On Unfolding Trees and Polygons on Various Lattices

Sheung-Hung Poon*

Abstract

We consider the problem of unfolding lattice trees and polygons in hexagonal or triangular lattice in two dimensions. We show that a hexagonal/triangular lattice chain (resp. tree) can be straightened in $O(n)$ (resp. $O(n^2)$) moves and time, and a hexagonal/triangular lattice polygon can be convexified in $O(n^2)$ moves and time. We hope that the techniques we used shed some light on solving the more general conjecture that a unit tree in two dimensions can always be straightened.

1 Introduction

Graph reconfiguration problems have wide applications in contexts including robotics, molecular conformation, animation, wire bending, rigidity and knot theory. The motivation for reconfiguration problems of lattice graphs arises in applications in molecular biology and robotics.

A graph is *simple* if non-adjacent edges do not intersect. A *unit tree* (resp. *polygon*) is a tree (resp. polygon) containing only edges of unit length. A *hexagonal/square/triangular lattice tree* (resp. *polygon*) is a simple tree (resp. polygon) containing only edges from a hexagonal/square/triangular lattice in two dimensions, respectively. We say two edges e and e' *cross* each other if one of edges penetrates into the interior of the other. We consider the problem about the reconfiguration of a simple chain, polygon, or tree through a series of continuous motions such that the lengths of all graph edges are preserved and no edge crossings are allowed. We remark that during the reconfiguration, touchings between rigid edges are allowed. A tree can be *straightened* if all its edges can be aligned along a common straight line such that each edge points "away" from a designated leaf node. In particular, a chain can be straightened if it can be stretched out to lie on a straight line. A polygon can be *convexfied* if it can be reconfigured to a convex polygon. We say a chain or tree is *locked* if it cannot be straightened. We say a polygon is *locked* if it cannot be convexified. We consider one move in the reconfiguration as a continuous monotonic change for the joint angle at some vertex, during which no edge crossings occur.

*Department of Mathematics and Computer Science, TU Eindhoven, 5600 MB, Eindhoven, the Netherlands. spoon@win.tue.nl

In four dimensions or higher, a polygonal tree can always be straightened, and a polygon can always be convexified [2]. In two dimensions, a polygonal chain can always be straightened and a polygon can always be convexified [3]. However, there are some trees in two dimensions that can lock [1, 4]. Poon [4] showed that a unit tree of diameter 4 in two dimensions can always be straightened. In their paper, they posed a challenging open question whether a unit tree in either two or three dimensions can always be straightened. Poon [5] showed that a square lattice tree in two and three dimensions can always be straightened, and a square lattice polygon in two dimensions can always be convexified.

In the following sections, we present our results for hexagonal and triangular lattices, respectively. We define that a *spring* in P is a maximal connected zig-zag path of edges in P coincident to a common lattice edge.

2 Hexagonal Lattice

In this section, we apply the techniques used in [5] for square lattice to unfold chains, trees or polygons in hexagonal lattice.

2.1 Hexagonal Lattice Chains

Given a hexagonal lattice chain $P = p_0p_1 \dots p_n$ in two dimensions. We fold up the chain from end edges iteratively to finally obtain a single spring, which can then be straightened straightforwardly. Starting from an end edge, say p_0p_1, we fold up the whole chain, edge by edge. At the end of step i, a hexagonal lattice polygon P_i is obtained such that a spring S_i containing a zig-zag path from p_0 to p_{i+1} is formed. Step $(i+1)$ of the algorithm tries to combine the spring S_i and the edge $p_{i+1}p_{i+2}$ to form a new spring S_{i+1} as shown in Figure 1, which takes constant number of moves and time. Thus the

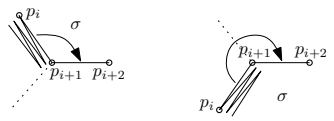

Figure 1: A folding step for a hexagonal lattice chain.

whole chain can be folded up, in $O(n)$ moves and time, into a final single spring, which can then be straightened straightforwardly.

Theorem 1 *A hexagonal lattice chain in two dimensions can be straightened in $O(n)$ moves and time.*

2.2 Hexagonal Lattice Trees

Given a hexagonal lattice tree P in two dimensions. The algorithm runs by pulling the whole tree to the left iteratively until the whole tree is straightened. We select a leftmost vertex r of P as its root. We use the convention that the parent of the root r as the lattice point to the left of r. Our algorithm proceeds by pulling P to the left successively until the whole tree is straightened. Each pulling step moves each vertex along its edge connecting to its parent until it coincides with its parent in the previous step. This step is repeated n times until, finally, P is straightened. Figure 2 shows the execution of the algorithm on a small tree. Step i generates a new pseudo-lattice tree P_i. The correctness

