Contemporary Mathematics

On the Complexity of the Linkage Reconfiguration Problem*

Helmut Alt, Christian Knauer, Günter Rote, and Sue Whitesides

Abstract. We consider the problem of reconfiguring a linkage of rigid straight segments from a given start to a given target position with a continuous nonintersecting motion. The problem is nontrivial even for trees in two dimensions since it is known that not all configurations can be reconfigured to a straight position. We show that deciding reconfigurability for trees in two dimensions and for chains in three dimensions is PSPACE-complete.

Section 1. Introduction

A *linkage* in d-space is a crossing-free straight line embedding of a graph in $\mathbb{R}^d$ where the edges are considered as rigid bars and the vertices are considered as hinges. A *reconfiguration* is a continuous motion of the vertices that preserves the lengths of the edges and never causes edges to collide.

We investigate several complexity issues concerning the reconfiguration of simple (non-crossing) polygonal chains of fixed-length segments in 3D and of simple trees of fixed-length segments in 2D. To be more precise, we consider the complexity of variants of the following

Linkage reconfiguration problem: Given two linkages S and T in d-space, can we reconfigure S into T?

In 2D, every simple polygonal chain of fixed length segments can be continuously moved to a straight configuration [**12, 7**]. Nevertheless, the general question is nontrivial since it is known that simple trees of fixed length segments cannot always be moved to configurations that are essentially flat, [**7**], and that there are polygonal chains in 3D that cannot be moved to a straightened configuration [**5, 2**].

In Section 2 we observe that the linkage reconfiguration problem is decidable in polynomial space for any fixed dimension d. In Section 3 we complement these upper bounds by showing that the reconfiguration problem for trees in 2D is PSPACE-hard. The technique we use is based on work of Joseph and Plantinga [**9**]. Finally,

1991 *Mathematics Subject Classification.* Primary 68U05, 68Q17; Secondary 68W40.

Key words and phrases. Computational Geometry, Linkage reconfiguration, PSPACE-completeness.

*This work was presented at the Proc. 19th Annu. ACM Symp. on Computational Geometry in San Diego, USA, c.f. [**1**].

Sue Whitesides' research was supported by NSERC and FCAR.

in Section 4 we sketch an extension of the PSPACE-hardness result to the reconfiguration problem for chains moving in 3D.

Section 2. Deciding the reconfiguration problem in polynomial space

Any embedding of a linkage consisting of n links in d-dimensional space can be described by giving the dn coordinates of the vertices. Thus, any embedding can be represented as a point in $\mathbb{R}^{dn}$. Furthermore, the condition that no two links may intersect can be written as a formula in the first order theory of the reals, i.e., a formula that is built by joining atomic formulas which involve the variables that represent the coordinates of the vertices, using addition and multiplication as operators, and $\leq$ as a predicate, with the usual boolean connectives. Furthermore, for each link, we can add to this formula the condition that its two endpoints lie a fixed distance (the length of the link) apart.

The region where this formula is true is a semialgebraic subset $\mathcal{F}$ of $\mathbb{R}^{dn}$ of the *feasible* embeddings of the linkage. Assume that two linkages S and T are given. Obviously linkage S can be reconfigured into linkage T precisely when there exists a path within $\mathcal{F}$ starting at S and ending at T.

The application of this technique to motion planning problems was investigated first by Schwartz and Sharir [**11**] and shown to be decidable using the techniques of Collins [**6**], which were improved later by Canny [**3**]. Afterwards, Renegar [**10**] showed that problems of this kind are in PSPACE (also claimed by Canny [**4**]). So we can conclude

THEOREM 1. *The reconfiguration problem for arbitrary linkages in d-dimensional space, $d \in \mathbb{N}$, is in PSPACE.*

Section 3. Hardness of the reconfiguration problem for planar trees

We show that it is PSPACE-hard to decide whether a given planar linkage S can be reconfigured to a prescribed target configuration T in the plane. In fact we show that the problem is already PSPACE-hard when S and T are trees.

