Solution and Optimization of Systems of Pseudo-Boolean Constraints

Fadi A. Aloul, *Member*, *IEEE*, Arathi Ramani, *Member*, *IEEE*,
Karem A. Sakallah, *Fellow*, *IEEE*, and Igor L. Markov, *Senior Member*, *IEEE*

Abstract—Optimized solvers for the Boolean Satisfiability (SAT) problem have many applications in areas such as hardware and software verification, FPGA routing, planning, and so forth. Further uses are complicated by the need to express "counting constraints" in conjunctive normal form (CNF). Expressing such constraints by pure CNF leads to more complex SAT instances. Alternatively, those constraints can be handled by Integer Linear Programming (ILP), but generic ILP solvers may ignore the Boolean nature of 0-1 variables. Therefore, specialized 0-1 ILP solvers extend SAT solvers to handle these so-called "pseudo-Boolean" (PB) constraints. This work provides an update on the on-going competition between generic ILP techniques and specialized 0-1 ILP techniques. To make a fair comparison, we generalize recent ideas for fast SAT-solving to more general 0-1 ILP problems that may include counting constraints and optimization. This generalization is embodied in our PB constraint solver and optimizer PBS, which is compared with state-of-the-art CNF and generic ILP solvers. Another aspect of our comparison is the evaluation on 0-1 ILP benchmarks that originate in Electronic Design Automation (EDA) but that cannot be directly solved by an SAT solver. Specifically, we solve instances of the Max-SAT and Max-ONEs optimization problems, which seek to maximize the number of satisfied clauses and the "true" values over all satisfying assignments, respectively. Those problems have straightforward applications to SAT-based routing and are additionally important due to reductions from Max-Cut, Max-Clique, and Min Vertex Cover. Our experimental results show that specialized 0-1 techniques implemented in PBS tend to outperform generic ILP techniques on Boolean optimization problems, as well as on general EDA SAT problems.

Index Terms—Boolean satisfiability (SAT), integer linear programming (ILP), backtrack search, conjunctive normal form (CNF), pseudo-Boolean (PB), Max-SAT, Max-ONE, global routing.

1 Introduction

Recent algorithmic advances in backtrack Boolean Satisfiability (SAT), along with highly efficient solver implementations, have enabled the successful deployment of SAT technology in a wide range of application domains and, particularly, in electronic design automation (EDA). Modern SAT solvers [5], [13], [21], [23], [31] have either displaced or have become essential companions to binary decision diagram (BDD) packages as the Boolean reasoning engines in such applications as formal hardware verification [26], routing of field-programmable gate arrays [24], and automatic test-pattern generation [18]. Their ability to readily solve SAT instances with tens of thousands of variables and millions of conjunctive normal form (CNF) clauses in a matter of seconds or minutes—an impressive feat that would have been impossible to even contemplate just a few years ago—has also encouraged their adaptation to solving some Boolean optimization problems that were traditionally handled as instances of Integer Linear Programming (ILP) [3], [20], [28]. These so-called 0-1 ILP problems call for the minimization or maximization of a linear objective function $c^T x$ subject to a set of m linear constraints[1] $Ax \leq b$, where $b, c \in Z^n$, $A \in Z^m \times Z^n$, and $x \in \{0,1\}^n$. These constraints are commonly referred to as *pseudo-Boolean* (PB) inequalities [3] (to distinguish them from those that admit unrestricted integer variables) and represent a natural generalization of the CNF constraints typically handled by SAT solvers. For example, the CNF clause $(x_1 \vee x_2 \vee \ldots \vee x_k)$ is equivalent to the PB constraint $x_1 + x_2 + \ldots + x_k \geq 1$. PB constraints are more expressive; however, a single PB constraint may, in some cases, correspond to an exponential number of CNF clauses.

