Information Processing Letters, 55, 1995, 159-165

Partitioning a Planar Assembly Into Two Connected Parts Is NP-Complete

Lydia E. Kavraki* Mihail N. Kolountzakis†

Abstract

Consider the following decision problem. Given a collection of non-overlapping (but possibly touching) polygons in the plane, is there a proper *connected* subcollection of it that can be separated from its complement moving as a rigid body, without disturbing the other parts of the collection, and such that the complement is also connected? We show that this decision problem is NP-complete. This had been known to be true without the connectedness requirement, and also *with* this requirement but in three-dimensional space.

Keywords: computational complexity, computational geometry, assembly planning, motion planning.

1 Introduction

This paper studies a variation of the *assembly partitioning* problem in the plane: given a collection of non-overlapping (but possibly touching) polygons, decide if there is a proper subcollection of it that can be removed as a rigid body without colliding with or disturbing the other parts of the collection, and such that both the subcollection and its complement form connected sets.

The partitioning problem arises in assembly planning, where a sequence of (possibly simultaneous) assembly motions are sought to bring separated parts into their relative goal positions in an assembly. It has been proved [8] that in its most general form, assembly planning is PSPACE-hard. In practice, however, most real assemblies can be constructed with *monotone two-handed* assembly plans [5]. Such plans consist of a sequence of operations, where each operation merges two

*Robotics Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305. E-mail: kavraki@cs.stanford.edu.

†School of Mathematics, Institute for Advanced Study, Olden Lane, Princeton, NJ 08540. E-mail: kolount@math.ias.edu.

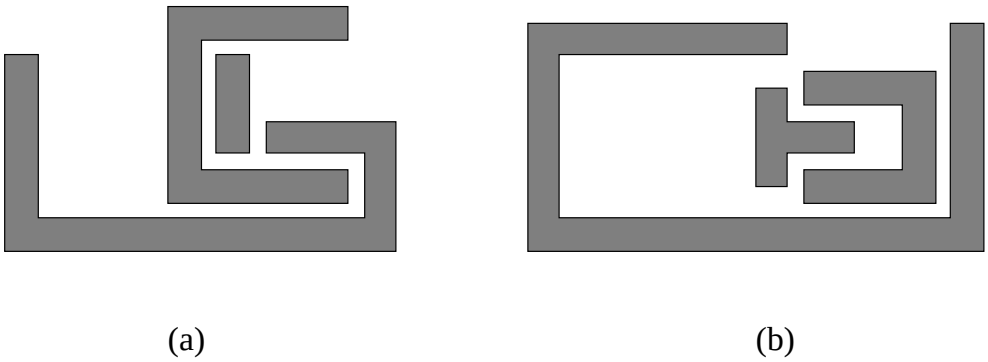

Figure 1: Assembly (a) admits a monotone two-handed plan, while (b) does not.

rigid subassemblies to make a larger one that stays rigid for the rest of the plan. For instance, the assembly in Figure 1(a) can be constructed with a monotone two-handed plan, while the assembly in Figure 1(b) cannot.

Since assembly is the reverse of disassembly, a monotone two-handed assembly plan can be found by *partitioning* the assembly – removing a rigid subassembly – and then partitioning each of the two subassemblies, etc., and finally reversing the motions. Thus partitioning is the key to assembly planning in this case.

Recent work has presented polynomial-time partitioning algorithms in special cases. Arkin, Connelly and Mitchell [1] give an algorithm to partition assemblies of polygons if the separating motions are limited to single infinite translations. Wilson [14] presented algorithms to partition assemblies of polyhedra, where the separating motions are either infinite translations or infinitesimal rigid motions (the latter identifying a superset of the removable subassemblies for general separating motions). Infinitesimal rigid motions are also treated in [3]. Snoeyink and Stolfi [12] present an assembly of convex polyhedra that cannot be partitioned. Other related geometric separation problems are studied in [4, 9, 10, 13].

In [6, 7, 16] it was shown that the partitioning problem for polygons in the plane is NP-complete. Let us remark here that the corresponding decision problem in space was known before [15] to be NP-complete. The construction given in [6, 7, 16] extended to some interesting variants of the problem, including the case where all motions are restricted to translations, and the case where all vertices of the polygons belong to an $N \times N$ grid (N is then the size of the problem) and all translations are by one unit up, down, left or right (no rotations). The reduction for hardness was from 3-Satisfiability.