THEOREM 2. *The reconfiguration problem for trees in the plane is PSPACE-hard.*

Our approach uses ideas from Joseph and Plantinga's PSPACE-hardness proof for the reachability problem for 2D linkages in the presence of polygonal obstacles [**9**].

We show directly that any language $L \in$ PSPACE can be reduced to the planar tree reconfiguration problem in polynomial time. Assume that L is recognized by the deterministic, polynomially space-bounded single-tape Turing machine M. Let P denote the polynomial that bounds the space complexity of M. We may assume without loss of generality that M has a unique accepting configuration in which it has erased the tape and moved the tape head to the first tape cell, and that there is no transition that leaves the accepting state.

We will first describe a reduction that transforms an input x to M in time polynomial in the length $|x|$ of x to a pair of *forests* S and T such that

$$M \text{ accepts } x \text{ if, and only if, } S \text{ can be reconfigured to } T.$$

The two forests will actually consist of only two trees each, and we will later describe an easy modification of the reduction that ensures that S and T are trees.

A *configuration* of the Turing machine M describes the current *state* and *tape head position* of the machine, along with the *content of the work tape*. In what follows, we assume that a configuration of M occurring during a computation with input x can be encoded with n bits. Since M is polynomially space-bounded, it follows that $n = \Theta(P(|x|))$. Our goal is to describe a reduction that is polynomial in n.

The reduction works by first building two trees. The first one, C, is a chain and is used to encode the configurations of M. The second tree, E, is an almost rigid structure that is built from elementary gadgets which are designed according to the transitions of M. The chain C will be confined to lie inside E; by moving chain C within the environment E, we will be able to simulate the computation of M. The forests S and T will be two different placements of C inside E. The first represents the initial configuration of M on x and the latter, the unique accepting configuration of M.

3.1. Encoding a configuration. The chain C is depicted in Figure 1. It consists of a sequence of n smaller chains $C_1, \ldots, C_n$, called *switches*, each of which is made up of 3 segments and used to encode a single bit of the configuration of M. The construction below will ensure that the chain is forced to move inside a narrow environment, so that the switches can only point towards the end of the chain as is indicated in Figure 1. When C_i is on one side of the chain C, the corresponding bit of the encoded configuration is true, and when it is on the other side, it is false.

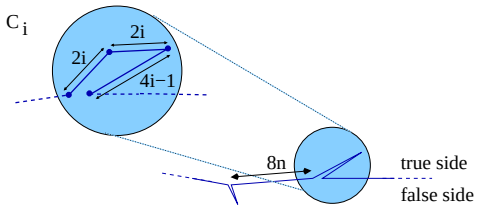

FIGURE 1. The chain encoding the configuration

The chains C_i are scaled copies of C_1, where the size increases with i. They are linked together (in consecutive order) by long intermediate segments. In addition, a long segment is placed before the first and after the last switch, respectively.

The i-th switch is depicted in the upper left part of Figure 1. The intermediate segments have length $8n$. The total length of the chain is therefore $8n + \sum_{i=1}^{n} 8n + 8i - 1 = 12n^2 + 11n$ units. If the chain is configured such that it is completely flat with all switches pointing forward, it has length $L = 8n + \sum_{i=1}^{n} 8n + 1 = 8n^2 + 9n$ units.

3.2. Simulating the transitions. Ideally, we would like the environment E to be a rigid structure. However, since it is a linkage and bars can rotate around hinges freely (if unconstrained), we design a nearly rigid structure instead. The idea is simple and is illustrated in Figure 2: in order to lock a hinge, we surround it with small chains and attach them to the hinge in such a way that the resulting tree is very nearly rigid.

