Synthesizing Optimal Switching Lattices

GRAEME GANGE, HARALD SØNDERGAARD and PETER J. STUCKEY,
Department of Computing and Information Systems, The University of Melbourne, Vic. 3010, Australia

The use of nanoscale technologies to create electronic devices has revived interest in the use of regular structures for defining complex logic functions. One such structure is the switching lattice, a two-dimensional lattice of four-terminal switches. We show how to directly construct switching lattices of polynomial size from arbitrary logic functions; we also show how to synthesize minimal-sized lattices by translating the problem to the satisfiability problem for a restricted class of quantified Boolean formulas. The synthesis method is an anytime algorithm which uses modern SAT solving technology and dichotomic search. It improves considerably on an earlier proposal for creating switching lattices for arbitrary logic functions.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis*; *Optimization*; G.2.3 [**Discrete Mathematics**]: Applications

General Terms: Algorithms, Design

Additional Key Words and Phrases: SAT, Switching lattice, Logic synthesis

ACM Reference Format:
Graeme Gange, Harald Søndergaard and Peter J. Stuckey. 2014. Synthesizing Optimal Switching Lattices. *ACM Trans. Des. Autom. Electron. Syst.* V, N, Article A (January YYYY), 13 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

We study the combinatorial problem of switching lattice synthesis. The idea of using regular arrays of switches, each cell connected to its neighbours, is old and has many forms. For example, in the "rectangular logic arrays" of Akers [1972], each switch is a relatively complex 3-input 2-output unit implementing either the median-of-3 function or the Boolean if-then-else function, and switches are arranged so that a current can flow top-down and left-to-right only (for appropriate orientation of the array).

The potentials of nanoscale technologies [Cui and Lieber 2001; Chen et al. 2003; Eshaghian-Wilner et al. 2006] have created interest in types of "crossbar" devices that can operate based on, say, molecular electronics. In this context a "switch" is ideally a very simple device, such as a four-terminal switch [Altun and Riedel 2012]. The four terminals of the switch link to the four neighbours of a lattice cell, so that these are either all connected (when the switch is on), or disconnected (when the switch is off). The direction of flow in the resulting lattice structure has fewer constraints, compared to the lattices considered by Akers, allowing for paths that meander through the lattice.

In more detail, an n-by-m switching lattice is a wire mesh that results from laying n parallel wires perpendicularly on top of m parallel wires, with a four-terminal switch inserted wherever two wires cross. (We picture the n wires laid down horizontally, so that we have n rows and m columns of switches.) The switching lattice can be used to

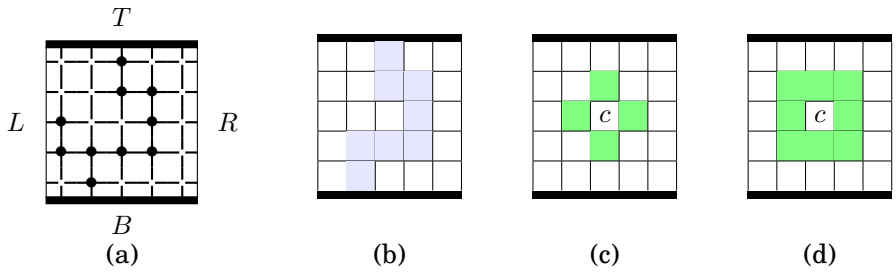


Fig. 1: (a) a 5-by-5 switching lattice, (b) its connecting path highlighted, (c) the four neighbouring cells to which cell c is 4-connected, and (d) the eight neighbouring cells to which c is 8-connected

implement a Boolean function by, firstly, connecting all the wire ends at the top to a "top plate" T and all the wire ends at the bottom to a "bottom plate" B and, secondly, having each four-terminal switch controlled by an input, see Figure 1(a). An input wire may be inverted, so we associate a Boolean literal, x or $\neg x$, with each four-terminal switch. An input may also be held constant (0 or 1), so in what follows, each switch, or wire junction, may be labelled with some literal ℓ, 0, or 1.

Some input combinations will create a (possibly meandering) path between the top and bottom plates; we associate those combinations with output being 1, see Figure 1(b). Combinations that leave the plates unconnected correspond to output 0.

A switching lattice can similarly be equipped with "left" and "right" plates and left-to-right connectivity can be associated with a Boolean function [Altun and Riedel 2012]. This way, a single lattice can implement two different functions, although not simultaneously. We will not make use of that in this paper, but it is worth keeping in mind that this provides a second aspect on minimisation, as it can be used to minimise the number of lattices required to implement a *set* of functions.

The plates T and B are connected if and only if it is possible to travel between T and B by stepping on a sequence of 4-connected *on* cells, see Figure 1(c). We call such a sequence a TB-path. As observed by Altun and Riedel [2012], there is no 4-connected *on* path between T and B if and only if there is a sequence of 8-connected *off* cells (Figure 1(d)) that connect L and R. We call such a sequence an *LR-path*.

Figure 2 shows an implementation of $(x_1 \wedge x_4) \vee (x_2 \wedge x_3)$.[1] To see that this is the function implemented, note that a connecting (TB-) path corresponds to a conjunction of literals, and the existence of a connecting path corresponds to a disjunction of those conjunctions, one disjunct per possible path. Enumerating the possible paths therefore leads to a sum-of-products representation of the Boolean function. Hence for the left lattice, we have (writing conjunction as juxtaposition): $\varphi = x_1x_4 \vee x_2x_3 \vee \neg x_1x_2x_3 \vee x_1x_2x_3x_4 \vee \neg x_1x_2x_3 \vee x_1x_2x_3x_4 \vee \neg x_1x_2x_3$, which is readily simplified to $(x_1 \wedge x_4) \vee (x_2 \wedge x_3)$. The 3-by-3 switching lattice in Figure 2 is not minimal. The 2-by-2 lattice in the same figure implements the same function $(x_1 \wedge x_4) \vee (x_2 \wedge x_3)$. We shall soon see a less trivial example of this kind of equivalence.