The question was posed in [6, 7, 16] whether the problem remains NP-complete if we restrict the possible subassemblies to being such that they are connected and their complement is connected. A collection of polygons is considered connected if the subset of the plane that the polygons jointly occupy is connected. A polygon is considered to be closed, so that our definition means that the interiors of

the polygons are not allowed to intersect and connectedness comes as a result of touching boundaries only. The motivation behind this constraint is that in practice connected assemblies are easier to grasp and manipulate.

In this paper we give a construction which is quite different from that in [6, 7, 16] and which answers this question in the affirmative. Let us also point out that this construction constitutes an alternative proof for all results given in [6, 7, 16].

2 The Reduction

Let us state the decision problem formally.

Planar Partitioning into Connected Subassemblies (PPCS). Given a set A of non-overlapping polygons in the plane, decide if there is a proper subset S of A that can be separated from $A\backslash S$ by a collision-free rigid motion of S, and such that both S and $A\backslash S$ are connected.

We show that PPCS is a NP-complete problem. That it is a problem in the complexity class NP is easy: a non-deterministic algorithm "guesses" S, checks whether S and $A\backslash S$ are both connected, and, if yes, checks whether S can be moved away to infinity without overlapping with $A\backslash S$. Both the connectedness test and the collision-free removal test can be done in polynomial time [11].

The proof that PPCS is a NP-hard problem is by reduction from the NP-complete problem of *3-Satisfiability* (3-SAT). An instance of 3-SAT consists of a set of *clauses* $C = \{c_1, \ldots, c_m\}$ on a set of boolean variables $U = \{u_1, \ldots, u_n\}$, where each clause is a disjunction of three terms. A term is either a variable u_j or its negation $\overline{u_j}$. For example clause c_1 may be the disjunction $u_2 \vee \overline{u_{15}} \vee \overline{u_{20}}$. The problem is to decide whether there is an assignment $U \longmapsto \{\text{TRUE}, \text{FALSE}\}$ that makes all the clauses true. It is well known [2] that 3-SAT is a NP-complete problem.

For any instance of 3-SAT we construct in polynomial time in m an instance of PPCS (that is, a specific collection A of polygons) for which the answer is YES if and only if the answer to the 3-SAT instance is YES. In other words, we construct a collection A of polygons which can be partitioned into S and $A\backslash S$ (both connected) such that S can be removed if and only if the given instance of 3-SAT is satisfiable.

First let us say a few words about how the figures that we present should be read. The only kind of polygons that we use are either rectangular boxes or collections of straight line segments of varying thickness. The assumption is that any two lines are rigidly connected to each other (that is they are parts of the same polygon) if and only if they touch each other in the drawing and they have the same thickness. In Figure 2 for example this means that the straight line segments da and ab are rigidly connected while ab and bc are not. Also, any two lines that appear to touch each other in the drawing, do so.

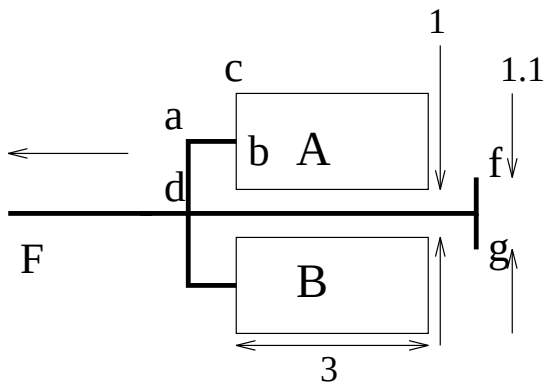

Figure 2: A basic selection mechanism.

2.1 A Basic Selection Mechanism

Let us start with Figure 2. In it we show an assembly of 3 polygons: the rectangles A and B and the fork-like collection of straight line segments which are all part of the same polygon, which we call F. Assume now that F is forced to move to the left. It is easy to see that this is not possible unless A or B or both follow it. Indeed, assume that the collection does not move as a whole and, say, B stays in its place. Then A has to move along with F as a rigid body according to the requirements of the problem. The motion is done as follows: first they both move forward until the rear T-shaped end of F reaches rectangle B, then they both move vertically upwards for distance equal to 0.1 so that the lower end of the T-shaped end is above B, then they move forward until the T-shaped end reaches the front of rectangle B, then they move vertically downwards by 0.1 and finally they continue to move forward, having passed the obstacle B forever.