Common examples of 0-1 ILPs include Min-COVER [12], Max-SAT, and Max-ONEs [8]. In Min-COVER, we have a collection of subsets of a given set and seek to find a cover of the set using the fewest number of subsets. Logic minimization of Boolean functions, as well as state minimization of finite-state machines are two important instances of this problem. In the Max-SAT problem, the goal is to find a variable assignment that maximizes the number of satisfied CNF clauses in an unsatisfiable SAT instance. Finally, in Max-ONEs, we seek a satisfying variable assignment that maximizes the number of variables set to 1. Max-SAT has direct

- *F.A. Aloul is with the Computer Engineering Department, American University of Sharjah, PO Box 26666, Sharjah, UAE. E-mail: faloul@aus.edu*
- *A. Ramani is with Microsoft Corp., One Microsoft Way, Redmond, WA 98052. E-mail: arathira@windows.microsoft.com.*
- *K.A. Sakallah and I.L. Markov are with the Department of Electrical Engineering and Computer Science, University of Michigan, 2260 Hayward St., Ann Arbor, MI 48109-2121. E-mail: {karem, imarkov}@umich.edu.*

Manuscript received 23 June 2004; revised 22 May 2005; accepted 22 Mar. 2007; published online 7 May 2007.
Recommended for acceptance by F. Lombardi.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0216-0604.
Digital Object Identifier no. 10.1109/TC.2007.1075.

[1] Greater than or equal to and equality constraints are easily accommodated by the equivalences $Ax \geq b \Leftrightarrow -Ax \leq -b$ and $(Ax = b) \Leftrightarrow (Ax \leq b) \wedge (Ax \geq b)$.

application in routing and routability estimation [29] and both Max-SAT and Max-ONEs are important due to reductions from Max-Cut, Max-Clique, and Min Vertex Cover.

Adapting a SAT solver for optimization purposes poses two questions: 1) What should be done with the objective function? and 2) How should the PB constraints be handled? An obvious answer to the first question is to use some form of branch-and-bound around the SAT engine and to prune the search space with best estimates of the objective function value. Alternatively, the objective function can be treated as an auxiliary PB constraint with an adjustable right-hand side, or *goal*. Starting with an easy-to-satisfy goal, a sequence of SAT instances, each with a successively tighter goal, is then constructed and solved. The process continues until an unsatisfiable instance is encountered, indicating that we have converged on the optimal value of the objective function, namely, the goal reached in the last satisfiable instance.

There are also two choices for dealing with the PB constraints: Each PB constraint can be converted into a set of equivalent CNF clauses or the SAT engine is modified to handle PB constraints directly. Conversion to CNF has the advantage of using the SAT solver as a black box but, as mentioned above, suffers from a potential exponential explosion in problem size. This is particularly true of "counting" constraints that impose upper or a lower bounds on the number of certain objects, for example, capacity constraints in routing applications. The increase in size can be reduced from exponential to linear by introducing auxiliary variables that, effectively, decompose a PB constraint into a "structure" of smaller constraints.

Given these various choices, the main question we address in this paper is whether and in what circumstances the currently best generic ILP techniques can compete with specialized SAT-powered 0-1 ILP techniques. Since both generic ILP solvers and specialized 0-1 ILP solvers have consistently improved since Barth's work [3] on the subject, we must ensure up-to-date comparisons. For example, ILOG [15] advertises great ILP performance improvements in CPLEX 7.* over 6.*. On the other hand, the Chaff SAT solver [23], viewed as a *narrowly specialized* 0-1 ILP solver, outperforms earlier competitors by an order of magnitude on many benchmarks. Therefore, in order to convincingly compare the state of the art in generic ILP to that in specialized 0-1 ILP, we need to ensure that the latest techniques are used. In particular, we generalize algorithms used in Chaff to solve 0-1 ILP problems that may include counting constraints and optimization. Our new specialized 0-1 ILP solver, PBS, handles CNF constraints and PB inequalities. Unlike previously proposed stochastic local search solvers [27], this solver is complete and is based on a backtrack search algorithm. We believe that our proposed algorithms to handle PB constraints can be added to any backtrack SAT solver.

The remainder of the paper is organized into five sections: In Section 2, we briefly review the latest enhancements in backtrack CNF-SAT solvers. Section 3 introduces PB constraints and describes how they might be incorporated into a SAT solving scenario. The new PBS solver is described in Section 4 and its performance against best-of-class ILP, specialized 0-1 ILP, and CNF-SAT solvers is analyzed in Section 5. We conclude in Section 6 with a few general observations and suggestions for further work.