In this paper we are interested in the automated synthesis of minimal lattices. Altun and Riedel [2012] propose a synthesis method for switching lattices—henceforth referred to as the *dual-product construction*, or DP—as follows. For the target function φ, find an irredundant sum-of-products (ISOP) form ψ of φ. Similarly let ψ' be φ's

[1] We use $\wedge$, $\vee$, and $\neg$ for conjunction, disjunction, and negation, resp.

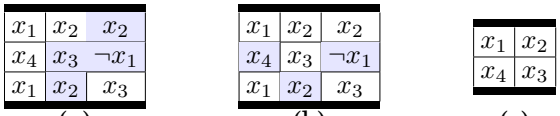


Fig. 2: (a) a 3-by-3 switching lattice with a TB-path highlighted, (b) an LR-path highlighted, and (c) an equivalent 2-by-2 switching lattice.

$$
\begin{aligned}
c_1 &: \quad (\neg x_2 \wedge \neg x_5 \wedge \neg x_6 \wedge x_7 \wedge \neg x_8)\\
c_2 &: \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_4 \wedge x_5)\\
c_3 &: \vee (x_3 \wedge \neg x_5 \wedge \neg x_6)\\
c_4 &: \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_4 \wedge x_8)\\
c_5 &: \vee (\neg x_1 \wedge x_3)\\
c_6 &: \vee (\neg x_2 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge x_8)\\
c_7 &: \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_5 \wedge x_7 \wedge \neg x_8)
\end{aligned}
$$

(a)

$$
\begin{aligned}
r_1 &: \quad (\neg x_2 \wedge x_3)\\
r_2 &: \vee (\neg x_1 \wedge x_6)\\
r_3 &: \vee (x_3 \wedge x_5 \wedge x_7 \wedge x_8)\\
r_4 &: \vee (x_3 \wedge \neg x_4 \wedge \neg x_8)\\
r_5 &: \vee (x_3 \wedge \neg x_4 \wedge \neg x_5)\\
r_6 &: \vee (\neg x_1 \wedge \neg x_5)
\end{aligned}
$$

(b)

Fig. 3: ISOP form formulas for (a) a function φ and (b) its dual

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
r_1	$\neg x_2$	$\neg x_2$	x_3	$\neg x_2$	x_3	$\neg x_2$	$\neg x_2$
r_2	$\neg x_6$	$\neg x_1$	$\neg x_6$	$\neg x_1$	$\neg x_1$	$\neg x_6$	$\neg x_1$
r_3	x_7	x_5	x_3	x_8	x_3	x_8	x_7
r_4	$\neg x_8$	$\neg x_4$	x_3	$\neg x_4$	x_3	$\neg x_4$	$\neg x_8$
r_5	$\neg x_5$	$\neg x_4$	x_3	$\neg x_4$	x_3	$\neg x_4$	$\neg x_5$
r_6	$\neg x_5$	$\neg x_1$	$\neg x_5$	$\neg x_1$	$\neg x_1$	$\neg x_5$	$\neg x_1$

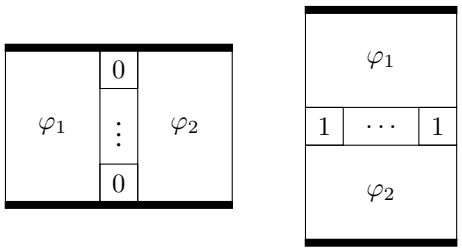
$\neg x_1$	$\neg x_1$	$\neg x_6$	$\neg x_1$
$\neg x_2$	$\neg x_4$	$\neg x_5$	1
$\neg x_4$	$\neg x_2$	$\neg x_8$	x_3
x_8	x_7	x_5	x_3

Fig. 4: A lattice generated by the DP construction (left) and a smaller but semantically equivalent lattice generated by our method (right)

dual (that is, φ' s.t. $\varphi(x_1, \ldots) = \neg\varphi'(\neg x_1, \ldots)$) in ISOP form. Let $t_1 \vee \cdots \vee t_m = \psi$ and $t'_1 \vee \cdots \vee t'_n = \psi'$. Now form an n-by-m switching lattice $\{A[i,j] \mid 1 \leq i \leq n, 1 \leq j \leq m\}$ by setting the input for $A[i,j]$ to some literal ℓ which is shared by t'_i and t_j. (The fact that ψ and ψ' are duals guarantees that such a shared ℓ exists for all i and j.)

Example 1.1. Consider a particular function φ, say, the function we have denoted b12.02 in Table I. Altun and Riedel use Espresso [Brayton 1984] to find an ISOP form for the function, such as the form shown in Figure 3(a), together with an ISOP form for its dual, say, the one shown in Figure 3(b). A switching lattice A for φ is now created by selecting, for entry $A[i,j]$, any literal that is shared by r_i and c_j, the ith product of the dual and the jth product of φ. Figure 4 (left) shows the resulting switching lattice.

The DP construction gives a systematic way of building the lattice, and it comes with the dual function implemented for free. However, it tends to lead to non-minimal lattices, partly because it ties the dimensions of the lattices to the sizes of ISOP form formulas, and partly because it does not utilise Boolean constants as input.



(a) (b)

Fig. 5: Constructing lattices (a) for $\varphi_1 \vee \varphi_2$, and (b) for $\varphi_1 \wedge \varphi_2$.

In this paper we compare the ISOP-based method with an "anytime" method which uses reduction to SAT. Our method uses the ISOP-based approach to find an upper bound on dimensions. It then searches for successively better implementations until either an optimal solution is found, or else a preset time limit has been exhausted. In our experiments, the alternative method can decrease lattice sizes considerably; Tables I and II show cases of reduction by 79%. For Figure 3's φ, our method generates the smaller switching lattice shown in Figure 4 (right). The reader is encouraged to ponder the possible top-to-bottom paths through this smaller switching lattice and test the equivalence with Figure 4 (left). The contributions of this paper are:

— an "anytime" algorithm for construction of switching lattices for arbitrary logic functions, based on reduction to SAT and dichotomic search;
— the first algorithm we are aware of for creating minimal-size switching lattices; and
— experiments showing that minimal-size switching lattices are often substantially smaller than those created by the DP construction.