A crucial property of this assembly is that the height (distance of points f and g) of the T-shaped end of F is, say, 1.1 while the distance of the two rectangles A and B is 1. This means that, assuming again that B stays in its place, A and F only have to move upwards by 0.1 in order to be separated from B. Notice also that during this motion F and A remain connected, since they touch at point b.

2.2 Enforcing an Assignment

Let us now focus our attention to a specific clause, say $c = \overline{u_j} \vee \overline{u_k} \vee u_l$. There are exactly 7 assignments to the variables u_j, u_k, u_l that make this clause true, the only one which makes it false being $u_j = \text{TRUE}, u_k = \text{TRUE}, u_l = \text{FALSE}$. And let us again focus our attention to one of those 7 assignments, say $u_j = \text{TRUE}, u_k = \text{FALSE}, u_l = \text{TRUE}$.

For this assignment we construct a mechanism, shown in Figure 3, which *en-*

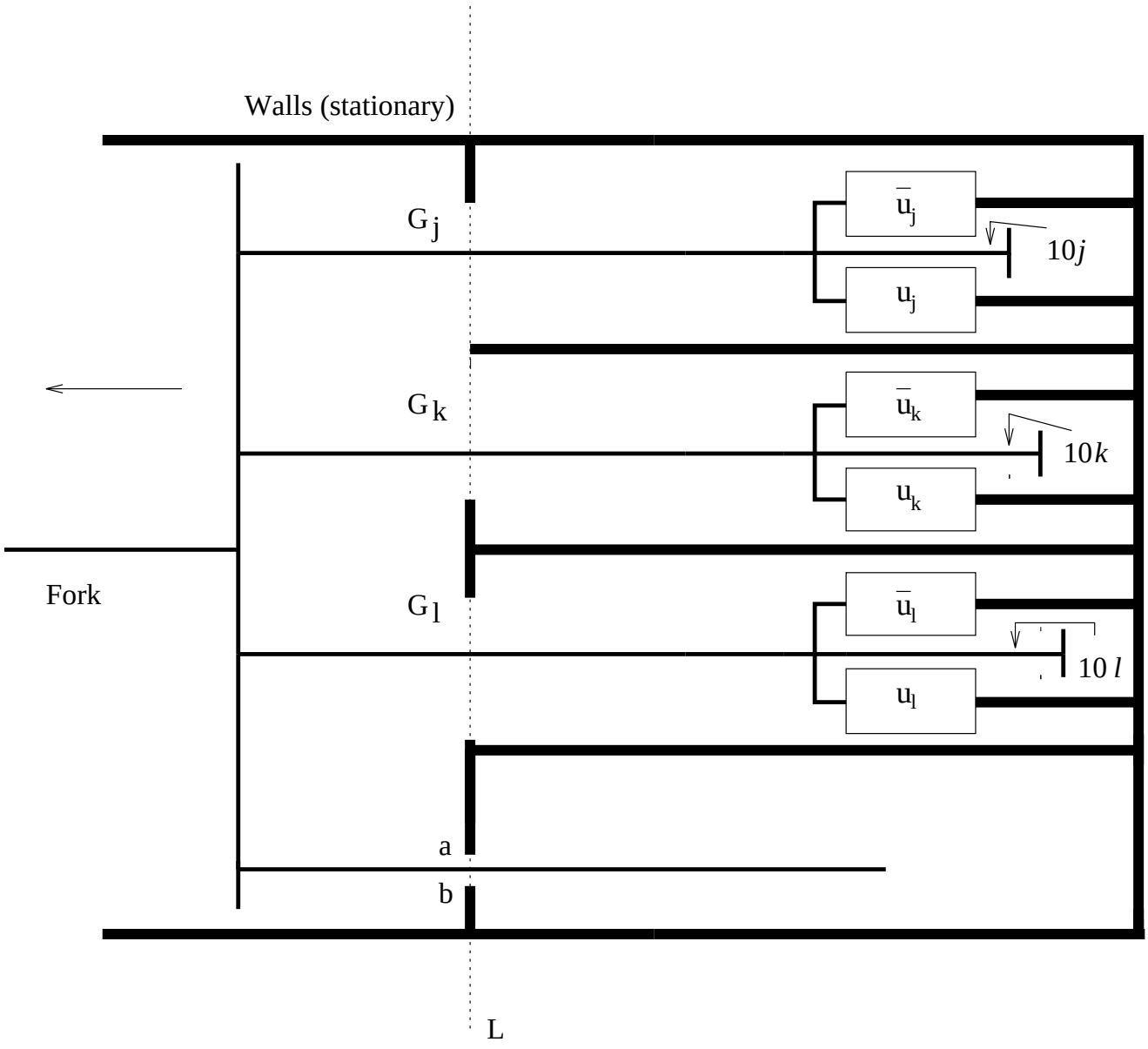

Figure 3: Enforcing the assignment u_j = TRUE, u_k = FALSE, u_l = TRUE (not drawn to scale).

forces it. Let us make clear what we mean by that. There are exactly 8 polygons in that figure: the 6 rectangles labelled $u_j, u_k, u_l, \overline{u_j}, \overline{u_k}, \overline{u_l}$, the *Walls* polygon (the thick lines shown) and the *Fork* polygon (thin lines).

We assume, as we may, that the walls remain stationary. (Indeed, given any admissible sequence of rigid motions that separates a subcollection of our collection from its complement, one can transform it to an admissible sequence of rigid motions that leave any desired polygon fixed, throughout the motion.) Then clearly the only way that any part can move out is when the fork moves to the left, as shown by the horizontal arrow. The fork, moving to the left, has to drag along at least one of the rectangles $u_j, \overline{u_j}$, one of the rectangles $u_k, \overline{u_k}$ and one of the rectangles $u_l, \overline{u_l}$, as explained when discussing Figure 2 (Section 2.1).

