Circuit-based Preprocessing of ILP and Its Applications in Leakage Minimization and Power Estimation

Donald Chai[1] Andreas Kuehlmann[1,2]
[1] University of California at Berkeley, CA, USA
[2] Cadence Berkeley Labs, Berkeley, CA, USA

Abstract

In this paper we discuss the application of circuit-based logical reasoning to simplify optimization problems expressed as integer linear programs (ILP) over circuit states. We demonstrate that a targeted restructuring of the problem formulation based on the circuit topology can significantly improve the performance and capacity of the overall optimization procedure. We further review two distinct application classes, one requiring a feasible, the other an infeasible bound of an ILP solution that cannot be computed optimally within resource limits and present algorithmic approaches to handle them. We use the problems of computing a minimal leakage state and finding the state transition with maximal peak current to exemplify these two unique classes and present results comparing our methods with alternative techniques.

1 Introduction

Many CAD problems require reasoning about the logical state of circuits. Algorithmic advances in this area have been driven mainly by functional verification which requires to check whether a particular state or sequence of states is reachable. The corresponding decision problem can often be formulated as a satisfiability (SAT) check for which exploiting the circuit structure has become a key component of efficient algorithms (e.g. [1]). Other CAD problems are more naturally formulated as integer linear programs seeking an "optimal valid state" with respect to an objective function given as a weighted sum of the variables. Examples include leakage state minimization with the goal to find a stand-by state that consumes as little leakage power as possible, or peak current analysis that searches for the state-transition that draws maximal current. Despite the significant progress in handling large ILP instances, many problems cannot be solved optimally with limited computing resources. In practice, however, suboptimal solutions are often acceptable.

There are two distinct approaches for deriving the optimal solution; both having unique applications. The first approach bounds the optimum from the infeasible side, we will further refer to this estimate as the *infeasible bound*. Clearly, there exist no actual variable assignment for this estimate unless it is the optimal solution. An example for using an infeasible bound is peak current analysis which demands a conservative estimate of the maximum current that cannot be exceeded under any circumstances. The second approach bounds the optimum from the feasible side, further referred to as the *feasible bound*. Here an actual variable assignment is desired to demonstrate this estimate. For example, leakage state minimization searches for a low leakage state, even if it is not the optimal one. Here a concrete state with the computed leakage value is needed to implement the stand-by function.

In this paper we present an approach to simplify large ILP instances in order to obtain better results. The paper is organized as follows. Section 2 discusses previous work in modeling leakage minimization and peak current analysis, and discusses the issues in solving the formulations provided by previous work. Section 3 presents a general approach to simplify ILP problems using circuit-based reasoning. Section 4 shows the application of the presented approach to two problems and provides experimental results.

2 Motivation and Preliminaries

2.1 Previous Work in Applications

Previous work on leakage current minimization through the control of input vectors formulates the task as a discrete optimization problem. The motivation is that the leakage current through each gate in the circuit is a strong function of its inputs [2]. However, because of the circuit structure, it is generally not possible to put *all* gates into their lowest leakage state by just controlling the primary inputs. To minimize the overall leakage current, the authors of [3] formulate an integer program over the circuit. For each gate, a number of terms are added to the objective to reflect each possible input state of the gate. A number of constraints express the logical relationships between the gates in a circuit. Solving the integer program provides the optimum input vector which minimizes leakage current.

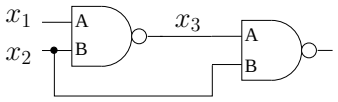
The problem of peak current estimation has been approached in [4]. In that work, the authors construct a MAX-SAT problem that models the possible switching activity in a circuit. Unfortunately, this approach requires that the problem be solved optimally—no suboptimal solution can be used as upper bounds of switching activity. The authors of [5] present a partial input enumeration algorithm, which is a variant of branch-and-bound used to iteratively improve estimates. The search procedure incrementally assigns values to primary inputs, and calls a procedure to obtain loose bounds on current. In this manner the authors simulate a fancy ILP solver, albeit with good branching heuristics.

All previous methods lack a uniform approach to exploit signal correlations within a circuit. Typical circuits contain large amounts of redundancies and don't care conditions that often go unused. The focus of this paper is on the use of this information to improve the performance of ILP solvers. In the next subsection we describe how ILP is applied to solve optimization problems over circuits.