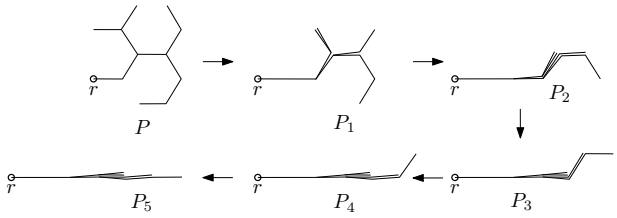

Figure 2: Straightening a hexagonal lattice tree by pulling it to the left successively.

proof of our algorithm is similar to what was done for the square lattice in [5]. Its details are thus omitted here. As each pulling step takes O(n) moves and time, the whole algorithm takes $O(n^2)$ moves and time.

Theorem 2 *A hexagonal lattice tree in two dimensions can be straightened in $O(n^2)$ moves and time.*

2.3 Hexagonal Lattice Polygons

Given a hexagonal lattice polygon. Our unfolding algorithm runs in the same fashion as that for lattice polygons in the square lattice [5]. We successively collapse leftmost collapsible block until we finally reach a lattice hexagon, which we can then be convexified straightforwardly. However, we need to define what we mean by a block in a hexagonal lattice. For our convenience, in this section, we suppose that the lattice lines are the vertical lines and lines of slopes $1/\sqrt{3}$ and $-1/\sqrt{3}$.

For the square lattice, Poon [5] define a block to be a rectangle of width one. However, for the hexagonal lattice, we need a new definition for a block. A *block* of a hexagonal lattice polygon is hexagonal lattice cell B such that the left three cell edges of B are coincident to a connected path of the given polygon lying on the cell boundary of B. Such a maximal connected path on the boundary of B is called the *block path* of B. We define the *opening* of B as the boundary of B complementing its block path. A block is called *collapsible* if its opening is not coincident to any edge of the given polygon. To collapse a block, we divide into three cases depending on the number of edges on the block opening. Figure $3(a), (b)$ show the results of collapsing a block to an opening containing exactly one lattice edge, say *Case*

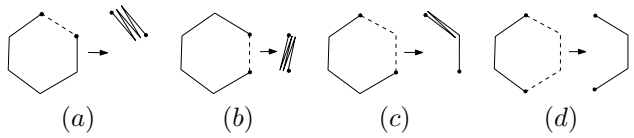

Figure 3: The block opening contains $(a), (b)$ only one lattice edge for *Case 1*, (c) two lattice edges for *Case 2*, or (d) three lattice edges for *Case 3*.

1, and Figure $3(c), (d)$ show the results of collapsing a block to an opening containing two or three lattice edges, say *Case 2* or *Case 3*, respectively. Observe that for *Case 3*, no edges are reduced during the block collapsing operation, and the vertical edge of its opening may contain a polygon edge so that the current block is not collapsible. As in each block collapsing step, one vertical edge of the given polygon is moved to its right, the number of occurrences of collapsing *Case 3* can be at most n until we reach one occurrence of *Case 1* or *Case 2*. Up to this point, it takes $O(n)$ moves and time. To fold up the polygon, we have to run $O(n)$ number of occurrences of *Case 1* or *Case 2*. Hence, it takes $O(n^2)$ moves and time in total. Finally, the folded polygon, a polygon containing a single hexagonal lattice face, can be convexified straightforwardly.

Theorem 3 *A hexagonal lattice polygon in two dimensions can be convexified in $O(n^2)$ moves and time.*

3 Triangular Lattice

In this section, we extend the techniques used in the previous sections to handle the graphs in a triangular lattice, which is in a sense denser than the hexagonal or square lattice. Thus more involved algorithms and analyses are needed.

3.1 Triangular Lattice Chains

Given a triangular lattice chain. We again apply the technique as in Section 2.1 by folding up the chain from end edges iteratively to finally obtain a single spring, which can then be straightened straightforwardly. However, for the triangular lattice case, more careful routine to fold up an end edge need to be designed so that there are no edge crossings occurring in the folding routine.

To fold up an end edge e towards the second last end edge e', we divide into three cases depending on

the angle α between e and e': *Case 1:* $\alpha = \pi/3$; *Case 2:* $\alpha = 2\pi/3$; and *Case 3:* $\alpha = \pi$. Remark that in each of these cases, we need to handle two subcases depending on which side of e the tail of the end spring resides, and at the end of each folding-up step, the result position of e' remains the same as its original position, or makes an angle of $\pi/2$ with the edge adjacent to e'. We illustrate the folding procedure for the case when $\alpha = \pi/3$ as shown in Figure 4. The folding procedures for the other two cases can be handled similarly. Note that in the figure, the interior of the dotted polygon does not contain any other chain edges except e and e' so that our folding-up procedure does not lead to any edge crossing.