The fork ends in 3 horizontal T shaped parts, all of them as described in Section 2.1. We assume that the distance of the end of each of these parts from the corresponding pair of rectangles is $10j$, $10k$ and $10l$ respectively. This means that (assume $j < k < l$), as the fork is moving to the left, it will first need to raise or lower itself by 0.1 when the T-shaped end reaches the u_j rectangles. But at that moment the other two T-shaped ends are still away from their respective rectangles and thus we need not be concerned with them. Thus, the fact that the three T-shaped ends initially have very different distances (equal to $10j, 10k$ and $10l$) from the corresponding rectangles allows us to move the fork up or down *independently* for each of the variables j, k, l.

Let us also point out that, since the distance between any two rectangles corresponding to the same variable is 1, the fork can move up or down by 0.1 (this is how much it needs to move for the T-shaped end to bypass a rectangle) without any collisions. Notice that those rectangles that are staying behind stay connected to the walls. Those rectangles that are moving with the fork keep touching it all the time at their leftmost edge, since according to our requirements all the parts that move, move as if rigidly connected to each other.

Along the vertical line L we have erected several barriers that restrict the motion of the fork. The lowest gap (between points a and b) is exactly 0.2 wide, to allow the fork to move up or down no more than it needs for the T-shaped ends to bypass the rectangles. The gaps G_j, G_k and G_l are such that the fork can only move out of them if the rectangles labelled $\overline{u_j}, u_k$ and $\overline{u_l}$ do *not* follow the fork. But since at least one rectangle of each variable has to move, it follows that exactly the rectangles labelled $u_j, \overline{u_k}$ and u_l move with the fork. The gaps are such that these rectangles can pass through them.

We have demonstrated that the mechanism of Figure 3 can only be separated if the fork along with the rectangles labelled $u_j, \overline{u_k}, u_l$ are the only parts that move. Furthermore, the fork never moves up or down by more than 0.1 from its original position. We say that this mechanism *enforces* the assignment $u_j = \text{TRUE}, u_k = \text{FALSE}$, and $u_l = \text{TRUE}$.

