Computing 3SAT on a Fold-and-Cut Machine

Byoungkwon An* Erik D. Demaine* Martin L. Demaine* Jason S. Ku*

Abstract

This paper introduces a computational model called a *fold-and-cut machine* which allows as operations simple folds and unfolds, straight-line cuts, and inspection for a *through-hole* (hole through all the layers of paper). We show that a (deterministic) fold-and-cut machine can decide a 3SAT instance with n variables and m clauses using $O(nm+m^2)$ operations (with just one cut and inspection), showing that the machine is at least as powerful as a nondeterministic Turing machine.

1 Introduction

Computational origami [DO08] is usually about using algorithms (on traditional computers) to design/analyze paper foldings. But what if we view the piece of paper as the computer itself, with folding as one of the operations provided by the model of computation? In this paper, we initiate the study of what computation is possible in such folding models.

A *fold-and-cut machine* operates on a polygonal 2D piece of paper, supporting four operations—Fold, Unfold, Cut, Look—that modify the current flat folded state of the piece of paper (initially flat) and the piece of paper itself.

1. A Fold operation is any simple fold [ADK, ABD+04]: a fold of some number of layers along a straight line by $\pm 180°$.

2. An Unfold operation undoes a Fold operation.

3. A Cut operation is a complete straight-line cut through (all layers of) the current flat folding, discarding all but one specified piece.

4. A Look operation decides whether the current flat folding has a *through-hole*, i.e., has nonzero genus (a hole) when imagining all touching layers to be fused together.

Each line (in a Fold or Cut operation) is specified by a distinct pair of points with integer coordinates using a polynomial number of bits. The initial polygon of paper is similarly described by integer vertex coordinates each using a polynomial number of bits. (In fact, in our constructions, the paper will be a rectangle and the lines will all be horizontal, vertical, or $45°$ diagonal.)

*CSAIL, Massachusetts Institute of Technology, {dran, edemaine,mdemaine,jasonku}@csail.mit.edu

Models involving just Fold and Unfold operations have been considered before [CDD+11, Ueh11], but where the goal was to achieve certain geometric folding properties instead of computation. We add the ability to make cuts, though in fact we will use just a single cut, in the style of the fold-and-one-cut problem [DO08, DDL98, BDEH01], also previously considered in the context of simple folds [DDH+10], but with geometric instead of computational goals.

A *3SAT instance* is a Boolean formula over n variables $X = \{x_1, x_2, \ldots, x_n\}$ in conjunctive normal form (CNF):

$$\bigwedge_{i=1}^{m}(a_i \vee b_i \vee c_i), \tag{1}$$

where a_i, b_i, c_i are called literals, each corresponding to some variable in X, or its negation. Each term $(a_i \vee b_i \vee c_i)$ is called a *CNF clause*. A 3SAT instance is *satisfiable* if there exists an assignment of each variable as either 0 or 1 such that the Boolean formula evaluates to 1. Deciding satisfiability of a given 3SAT instance is a classic NP-complete problem [GJ79].

Theorem 1 *A fold-and-cut machine can decide 3SAT in $O(nm+m^2)$ operations, using just one* Cut *and just one* Look.

As a consequence, all decision problems in NP can be solved by a polynomial number of operations on a fold-and-cut machine, making it at least as powerful as a nondeterministic Turing machine.

Inspired by the recently introduced *fold-and-one-punch problem* [ADD+], we also consider an alternative folding model of computation that replaces the Cut operation as follows:

3′. A Punch operation cuts a point hole (or a small circular hole) through (all layers of) the current flat folding.

(Again the point has integer coordinates specified by a polynomial number of bits.) We prove that Theorem 1 also holds on this *fold-and-punch machine*. In fact, this construction works even with "1.5D paper": a narrow rectangular strip that can be folded only perpendicular to the strip direction, but which remains connected after punching a hole.