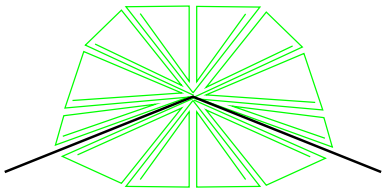

FIGURE 2. Fixing a hinge

The environment tree E simulates a simple polygon that confines the ways in which C can move. It is built by combining several elementary gadgets. For each possible transition t of M and each possible tape position p, we build an elementary gadget $G_{t,p}$. Each gadget consists of two parts. The first part is a tunnel-like structure. It is called the *checker* $E_{t,p}$, and it can be passed by C if the configuration c that is encoded allows the application of the transition t at position p of the head. We call this property the *precondition* of the transition-position pair (t, p).

The second part of the gadget is a dead-end tunnel-like structure $S_{t,p}$ called the *setter*. Once C has entered this tunnel, the parts of the chain that correspond to the bits that are changed by the corresponding transition are provided enough space to move to the other side of the chain. The gadgets are combined in such a way that the setters provide the only possibilities for the chain to reconfigure. The chain can be pulled out of the gadget through the checker again if the *postcondition* of the pair (t, p) is satisfied, i.e., if the bits have been changed in such a way that the configuration encoded is the one obtained from c by applying transition t at position p of the tape.

The fact that the checker can be passed by satisfying either the pre- or the postcondition makes it possible to go through a gadget without changing the configuration c or even to simulate a step of M backwards. We will deal with this problem later. We will now describe the gadgets and their parts in more detail and explain how they are assembled into E.

3.2.1. *Checkers.* When configurations are encoded by boolean variables $z_1, z_2, \ldots, z_n$, the pre- and postcondition of a transition-position pair can be written as a *conjunction* of these variables or their negations: a tape position can be encoded with $O(\log P(|x|)) = O(\log n)$ bits, and a state and tape symbol can be encoded with $O(1)$ bits. So all in all, the pre- and postconditions of a transition-position pair (t, p) can be written as *conjunctions* $P_{t,p} = x_1 \wedge \cdots \wedge x_r$ and $Q_{t,p} = y_1 \wedge \cdots \wedge y_r$ of the variables z_i or their negations with $r = O(\log n)$.

2-clause checkers. Our aim is to build a gadget that checks whether the chain C encodes a configuration that fulfills the pre- or the postcondition of a transition-position pair (t, p). Note that we can rewrite

$$\begin{aligned} P_{t,p} \vee Q_{t,p} &= (x_1 \wedge \cdots \wedge x_r) \vee (y_1 \wedge \cdots \wedge y_r) \\ &= (x_1 \vee y_1) \wedge \cdots \wedge (x_i \vee y_j) \wedge \cdots \wedge (x_r \vee y_r) \end{aligned}$$

as a 2-CNF formula with r^2 2-clauses.

First we describe a gadget that can check a 2-clause $k = x \vee y$. This is a tunnel-like structure E_k with the property that C can enter E_k at one end and leave it at the other end if, and only if, it encodes a configuration that satisfies the clause

k. Inside E_k the parts of the chain C that do not correspond to the literals x and y cannot change their status (i.e., from true to false or vice versa), and when the chain leaves E_k it encodes the same configuration as when it entered. The gadget is depicted in Figure 3. Speaking with reference to Figures 4 and 3, the entrance

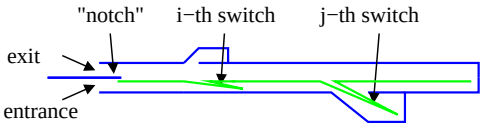

FIGURE 3. Checking the 2-clause $(z_i \vee \bar{z}_j)$

is below the exit on the extreme left of the diagram. The chain C can always enter the gadget at the entrance. In order to be able to leave via the exit, it has to be shortened a little bit, so that the leftmost segment of C can be moved above the small notch that separates the entrance from the exit. After that, the chain can be pulled out of the gadget via the exit. For each of the two literals that are to be checked in E_k, there is a small *box* on the top (if the literal is positive) or at the bottom (if the literal is negative) of the tunnel. The position of the structure corresponds to the position of the switch of C that encodes the variable that is being checked. When the corresponding variable has the right value, the piece of the chain that encodes the variable can enter the box, which allows the chain to be shortened. The process is depicted in Figure 4.