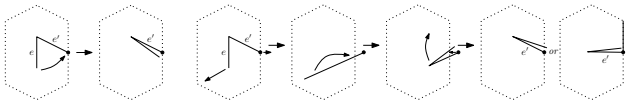

Figure 4: The case when $\alpha = \pi/3$.

After one folding-up step, the resulting end spring e may make an angle of $\pi/2$ with the edge e' adjacent to e as we can see above. Thus there is one more case for folding up the end edges later on. However, this case can be handled in a similar fashion as shown in Figure 4(b). As each folding-up step takes constant number of moves, we thus have the following theorem.

Theorem 4 *A triangular lattice chain in two dimensions can be straightened in $O(n)$ moves and time.*

3.2 Triangular Lattice Trees

Given a triangular lattice tree P. We apply the technique as in Section 2.2 by pulling the tree to the left iteratively until the whole tree is straightened. Suppose the lattice lines are horizontal lines and lines of slopes $\sqrt{3}$ and $-\sqrt{3}$. We select a leftmost vertex r of P as its root. We use the convention that the parent of the root r as the lattice point of distance one to the left of r. Each pulling step moves each vertex along the edge (or its extension) connecting itself to its parent at the beginning of the step until r coincides with its parent. We first show that no edge crossings can occur during one pulling step in the following lemma.

Lemma 5 *No edge crossings can occur during the pulling step $i (1 \leq i \leq n)$ for straightening a triangular lattice tree.*

Proof. (Sketch) If all the angles between adjacent edges are at least $2\pi/3$, then the same argument as for straightening hexagonal lattice trees holds. Otherwise there may be some new situations we will encounter.

Consider a moving edge uv during pulling step i. Suppose at the beginning of step i, the two adjacent edges TU and UV in P_{i-1} make an angle of $\pi/6$, i.e., $\angle TUV = \pi/6$. Suppose u moves from U to T along edge TU, and v first moves away from V on the extension of UV and then moves back to V along the extension of UV. See Figure 5. We will estimate how far v goes from

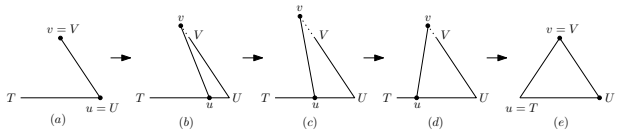

Figure 5: The motion of uv when $\angle TUV = \pi/3$.

V along the extension of UV. The maximum distance d of v from V during the pulling step i is acquired when u reaches the middle point of TU. By cosine rule,

$$d^2 + (\sqrt{3}/2)^2 - 2(d)(\sqrt{3}/2)\cos(5\pi/6) = 1.$$

By solving this quadratic equation and neglecting the negative root, we have $d = (\sqrt{13} - 3)/4 < 0.1514$.

Then we consider how the descendent vertices of V moves. Suppose WXY is a path of P_{i-1} such that V is the ancestor of W, X and Y. Consider a moving edge xy during pulling step i such that x (resp. y) starts at and moves away from X (resp. Y) along edge WX (resp. XY) or its extension, and finally return to X (resp. Y). We claim that if $xX < 0.1514$, then $yY < 0.1514$. With this claim, we can then apply it recursively to all descendent vertices of V. Now, let's prove the claim. We divide into three cases depending on whether $\angle xXY$ is equal to $\pi, \pi/3$ or $2\pi/3$. If $\angle xXY = \pi$, then $yY = xX$ follows immediately. We then consider the case that $\angle xXY = \pi/3$. Refer to Figure 5(b) by replacing TUV and uv by WXY and xy, respectively. By applying cosine rule on $\triangle xXy$, we have

$$(yY + 1)^2 + (xX)^2 - 2(yY + 1)(xX)\cos(\pi/3) = 1.$$

By solving this quadratic equation and neglecting the negative root, we have $yY < 0.12061$ as $xX < 0.1514$. Finally, we consider the case that $\angle xXY = 2\pi/3$. Again by applying cosine rule on $\triangle xXy$, we have

$$(1 - yY)^2 + (xX)^2 - 2(1 - yY)(xX)\cos(2\pi/3) = 1.$$

By solving this quadratic equation and neglecting the negative root, we have $yY < 0.08433$ as $xX < 0.1514$. Thus we are done.

Using the above properties and by performing case analysis on the motion of any pair of edges, it is not hard for us to show that they do not cross each other. Details are omitted in this abstract. □

In all, as each pulling step takes $O(n)$ moves and time, the whole algorithm takes $O(n^2)$ moves and time.