This paper is organized as follows. Section 2 introduces the structure of our approach, representing a 3SAT instance as a DNF formula with a symmetric clause ordering. Section 3 describes a set of holes in a piece of paper that correspond to a 3SAT instance. Section 4 shows that the resulting paper can be folded so that the outcome of a single LOOK operation is equivalent to the solution to the 3SAT instance. Section 5 shows how to produce the described hole pattern using polynomially many FOLD operations and one CUT operation. Section 6 puts all of these pieces together to prove the theorem. We conclude in Section 7 with open problems for future work.

2 Approach

Conceptually, we use the known conversion from any 3CNF formula into disjunctive normal form (DNF) [GJ79], which results in an OR of 3^m DNF clauses, where each DNF clause is an AND of m of the CNF literals, one from each CNF clause. Our reduction from 3SAT cannot afford to actually compute this DNF formula, but the fold-and-cut machine we produce will end up physically constructing a truth table for the DNF formula where holes represent 1s.

Deciding whether the 3SAT instance has a solution is equivalent to finding an assignment of the variables $\alpha : X \to \{0,1\}$ for which some DNF clause evaluates to 1. There are 3^m DNF clauses and 2^n possible assignments of variables. Let $d(i,j,k)$ represent the Boolean value of the ith CNF clause's literal present in the jth DNF clause when using the kth assignment for the variables, with $i \in [1,m]$, $j \in [1,3^m]$, and $k \in [1,2^n]$. In particular, if the literal from the ith CNF clause present in the jth DNF clause is variable x_r, then $d(i,j,k) = \alpha_k(x_r)$, while if the literal is the negation of x_r, then $d(i,j,k) = \neg\alpha_k(x_r)$. Thus, there are $m(3^m)(2^n)$ possible $d(i,j,k)$. Then the DNF formula associated with a 3SAT instance is equivalent to evaluating the exponentially sized Boolean formula:

$$\bigvee_{k=1}^{2^n} \bigvee_{j=1}^{3^m} \bigwedge_{i=1}^{m} d(i,j,k). \tag{2}$$

To prove Theorem 1, we will use a fold-and-cut machine to operate on a rectangular strip of paper P with unit width and length $m(3^m)(2(2^n)+2)$. We will conceptually divide P evenly into unit-square *cells*. We will associate two cells with each possible $d(i,j,k)$, with the $m(3^m)2$ remaining cells not associated with any $d(i,j,k)$. We will cut a hole in the center of $m(3^m)(2^n)$ cells of P, resulting in a modified paper P', cutting a hole exactly when a cell has a $d(i,j,k)$ associated with it which evaluates to 1.

To decide whether the 3SAT instance has a satisfying assignment, we will give a folding of P' such that every

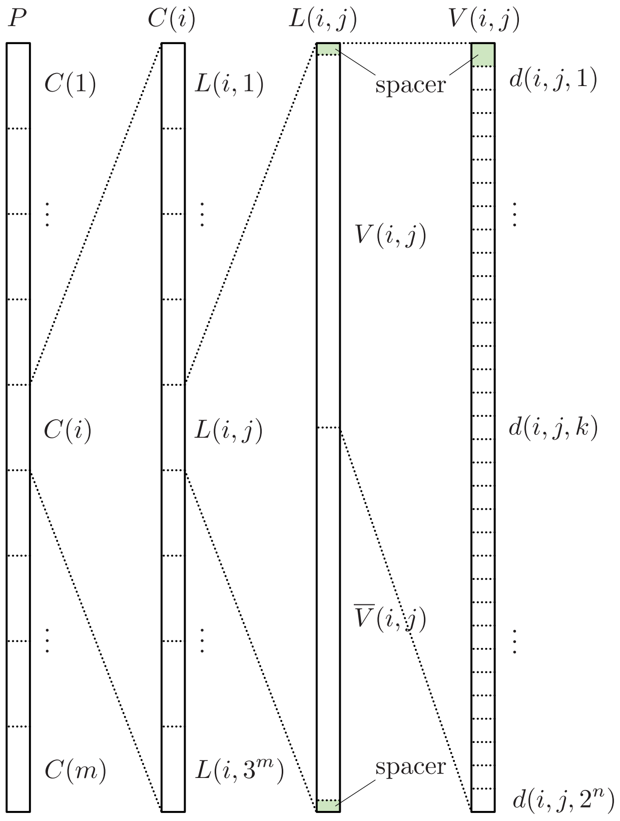

Figure 1: Layout of blocks and cells of P.