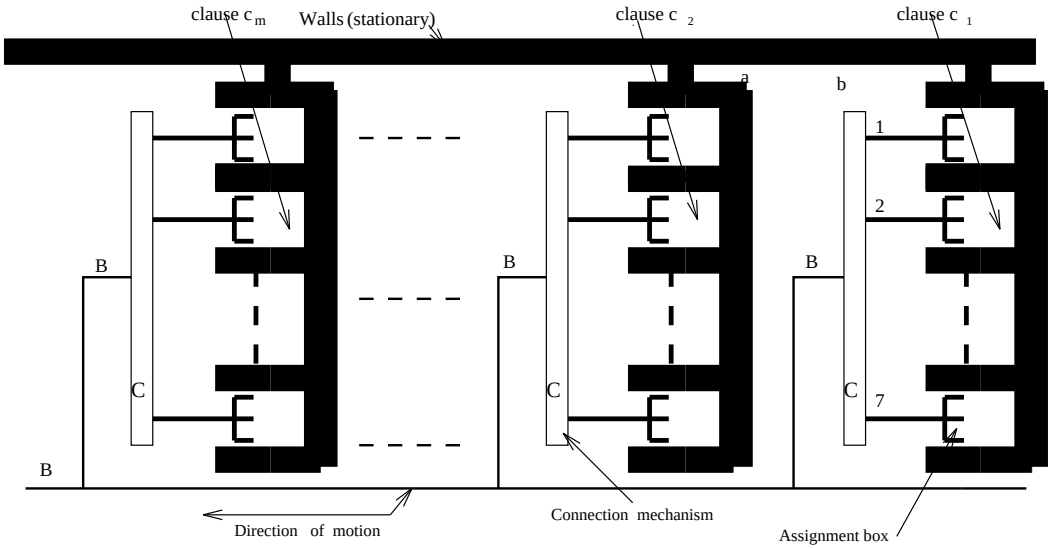

Figure 4: Forcing all clauses to be true (not drawn to scale).

2.3 Satisfying the Clauses

Next we stack together on top of each other (as shown in Figure 4) the 7 mechanisms that correspond to the 7 different assignments that make the clause in question, say again $c = \overline{u_j} \vee \overline{u_k} \vee u_l$, true. The forks of all 7 assignments are joined (not rigidly) via a, yet to be described, connection mechanism C to a horizontal bar B. The bar B is shown at the bottom of Figure 4. All the fat lines (walls, assignment boxes and their backbones but not the forks) are rigidly connected to each other.

The functional description of the connection mechanism C is the following. It allows the bar B to move to the left only if *at least one* of the forks that enter each mechanism C from the right, moves along with it. But, because the forks that enter C at each assignment box correspond to pairwise incompatible assignments (of a certain clause), *at most one* of these forks is allowed to move. Thus, for each clause of $c_1, \ldots, c_m$ there is exactly one valid assignment chosen at its corresponding stack of seven boxes.

For each fixed j we have many rectangle pairs labelled $u_j, \overline{u_j}$ in our collection. We have to ensure that in every occurence of such a rectangle pair it is always the same of the two rectangles that moves. That is, either all u_j rectangles move, or all $\overline{u_j}$ rectangles move, so that the local assignments in each box are restrictions of a global assignment to the variables $u_1, \ldots, u_n$.

This is automatic since at each occurence of a rectangle pair for the variable u_j the length from the end of the T-shaped fork to the rectangles is always the same and equal to $10j$. Consequently, the forks have to move up or down *at the same*

time for all such rectangles, resulting in the same choice of u_j (if they move down) or $\overline{u_j}$ (if they move up), for each occurence of a $u_j, \overline{u_j}$ rectangle pair. (Remember that all moving parts perform the same motion so that all forks move together up or down.)

Thus, the bar B can move out of the rest of the assembly, dragging several other polygons with it, if and only if there is an assignment to the variables that satisfies all the clauses. This is done, as described in Section 2.1, by moving to the left and sometimes also moving up or down by 0.1 until all the forks have moved past the connection mechanisms C, and then moving vertically downwards to be completely separated from the rest of the assembly, which remains connected. The distance between points a and b in Figure 4 is large enough to accommodate the whole length of the fork.

2.4 The Connection Mechanism C

We still have to describe how the connection mechanism C works. It is shown in Figure 5. It is a straightforward generalization of the basic selection mechanism shown in Figure 2. That mechanism allowed the choice of one or both of the rectangles A and B to move along with the fork. We repeat that selection mechanism in a binary tree of depth 3. Note that each of the rectangles shown is connected rigidly to the straight line segment to its right, as the line thicknesses indicate. If part B, which is shown to the left, moves in the direction of the horizontal arrow, then at least one of the 8 "inputs" to the right will move with it. We have enclosed (Figure 5) in a closed dashed line all the parts that move if we choose input 3 to move along with B. Input 8 of this mechanism is never used; it can for example be joined rigidly to the walls so that it cannot move.