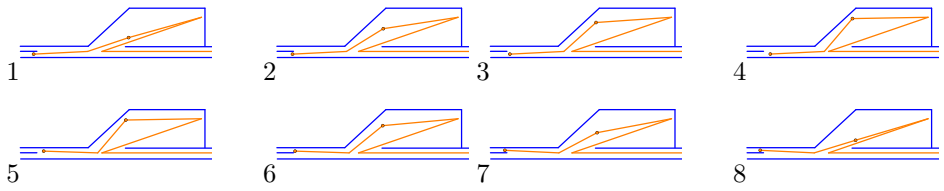

FIGURE 4. Animation showing how the chain can move around the notch when the checked variable has the right value

The box that checks for the (positive) literal z_i is depicted in Figure 5. From the drawing it is clear that the chain can be shortened by almost one length unit when the chain can enter a box.

The piece of the chain that encodes a variable z_i cannot reach the boxes that check literals corresponding to variables z_j for $j > i$, because the chain gets stuck at the dead-end of the gadget. On the other hand, the boxes that check literals corresponding to variables z_j for $j < i$ do not provide enough space for the piece of the chain that encodes z_i to enter and thus shorten the chain.

The width of the tunnel below/above the two boxes, called the *base tunnel*, is $h = 1/(2mn)$, where $m = \Theta(P(|x|)) = \Theta(n)$ denotes the number of transition-position pairs of M. In particular, the width of the tunnel is smaller than one unit,

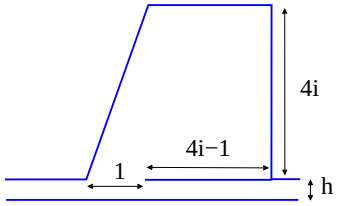

FIGURE 5. The box checking for z_i

so the chain cannot reconfigure inside the tunnel. The length of the tunnel is L, so the chain, if completely flattened, would exactly fit into the gadget.

Note that the chain can shorten a little bit by zig-zagging inside the base tunnel, cf., Figure 6. In that case the height of the tunnel is an upper bound on how much the chain can be shortened when a single switch folds up, so in total, the chain cannot shorten by more than $n \cdot h = 1/(2m) < 1/2$ length units. The length of the notch that separates the entrance from the exit is $3/4$, so the only way for the chain to leave the gadget is to be shortened in a box.

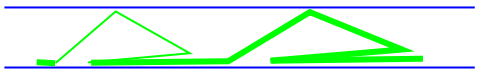

FIGURE 6. The chain can shorten in the base tunnel

Combining 2-clause checkers. The final checker $E_{t,p}$ for the condition $P_{t,p} \vee Q_{t,p}$ is made by combining all the corresponding 2-clause checkers $k_1, \ldots, k_{r^2}$ by appropriately attaching the exit of tunnel k_i to the entrance of tunnel k_{i+1} to simulate the conjunction of the clauses. The scheme is shown in Figure 7.

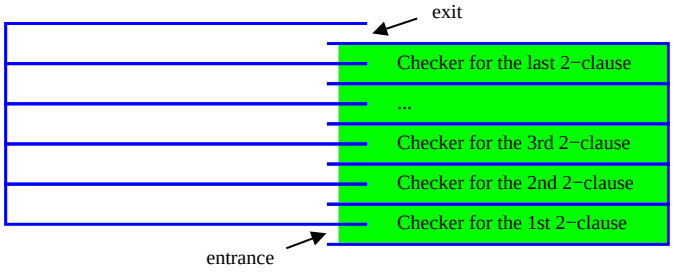

FIGURE 7. Scheme showing how the 2-clause checkers are combined

Since a box checking for a literal in a 2-clause checker can have height up to $\Theta(n)$ length units, the chain C could reconfigure in the tunnels on the left side of Figure 7. This is avoided by replacing each such tunnel by a staircase-like structure of narrow tunnels, called a *wire gadget*, through which C has to move in order to get to the next checker.