cell overlaps with exactly $m-1$ other cells. Further, any cell associated with $d(i,j,k)$ overlaps with cells associated with each $d(i',j,k)$ for $i' \in [1,m]$. Thus, if the folding has a through-hole passing through a cell corresponding to $d(i,j,k)$, then DNF clause j evaluates to 1 under variable assignment k. And if a cell associated with $d(i,j,k)$ does not have a through-hole, then DNF clause j evaluates to 0 under variable assignment k. A LOOK operation restricted to the folded location of a cell associated with $d(i,j,k)$ will evaluate the formula

$$\bigwedge_{i=1}^{m} d(i,j,k), \tag{3}$$

while the entire LOOK operation performs the same test in parallel for all DNF clauses j, over all variable assignments k. We will explicitly evaluate each of the exponentially many $d(i,j,k)$ values in some cell, but we will be able to produce them using only polynomially many operations.

3 Hole Locations

First, we will describe how cells of P are associated with each $d(i,j,k)$; see Figure 1. Conceptually, we will divide P into consecutive equally-sized sets of cells called *blocks*. Dividing P into a different number of equally-sized sets will partition P at different levels of detail, into blocks of different size. At the coarsest level, the paper is divided into m *CNF blocks*, one for each 3CNF clause. Each CNF block $C(i)$ contains 3^m literal blocks, one for each DNF clause. Each literal block $L(i,j)$ contains two variable blocks $V(i,j)$ and $\overline{V}(i,j)$, which are mirror reflections of each other. Each variable block $V(i,j)$ is made up of 2^n+1 cells.

The first cell $\sigma(i,j,1)$ in a variable block is a blank cell not associated with any $d(i,j,k)$, which we call a *spacer*. The remaining 2^n cells in variable block $V(i,j)$ are associated with $d(i,j,k)$, one for each variable assignment, with cell $\sigma(i,j,k+1)$ associated with $d(i,j,k)$. The desired hole pattern P' contains a hole in a cell exactly when it is associated with a $d(i,j,k)$ equal to 1. We call a cell associated with a $d(i,j,k)$ equal to 1 a *hole cell*, with *blank cells* referring to any other cell. We will show in Section 4 that we can fold P' so that a LOOK operation can decide the 3SAT instance, and in Section 5 we can produce P' from P in polynomially many operations.

We have not yet fully specified the hole pattern for P' because we have not fixed an ordering for the DNF clauses or the variable assignments. For our construction, the DNF clauses and the variable assignments must be ordered in a highly symmetric way to allow blocks to be aligned with a polynomial number of folds, though the CNF clauses may be ordered arbitrarily. We order the DNF clauses in the following way. Let DNF clause $\mathcal{D}(j)$ be defined as:

$$\mathcal{D}(j) = \bigwedge_{i=1}^{m} l(i,j), \tag{4}$$

where $l(i,j)$ represents the literal from CNF clause i appearing in DNF clause j, according to:

$$l(i,j) = \begin{cases} a_i & \text{if } \left\lfloor \frac{j-1}{3^{m-i}} \right\rfloor \equiv 0 \text{ or } 5 \mod 6 \\ b_i & \text{if } \left\lfloor \frac{j-1}{3^{m-i}} \right\rfloor \equiv 1 \text{ or } 4 \mod 6 \\ c_i & \text{if } \left\lfloor \frac{j-1}{3^{m-i}} \right\rfloor \equiv 2 \text{ or } 3 \mod 6 \end{cases}. \tag{5}$$

Conceptually, this order corresponds to the following layout. Each CNF block $C(i)$ contains 3^{i-1} *unit blocks* made up of three adjacent *subunit blocks*, one for each literal in $\{a_i, b_i, c_i\}$. Each subunit block contains 3^{m-i} literal blocks, all of which are associated with the same literal. Further, every unit block is a reflection of each adjacent unit block.