2 Anatomy of a Modern CNF-SAT Solver

The satisfiability problem involves finding an assignment to a set of binary variables that satisfies a given set of constraints. In general, these constraints are expressed in CNF or, as it is commonly known as, a product-of-sum form. A CNF formula φ on n binary variables, $x_1, \dots, x_n$, consists of the conjunction (AND) of m clauses, $\omega_1, \dots, \omega_m$, each of which consists of the disjunction (OR) of k literals. A literal l is an occurrence of a Boolean variable or its complement. We will refer to a CNF formula as a clause database.

A variable x is said to be *assigned* when its logical value is set to 0 or 1 and *unassigned* otherwise. A literal l is a *true* (*false*) literal if it evaluates to 1 (0) under the current assignment to its associated variable and a *free literal* if its associated variable is *unassigned*. A clause is said to be *satisfied* if at least one of its literals is true, *unsatisfied* if all of its literals are set to false, *unit* if all but a single literal are set to false, and *unresolved* otherwise. A formula is said to be satisfied if all of its clauses are satisfied and unsatisfied if at least one of its clauses is unsatisfied.

As an example, the CNF instance $f(a,b,c) = (a \vee b)(\bar{b} \vee c)$ consists of three variables, two clauses, and four literals. The assignment $\{a = 0, b = 1, c = 0\}$ leads to a conflict, whereas the assignment $\{a = 0, b = 1, c = 1\}$ satisfies f.

Most modern SAT solvers [13], [21], [23], [31] are based on the original Davis-Putnam-Logemann-Loveland (DPLL) backtrack search algorithm [9]. The algorithm performs a search process that traverses the space of 2^n variable assignments until a satisfying assignment is found (the formula is satisfiable) or all combinations have been exhausted (the formula is unsatisfiable). It maintains a *decision tree* to keep track of variable assignments and can be viewed as consisting of three main engines: the *Decision*, *Deduction, and Diagnosis* engines.

Originally, all variables are unassigned. The algorithm begins by choosing a decision assignment to an unassigned variable. After each decision, the deduction engine determines the implications of the assignment on other variables. This is obtained by forcing the assignment of the variable representing an unassigned literal in an unresolved clause, for which all other literals are assigned to 0, to satisfy the clause. This is referred to as the *unit clause* rule and the repeated application of the unit clause rule over the given clause database is known as Boolean constraint propagation (BCP). If no conflict is detected, the algorithm makes a new decision on a new unassigned variable. Otherwise, the diagnosis engine backtracks by unassigning one or more recently assigned variables and the search continues in another area of the search space.

The power of modern CNF-SAT solvers can be attributed to a few key algorithmic advances and implementation optimizations to the DPLL backtrack procedure, which we summarize below.

2.1 Conflict Diagnosis and Clause Recording

A major advance in backtrack CNF-SAT solvers was the introduction of conflict diagnosis [21] and its tight integration with BCP, nonchronological backtracking, and clause recording. Conflict diagnosis refers to analysis of the implication chains, initiated by elective variables assignments, which cause one or more clauses to become unsatisfied. Such an analysis can identify a small subset of variables whose current assignments can be blamed for the conflict. In addition, these assignments can be turned into a conflict-induced clause that, when added to the clause database, prevents the future occurrence of the same conflict and can be viewed as a form of on-demand *learning*. Finally, recognizing that the current conflict is caused by variable assignments from earlier levels in the decision tree enables nonchronological backtracking, potentially pruning large portions of the search space.

Conflict diagnosis is implemented in most modern backtrack SAT solvers and its effectiveness in pruning the search space has been amply demonstrated empirically. A number of variations have also been studied, including alternative ways of generating conflict clauses and schemes that learn several clauses at each conflict [21], [23]. Recent experimental evidence [32], however, has shown that creating a single conflict clause—based on the unique implication point closest to the conflict—outperforms other schemes on hard instances.

Notwithstanding its effectiveness in pruning the search space, conflict-based learning runs the risk of exponentially increasing the size of the clause database. This is typically avoided by either 1) recording only those conflict-induced clauses with k or fewer literals or 2) deleting conflict-induced clauses after k or more of their literals become unassigned. This clause addition/deletion threshold k is typically set to a value between 100 and 200, indicating that fairly large clauses are created and kept.

2.2 Random Restarts and Backtracking

Besides conflict-based learning, recent studies have shown that using random restarts can be very effective in solving hard SAT instances [2], [23]. An SAT solver may often get stuck in a "bad" region of the search space because of the sequence of decision assignments it had made. The restart process helps to extricate the solver from such regions by periodically resetting all decision and implication assignments and randomly selecting a new sequence of decisions, thus insuring that different subtrees are explored each time. Additionally, all conflict-learned clauses in the various probes of the search space are kept and help boost the effectiveness of subsequent restarts.