In Section 2 we show how to construct switching lattices in a compositional way. In Section 3 we show that the problem of deciding the satisfiability of a Boolean function, represented by a switching lattice, is NP-complete. Section 4 gives details of how we synthesize minimal lattices using reductions to SAT. Section 5 provides experimental results, and Section 6 concludes.

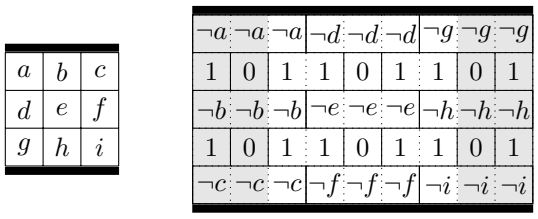
2. COMPOSITION OF SWITCHING LATTICES

Given switching lattices L_1 and L_2 implementing functions φ_1 and φ_2, we can easily construct lattices for $\varphi_1 \vee \varphi_2$ and $\varphi_1 \wedge \varphi_2$, as illustrated in Figure 5.

In some cases, we can avoid introducing the padding rows/columns. When constructing $\varphi_1 \wedge \varphi_2$, we can omit the padding row if both lattices are of width 1. Similarly, we can omit padding for $\varphi_1 \vee \varphi_2$ if both lattices are of height at most 2.

This composition allows us to directly construct a switching lattice implementing an arbitrary Boolean formula φ—we convert φ to negation normal form (NNF), and then construct the lattice in a bottom-up fashion. Let $|\varphi|$ denote the number of $\{\wedge, \vee\}$ terms in the input encoding of φ. The transformation to NNF does not increase $|\varphi|$; therefore in the worst case, this yields a lattice of size $O(|\varphi|^2)$.

We can also directly construct a lattice for the complement $\neg\varphi$. However, the transformation is slightly more involved; given a lattice L for φ, we must construct some lattice L' that is connected for every truth assignment that makes L disconnected, and *vice versa*. The difficulty is the asymmetry in connectivity; as falsifying (LR-) paths in φ are 8-connected, each literal in $\neg\varphi$ must be connected to all 8 neighbours. We can simulate this by padding each literal with blocks of 1; however, we must be careful not



(a) (b)

Fig. 6: Constructing the complement of a 3×3 lattice.

to permit the signal to *skip* literals. We can achieve this by transposing the original lattice, and replacing every element with at most 3 copies, and adding rows of 0 and 1 to connect diagonals. Example 2.1 shows the idea.

Example 2.1. Consider the 3×3 lattice L shown in Figure 6(a). Every TB-path path in L (say, $[a, d, e, h]$) must correspond to an LR-path in L'. Figure 6(b) shows a possible construction for L'. The added 1 blocks ensure that $\neg e$ is connected to all eight neighbours on vertical paths, but horizontal (false) paths *must* pass through the originally 4-connected literals. We could discard the left- and right-most (shaded) columns; they are included to better illustrate the repeated structure of the construction.

Given an $n \times m$ lattice L, this construction will produce a $(2m-1) \times 3n$ lattice L' for the complement. More precisely, we define, for $i \in [1, 2m-1]$ and $j \in [1, 3n]$:

$$L'(i,j) = \begin{cases} \neg L(\lceil \frac{j}{3} \rceil, \frac{i+1}{2}) & \text{for } i \text{ odd}, j \in [1, 3n] \\ 0 & \text{for } i \text{ even}, j = 3k-1, k \in [1, n] \\ 1 & \text{for } i \text{ even}, j \neq 3k-1, k \in [1, n] \end{cases}$$

3. SATISFIABILITY FOR SWITCHING LATTICES

At a first glance, the regular shape of a switching lattice may suggest that satisfiability may be easier determined in this form. We show, however, that it is no easier than generalised SAT.

Define the problem LATTICE-SAT as given a switching lattice, determine if there is an assignment θ which connects the top T to the bottom B.

THEOREM 3.1. LATTICE-SAT *is NP-complete.*

Proof. Given a candidate assignment θ, we can easily test if θ satisfies the lattice L by checking if there is some path between the top and bottom plates using only literals that are true under θ. LATTICE-SAT is therefore in NP.

We use reduction from 3SAT to show that LATTICE-SAT is NP-hard. Consider an instance $P = \langle V, C \rangle$ of 3SAT, with $C = \{\{l_{11}, l_{12}, l_{13}\}, \ldots\}$. We build a $3\times(2n-1)$ switching lattice L as shown to the right, by introducing a row for each clause, and adding a row of 1s between successive clauses. This construction is linear in the size of P.

l_{11}	l_{12}	l_{13}
1	1	1
l_{21}	l_{22}	l_{23}
1	1	1
l_{31}	l_{32}	l_{33}
⋮		

Consider some assignment satisfying P. For each clause, there is some literal which is true. In the corresponding row of L, that literal will also be true, connecting the adjacent 1 rows. Since such a literal exists for every row, there must be some vertical path through the lattice. Conversely, consider an assignment θ which admits a path through L. Each row must have some true literal, connecting the adjacent 1 rows. Since each clause in

P has a corresponding row in L, θ must also satisfy each clause in P. Therefore, L is satisfiable iff P is satisfiable. Hence LATTICE-SAT is NP-hard.

We conclude that LATTICE-SAT is NP-complete. □

4. SYNTHESIZING $N \times M$ SWITCHING LATTICES

The DP construction [Altun and Riedel 2012] computes ISOP formulas for the function φ and its dual φ', and constructs a lattice with one column for each term in φ, and one row for each term in φ'. While this construction is general, there are drawbacks. First, the resulting lattice may become quite large, as typically at least one of φ and φ' will require exponentially many terms, and the circuit is formulated so that there can be no sharing between paths. Second, it prevents us from using arbitrary functions for the vertical and horizontal components; the "horizontal" function will always be the dual of the "vertical" function. Finally, it does not allow the use of fixed inputs.

We instead formulate the synthesis of an $n \times m$ lattice as a satisfiability problem in quantified Boolean logic, which can be solved by quantified Boolean formula (QBF) solvers, for example, DepQBF [Lonsing and Biere 2010]. We can then synthesize a minimal circuit by repeatedly solving formulas for varying values of n and m.