Figure 8 shows how two 2-clause checkers are connected. The checkers are depicted on the right side. The wire gadget is on the left side. It is a tunnel of

width twice the length of the flattened chain. It consists of a sequence of interleaved tunnels of height $2h$ length units (the first and the last tunnel that connect to the checker gadgets are actually only h length units high).

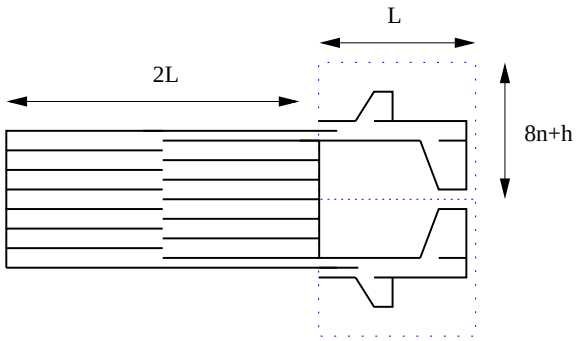

FIGURE 8. Combining two 2-clause checkers with a wire gadget

The distance between the exit of the lower 2-clause checker and the entrance of the upper one can be up to $\Theta(n)$ length units. Therefore the number of tunnels in the wire gadget is $O(n^2m)$. The number of segments is of the same order.

The final gadget $E_{t,p}$ combines r^2 2-clause checkers, and thus consists of $O(r^2n^2m)$ segments. It is $O(n^2)$ length units wide and $O(r^2n)$ length units high. Note also that, by construction, it is a tree.

3.2.2. *Setters.* For each transition-position pair (t,p) we build a setter $S_{t,p}$ that allows the switches that correspond to the bits of a configuration c that would be affected by t at position p to reconfigure; note that t changes at most $O(r)$ bits of c. The gadget is depicted in Figure 9.

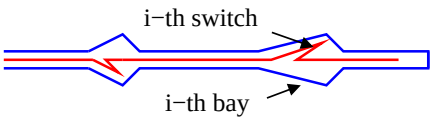

FIGURE 9. Setter allowing the i-th bit (among others) to flip

It is a dead-end tunnel, and for each variable z_i that is changed by (t, p) there is a small bay widening in the tunnel. The position of the bay corresponds to the position of the switch of C that encodes the variable that is being changed. Apart from the bays, the gadget is so narrow that the chain cannot reconfigure inside.

If the piece of the chain that encodes the variable z_i has reached the corresponding bay, the switch has enough space to reconfigure. This process is depicted in Figure 10.

The length of the tunnel is $L+8n-1$ length units, the height of the tunnel is h. This ensures that the piece of the chain that encodes the variable z_i cannot reach the bays that correspond to the variables z_j for $j > i$, because the chain gets stuck in the dead-end of the gadget. On the other hand, since the size of the switches increases along the chain, the bays that correspond to variables z_j for $j < i$ do not provide enough space for the piece of the chain that encodes z_i to reconfigure. The checker $S_{t,p}$ consists of $O(r)$ segments.

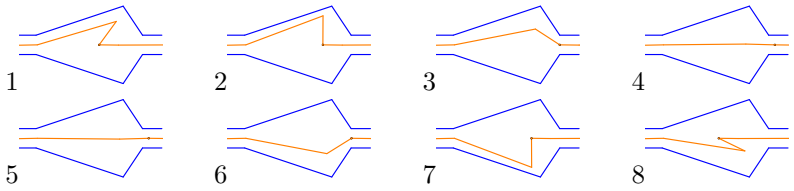

FIGURE 10. Animation showing how a bit of the configuration can flip in the corresponding setter

3.2.3. *Assembling the environment.* For each transition-position pair (t, p), we first combine the checker $E_{t,p}$ and the setter $S_{t,p}$ (with an additional wire-gadget) to build the gadget $G_{t,p}$, which is again a tree; it consists of $O(r^2n^2m)$ segments, and it is $O(n^2)$ length units wide and $O(r^2n)$ length units high.