2.2 Preliminaries

Integer programming (IP) [6] can be used as a computational kernel to perform many optimization tasks. An integer program is of the form:

$$\begin{array}{rll} \min & f(x) & \\ \text{such that} & \mathbf{g}(x) & \geq 0 \\ & x \in \mathbb{Z}^n & \end{array}$$



Gate inputs A	Gate inputs B	Leakage
0	0	$58 = \alpha$
0	1	$157 = \gamma$
1	0	$146 = \beta$
1	1	$170 = \delta$

Figure 1: Example circuit for leakage minimization and leakage power values dissipated in both NAND gates as a function of their inputs.

where f is the objective function and $\mathbf{g}$ represents a set of constraints. If $x \in \{0,1\}^n$ then the problem is called 0-1 IP. An integer linear program (ILP) restricts f and $\mathbf{g}$ to consist of linear functions.

As an example, consider the problem of leakage state minimization of the circuit depicted in Figure 1. Using the values for the individual gate states, we can formulate the following IP:

$$\begin{aligned} \min \quad & 58\bar{x}_1\bar{x}_2 + 146x_1\bar{x}_2 + 157\bar{x}_1x_2 + 170x_1x_2 \qquad (1)\\ & +58\bar{x}_3\bar{x}_2 + 146x_3\bar{x}_2 + 157\bar{x}_3x_2 + 170x_3x_2 \\ & x_1 + x_3 \geq 1 \\ & x_2 + x_3 \geq 1 \\ & \bar{x}_1 + \bar{x}_2 + \bar{x}_3 \geq 1 \\ & x_i \in \{0,1\} \end{aligned}$$

A complemented variable $\bar{x}_i$ represents the arithmetic expression $(1 - x_i)$. The first expression in (1) computes the circuit leakage and provides the minimization objective. The first (second) four terms in the objective represent the four leakage modes of the first (second) gate in the circuit. The three inequalities are Boolean constraints ensuring logical consistency of the gate inputs with respect to the circuit topology.

There are two methods to approach the above given IP. First, due to the lack of efficient non-linear solvers, all common approaches linearize the objective function by substituting a fresh Boolean variable for each non-linear term and adding constraints that force their logical equivalence [7]. For example, the first part of the objective (1), $58\bar{x}_1\bar{x}_2$, can be rewritten as $58y_1$ with the additional constraints $(\bar{x}_1 + \bar{y_1} \geq 1)$, $(\bar{x}_2 + \bar{y_1} \geq 1)$, and $(x_1 + x_2 + y_1 \geq 1)$. Note that in the circuit representation of Figure 1 this would be equivalent to adding an AND gate to the original circuit that represents the state $(A = 0, B = 0)$ of gate g_1. In this manner all non-linear terms can be expressed as circuit gates resulting in a objective function that is linear in the variables of these gate outputs.

Second, one can simplify the objective function in (1) by multiplying out the $(1 - x_i)$ expressions and combining the coefficients of the resulting polynomial [8, 3]. This would lead to the objective:

$$\min \quad -75x_1x_2 - 75x_3x_2 + 88x_1 + 198x_2 + 88x_3 + 116$$

In the example, we know that x_3 is a function of x_1 and x_2, i.e., $x_3 = (1 - x_1x_2)$. With careful manipulation, and knowledge of properties of Boolean algebra (for example $x^2 = x$), the original quadratic problem in (1) can be restated as

$$\begin{aligned} \min \quad & -88x_1x_2 + 88x_1 + 123x_2 + 204 \qquad (2)\\ & x_i \in \{0,1\} \end{aligned}$$

with no further constraints. Thus, we may often reduce the number of terms in the objective without affecting the solution to the problem. By reducing the count of nonlinear terms in the objective, we also reduce the number of constraints necessary for linearization. However, for large optimization instances derived from realistic circuit problems this algebraic approach is infeasible.

In this paper we present how 0-1 ILP problems that are derived from circuits can be compacted by analyzing the circuit structure.

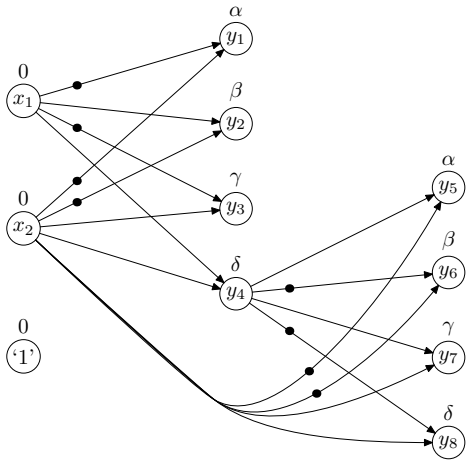


Figure 2: AND/INVERTER graph for circuit from Figure 1. Nodes represent the inputs x_1, x_2, the constant '1', and AND gates. Labels above each node are weights representing leakage.