Roughly, the problem is formulated as:

$$\exists L \,.\, \forall V \,.\, \varphi(V) \Leftrightarrow \gamma(L, V)$$

where L is an $n \times m$ lattice, V is the set of inputs, and γ evaluates L under a given input. If the formula is true, the corresponding assignment to L gives us an implementation of φ. Let $V^+ = V \cup \{1\}$, and $S = V^+ \cup \{\neg v \mid v \in V^+\}$ be the possible literals over V together with 1 and 0. The crucial decision is, for each lattice position l_{ij}, which literal $\ell \in S$ to place at that position. We represent this decision by the literals $[\![l_{ij} = \ell]\!]$. *Some* literal must be used, so we require that $\forall i, j. \sum_{l \in S} [\![l_{ij} = \ell]\!] = 1$.

We must be aware of some subtleties in the formulation of γ. The obvious approach is to express a path from top to bottom using path variables p_{ij}. A lattice element can be on the path only if it evaluates to true:

$$p_{ij} \Rightarrow \bigvee_{v \in V^+} ([\![l_{ij} = v]\!] \wedge v) \vee ([\![l_{ij} = \neg v]\!] \wedge \neg v)$$

These must be connected to a neighbour to form a path. Let $adj_4(i, j)$ denote the positions adjacent to (i, j), that is

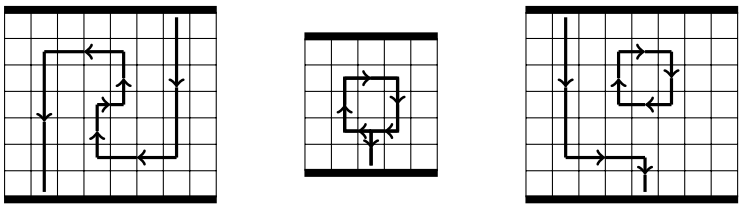
$$adj_4(i, j) = \{(i', j') \mid 1 \leq i' \leq n, 1 \leq j' \leq m, |i - i'| + |j - j'| = 1\}$$

Then each path position should be adjacent to another path position:

$$p_{ij} \Rightarrow \bigvee \{p_{i'j'} \mid (i', j') \in adj_4(i, j)\}$$

Unfortunately, this formulation gives rise to spurious paths such as that in Figure 7(b), where a set of elements provide mutual reachability, without any incoming edge. This is a common issue in path-finding problems, motivating the adoption of stable-model semantics [Gelfond and Lifschitz 1988] in answer-set programming solvers [Marek and Truszczyński 1999]. We are not aware of any QBF solvers that support stable model semantics.

Observe, however, that for any valid (non-redundant) path, each internal node on the path is adjacent to exactly two other nodes on the path. For a spurious path to B to exist, there must not only be a cycle of mutually-supporting elements, there must also be some path from the cycle to B – that is, there must be some element that is *both* part of a cycle, *and* has some path to B. This element has 3 neighbours in the path.



(a) (b) (c)

Fig. 7: (a) A complex but valid connection; (b) a spurious path permitted by the naive formulation; (c) an assignment that leads to a spurious path, but also contains a valid connection.

Another convenient simplification is that we can tolerate the existence of spurious paths, so long as *some* TB-path exists. Consider the case shown in Figure 7(c); even though there is a spurious cycle, the requirement for degree 1 elements at the source and destination ensures the existence of a valid path.

We then add the constraints:

$$(\sum_j p_{1j} \leq 1) \wedge (\sum_j p_{nj} \leq 1)$$

to ensure that a path cannot both start and end at the top or bottom. Moreover, we add

$$\forall\, j \;.\; \neg p_{2j} \Rightarrow \neg p_{1j}$$

$$\forall\, j \;.\; \neg p_{(n-1)j} \Rightarrow \neg p_{nj}$$

$$\forall\, i \in [2, n-1],\; j \;.\; ((\sum_{(i',j') \in adj_4(i,j)} p_{i'j'}) \neq 2) \Rightarrow \neg p_{ij}$$

Of these, the first two handle connectivity for the first and last row; elements in these rows are effectively only adjacent to one other element. The third handles connectivity of the remaining elements, ensuring that an intermediate element can only appear in a path if it has degree 2. We finally ensure that a path must exist whenever φ is true:

$$\varphi(V) \Rightarrow p_{n1} \vee \ldots \vee p_{nm}$$

We encode the sum expressions in these constraints using sorting networks (see for example, Asín et al. [2011]).

The formulation thus far can be trivially satisfied by setting every element to 0. We must also ensure that there is *no* path from T to B whenever $\varphi(V)$ evaluates to 0. As already pointed out, there is no path from T to B iff there is some falsifying LR-path. We can encode these LR-paths in exactly the same manner as the TB case; the only difference is that a path variable is forced to false if the switch is *connected*, and we must include additional adjacent vertices. We introduce new variables $\bar{p}_{ij}$ to represent the path from L to R. They require the corresponding lattice element to be false:

$$\bar{p}_{ij} \Rightarrow \bigvee_{v \in V^+} ([\![l_{ij} = v]\!] \wedge \neg x_v) \vee ([\![l_{ij} = \neg v]\!] \wedge x_v)$$

Let $adj_8(i,j)$ denote the set of positions that are either adjacent or diagonal to (i,j), that is,

$$adj_8(i,j) = \{(i',j') \mid 1 \leq i' \leq n, 1 \leq j' \leq m, |i-i'| \leq 1, |j-j'| \leq 1, (i,j) \neq (i',j')\}$$

and $sum_8(i,j) = \sum_{(i',j') \in adj_8(i,j)} \bar{p}_{i'j'}$. Then the constraints are

$$(\sum_i \bar{p}_{i1} \leq 1) \wedge (\sum_i \bar{p}_{im} \leq 1)$$

$$\forall\, i\ .\ (sum_8(i,1) \neq 1) \Rightarrow \neg \bar{p}_{i1}$$

$$\forall\, i\ .\ (sum_8(i,m) \neq 1) \Rightarrow \neg \bar{p}_{im}$$

$$\forall\, i,\ j \in [2, m-1]\ .\ (sum_8(i,j) \neq 2) \Rightarrow \neg \bar{p}_{ij}$$

$$\neg\varphi(V) \Rightarrow \bar{p}_{1m} \vee \ldots \vee \bar{p}_{nm}$$

4.1. Unfolding to SAT

Unfortunately, QBF solvers are nowhere near as robust as modern SAT solvers. We ran the QBF model over night using a standard off-the-shelf QBF solver on a 3×3 lattice with 7 inputs, and it had not completed in 16 hours. This is clearly unacceptable.

