly without this restriction.

References

[1] P. K. Agarwal, M. de Berg, D. Halperin, M. Sharir. Efficient Generation of k-Directional Assembly Sequences. *Proc. 7th ACM-SIAM Symp. Discr. Algorithms (SODA)*, 122-131, 1996.

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

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

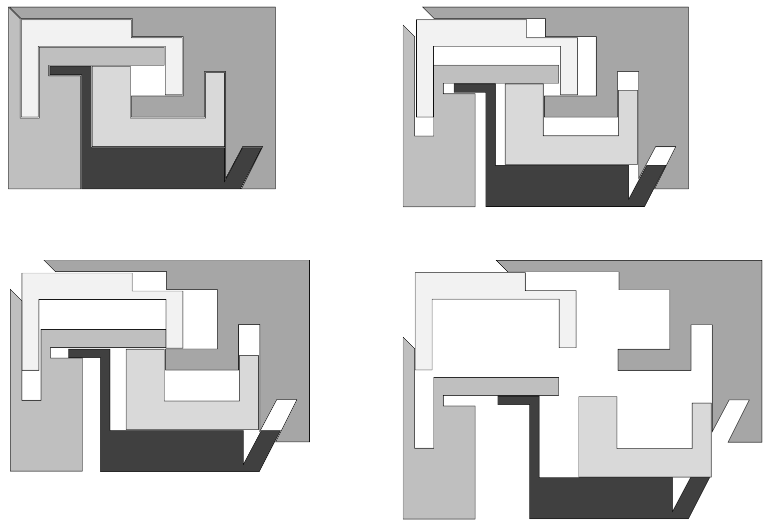

Figure 4: Motions computed for the assembly in fig. 3.

[4] L. Joskowicz, R. H. Taylor. Interference-free insertion of a solid body into a cavity: an algorithm and a medical application. *International Journal of Robotics Research*, Vol. 15 No. 3, 1996.

[5] K. Mehlhorn, S. Naeher. LEDA: A Platform for Combinatorial and Geometric Computing. *Max-Planck-Institut fuer Informatik, Saarbruecken*, 1995.

[6] A. Schweikard and R. H. Wilson. Assembly Sequences for Polyhedra. *Algorithmica*, 13(6): 539–552, June 1995.

[7] A. Schweikard and F. Schwarzer. General Translational Assembly Planning. Proc. IEEE Conference on Robotics and Automation, 162 - 169, 1997.

[8] J. Snoeyink, J. Stolfi. Objects that Cannot be Taken Apart with Two Hands. *Discrete and Computational Geometry*, 12: 367-384, 1994.

[9] J. Stoer, C. Witzgall. *Convexity and Optimization in Finite Dimensions*, I, Berlin, New York: Springer Verlag, 1970.

[10] Wright, M. H. Interior Methods for Constrained Optimization. in: Acta Numerica, Iserles, A. (ed.), New York: Cambridge University Press, 1992.