We show that the above mentioned simplifications and linearization can be handled natively in a circuit-based representation and thus be applied to large problem instances. We also show how the identification of general equivalences can be used to derive redundant constraints which tighten the solution to the continuous relaxation of the ILP. This technique helps the computation of infeasible bounds.

3 Circuit-based Equivalences for ILP

3.1 Circuit Representation

We use an AND/INVERTER graph representation to efficiently store and manipulate circuit structures. The AND/INVERTER graph is composed of three types of nodes: A unique terminal node represents the constant '1' ('0') value when it is referenced by a non-complemented (complemented) arc. A second type of node has no incoming arcs and models primary inputs. The third node type has two incoming arcs and represents the AND of the node functions referenced by the two arcs. INVERTER attributes on the graph arcs indicate Boolean complementation. The AND/INVERTER graph for a given circuit is simply constructed by rewriting each gate in terms of ANDs and INVERTERs. The motivation for using an AND/INVERTER graph instead of a BDD (or a BDD variant) is that BDDs tend to grow impractically large for many circuits.

To illustrate the preprocessing on an AND/INVERTER graph, we revisit the circuit from Figure 1. We first construct an AND/INVERTER graph from the circuit. To represent the nonlinear objective (1), AND nodes are introduced for each term and assigned a weight equal to its respective coefficient. The resulting graph is shown in Figure 2. Note that the nonlinear term x_1x_2 and x_3 are equivalent modulo complementation ($\bar{x_3} \equiv x_1x_2$) and are represented by the same node $y_4 (\equiv x_1x_2)$. The node labeled y_5 has a weight of α to represent the term $58\bar{x_3}\bar{x_2}$ in the objective of the original problem.

The problem can now be restated as follows. Nodes in the AND/INVERTER graph can be assigned 0 or 1, and the cost of the assignment is evaluated as the sum of the weights of all nodes assigned to 1. The objective of the optimization problem is to find an assignment logically consistent with the graph that has lowest cost.

The above process produces an AND/INVERTER graph which maps directly to the original integer program. We use the term *weight* and *coefficient* interchangeably depending on whether the

emphasis is on the graph or its associated integer program. Similarly we use the terms *variable* and *node* interchangeably. The key idea of this paper is that we may modify the graph in any way that does not change the logical relationships between nodes and does not alter the value of the objective under any valid assignment. Doing so will yield a new graph, which maps to a syntactically different, but logically equivalent, integer program. Theoretically, one could do the same type of modifications in the ILP formulation directly, however, this is impractical for large problem instances.

The approach we take in this paper is as follows. We find logical equivalences in a graph which permit us to perform restructuring. From these we derive corresponding arithmetic equations which permit us to rewrite the objective and further compact the graph.

3.2 Simple Equivalences ($y \equiv x$)

Practical circuits contain a significant number of equivalent internal gate functions which cause redundancies in their SAT formulation and a corresponding significant slow-down of SAT solvers [9]. Finding such equivalences is key for compacting their SAT formulation. In the same vein, these redundancies are likely to adversely affect the performance of ILP solvers.

We use three methods, similar to those described in [1, 9], to identify functional equivalences. First, during the construction of the AND/INVERTER graph, isomorphic subgraphs are identified by simple structural hashing. Second, local rewriting rules are used to identify equivalent subgraphs which are only slightly different in structure. Third, to find equivalent vertices that are not identified by the previous two techniques, we use a variation of BDD SWEEPING [9].[1]

3.3 Generalized Equivalences ($y \equiv x_1 \vee x_2$)

More general equivalences between variables may also be exploited. The following theorem is familiar from probabilistic simulation of logical circuits [10]:

$$y \equiv x_1 \vee x_2 \equiv \overline{\bar{x}_1 \wedge \bar{x}_2} \quad \Longleftrightarrow \quad y = x_1 + x_2 - x_1 x_2$$

Informally, a Boolean-OR of two operands is the same as their arithmetic sum minus any "overlap". If these conditions on y hold, we may replace y by $x_1 + x_2 - x_1 x_2$ in the objective of our original ILP formula without changing the problem. This may be beneficial if no node implementing $\bar{x_1} \wedge \bar{x_2}$ yet exists in the AND/INVERTER graph, but a node exists for $x_1 \wedge x_2$.