A simple solution is to unfold the universally quantified variables – that is, introduce a copy of $\varphi(V) \Leftrightarrow \gamma(L,V)$ for each input assignment θ to V. This gives us a quantifier-free formula that can be passed to a SAT solver.

This unfolding is clearly infeasible for circuits with many inputs, as we must introduce $O(2^{|V|}nm)$ path variables (as well as the corresponding sorting networks). However, each unfolded copy can be simplified considerably. $\varphi(V)$ becomes a constant – so we only need to encode either a positive or negative path for each copy, rather than both, and the connectivity constraint on p_{ij} becomes a set of clauses. Where previously we had p_{ij} and $\bar{p}_{ij}$, we now introduce variables p^θ_{ij} for each lattice element (i,j) and assignment θ. Where φ evaluates to 1, only connected elements may be on a path:

$$\forall\, \theta \in \Theta, \varphi(\theta) = 1\ .\ p^\theta_{ij} \Rightarrow \bigvee_{s \in S, \theta(s)=1} [\![l_{ij} = s]\!]$$

These path variables are constrained to form a TB-path as described above. And, if $\varphi(\theta) = 0$, only disconnected elements may be on a path:

$$\forall\, \theta \in \Theta, \varphi(\theta) = 0\ .\ p^\theta_{ij} \Rightarrow \bigvee_{s \in S, \theta(s)=0} [\![l_{ij} = s]\!]$$

These path variables are constrained to form an LR-path as described above.

4.2. Restricted formulation

The SAT formulation, as mentioned above, requires solving a series of SAT instances with sizes that grow exponentially with $|V|$. We would obviously prefer a model without this up-front cost.

Rather than maintain a separate set of path variables for each assignment, we would prefer to maintain only one path per term in the ISOP formulation. Unfortunately, in an arbitrary switching lattice, different assignments satisfying a given product may follow completely different energized paths; we must ensure that a path exists for all assignments satisfying the product. This leads to a restricted synthesis problem, where we forbid paths from passing through literals that are don't-care in the given product.

For each term $t = \wedge_k \ell_k$ in the ISOP of φ we construct a set of path variables p^t_{ij} which are definitely true in any model of this term, that is

$$p^t_{ij} \Rightarrow [\![l_{ij} = 1]\!] \vee \bigvee_k [\![l_{ij} = \ell_k]\!]$$

These path variables are constrained to form a TB-path.

Similarly for each term $t' = \wedge_k \ell'_k$ in the ISOP of the dual of φ we construct a set of path variables p^t_{ij} which are definitely false when this term is false, that is

$$p^{t'}_{ij} \Rightarrow [\![l_{ij} = 0]\!] \vee \bigvee_k [\![l_{ij} = \neg \ell'_k]\!]$$

These path variables are constrained to form an LR-path.

Example 4.1. Consider formulating a lattice for the function φ whose ISOP form is given in Figure 3(a), and ISOP form of the dual is given in Figure 3(b). Unfolding would normally create a set of path variables for each of the 128 different possible assignments to $\{x_1, \ldots, x_8\}$. In the restrict formulation we create 7 sets for the terms in the ISOP form and 6 sets for the terms in the ISOP form of the dual. For example for the term $x_3 \wedge \neg x_5 \wedge \neg x_6$ in the ISOP of φ we would create path variables that hold for any solution to this term

$$p^{x_3 \wedge \neg x_5 \wedge \neg x_6}_{ij} \Rightarrow [\![l_{ij} = 1]\!] \vee [\![l_{ij} = x_3]\!] \vee [\![l_{ij} = \neg x_5]\!] \vee [\![l_{ij} = \neg x_6]\!]$$

And for the term $\neg x_2 \wedge x_3$ in the ISOP of the dual of φ we create path variables that are constrained to hold when this term is false:

$$p^{\neg x_2 \wedge x_3}_{ij} \Rightarrow [\![l_{ij} = 0]\!] \vee [\![l_{ij} = x_2]\!] \vee [\![l_{ij} = \neg x_3]\!] .$$

Because the restricted form forces a single path for each assignment satisfying a term in the ISOP, or dissatisfying a term in the ISOP of the dual, it is not guaranteed to allow minimal solutions. For this formula the full formulation finds a better answer, but the size of the formula is substantially smaller, so the restricted formulation is solved much faster.

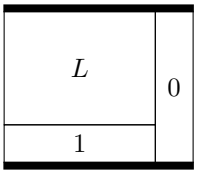
Note that the circuit computed by Altun and Riedel [2012] is a correct upper bound for the restricted formulation, since each column provides a correct path for the terms in the ISOP and each row a correct path for terms in the ISOP of the dual.

4.3. Synthesizing minimal lattices

Of course, these formulations only allow us to test whether there exists some $n \times m$ lattice implementing φ. Given that we have direct methods for computing a feasible solution L_U, we can simply progressively try all lattice sizes smaller than $|L_U|$, and return the smallest one that is feasible. We can improve this naive method by making the following observation:

THEOREM 4.2. *If some $n \times m$ lattice L implements φ, there is also an $n' \times m'$ lattice L' implementing φ, for any $n' \geq n$ and $m' \geq m$.*

Proof. Assume L is annotated with literals $l_{11}, l_{21}, \ldots, l_{nm}$. We now construct an $n' \times m'$ lattice L' by filling additional rows with 1, then filling additional columns with 0, as shown here. Any TB-path in L can be extended to a TB-path in L; similarly, any LR-path can be extended through the 0 elements in L'. Therefore, L' is equivalent to L. □



This means that, conversely, if we find that there is no $n \times m$ lattice implementing φ, we need never consider any strictly smaller lattices (that is, $n' \times m'$ for $n' \leq n, m' \leq m$). This approach will solve O($|L_U|$) subproblems.