For variable assignments, we represent an assignment as a binary string and list them in lexicographical order, with the first variable being the left most digit in the binary string. So for a 3SAT instance with 5 variables, the first variable assignment among the 2^n assignments will be 00000, while the tenth variable assignment will be 01001.

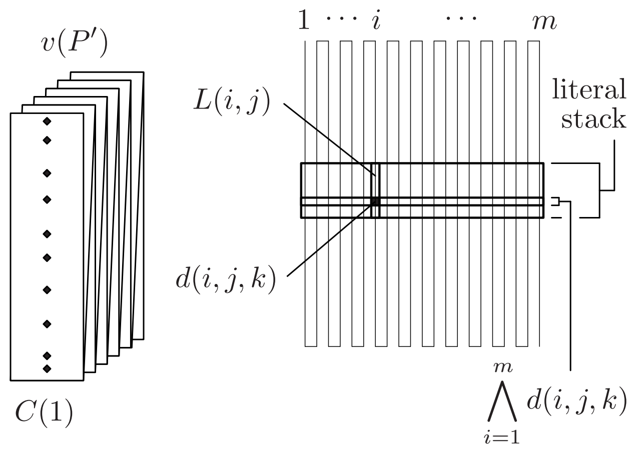
$v(P')$
$C(1)$
$1 \cdots i \quad \cdots \quad m$
$L(i,j)$
literal stack
$d(i,j,k)$
$\bigwedge_{i=1}^{m} d(i,j,k)$

Figure 2: Flat folding $v(P')$ aligns cell stacks containing cells corresponding to $d(i,j,k)$ for all $i \in [1,m]$, and in doing so, enables verification of the instance using a single LOOK operation.

4 Verification

Let $v(P')$ be a flat folding of P' produced by pleating the m CNF blocks back and forth on top of each other. This folding can easily be produced by a sequence of simple folds by folding each crease in order from top to bottom.

Flat-folding $v(P')$ has m layers uniformly at every point. In particular, every cell folds onto $m-1$ other cells, while each literal block overlaps and aligns with $m-1$ other literal blocks, one from each CNF block. We call a set of aligned and overlapping blocks of the same size a *stack*. For example, we call m overlapping literal blocks a literal stack.

Lemma 2 *Flat-folding $v(P')$ has a through-hole if and only if the 3SAT instance associated with P' is satisfiable.*

Proof. We first prove a bijection between literal stacks and DNF clauses, by showing that, for any DNF clause $\mathcal{D}(j)$, there exists a literal stack containing literal blocks corresponding to the CNF clause literals present in $\mathcal{D}(j)$. Let $l(i,j)$ be the literal in $\mathcal{D}(j)$ associated with CNF clause i. CNF block $C(1)$ contains a single unit block with three subunit blocks, one for each literal. If some literal stack represents $\mathcal{D}(j)$, it must contain a literal block from the subunit block corresponding to literal $l(1,j)$. Now consider CNF block $C(i)$ for $i>1$. By construction, every unit block of $C(i)$ is exactly the same size as a subunit block of $C(i-1)$, and every subunit block of $C(i-1)$ will exactly align and overlap with some unit block of $C(i)$ in $v(P')$. Thus by induction, there exists a literal stack containing each $l(i,j)$ in $\mathcal{D}(j)$. There are 3^m DNF clauses and 3^m literal stacks, so there is a bijection between them.