Note that the above equation is nonlinear, while the ILP solvers we use require us to make it linear. Ordinarily, we would linearize the equation by introducing a fresh Boolean variable z representing $x_1 \wedge x_2$. In that case, we would obtain:

$$y = x_1 + x_2 - z$$

At first glance, this seems to complicate our formulation with more variables and constraints. On the other hand, we can exploit the first two simple equivalence identification mechanisms mentioned in the previous section to search for some existing node z' equivalent to z. If such a z' is found, then the equation $y = x_1 + x_2 - z'$ can be used instead with no increase in complexity. For example, the term $58\bar{x_1}\bar{x_2}$ from the objective in (1) can be simply rewritten as $58(1 - (x_1 + x_2 - x_1 x_2))$, thus reducing the number of nonlinear terms. This can be seen in Figure 2 as moving the label from y_1 to y_4, x_1, x_2, and '1'.

The result of such rewriting is twofold: First, the number of nonlinear terms in the objective function is reduced which helps solving the optimization task and also decreases the number of additional variables needed for linearization. Second, the AND/INVERTER graph nodes that yield a weight of zero and have no fanout can be removed which reduces the number of required constraints for the ILP formulation. In the next subsection we outline how rewriting opportunities can be identified and how they are processed in a systematic manner.

[1] Our implementation of the SWEEPING procedure uses SAT for speed and robustness. However, the principle is the same.

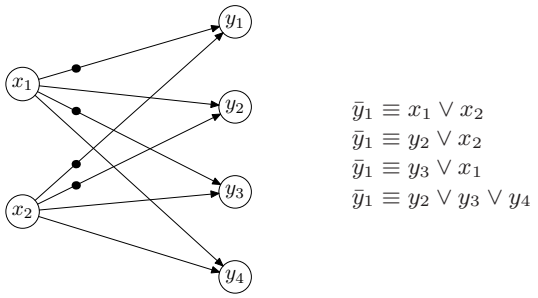


Figure 3: Valid facts about $\bar{y_1}$.

3.3.1 Identification

Generalized equivalences of the form $y \equiv x_1 \vee x_2$ are often obtainable by inspecting an AND/INVERTER graph structurally. We list the three easily discernible types below, exemplified in Figure 3:

- $\bar{y}_1 \equiv x_1 \vee x_2$
  For any AND node $y_1 \equiv \bar{x}_1 \wedge \bar{x}_2$, De Morgan's theorem yields $\bar{y}_1 \equiv x_1 \vee x_2$. Finding a node y_4 where $y_4 \equiv x_1 \wedge x_2$, yields
  $$1 - y_1 = x_1 + x_2 - y_4$$
- $\bar{y}_1 \equiv y_2 \vee x_2$
  In this case, y_1 and y_2 share common fan-ins, except one is complemented. This is justified as follows:
  $$\begin{aligned} \bar{y}_1 &\equiv y_2 \vee x_2 \\ &\equiv (x_1 \wedge \bar{x}_2) \vee x_2 \\ &\equiv x_1 \vee x_2 \end{aligned}$$
  For these structures, $1 - y_1 = y_2 + x_2 - y_2 x_2$. Furthermore, $y_2 \wedge x_2 \equiv x_1 \wedge \bar{x}_2 \wedge x_2 \equiv$ false, which leads to
  $$1 - y_1 = y_2 + x_2$$
- $\bar{y}_1 \equiv y_3 \vee x_1$
  This is the same as the previous case, where the other input of the gate implementing y_1 is complemented.

While these examples focus on node y_1, the same ideas apply to the other nodes in the figure. For example, we may use the facts $\bar{y}_2 \equiv \bar{x}_1 \vee x_2$, $\bar{y}_2 \equiv y_1 \vee x_2$, and $\bar{y}_2 \equiv y_4 \vee \bar{x}_1$ to derive similar equations.

These concepts may be generalized to equivalences of the form:

$$y \equiv \bigvee_i x_i \equiv \overline{\bigwedge_i \bar{x}_i} \quad \Longleftrightarrow \quad y = 1 - \prod_i (1 - x_i)$$

However, the identification of these more general cases is more complex. Therefore, this paper focuses on simple equivalences and the special cases listed above.

3.4 Use of Equivalences And Equalities

The previous discussion introduced a set of equivalences for reasoning about a circuit. We call a transformation on a circuit subgraph a valid "move" if it preserves the objective.

Figure 4 illustrates how a set of valid moves can be applied to a graph. For example, the move from (4a) to (4b) can be justified as follows: If x_1 is true, then either y_2 or y_4 is true. Therefore, we can "factor out" a common cost β from the two cases and attribute β to x_1 by deducting it from the costs of y_2 and y_4. This reasoning is the

Table 1: Simplification of problem in Figure 2: $\alpha = 58, \beta = 146, \gamma = 157, \delta = 170$.