In practice, the time to solve each subproblem grows exponentially with $n \times m$. When the size of the optimum lattice $\hat{L}$ is much smaller than $|L_U|$, we will solve many (expensive) feasible instances before eventually finding the optimum. It is better to use a

```
minimize_split(φ, max)
    lb := 1; ub := max
    best := none
    failed := ∅
    while (lb ≤ ub)
        mid := (lb+ub)/2
        found := false
        % Collect all non-dominated configurations, no larger
        % than mid, that haven't already been eliminated
        C := {(n, m) | n × m ≤ mid,
                       n × (m + 1) > mid,
                       (n + 1) × m > mid,
                       ∀ (n′, m′) ∈ failed . n > n′ ∨ m > m′}
        while (C ≠ ∅ ∧ ¬found)
            (n, m) := (n′, m′) ∈ C with maximal n′ × m′
            C := C \ {(n, m)}
            res := synth(φ, n, m)
            if (res = fail)
                % Eliminate configurations dominated by (n, m)
                failed := failed ∪ {(n, m)}
            else
                best := res
                found := true
                % Restrict search to smaller lattices
                ub := n × m − 1
        if (¬found)
            lb := mid + 1
    return best
```

Fig. 8: A dichotomic search strategy. Note that we check all maximal configurations no larger than mid, rather than just those with $w \times h = mid$.

binary-search style approach. Of course, in the worst case, we will need to solve problems for $O(\log |L_U|)$ different areas; however, when $\hat{L}$ is small, we will avoid the need to solve a sequence of large feasible instances.

Pseudo-code for the dichotomic search strategy is given in Figure 8. Note that we must consider *all* non-dominated configurations before concluding that there are no smaller lattices, not just those instances with $n \times m = mid$.

Example 4.3. Consider a circuit with a minimal lattice of size (2×8). Starting with $max = 34$, we initially pick $mid = 17$. The only configurations with $n \times m = 17$ are $(1, 17)$ and $(17, 1)$. Both of these configurations may be infeasible, despite the existence of a smaller feasible configuration.

5. EXPERIMENTAL RESULTS

We have computed minimal lattices for the circuits used by Altun and Riedel [2012], to test our method and to see how close the DP construction gets to finding the minimum.

Experiments were performed on a 3.4GHz Intel Core i7 with 8Gb RAM running Ubuntu 12.04. The circuits were converted into SAT[2], and solved with `plingeling`, the parallel version of `lingeling` [Biere 2013], running on 4 cores. Experiments were performed with a 4 hour time limit.

Results are given in Tables I and II. All times are given in seconds. Instances which failed to terminate within 4 hours are denoted by '—'. `DP` denotes the upper bound computed using the dual-product construction, and `C` is the upper bound computed with

[2]The script is available at http://people.eng.unimelb.edu.au/gkgange/synth

Circuit	$\|V\|$	bounds			optimal		restricted	
		DP	C	lb	dim	time	dim	time
alu1_00	4	**(2x3)**	(2x5)	6	**(2x3)**	**0.19**	**(2x3)**	0.22
alu1_01	4	**(3x2)**	(3x3)	6	**(3x2)**	**0.18**	**(3x2)**	0.22
alu1_02	3	**(3x1)**	**(3x1)**	3	**(3x1)**	**0.09**	**(3x1)**	0.11
b12_00	6	(6x4)	(4x7)	9	**(4x3)**	1.16	**(4x3)**	**0.80**
b12_01	7	(5x7)	(4x13)	12	**(4x4)**	5.37	**(4x4)**	**0.96**
b12_02	8	(6x7)	(5x13)	12	**(4x4)**	**18.01**	(5x4)	1.90
b12_03	4	(2x4)	(2x7)	4	**(3x2)**	**0.18**	**(3x2)**	0.24
b12_04	5	**(2x4)**	(2x7)	8	**(2x4)**	**0.30**	**(2x4)**	0.31
b12_05	5	**(1x5)**	(1x9)	5	**(1x5)**	**0.20**	**(1x5)**	0.25
b12_06	9	(6x9)	(6x17)	12	**(5x4)**	238.88	**(5x4)**	**2.89**
b12_07	7	(4x6)	(4x11)	15	**(3x6)**	**3.88**	(4x5)	1.86
b12_08	8	**(2x7)**	(2x13)	14	**(2x7)**	3.70	**(2x7)**	**0.88**
c17_00	4	(3x3)	(2x5)	6	**(2x3)**	**0.22**	**(2x3)**	0.32
c17_01	4	(2x4)	(2x7)	4	**(3x2)**	**0.18**	**(3x2)**	0.23
clpl_00	7	(4x4)	(4x7)	12	**(3x4)**	1.42	**(3x4)**	**0.64**
clpl_01	5	**(3x3)**	(3x5)	9	**(3x3)**	**0.46**	**(3x3)**	0.51
clpl_02	3	**(2x2)**	(2x3)	4	**(2x2)**	**0.10**	**(2x2)**	0.15
clpl_03	11	(6x6)	(6x11)	15	**(3x6)**	99.91	**(3x6)**	**1.71**
clpl_04	9	(5x5)	(5x9)	12	**(3x5)**	13.94	**(3x5)**	**0.97**

Table I: Results on a set of circuit synthesis benchmarks

the composition approach, inserting false (0) columns between product columns as in Figure 5(b). lb denotes the lower bound computed by Altun and Riedel [2012]. optimal gives the running time and the minimal lattice given by the SAT model described in Section 4.1. The SAT model successfully found the minimal layout (and proved optimality) for all but one instance (mp2d_02) in under 4 hours. For this set of benchmarks, the optimal lattices appear to be much smaller than the ISOP-based construction. restricted gives results using the restricted model described in Section 4.2. The minimal lattice size for the restricted formulation is typically quite close to that for the full formulation. Interestingly, while the restricted formulation is much faster on some moderate instances, this is not consistently the case; indeed, the restricted approach failed to solve 3 instances that were successfully solved by the optimal.