Lastly, given a literal stack corresponding to DNF clause $\mathcal{D}(j)$, we show that there is a through-hole in the stack if and only if the DNF clause is satisfiable. Each literal block is mirror symmetric containing a variable block $V(i,j)$ and its reflection $\overline{V}(i,j)$, and variable blocks exactly overlap other variable blocks in two symmetric stacks. In particular, within a variable stack, each cell associated with $d(i,j,k)$ overlaps a cell corresponding to every other $d(i',j,k)$ for $i' \in [1,m]$, and $d(i,j,k)$ has a hole exactly when $d(i,j,k)$ is 1 by construction. Thus, if $\mathcal{D}(j)$ evaluates to 1 using variable assignment k, then there will be a through-hole, and there will be no through-hole if some $d(i,j,k)$ in the variable stack is 0, completing the proof. □

5 Making Holes

In this section, we show how to produce the holes in P' from paper P in $O(nm+m^2)$ FOLD operations and one CUT operation. We will fold P in two stages. First, we will show how to fold a CNF block $C(i)$ into three literal stacks, with each literal stack containing the 3^{m-1} literal blocks in $C(i)$ corresponding to the same literal. Second, we will show how to fold a literal block $L(i,j)$ to align all hole cells to the same location.

5.1 Align Literals

Given a CNF block $C(i)$, we align literal blocks corresponding to the same literal by folding along a sequence of *symmetric two-fold pleats*. A symmetric two-fold pleat folds a block along two lines dividing the block into equal thirds. The upper crease will be a valley fold, and the lower crease will be a mountain fold.

Lemma 3 *We can fold any CNF block $C(i)$ into three adjacent literal stacks using a sequence of $2(i-1)+6(m-i)$ simple folds, with each literal stack containing the 3^{m-1} literal blocks corresponding to the same literal.*

Proof. The folding follows directly from the order of $\mathcal{D}(j)$ defined by $l(i,j)$; see Figure 3. First, we overlap every unit block on top of one another by repeatedly folding through all layers along $i-1$ symmetric two-fold pleats. These folds overlap the unit blocks in the same orientation because unit blocks alternate in orientation in the layout defined by $l(i,j)$. Then, we overlap the literal blocks in each of the three adjacent subunit blocks by using $m-i$ symmetric two-fold pleats. Each symmetric two-fold pleat can be folded using two simple folds by first valley folding along the higher fold, then folding back down along the lower fold. The first step uses $2(i-1)$ FOLD operations, while the second step uses $3(2)(m-i)$. □

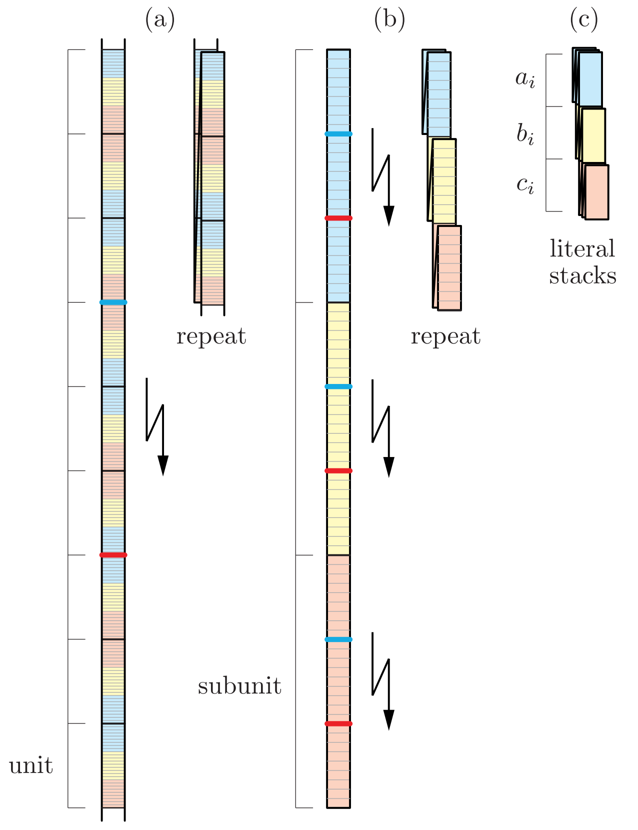

Figure 3: Aligning literal blocks associated with the same literal in the CNF clause corresponding with CNF block $C(i)$. (a) Align unit blocks from $C(i)$ on top of each other using $i-1$ pleats. (b) Collapse each subunit stack into one literal stack using $m-i$ pleats. (c) The folding after aligning the literal blocks into three stacks.