Theorem used		Coefficients of nodes in Figure 2										
Equivalence	Associated Equation	x_1	x_2	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	const
				58	146	157	170	58	146	157	170	
				α	β	γ	δ	α	β	γ	δ	
$\bar{y}_8 \equiv \bar{x}_2 \vee y_4$	$-y_8 + x_2 - y_4 + y_5 = 0$		δ	α	β	γ	0	$\alpha + \delta$	β	γ	0	
$\bar{y}_7 \equiv y_5 \vee \bar{y}_4$	$-y_7 - y_5 + y_4 = 0$		δ	α	β	γ	γ	$\alpha + \delta - \gamma$	β	0		
$\bar{y}_6 \equiv y_5 \vee x_2$	$1 - y_6 - y_5 - x_2 = 0$		$\delta - \beta$	α	β	γ	γ	$\alpha + \delta - \gamma - \beta$	0			β
$y_5 \equiv$ false	$y_5 = 0$		$\delta - \beta$	α	β	γ	γ	0				β
$\bar{y}_1 \equiv x_1 \vee x_2$	$1 - y_1 - x_1 - x_2 + y_4 = 0$	$-\alpha$	$\delta - \beta - \alpha$	0	β	γ	$\gamma + \alpha$					$\beta + \alpha$
$\bar{y}_2 \equiv \bar{x}_1 \vee y_4$	$-y_2 + x_1 - y_4 = 0$	$\beta - \alpha$	$\delta - \beta - \alpha$		0	γ	$\gamma + \alpha - \beta$					$\beta + \alpha$
$\bar{y}_3 \equiv \bar{x}_2 \vee y_4$	$-y_3 + x_2 - y_4 = 0$	$\beta - \alpha$	$\delta - \beta - \alpha + \gamma$			0	$\alpha - \beta$					$\beta + \alpha$
		88	123			0	-88					204

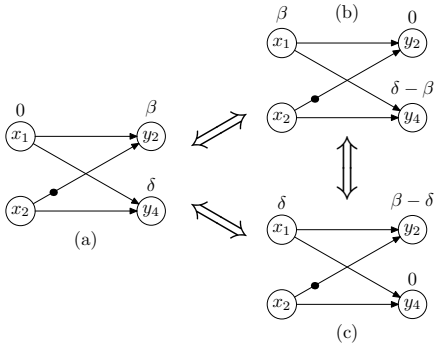


Figure 4: Valid moves preserve the objective.

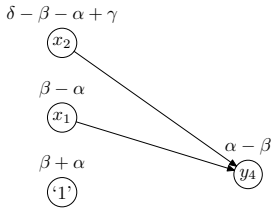


Figure 5: Simplification of graph in Figure 2.

same as using the fact $x_1 - y_2 - y_4 = 0$. After this transformation node y_2 is unused and may be discarded or used for further moves. All other moves in the figure can be justified in a similar way.

We apply the moves in a general preprocessing loop that aims at simplifying the AND/INVERTER graph which, in turn, results in a simpler ILP formulation. The preprocessing loop makes a series of moves until some stopping condition:

do
  make a valid move
  update gains
while there are beneficial moves
discard unused nodes

The stopping condition and move values depend on the application and will be discussed in detail in Section 4. In general, we use an empirical quality metric to measure the anticipated ILP runtime for a given circuit structure and interleave rewriting with quick attempts to solve intermediate formulations.

Table 1 shows a sequence of moves that greatly simplify the graph of Figure 2 for the original example given in Figure 1. The resulting graph is depicted in Figure 5 and reflects the ILP formulation (2) after the term $x_1 x_2$ is linearized.

4 Applications

In this paper, we focus on two applications. The leakage minimization problem seeks a solution showing a feasible bound, whereas the peak current problem seeks a proof for an infeasible bound. As stated in the introduction, leakage minimization requires a variable assignment which induces a minimal leakage current in the rest of the circuit. This assignment must be feasible so that it can actually be used in real devices. On the other hand, the peak current problem requires a guaranteed estimate of the maximum current, which cannot be exceeded under any circumstances. Obtaining an actual pair of input vectors is not as important as obtaining a conservative upper bound in this case. In the next two subsections, we show how our approach can be used to obtain tight bounds of both feasibility and infeasibility, respectively.

4.1 Minimizing leakage through input vector control

The leakage current dissipated in a circuit has been shown to depend on the assigned logic values as shown in [2]. Figure 1 shows an example of the varying leakage amounts for different logic values at the gate inputs. The problem of leakage state minimization is to find a circuit-wide consistent assignment of logic values to all gates such that the overall leakage of minimal.