Instances in Table III illustrate the worst-case behaviour of the DP approach. An instance (t, c, s) is a disjunction of t terms, each of cardinality c, with s terms in common between each adjacent pair. Instance $(3, 3, 1)$, for example, is the function $(x_1 \wedge x_2 \wedge x_3) \vee (x_3 \wedge x_4 \wedge x_5) \vee (x_5 \wedge x_6 \wedge x_7)$. We can see that the size of the DP construction grows rapidly as either the number of terms or cardinality increases, whereas the lattice constructed by the composition method grows only polynomially with the input size.

6. CONCLUSION

We have shown how to efficiently construct switching lattices for arbitrary Boolean formulas which are of size polynomial in the input. We have also shown how to synthesize optimal switching lattices using SAT solving on a carefully chosen formulation of the problem. Our method is an anytime algorithm: stopped prematurely it will deliver a correct (albeit not necessarily minimal) switching lattice. While the problem is clearly challenging, our synthesis method generates considerably smaller lattices than previous approaches.

Circuit	$\|V\|$	bounds			optimal		restricted	
		DP	C	lb	dim	time	dim	time
dc1_00	4	(4x4)	(3x7)	9	**(3x3)**	**0.44**	**(3x3)**	0.50
dc1_01	4	**(3x2)**	(3x3)	6	**(3x2)**	**0.19**	**(3x2)**	0.23
dc1_02	4	(4x4)	(3x7)	12	**(3x4)**	**0.48**	**(3x4)**	0.57
dc1_03	4	(5x4)	(4x7)	9	**(4x3)**	**0.45**	**(4x3)**	0.47
dc1_04	4	(3x3)	(2x5)	6	**(2x3)**	**0.22**	**(2x3)**	0.27
misex1_00	4	(5x2)	(4x3)	6	**(4x2)**	**0.29**	**(4x2)**	0.33
misex1_01	6	(7x5)	(4x9)	12	**(3x5)**	**2.05**	(4x4)	0.98
misex1_02	7	(8x5)	(5x9)	12	**(5x4)**	21.86	**(5x4)**	**2.45**
misex1_03	7	(7x4)	(5x7)	9	**(4x3)**	**1.43**	(5x3)	0.81
misex1_04	4	(5x5)	(4x9)	12	**(3x4)**	**0.51**	(3x5)	0.89
misex1_05	6	(7x6)	(4x11)	12	**(4x4)**	**3.16**	(5x4)	2.05
misex1_06	6	(7x5)	(4x9)	9	**(5x3)**	2.24	**(5x3)**	**0.98**
mp2d_00	11	**(1x11)**	(1x21)	11	**(1x11)**	1111.86	**(1x11)**	**9.64**
mp2d_01	10	(7x8)	(5x15)	20	**(3x9)**	**7279.19**	(4x7)	64.93
mp2d_02	11	(5x10)	(4x19)	24	**(6x6)**	—	(5x10)	—
mp2d_03	10	(19x5)	(8x9)	15	**(4x6)**	**1508.76**	(5x5)	28.76
mp2d_04	10	(11x6)	(9x11)	15	**(7x3)**	4635.74	**(7x3)**	**31.63**
mp2d_05	5	**(5x1)**	**(5x1)**	5	**(5x1)**	**0.21**	**(5x1)**	0.24
mp2d_06	5	(8x3)	(5x5)	8	**(4x3)**	**0.83**	(7x2)	0.74
mp2d_07	8	**(8x1)**	**(8x1)**	8	**(8x1)**	1.18	**(8x1)**	**0.48**
mp2d_08	5	**(1x5)**	(1x9)	5	**(1x5)**	**0.20**	**(1x5)**	0.23
newapla2_00	6	**(6x1)**	**(6x1)**	6	**(6x1)**	**0.27**	**(6x1)**	0.28
newbyte_00	5	**(5x1)**	**(5x1)**	5	**(5x1)**	**0.21**	**(5x1)**	0.26
newtag_00	8	(4x8)	(3x15)	15	**(3x6)**	9.47	**(3x6)**	**1.38**
ex5_00	3	**(3x1)**	**(3x1)**	3	**(3x1)**	**0.08**	**(3x1)**	0.10
ex5_01	5	**(5x1)**	**(5x1)**	5	**(5x1)**	**0.22**	**(5x1)**	0.25
ex5_02	4	**(4x1)**	**(4x1)**	4	**(4x1)**	**0.11**	**(4x1)**	0.14
ex5_03	7	**(7x1)**	**(7x1)**	7	**(7x1)**	0.63	**(7x1)**	**0.39**
ex5_04	8	**(8x1)**	**(8x1)**	8	**(8x1)**	1.22	**(8x1)**	**0.48**
ex5_05	6	**(6x1)**	**(6x1)**	6	**(6x1)**	0.28	**(6x1)**	**0.26**
ex5_06	7	(4x8)	(3x15)	15	**(3x6)**	**5.61**	(3x7)	3.87
ex5_07	8	(4x10)	(3x19)	20	**(4x6)**	**210.94**	(3x9)	1169.00
ex5_08	8	**(3x7)**	(3x13)	18	**(3x7)**	12.63	**(3x7)**	**3.66**
ex5_09	8	(4x10)	(4x19)	20	**(4x6)**	40.21	**(3x8)**	**28.63**
ex5_10	6	(3x7)	(3x13)	15	**(3x6)**	2.13	**(3x6)**	**1.35**
ex5_11	8	**(2x8)**	(2x15)	16	**(2x8)**	5.54	**(2x8)**	**1.21**
ex5_12	8	(3x9)	(3x17)	12	**(3x5)**	6.12	**(3x5)**	**0.99**
ex5_13	8	(4x9)	(3x17)	20	**(4x6)**	24.40	**(3x8)**	**22.83**
ex5_14	8	**(2x8)**	(2x15)	14	**(2x8)**	5.50	**(2x8)**	**1.18**
ex5_15	8	(6x12)	(4x23)	18	**(3x8)**	**4697.57**	(6x12)	—
ex5_16	5	**(2x5)**	(2x9)	10	**(2x5)**	**0.50**	**(2x5)**	0.59
ex5_17	8	(9x14)	(4x27)	18	**(3x9)**	**14199.63**	(9x6)	—
ex5_18	7	**(2x7)**	(2x13)	14	**(2x7)**	1.97	**(2x7)**	**0.91**
ex5_19	8	**(3x6)**	(3x11)	15	**(3x6)**	7.49	**(3x6)**	**1.52**
ex5_20	7	**(2x6)**	(2x11)	12	**(2x6)**	1.30	**(2x6)**	**0.69**
ex5_21	8	(7x10)	(3x19)	18	**(3x7)**	**20.23**	(4x7)	1726.31
ex5_22	7	(6x6)	(3x11)	15	**(3x6)**	5.58	**(3x6)**	**1.37**
ex5_23	8	(10x12)	(4x23)	20	**(3x9)**	**9171.67**	(3x10)	4507.30
ex5_24	8	(8x14)	(5x27)	18	**(3x8)**	**1209.65**	(8x7)	—
ex5_25	8	(5x8)	(3x15)	18	**(3x7)**	18.32	**(3x7)**	**9.03**
ex5_26	8	(8x10)	(3x19)	18	**(3x7)**	**17.80**	(3x9)	272.40
ex5_27	8	(7x11)	(4x21)	18	**(3x8)**	4952.00	**(3x8)**	**50.41**
ex5_28	8	(3x9)	(3x17)	20	**(4x6)**	26.75	**(3x8)**	**23.48**
Total		2077	2643	834	1711		1829	