5.2 Align Hole Locations

We align hole locations of a literal block using the following procedure; see Figure 4. First, valley fold the literal block down along its lower boundary, and fold back up along the line separating $V(i,j)$ and $\overline{V}(i,j)$. Because $V(i,j)$ and $\overline{V}(i,j)$ are symmetric (recall that they are mirror reflections of each other), the resulting flat-folded pleat will have through-holes at the same locations as $V(i,j)$.

Lemma 4 *We can fold any variable block $V(i,j)$ onto the first three cell locations of the block using at most $2n$ simple folds, such that a cell folds to the second cell location from the top if and only if it is a hole cell.*

Proof. We prove by constructing such a folding. By definition, each variable block comprises a spacer cell followed by 2^n cells, with cell $\sigma(i,j,k+1)$ associated with $d(i,j,k)$. Also, each variable block is associated with some variable x_r or its negation. Because of the lexicographical order of the variable assignments, the

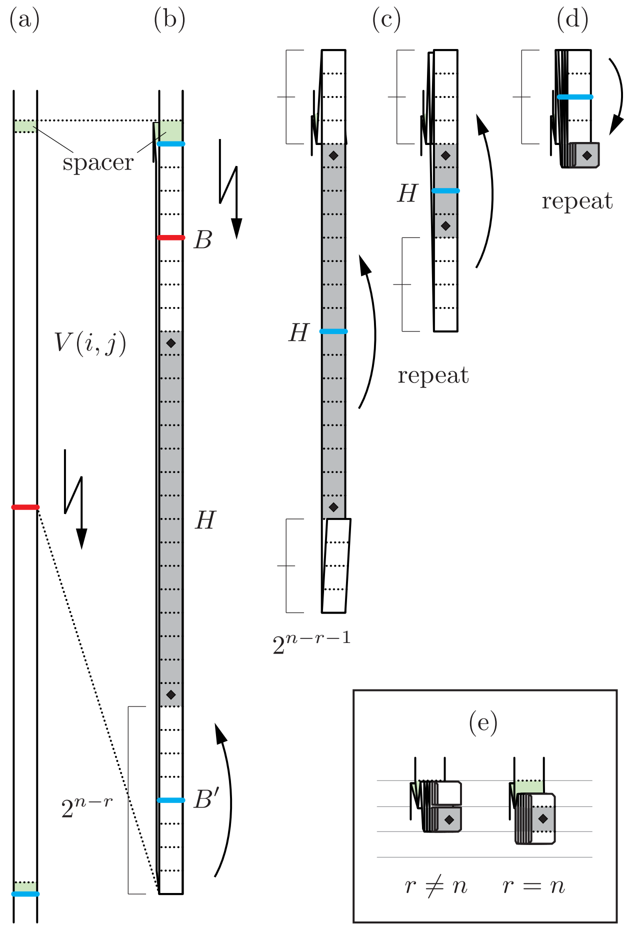

Figure 4: Aligning hole cells in literal block $L(i,j)$. (a) Fold $L(i,j)$ to align variable blocks. (b) Pre-process leading or trailing blank cells B or B' of $V(i,j)$. (c) Align hole cells by folding H in half $n-1$ times. (d) Get rid of extra blank cells by folding the excess with height 2^{n-r-1} in half $n-r-1$ times. (e) Final state for two cases, when $r \neq$ and when $r = n$.

hole pattern described by $d(i,j,k)$ within the variable block is an alternating sequence of 2^{n-r} hole cells and 2^{n-r} blank cells. If the variable block is associated with variable x_r, then cell $\sigma(i,j,1)$ is a blank cell, while if the block is associated with $\neg x_r$, then $\sigma(i,j,1)$ is a hole cell.