To find feasibility bounds, we focus on using a SAT-based 0-1 ILP solver to successively search for increasingly better solutions for this optimization problem. We focus on SAT-based methods because modern solvers are tuned for efficiently finding feasible solutions [11, 12].

All modern SAT-based methods are based on branch-and-bound search and are highly sensitive to the branching order which is controlled by the *decision heuristic*. Popular decision heuristics [12] tend to prefer assignments to variables which occur in many constraints. However, in optimization, it seems sensible to favor variables with large coefficients c_i. To assist the solver, we try to align the two decision heuristics by maximizing the value of a formulation as computed below:

$$value = \frac{\sum_{i \in I} |c_i| * fanout_i^2}{|I|} \quad I = \{i | c_i \neq 0\}$$

Here, $fanout_i$ denotes the out-degree of node i in the AND/INVERTER graph. There is a linear relationship between $fanout_i$ and the number of constraints in the ILP formulation which contain the variable representing node i. It has been observed empirically that using $fanout_i^2$ instead of $fanout_i$ leads to objectives with fewer terms and smaller coefficients.

Preliminary experiments show that there is a weak correlation between the value of an ILP formulation as computed above and the observed quality of the solutions from the SAT solver. Therefore,

we use the value only as a guide for preprocessing. To heuristically obtain a good solution, we iterate between applying graph moves and running the SAT-based ILP solver with a short time-out. The moves aim at increasing the value of the formulation whereas the inexpensive attempts to solve the formulation address the typically erratic relationship between the structure of the ILP formulation and the solver efficiency.

In addition to the interleaved application of greedy moves and solver attempts, we regularly apply random moves to the formulation in order to move it out of a local optimum of its value. This is accomplished by a series of random moves before entering into the preprocessing loop. For assisting the solver, we further include constraints given by the equations described in Section 3.3.

4.1.1 Experimental Results

We run the proposed algorithms to compute a minimum leakage vector on MCNC '91 circuits. We chose circuits with many inputs as to make exhaustive simulation infeasible. The circuits are synthesized in SIS [13] using `script.delay` to a library consisting of inverters and 2- and 3-input NANDs and NORs. We use the SAT-based ILP solver `galena` with its CNF learning option [11]. For comparison we also apply the LP-based solver OSL [14] from IBM. All experiments are run on a Pentium III 800MHz processor with 256MB of memory.

For providing a baseline, we run the problems using two variants of random search. We run random simulation with 50000 vectors to obtain a bound on the leakage amount and its mean value. The corresponding results are shown in Column "Random Simulation" in Table 2. Furthermore, we apply a genetic algorithm [15] using 21 generations of 1000 vectors (total of 21000 vectors); the results are shown in column "GA".

We then compare the minimization with a SAT-based solver with and without applying the rewriting techniques described in this paper. Without rewriting, the problem is formulated as in [3] using a direct translation from the original circuit to a 0-1 ILP, which is then run with a timeout of 600 seconds. With rewriting, the problem is converted to an AND/INVERTER graph which is amenable to preprocessing and the iterative solve loop using short time-outs. The solver is run at most 11 times for 3 seconds each then once more for 30 seconds.

The best results obtained before termination are shown in columns 7 and 8 of Table 2. Provably optimal results are marked by parentheses along with the used run time. The reported runtimes include only the time spent in the solver. In almost all cases, running the solver with preprocessing shows an improvement compared to running the solver without preprocessing. Note that for `cht`, optimality is provable only after preprocessing.

Compared to the GA bounds, the SAT-based approach does comparatively poorly on `frg2`, `i6`, and `x3`. Our explanation for this is that these circuits are relatively shallow; SAT solvers are able to reason well in deep circuits using logical implications, while in these experiments, we force the solver to reason laterally.

For further comparison, we run the same problems in the LP-based solver OSL using the simplex method to search for feasible solutions given a 600s time-out period. Dashed entries indicate that OSL was not able to find any feasible 0-1 solution within that time limit. The same preprocessing loop is used as before to obtain the final formulation; however, no intermediate solves are performed. Instead of performing moves on the AND/INVERTER graph, the associated equalities described in Section 3.3.1 are emitted to the ILP solver. This is because LP-based solvers perform the same moves internally when given the proper equations. For the examples `C2670` and `frg2`, the solver experiences a dramatic speedup over the original formulation. For `C7552`, the solver is able to find a feasible solution only after preprocessing.

Interestingly, the problems that were difficult for the SAT-based ILP solver are simple for the LP-based ILP solver, while the problems intractable for the LP-based solver are approachable for the SAT-based solver. This demonstrates how the two solvers may complement one another. Overall, the results show that both types of solvers may significantly benefit from preprocessing the ILP instances.