Table II: Results on a set of circuit synthesis benchmarks—continued

Circuit	$\|V\|$	bounds			optimal		restricted	
		DP	C	lb	dim	time	dim	time
$(2,6,0)$	12	(36x2)	**(6x3)**	9	**(6x3)**	7076.22	**(6x3)**	**119.40**
$(3,3,0)$	9	(27x3)	**(3x5)**	9	**(3x5)**	20.27	**(3x5)**	**1.55**
$(3,3,1)$	7	(9x3)	(3x5)	9	**(4x3)**	1.36	**(4x3)**	**0.45**
$(3,3,2)$	5	(4x3)	(3x5)	6	**(3x2)**	**0.08**	**(3x2)**	0.22
$(3,4,0)$	12	(64x3)	**(4x5)**	9	**(4x5)**	—	**(4x5)**	**27.48**
$(3,4,1)$	10	(25x3)	**(4x5)**	9	**(4x5)**	4186.20	**(4x5)**	**26.47**
$(3,4,2)$	8	(12x3)	(4x5)	6	**(6x2)**	8.37	**(6x2)**	**0.82**
$(3,5,0)$	15	(125x3)	**(5x5)**	9	**(5x5)**	—	**(5x5)**	**12408.54**
$(3,5,1)$	13	(57x3)	**(5x5)**	9	**(5x5)**	—	**(5x5)**	—
$(3,5,2)$	11	(25x3)	(5x5)	9	**(5x4)**	6530.91	**(5x4)**	**25.93**
$(4,3,0)$	12	(81x4)	(3x7)	12	(3x7)	—	**(4x5)**	**58.52**
$(4,3,1)$	9	(17x4)	(3x7)	12	**(3x5)**	24.26	**(3x5)**	**1.36**
$(4,3,2)$	6	(6x4)	(3x7)	6	**(4x2)**	0.42	**(4x2)**	**0.28**
$(4,4,0)$	16	(256x4)	**(4x7)**	12	**(4x7)**	—	**(4x7)**	**1787.58**
$(4,4,1)$	13	(64x4)	(4x7)	12	(4x7)	—	**(5x5)**	**858.35**
$(4,4,2)$	10	(28x4)	(4x7)	9	**(3x6)**	2052.07	**(3x6)**	**350.54**
$(6,2,0)$	12	(64x6)	(2x11)	12	**(2x6)**	71.92	**(2x6)**	**2.30**

Table III: Results on instances consisting of sums of disjoint products. An instance (t, c, s) consists of t terms of cardinality c, with s literals shared between each pair of adjacent terms.

REFERENCES

Sheldon B. Akers, Jr. 1972. A Rectangular Logic Array. *IEEE Trans. Comput.* C-21, 8 (1972), 848–857.

Mustafa Altun and Marc D. Riedel. 2012. Logic Synthesis for Switching Lattices. *IEEE Trans. Comput.* 61, 11 (2012), 1588–1600.

Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. 2011. Cardinality Networks: A Theoretical and Empirical Study. *Constraints* 16, 2 (2011), 195–221.

Armin Biere. 2013. Plingeling and Treengeling Entering the SAT Competition 2013. In *Proceedings of the SAT Competition 2013 (Department of Computer Science Series of Publications B)*, A. Balint, A. Belov, M. Heule, and M. Järvisalo (Eds.), Vol. B-2013-1. University of Helsinki, 51–52.

Robert K. Brayton. 1984. *Logic Minimization Algorithms for VLSI Synthesis*. Vol. 2. Springer.

Yong Chen, Gun-Yong Jung, Douglas A. A. Ohlberg, Xuema Li, Duncan R. Stewart, Jan O. Jeppesen, Kent A. Nielsen, J. Fraser Stoddart, and R. Stanley Williams. 2003. Nanoscale Molecular-Switch Crossbar Circuits. *Nanotechnology* 14 (2003), 462–468.

Y. Cui and C. M. Lieber. 2001. Functional nanoscale electronic devices assembled using silicon nanowire building blocks. *Science* 291, 5505 (2001), 851–853.

Mary M. Eshaghian-Wilner, Amar H. Flood, Alex Khitun, J. Fraser Stoddart, and Kang Wang. 2006. Molecular and Nanoscale Computing and Technology. In *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Methods with Emerging Technologies*, Albert Y. Zomaya (Ed.). Springer, 477–509.

Michael Gelfond and Vladimir Lifschitz. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, Robert A. Kowalski and Kenneth A. Bowen (Eds.). MIT Press, 1070–1080.

Florian Lonsing and Armin Biere. 2010. DepQBF: A Dependency-Aware QBF Solver. *Journal on Satisfiability, Boolean Modeling and Computation* 7, 2-3 (2010), 71–76.

Victor W. Marek and Mirosław Truszczyński. 1999. Stable Models and an Alternative Logic Programming Paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*, K. R. Apt, V. W. Marek, M. Truszczyński, and D. S. Warren (Eds.). Springer, 375–398.