First, we perform a preprocessing step. Let B be the block of blank cells between the spacer and the first hole cell exclusive. If B is nonempty, it has length 2^{n-r}. Valley fold B up along its top boundary line, and valley fold it back down along the line dividing its cells equally. This aligns the first hole cell onto the second cell location. Otherwise, let B' be the block of blank cells after the last hole cell. If B' is nonempty, it has length 2^{n-r}. Valley fold B' in half, bringing its bottom edge to its top. Note that exactly one of B and B' will be nonempty.

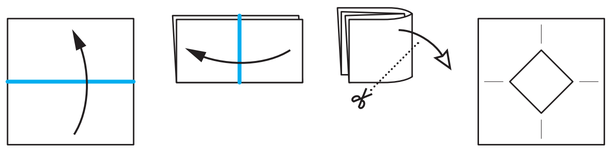

Figure 5: How to cut a diamond-shaped hole.

Second, we perform hole alignment. Let H be the block of cells between the first hole cell and the last hole cell inclusive. Because the hole pattern is an alternating sequence of 2^{n-r} hole cells and blank cells, H is symmetric. Valley fold H in half by folding the bottom half of H along with any attached blank cells upward. Because H is symmetric, hole cells of this variable block will only overlap other hole cells from the block. Because of the alternation and symmetry of the hole pattern, the block of cells on the top-most layer of this resulting flat folding between the first hole cell and the last hole cell will again be symmetric. So we can repeatedly fold the block containing the hole cells in half until all hole cells are aligned in the second cell location. After repeating this procedure, there will either be no paper below the second cell, or if $r = n$, half a cell of paper will extend below the second cell. In any case, no paper from $V(i,j)$ will extend below the third cell location.

Next, we fold away blank cells extending above the first cell location. After we align the hole cells, excess blank cells will exist above the second cell location. These cells extend above the second hole location by length 2^{n-r-1}, equal to half the width of a 2^{n-r} block of blank cells. We can fold all empty cells extending above the second cell location onto the spacer cell by folding the excess cells in half $n-r-1$ times, or not at all if $n-r-1 < 1$.

The preprocessing step requires either 1 or 2 simple folds, the hole alignment steps require $n-1$ simple folds, while collapsing empty cells requires at most $n-r-1$ simple folds. Since $r \in [1,n]$, this folding uses at most $2n$ folds. □

5.3 Cutting a Hole

Lemma 5 *We can make a diamond-shaped hole centered on a hole location on the interior of a flat folding using two* FOLD *operations and one* CUT *operation.*

Proof. See Figure 5. Fold in half horizontally through the hole location and then vertically though the same point. Then cut off the paper containing the hole location with a cut line at 45°. □

On a fold-and-punch machine, this lemma can be replaced by a direct PUNCH operation.

6 Proof of Theorem

Now we are ready to prove Theorem 1.

Proof. We produce hole pattern P' from P according to a given 3SAT instance by doing the following. Fold each CNF block of P to align literal blocks as described in Lemma 3, which folds CNF block $C(i)$ using $2(i-1)+6(m-i)$ FOLD operations, for a total of $4(m^2-m)$. Then, for each of the $3m$ resulting literal stacks, pleat it in half to align each variable block $V(i,j)$ and $\overline{V}(i,j)$ onto each other, and align hole locations as described in Lemma 4, using at most $2n$ FOLD operations per literal stack. Perform this step on literal stacks starting from the lowest literal stack up to the highest. This ordering ensures that we are always performing simple folds on a set of topmost layers. The result is a flat folding contained within the space of three cells, with all hole cells overlapping at the second cell location. Then use the construction in Lemma 5 to construct a hole centered on the second cell location, using two FOLD operations and one straight CUT. This folding uses $O(nm+m^2)$ FOLD operations.