Finally all the gadgets $G_{t,p}$ are connected to a *root tunnel* of height $mh = 1/(2n) < 1/2$ length units and width L length units. The choice of the tunnel width ensures that the chain cannot reconfigure inside the root tunnel.

The transition gadgets are lined up at the right end of the tunnel. A long horizontal corridor of height h length units connects a transition gadget to the root tunnel. The length of the corridor leading to the i-th gadget is $O(in^2)$. After combining the m gadgets, the environment consists of $O(r^2n^2m^2)$ segments and is $O(n^2m)$ length units wide and $O(r^2n)$ length units high. Note that the environment

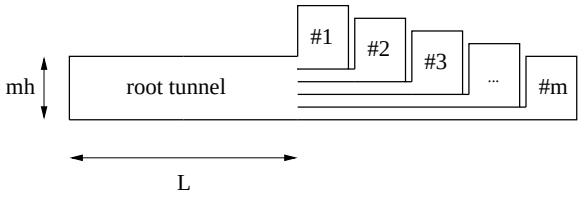

FIGURE 11. A schematic drawing of the environment

as depicted in Figure 11 is not a tree. To make it into a tree, we cut open the root tunnel in the lower left corner and place a rigid *lock-gadget* there, cf. Figure 12 and Figure 2.

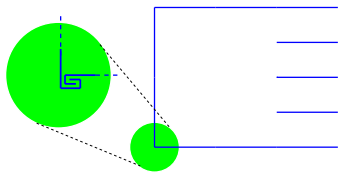

FIGURE 12. Making the environment-tree rigid

3.2.4. *The forests S and T.* Remember that there is a unique accepting configuration c_{acc} in which M has erased the tape and moved the tape head to the first tape cell, and that there is no transition that leaves the accepting state. By

c_x, we denote the initial configuration of M on x. By C_x and C_{acc}, we denote the configurations of the chain C encoding the configurations c_x and c_{acc}, respectively.

The forest S is obtained by placing the chain C_x into the root tunnel. Let us assume without loss of generality that the precondition of the first transition gadget checks for the accepting configuration c_{acc}. Since there is no transition that leaves the accepting state, the associated setter is just a tunnel without any bays. The forest T is obtained by placing the chain C_{acc} in that setter.

3.3. Computing the coordinates. The coordinates of the vertices of S and T are computed approximately with a precision of $\Theta(n)$ bits, so that the encoding length of the coordinates of these vertices is polynomial in the input size, and S and T can be constructed in polynomial time.

As a consequence, the vertices of the construction described above are rounded to a grid where the distance between two grid points is $\Theta(1/2^{cn})$ for some constant $c > 0$. The grid size (and thus the roundoff error) is negligible compared to the deviations of the construction, and so the rounding does not influence the functionality of the gadgets.

The issue of rigidity. Since the coordinates of the construction are only computed approximately, the environment is not completely rigid. The mechanism depicted in Figure 2 will not keep the angle between the two hinges fixed but allow them to rotate slightly by an angle of $O(1/2^{cn})$ degrees. If we add up all these angular deviations and multiply them by the total length of all segments used in the construction we get an upper bound of $O(r^4n^4m^3/2^{cn})$ length units on how much the distance between any two vertices of the construction can vary. Again this is negligible compared to the deviations of the construction, and does not influence the functionality of the gadgets.

3.4. Correctness of the reduction. We need to show that S can be reconfigured to T precisely when M accepts x.

On the one hand, an accepting computation of M on x can be turned into a reconfiguration that transforms S to T. To see this, look at the sequence of configurations of the accepting computation of M on x. Each transition $c \to c'$ in that sequence that transforms a configuration c into another configuration c' via some transition t (applied at some tape position p) can be simulated by moving C encoding c into the gadget that corresponds to the transition-position pair (t, p), changing C in the setter of that gadget to C' encoding c' and pulling C' out of the gadget.