Recently, Lynce et al. [19] proposed and empirically evaluated combining random restarts with random backtracking. In this scheme, the diagnosis engine periodically backtracks nonchronologically to a decision level involving *any* literal in the conflict-induced clause. The completeness of the search is preserved by monotonically increasing the clause addition/deletion threshold between random backtracks.

2.3 Improved BCP

On the implementation side, it was observed that a significant fraction of an SAT solver's runtime is spent in the BCP procedure [23]. In a conventional implementation of BCP, an assignment to a variable triggers a traversal of all clauses that contain literals of that variable to check whether they have become unit or are in conflict. In other words, an implication step requires time bounded by the number of occurrences of literals of the assigned variable. This overhead can be significantly reduced by adopting a form of "lazy" evaluation that avoids unnecessary traversals of the clause database. Specifically, rather than keep track of all literals in each clause, the enhanced procedure picks and updates only two unassigned literals per clause (the "watched" literals) and yields a very efficient mechanism for detecting unit clauses [23], [33]. In an SAT instance consisting of n $k\text{-literal}$ clauses, this enhancement reduces BCP overhead from kn to $2n$, which is substantial for typical instances with $k \gg 2$.

2.4 Decision Strategy

Numerous decision (or branching) heuristics have been proposed over the years, with no single heuristic emerging as a clear winner in most cases. One that has been found to be particularly effective in a variety of problems is the Variable State Independent Decaying Sum (VSIDS) heuristic introduced in [23]. The heuristic maintains two counters for every variable that are incremented if a positive (respectively, negative) literal of that variable is identified in a new conflict-induced clause. The variable with the highest counter is selected for the next decision. Counters are also periodically divided by a constant to emphasize variables identified in recent conflicts.

3 Processing of PB Constraints

A PB constraint is in *normal* form if it is expressed as

$$a_1\dot{x}_1 + a_2\dot{x}_2 + \ldots + a_n\dot{x}_n \leq b, \qquad (1)$$

where a_i, $b \in Z^+$, and $\dot{x}_i$ denotes either x_i or $\bar{x}_i$. We will say that $\dot{x}_i$ is a true literal if it evaluates to 1 under the current assignment to its associated variable. An arbitrary PB constraint can be converted to normal form by noting that $\bar{x}_i = 1 - x_i$. For example, $-3x_1 + 2x_2 - x_3 \geq 1$ is first transformed to the "$\leq$" inequality $3x_1 - 2x_2 + x_3 \leq 1$, which, upon substituting $x_2 = 1 - \bar{x}_2$ and rearranging the terms, yields $3x_1 + 2\bar{x}_2 + x_3 \leq 3$.

The two choices for handling PB constraints in an SAT solver are 1) to convert them, in a preprocessing step, to equivalent CNF constraints and 2) to process them directly within the SAT solver.

3.1 PB-to-CNF Conversion

The PB constraint in (1) corresponds to a *threshold* Boolean function [16]. Such functions are unate (monotone) in each of their variables and have unique minimal CNF representations. Minimality here refers to the smallest CNF formula, among all functionally equivalent CNF formulas, namely, the formula that has the fewest number of clauses provided that there is no other such formula with the same number of clauses but with fewer literals. This minimal formula can be

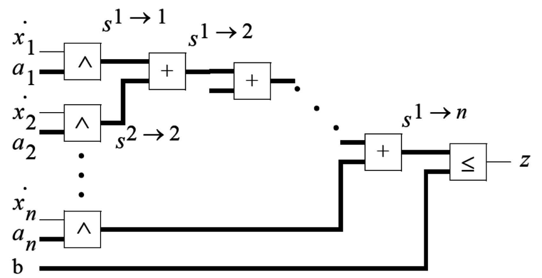


Fig. 1. Circuit representation of (4). Single and multibit signals are denoted by thin and bold lines, respectively.