4.2 Peak current estimation

In the peak current problem, we would like to estimate the maximum amount of current that is drawn by a circuit during any state transition. For simplification, we use a zero-delay model, however, more elaborate delay models can be accommodated in a straightforward manner.

For modeling dynamic current, we create two copies of a circuit, and create an XOR between each net and its copy to represent the net switching for a pair of inputs. The XOR is weighted with the the associated switching capacitance; in our case, we simply use the fanout count.

Linear programming can effectively obtain a bound on the optimum. By relaxing the integral constraints, we can often achieve an approximate solution relatively fast. However, typically solutions obtained without preprocessing are not useful. For example, the following inequalities reflect the circuit graph depicted in Figure 3:

$$\begin{aligned} x_1 + (1 - y_2) &\geq 1 \\ (1 - x_2) + (1 - y_2) &\geq 1 \\ (1 - x_1) + x_2 + y_2 &\geq 1 \\ x_1 + (1 - y_4) &\geq 1 \\ x_2 + (1 - y_4) &\geq 1 \\ (1 - x_1) + (1 - x_2) + y_4 &\geq 1 \end{aligned}$$

For this formulation, the relaxed solution $x_1 = x_2 = y_2 = y_4 = 0.5$ is valid. This means that the estimation based on the linear program yields an upper bound for which all circuit nets may switch, which is of little use. The following equation tightens the constraints, by excluding the case where $x_1 = y_2 = y_4 = 0.5$.

$$-y_2 + x_1 - y_4 = 0$$

In general, ILP solvers based on LP will find these types of equalities, albeit extremely slowly. By including these equations in the formulation beforehand during the preprocessing step, we may often get a better LP solution and speed up ILP.

Linear programming is less sensitive than SAT to the structure of the problem, because it rewrites the problem as part of the solution process. However, LP can still benefit from a more compact formulation. For this reason, the specific version of the preprocessing loop outlined previously for this application aims to minimizes the number of terms in the objective for reducing the number of auxiliary constraints.

4.2.1 Experimental Results

To measure the effectiveness of adding the derived equations from Section 3.3, we allow variables to take on continuous values in $[0, 1]$ and run the problems through OSL using the dual simplex method for obtaining infeasibility bounds. Without the derived equations, the obtained upper bound is invariably the trivial upper bound. As the results in Table 3 show, easily identifiable constraints

Table 2: Minimizing Leakage with SAT- and LP- based ILP.

Circuit			Random Simulation		GA	SAT-based ILP (galena)		LP-based ILP (OSL)	
Name	PIs	Levels	Mean	Min	Min	Unprepped	Preprocessed	Unprepped	Preprocessed
C1908	33	33	11059.4	10562.2	10536.1	10669.2	10498.4	(104435) 72s	(104435) 64s
C2670	233	35	14112.2	13616.2	13407.2	13537.9	13029.8	(12875) 23s	(12875) 5s
C6288	32	111	53027.9	50925.3	50463	50317.8	50317.8	———	———
C7552	207	36	49527	48199.7	47998.3	47584.3	47981.2	———	469946
C880	60	25	6759.69	6225.2	5932.4	5870.7	5868.5	(5868.5) 3.8s	(5868.5) 1.8s
apex6	135	15	11925.2	11062.	10931	10819.3	10693.3	(10289.8) 8.9s	(10289.8) 8.0s
b9	41	9	1714.06	1510.2	1462	(1438.8) 20s	(1438.8) 2.75s	(1438.8) 0.2s	(1438.8) 0.1s
cht	47	7	2918.9	2602.3	2506.6	2458.0	(2458.0) 33s	(2458) 0.2s	(2458) 0.1s
cm150a	21	11	869.103	788.8	779.2	(777.8) 0.1s	(777.8) 0.2s	(777.8) 0.1s	(777.8) 0.1s
des	256	23	70153.2	68861.	67951.1	67223.7	66794.4	———	———
frg1	28	17	1913.91	1658.7	1593.7	(1587.3) 8.2s	(1587.3) 6.1s	(1587.3) 0.2s	(1587.3) 0.1s
frg2	143	15	13439.8	12404.	12343.6	12400.8	12863.0	(11827.9) 226s	(11827.9) 11s
i10	257	39	38170.1	36873.4	36497.4	36388.0	35901.2	———	———
i6	138	9	8633.88	8236.4	8140.7	8184.1	8184.1	(7878.5) 3.3s	(7878.5) 1.0s
k2	45	18	21871.6	21037.1	20936.8	21478.9	20997.6	———	———
pair	173	20	24731.5	23520.5	23134	23161.8	23055.7	22464.8	22496.2
vda	17	15	14023.9	13495	13495	(13438.2) 2.4s	(13438.2)2.2s	———	———
x3	135	13	11236.7	10445.6	10247.2	10581.8	10332.4	(9925.9) 4.3s	(9925.9) 6.8s