Theorem 6 *A triangular lattice tree in two dimensions can be straightened in $O(n^2)$ moves and time.*

3.3 Triangular Lattice Polygons

Given a triangular lattice polygon P. In this section, we have to extend the block-collapsing technique used in Section 2.3 by defining a block in a more general way. For our convenience, we suppose that the lattice lines are the vertical lines and lines of slopes $1/\sqrt{3}$ and $-1/\sqrt{3}$.

A *parallelogram block* (resp. *trapezoidal block*) of a triangular lattice polygon is a parallelogram (resp. trapezoid) of width one (along the lattice) such that its two width-sides and one of its non-width side coincides with the edges of the given polygon. A *triangular block* is a triangle with one of its edges not coinciding with the edges of the given polygon. See Figure 6 for illustration. A *block* is either a parallelogram, trapezoidal or triangular block. A *collapsible block* of a lattice polygon is a block such that its side not completely lying on the given polygon complementing the given polygon is a single segment. Such a segment is called the *opening segment*, or simply the *opening*, of the corresponding collapsible block. And the two endpoints of an opening segment are called *opening vertices*. The path between its two opening vertices on a block is called the *path* of the block. A block with its path containing no spring on its opening side is called *unsheltered*; otherwise it is called *sheltered*.

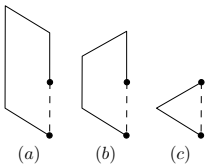

Figure 6: (a) A parallelogram block, (b) a trapezoidal block, & (c) a triangular block.

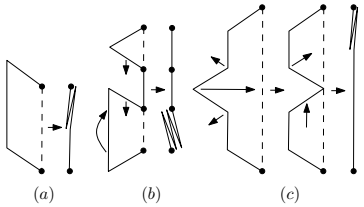

Figure 7: (a) Collapse a parallelogram block; (b) Collapse two trapezoidal/triangular blocks; & (c) Collapse an extended block.

Below we sketch the main ideas of the unfolding algorithm. Observe that an unsheltered collapsible parallelogram block can be folded up directly as shown in Figure 7(a). By sweeping a vertical line from left to right, we must find a collapsible block B. If B is sheltered, then from the sheltered part of B, we can find an unsheltered parallelogram block, which can then be folded up directly. If B is unsheltered and is a parallelogram, then we can fold up B. We need to consider the case when B is an unsheltered trapezoidal or triangular block. For this case, if there is another unsheltered trapezoidal or triangular block vertically adjacent to B, then we can fold them up to align on their openings as shown in Figure 7(b). Thus the only case remained is when B is isolated. If it is the case, we need to define the extension of B. The *extended block* B' of B is defined by stacking another block on B such that by replacing the block path of B by its opening segment, B' becomes also a block. B' is called *collapsible* if B is collapsible, and the extra stacked block is collapsible. A collapsible extended block can be folded up by first rearranging the edges of B in a way simulating the flipping of B along its opening side, and then collapse the two adjacent blocks resulted in. This is shown in Figure 7(c). Up to this moment, we still haven't handled the case when B is an unsheltered collapsible trapezoidal or triangular block, and either it does not have an extended block or its extended block is non-collapsible. For this complicated case, we then need to consider the sweeping direction from right to left for the vertical line or the other four sweeping directions so that we can find some collapsible block/extended block. The details are omitted in this abstract. Ultimately, the given polygon P is folded up as a triangle or a parallelogram containing two triangular lattice faces depending on whether P contains odd or even number of edges. Such a nearly folded polygon can then be convexified straightforwardly. Hence, we obtain the following theorem.

Theorem 7 *A triangular lattice polygon in two dimensions can be convexified in $O(n^2)$ moves and time.*

References

[1] T. Biedl, E. Demaine, M. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, S. Robbins, I. Streinu, G. Toussaint, and S. Whitesides. A Note on Reconfiguring Tree Linkages: Trees Can Lock. *Discrete Applied Mathematics*, volume 117, number 1-3, pages 293-297, 2002.

[2] R. Cocan and J. O'Rourke. Polygonal Chains Cannot Lock in 4D. *Computational Geometry: Theory & Applications*, 20, 105–129, 2001.

[3] R. Connelly, E.D. Demaine, and G. Rote. Straightening Polygonal Arcs and Convexifying Polygonal Cycles. *Discrete & Computational Geometry*, volume 30, number 2, 205–239, 2003.

[4] S.-H. Poon. On Straightening Low-Diameter Unit Trees. In *Proc. 13th International Symposium on Graph Drawing*, 519–521,2005.

[5] S.-H. Poon. On Unfolding Lattice Polygons/Trees and Diameter-4 Trees. In *Proc. 12th Annual International Computing and Combinatorics Conference (COCOON)*, 186–195, 2006.