Now unfold to P' using $O(nm+m^2)$ UNFOLD operations. Then, folding P' to $v(P')$ using $m-1$ FOLD operations results in a flat folding which, by Lemma 2, has a through-hole if and only if the original 3SAT instance is satisfiable. Thus a single LOOK operation completes the computation. $\square$

7 Open Problems

We suggest some open problems for future study. What is the full computation power of a fold-and-cut machine? We can simulate t time units of a fold-and-cut machine on a RAM in $2^{O(t)}$ time. Can a polynomial-time fold-and-cut machine simulate all of EXPTIME, or at least PSPACE? The model seems related to parallel computation; a natural goal would be to solve problems in EXPTIME representable by a polynomial-depth circuit. More generally, if we allow input coordinates to be arbitrary real numbers, do we get $\mathbb{R}$ versions of complexity classes such as $\mathrm{NP}_{\mathbb{R}}$ [BCSS98]? Is it possible to solve NP with just FOLD/UNFOLD operations, without a CUT? (This may require adding a different query operation to the model.)

Acknowledgments The authors would like to thank Alex Cornejo, Eli Groban, Owen Macindoe, Shuhei Miyashita, Jimmy Wong, and Damien Woods for their insightful discussions and support.

References

[ABD+04] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S. B. Mitchell, Saurabh Sethia, and Steven Skiena. When can you fold a map? *Computational Geometry: Theory and Applications*, 29(1):23–46, 2004.

[ADD+] Yasuhiko Asao, Erik D. Demaine, Martin L. Demaine, Hideaki Hosaka, Akitoshi Kawamura, Tomohiro Tachi, and Kazune Takahashi. Folding and punching paper. *Journal of Information Processing.* To appear. Special issue of papers from the 19th Japan Conference on Discrete and Computational Geometry, Graphs, and Games, September 2016.

[ADK] Hugo Akitaya, Erik D. Demaine, and Jason S. Ku. Simple folding is really hard. *Journal of Information Processing.* To appear. Special issue of papers from the 19th Japan Conference on Discrete and Computational Geometry, Graphs, and Games, September 2016.

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation.* Springer-Verlag, 1998.

[BDEH01] Marshall Bern, Erik Demaine, David Eppstein, and Barry Hayes. A disk-packing algorithm for an origami magic trick. In *Origami^3: Proceedings of the 3rd International Meeting of Origami Science, Math, and Education*, pages 17–28. A K Peters, 2001.

[CDD+11] Jean Cardinal, Erik D. Demaine, Martin L. Demaine, Shinji Imahori, Tsuyoshi Ito, Masashi Kiyomi, Stefan Langerman, Ryuhei Uehara, and Takeaki Uno. Algorithmic folding complexity. *Graphs and Combinatorics*, 27(3):341–351, 2011.

[DDH+10] Erik D. Demaine, Martin L. Demaine, Andrea Hawksley, Hiro Ito, Po-Ru Loh, Shelly Manber, and Omari Stephens. Making polygons by simple folds and one straight cut. In *Revised Papers from the China-Japan Joint Conference on Computational Geometry, Graphs and Applications*, Lecture Notes in Computer Science, pages 27–43, Dalian, China, November 2010.

[DDL98] Martin L Demaine, Erik D Demaine, and Anna Lubiw. Folding and cutting paper. In *Revised Papers from the Japan Conference on Discrete and Computational Geometry*, volume 1763 of *Lecture Notes in Computer Science*, pages 104–118, December 1998.

[DO08] Erik D Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge University Press, August 2008.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[Ueh11] Ryuhei Uehara. Stamp foldings with a given mountain-valley assignment. In *Origami^5: Proceedings of the 5th International Meeting of Origami Science, Math, and Education*, pages 585–597. A K Peters/CRC Press, November 2011.