It remains to show that a reconfiguration that transforms S to T yields an accepting computation of M on x. To this end we subdivide the reconfiguration into phases that start when the linkage leaves a transition gadget and that end when it enters the next one, so within a phase the chain is in the root tunnel. We ignore the steps of the reconfiguration process where the chain is inside a gadget. As was mentioned before, the construction ensures that the chain C encodes a unique configuration c during a phase. Thus the reconfiguration gives a sequence $c_1, \ldots, c_k$ of configurations of M on x, where $c_1 = c_x$ and $c_k = c_{acc}$. The construction ensures that either $c_i = c_{i+1}$, $c_i \to c_{i+1}$ or $c_{i+1} \to c_i$ for $i = 1, \ldots, k-1$. If we delete successive duplicates $c_i = c_{i+1}$ from that sequence we are left with a sequence

$$c_1 \leftrightarrow c_2 \leftrightarrow \cdots \leftrightarrow c_{k-1} \leftrightarrow c_k,$$

where $\leftrightarrow$ is either $\rightarrow$ or $\leftarrow$. Now assume that

$$\cdots \leftrightarrow c_{j-1} \leftarrow c_j \rightarrow c_{j+1} \leftrightarrow \ldots$$

Since M is a deterministic machine, we can conclude that $c_{j-1} = c_{j+1}$, so the backward transition $c_{j-1} \leftarrow c_j$ can be eliminated from the sequence. If we iterate this process we are left with a sequence of configurations of M forming an accepting computation of M on x. Note that $c_k = c_{acc}$ is never eliminated during that process since, according to our assumption, there is no transition that leaves the accepting state.

3.5. From forests to trees. In order to prove Theorem 2 we have to modify the reduction slightly. The idea is simple: we use a chain of very short segments – called the *leash* – to attach the chain C to the environment. With reference to Figure 11, one end of the leash is tied to the upper left corner of the root tunnel, and the other end of the leash is attached to the initial endpoint of chain C.

The leash itself is assembled from segments small enough not to restrict the possible motions of the chain inside the environment compared with the unconstrained setting. A detailed analysis shows that a leash of length $O(n^7)$ allows the chain to reach the same positions as before and that a total number of segments in the leash of $O(n^{10})$ makes it flexible enough.

Section 4. Hardness of the reconfiguration problem for chains in $\mathbb{R}^3$

We describe a modification of the construction from Section 3 that shows that the problem of deciding whether a given polygonal chain S' can be reconfigured into a prescribed target chain T' in 3-space is also PSPACE-hard.

THEOREM 3. *The reconfiguration problem for polygonal chains in $\mathbb{R}^3$ is PSPACE-hard.*

Let us first sketch the idea behind the modified reduction. Assume that we turn the two-dimensional environment E of the previous section into a solid 3D maze by erecting vertical solid walls above the edges of E and putting a solid lid on top and a solid bottom underneath. It is clear from the construction of the previous section that C can be moved from the root tunnel to the setter of the first gadget without self-intersecting and without penetrating the walls of the maze if, and only if, M accepts x (under the assumption that the distance between the bottom and the lid is sufficiently small). This remains true if the chain is tied to the maze with a leash as above. Now instead of using solid walls as the building blocks of the maze we use a fine mesh woven with a polygonal chain, and we blow up the vertices of C slightly (using a structure also made from a polygonal chain). The width of the mesh is chosen small enough and the size of the vertex structures is made so large that the chain cannot leave the maze.

The basic building block of the construction is a (nearly) rigid tetrahedron τ made from a polygonal chain as follows.

We start with four pairs of interlocked polygonal chains whose construction is given in [**8**]. These pairs are placed at the four corners of a tetrahedron and connected by more bars along the edges of the tetrahedron as shown in Figure 13.

First, every vertex of the chain C is replaced by a small copy of τ so that it cannot be moved through a sufficiently fine-meshed grid.