In each node of the binary tree we have a copy of the basic selection mechanism of Figure 2. It is then easy to prove inductively that if the "root" B moves to the left then at least one of the leaves moves with it. The moving parts are all connected and the remaining parts are all connected to the remaining leaves, which in turn are connected to the stationary walls. These remaining leaves correspond to the assignments that cannot move out of their boxes. The dimensions here must be such that the vertical dimension of each of the rectangles shown in Figure 5 is at least as large as the vertical dimension of each of the assignment boxes of Figure 4. That assignment box must also be placed at the same height as the rectangle.

Again the distances from the rectangles to the T-shaped ends should be different (by at least 10) for each node of the tree (this is not shown in Figure 5), so that the selection for each node happens at a different time, and thus all combinations of selections can happen. Since there are 7 nodes for each such tree and m clauses we need $7m$ distinct distances, say $d_1, \ldots, d_{7m}$. These numbers must also be different from the lengths that we used in the boxes, that is they must be different from

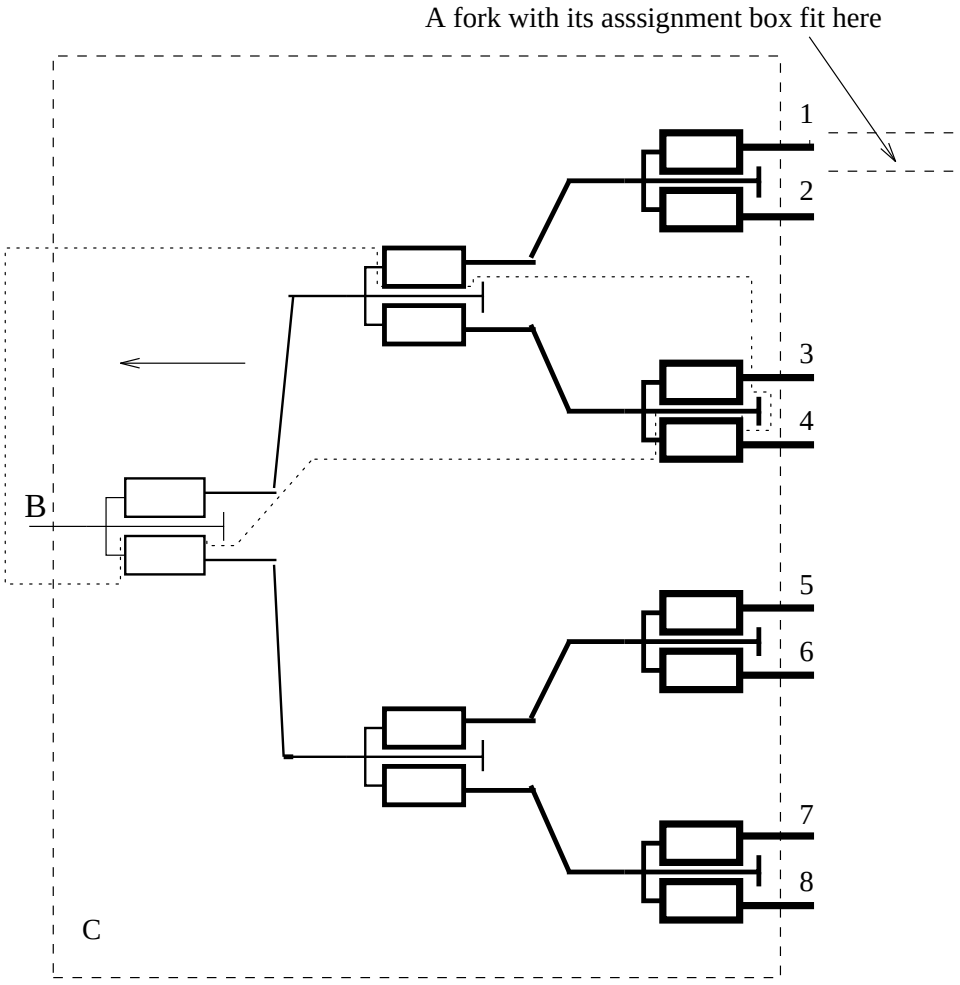

Figure 5: A 3-level binary tree for selecting one or more of the 8 inputs (not drawn to scale).

$10, 20, \ldots, 10n$. We can choose $d_j = 10n + 100j$, for $j = 1, \ldots, 7m$.