Table 3: Peak current estimation

Circuit	Original Bound	Tightened Bound	% Difference
C1908	1455	1259	13.5
C2670	1713	1479.5	13.6
C6288	6884	6104	11.3
C7552	6320	5521.5	12.6
C880	797	756	5.1
apex6	1494	1339.5	10.3
b9	194	189	2.6
cht	357	320	10.4
cm150a	90	75	16.7
des	9333	7302.5	21.8
frg1	194	192	1
frg2	1807	1641	9.2
i10	5040	4297	14.7
i6	1026	896	12.7
k2	2837	1953.25	31.2
pair	3027	2756.5	9
vda	1831	1217	33.5
x3	1462	1305.5	10.7

are often quite effective in tightening the bounds returned by an LP solver.

5 Conclusions

In this paper we illustrate a general framework for simplifying an integer program defined over an AND/INVERTER graph. By using structural properties of the graph, effective simplification can be achieved. Assuming that the integer program is hard to solve exactly, we show how these inexpensive manipulations can lead to the optimum or to tighter approximations.

When searching for feasibility bounds with a SAT-based ILP solver, preprocessing allows us to find higher-quality solutions in a shorter amount of time. When searching for a feasible solution with an LP-based ILP solver, the same preprocessing allows a dramatic speedup in the search time.

We show that the structure of the graph can be used for deriving equations to help eliminate logically inconsistent assignments from the continuous relaxation of an ILP. For the peak current problem, these equations result in tighter bounds on the maximum power dissipated.

References

[1] A. Kuehlmann, M. K. Ganai, and V. Paruthi, "Circuit-based Boolean reasoning," in *Proceedings of the 38th ACM/IEEE Design Automation Conference*, pp. 232–237, June 2001.

[2] J. P. Halter and F. N. Najm, "A gate-level leakage power reduction method for ultra-low-power CMOS circuits," in *IEEE Custom Integrated Circuits Conference*, pp. 442–445, 1997.

[3] S. R. Naidu and E. Jacobs, "Minimizing stand-by leaking power in static CMOS circuits," in *Design Automation and Test in Europe*, pp. 370–376, 2001.

[4] S. Devadas, K. Keutzer, and J. White, "Estimation of power dissipation in CMOS combinational circuits using boolean function manipulation," *IEEE Transactions on CAD*, vol. 11, pp. 373–383, Mar. 1992.

[5] H. Kriplani, F. Najm, and I. Hajj, "Pattern independent maximum current estimation in power and ground buses of CMOS VLSI circuits: Algorithms, signal correlations and their resolution," *IEEE Transactions on CAD*, vol. 14, pp. 998–1012, Aug. 1995.

[6] G. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. Wiley, 1988.

[7] R. Fortet, "Applications de l'algebre de Boole en recherche operationelle," *Revue Francaise de Recherche Operationelle*, vol. 4, pp. 17–26, 1960.

[8] R. E. Bryant and Y.-A. Chen, "Verification of arithmetic circuits with binary moment diagrams," in *Design Automation Conference*, pp. 535–541, 1995.

[9] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proceedings of the 34th ACM/IEEE Design Automation Conference*, pp. 263–268, June 1997.

[10] J. Jain, J. Bitner, D. S. Fussel, and J. A. Abraham, "Probabilistic design verification," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 468–471, November 1991.

[11] D. Chai and A. Kuehlmann, "A fast pseudo-Boolean constraint solver," in *Design Automation Conference*, pp. 830–835, 2003.

[12] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th ACM/IEEE Design Automation Conference*, (Las Vegas, Nevada), pp. 530–535, June 2001.

[13] E. Sentovich *et al.*, "SIS: A system for sequential circuit synthesis," Tech. Rep. UCB/ERL M91/41, UC Berkeley, May 1992.

[14] M. S. Hung, W. O. Rom, and A. D. Waren, *Optimization with IBM OSL*. Scientific Press, 1993.

[15] Z. Chen, M. Johnson, L. Wei, and K. Roy, "Estimation of standby leakage power in cmos circuits considering accurate modeling of transistor stacks," in *International Symposium on Low Power Electronic Design*, pp. 239–244, 1998.