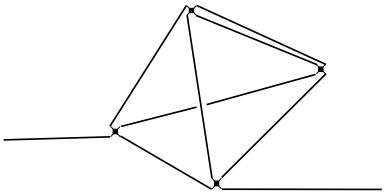

FIGURE 13. A polygonal chain forming a rigid tetrahedron τ

The maze is constructed as follows: we build the environment tree E as in the previous section, laid out in the xy-plane. We turn E into a path P by doubling all vertices and edges, and connecting them according to a walk around E; we choose the lower left corner of the root tunnel as the start and end point of the walk. Now we place a copy P' of P directly above P (so that P' projects vertically to P in the xy-plane), and make them into a single polygonal chain E' by connecting two corresponding endpoints. The vertical distance between the two copies is small enough so that the chain C cannot flip the bits encoding the configuration of M. The floor, ceiling and walls of the maze are built by wrapping a polygonal chain G around E' in a grid-like fashion and connecting it to one end of E' to form a chain E''. The width of the grid is chosen so that it does not provide enough room for C to pass through. We surround every intersection point of the grid with a small copy of the tetrahedral chain τ and make a single chain from these chains (that connects to one end of E'') by connecting them appropriately. The whole maze is then made almost rigid by adding all the diagonals to the grid. This again can be done with a polygonal chain that pierces the copies of the tetrahedron. Finally the chain C is attached to the environment tree as in the previous section to form S' and T'

More detailed considerations show that the number of segments used to build the chains S' and T' is polynomial in n and that it is sufficient to compute the coordinates of the vertices of the two paths approximately with a precision of $O(n)$ bits. In particular the two paths can be constructed in time polynomial in n.

Section 5. Conclusion and open problems

We have shown that the linkage foldability problem is PSPACE-complete for trees moving in 2D and chains moving in 3D. The hardness proofs use rigid or inseparable substructures to simulate a motion planning problem and mimic the known hardness proofs for these problems. One interesting question that remains open (see [**2**]) and leaves room for further work is to determine the complexity of the problem of deciding whether trees in 2D or chains in 3D can be straightened out flat (in the case of trees in 2D, made arbitrarily close to flat).

References

[1] H. Alt, C. Knauer, G. Rote, and S. Whitesides, The Complexity of (Un)folding. In *Proc. 19th Annu. ACM Symp. on Computational Geometry*, pages 164–170, 2003.

[2] T. C. Biedl, E. D. Demaine, M. L. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, M. H. Overmars, S. Robbins, I. Streinu, G. T. Toussaint, and S. Whitesides. Locked and unlocked polygonal chains in three dimensions. *Discrete and Computational Geometry*, 26:269–281, October 2001.

[3] J. Canny. *The Complexity of Robot Motion Planning*. ACM – MIT Press Doctoral Dissertation Award Series. MIT Press, Cambridge, MA, 1987.
[4] J. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 460–467, 1988.
[5] J. Cantarella and H. Johnston. Nontrivial embeddings of polygonal intervals and unknots in 3-space. *Journal of Knot Theory and its Ramifications*, 7:1027–1039, 1998.
[6] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conference on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes Comput. Sci.*, pages 134–183. Springer-Verlag, 1975.
[7] R. Connelly, E. D. Demaine, and G. Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete and Computational Geometry*, 2003. To appear.
[8] E. D. Demaine, S. Langerman, J. O'Rourke, and J. Snoeyink. Interlocked open linkages with few joints. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 189–198. ACM Press, 2002.
[9] D. A. Joseph and W. H. Plantinga. On the complexity of reachability and motion planning questions. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 62–66, 1985.
[10] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, I-III. *Journal of Symbolic Computation*, 13:255–352, 1992.
[11] J. T. Schwartz and M. Sharir. On the "piano movers" problem II: general techniques for computing topological properties of real algebraic manifolds. *Advances in Appl. Math,*, 4:298–351, 1983.
[12] I. Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *IEEE Symposium on Foundations of Computer Science*, pages 443–453, 2000.

Freie Universität Berlin, Institute of Computer Science, Takustrasse 9, D-14195 Berlin, Germany
E-mail address: {alt,knauer,rote}@inf.fu-berlin.de

School of Computer Science, McGill University, 3480 University St. #318, Montreal, Canada H3A 2A7
E-mail address: sue@cs.mcgill.ca