To repeat, more generally, there are several copies of the basic selection mechanism of Figure 2 throughout the construct that we have described. To go through one of them the corresponding fork and, because of the required rigidity of motion, the whole moving subassembly has to move up or down by 0.1 (the geometry of these selection mechanisms is the same everywhere). There are some groups of those selection mechanism for which we want to ensure that it is always the same rectangle that moves, for instance the selection mechanisms corresponding to a specific variable u_j, but apart from those we want to ensure independence of choice. For instance we want that, for two different variables u_j and u_k, all selections are possible. As another example of that, we want any two basic selection mechanisms, used by connection mechanisms C, to be able to move independently up or down. To ensure the above it suffices, as we have explained, that any two forks we want to move independently have T-shaped-end-lengths differing by at least 10 (the length of a rectangle in the basic selection mechanism being 3) and any two forks that we want to bind to each other (to move together up or down) have the same T-shaped-end-length.

We have constructed a collection of polygons which can be partitioned into two connected subcollections if and only if the given instance of 3-SAT is satisfiable. Since this construction can obviously be carried out in polynomial time, we have shown the decision problem PPCS to be NP-complete.

Acknowledgement: We would like to thank one of the referees for pointing out an error in Figure 5. L. Kavraki was partially supported by NSF grant IRI-9306544-001 and M. Kolountzakis was supported by NSF grant DMS 9304580.

References

[1] E.M. Arkin, R. Connelly, J.S.B. Mitchell, "On monotone paths among obstacles with applications to planning assemblies", *Proc. of the 5th ACM Symposium on Computational Geometry*, pp. 334-343, 1989.

[2] M.R. Garey, D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[3] L.J. Guibas, D. Halperin, H. Hirukawa, J,C, Latombe and R. Wilson, "A Simple and Efficient Procedure for Polyhedral Assembly Partitioning under Infinitesimal Motions", *Proc. IEEE Internat. Conf. Robotics and Automation*, Nagoya, Japan, 1995, to appear.

[4] L. Guibas, F. Yao, "On translating a set of rectangles", *Computational Geometry*, F. P. Preparata (ed.), series: Advances in Computing Research, Vol. 1, JAI Press, London, pp. 61-67, 1983.

[5] L.S. Homem de Mello, S. Lee editors, *Computer-Aided Mechanical Assembly Planning*, Kluwer Academic Publishers, Boston, 1991.

[6] L. Kavraki, J.-C. Latombe, "Complexity of partitioning a planar assembly", TR STAN-CS-93-1467, Dept. of Computer Science, Stanford Univ., 1993.

[7] L. Kavraki, J.-C. Latombe and R. H. Wilson, "On the complexity of assembly partitioning", *Inf. Proc. Letters* 48 (1993), 229-235.

[8] B.K. Natarajan, "On planning assemblies", *Proc. of the 4th ACM Symposium on Computational Geometry*, pp. 299-308, 1988.

[9] D. Nussbaum, J. R. Sack, "Disasembling Two-Dimensional Composite Parts Via Translations", *Inter. J. of Computational Geometry*, 3, pp. 71-84, 1993.

[10] R. Pollack, M. Sharir, S. Sifrony, "Separating two polygons by a sequence of translations", *Discrete and Computational Geometry*, 3, pp. 123-136, 1988.

[11] J.T. Schwartz and M. Sharir, "On the piano movers' problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers", *Communications on Pure and Applied Mathematics*, 36, pp. 345-398, 1983.

[12] J. Snoeyink and J. Stolfi, "Objects that cannot be taken apart with two hands", *Proc. of the 9th ACM Symp. on Computational Geometry*, pp. 247-256, 1993.

[13] G.T. Toussaint, "Movable separability of sets", *Computational Geometry*, Elsevier Science Publishers, North Holland, 1985.

[14] R. H. Wilson, *On Geometric Assembly Planning*, PhD Thesis, Stanford University, March 1992, Stanford Technical Report STAN-CS-92-1416.

[15] R. H. Wilson, J.C. Latombe, T. Lozano-Pérez, "On the complexity of partitioning an assembly", TR STAN-CS-92-1458, Dept. of Comp. Science, Stanford Univ., 1992.

[16] R. H. Wilson, L. Kavraki, T. Lozano-Perez and J. C. Latombe, "Two-handed assembly sequencing", to appear in *Inter. J. Robot. Res*, Also available as T.R. STAN-CS-93-1478, Stanford, CA, 1993.