derived by recursive application of Boole's expansion theorem [6, p. 36]. Let $\varphi(x_1, x_2, \ldots, x_n)$ denote the function of the PB constraint in (1). Expanding around x_i, we get

$$\varphi = (\bar{x}_i \vee \varphi_{x_i})(x_i \vee \varphi_{\bar{x}_i}), \tag{2}$$

where φ_{x_i} and $\varphi_{\bar{x}_i}$ are, respectively, the positive and negative cofactors of φ with respect to x_i. Noting further that φ is either negative or positive unate in x_i allows (2) to be simplified to

$$\varphi = \begin{cases} (\varphi_{x_i})(x_i \vee \varphi_{\bar{x}_i}) & \text{if } \dot{x}_i = \bar{x}_i \\ (\varphi_{\bar{x}_i})(\bar{x}_i \vee \varphi_{x_i}) & \text{if } \dot{x}_i = x_i. \end{cases} \tag{3}$$

Repeated application of (3), distributing $\vee$ over $\wedge$, and making obvious simplifications yields the desired CNF formula. For example, conversion of $3x_1 + 2\bar{x}_2 + x_3 \leq 3$ proceeds as follows:

$$\begin{aligned} \varphi &= (3x_1 + 2\bar{x}_2 + x_3 \leq 3) \\ &= (2\bar{x}_2 + x_3 \leq 3)(\bar{x}_1 \vee (2\bar{x}_2 + x_3 \leq 0)) \\ &= (\bar{x}_1 \vee [(\bar{x}_3)(x_2 \vee 0)]) \\ &= (\bar{x}_1 \vee [(\bar{x}_3)(x_2)]) \\ &= (\bar{x}_1 \vee \bar{x}_3)(\bar{x}_1 \vee x_2). \end{aligned}$$

As mentioned earlier, however, the minimal CNF representation of a PB constraint may have an exponential number of clauses. Specifically, a counting constraint that chooses at most k out of n objects yields $\binom{n}{k+1}(k+1)$-literal clauses [30]. For example, choosing at most 15 out of 30 objects yields 150 million 16-literal clauses and clearly demonstrates the infeasibility of this type of transformation.

The associativity of addition suggests an alternative transformation that yields a CNF formula whose size is linear in n. This transformation can be obtained by introducing auxiliary "partial sum" variables that decompose the monolithic PB constraint into a set of smaller constraints. Letting $s^{i\to i} \equiv a_i\dot{x}_i$ and $s^{i\to j} \equiv \sum_{i\leq k\leq j} s^{k\to k}$, we can rewrite (1) as follows:

$$\underbrace{\underbrace{\underbrace{(((s^{1\to 1} + s^{2\to 2})}_{s^{1\to 2}} + s^{3\to 3})}_{s^{1\to 3}} + \ldots + s^{n\to n})}_{s^{1\to n}} \leq b. \tag{4}$$

Schematically, (4) can be viewed as a multilevel prefix computation [17] "circuit" (see Fig. 1) whose modules represent AND gates, adders, and an output comparator. From this construction, it should be evident that the truth assignments that satisfy (1) are precisely those assignments that set the circuit output z to 1. Specifically

$$\varphi = \exists z, s^{1\to 1}, \ldots, s^{n\to n}, s^{1\to 2}, \ldots, s^{1\to n}(z \wedge \psi), \tag{5}$$

where ψ is the *circuit consistency* function:

$$\begin{aligned} \psi = \quad & (z \leftrightarrow s^{1\to n} \leq b) \wedge \\ & \bigcap_{2\leq i\leq n} (s^{1\to i} \leftrightarrow s^{1\to i-1} + s^{i\to i}) \wedge \\ & \bigcap_{1\leq i\leq n} (s^{i\to i} \leftrightarrow a_i\dot{x}_i). \end{aligned} \tag{6}$$

In other words, the satisfiability of the formula $(z \wedge \psi)$ is equivalent to the satisfiability of the PB formula φ.

The final step in this transformation is the translation of each conjunct in (6) to a set of CNF clauses. This is accomplished by expressing each multibit coefficient and variable in terms of a suitable number of binary encoding variables and invoking the module function (AND, add, and compare) to relate those variables.

There are obvious simplifications in this construction that can eliminate redundant variables and clauses (for example, some of the equivalences in (6) can be replaced by one-way implications). Furthermore, unlike the first transformation in (3), this construction is not unique: Associativity of addition allows the terms in (4) to be grouped in other ways that may reduce the number of CNF variables and yield fewer clauses. Finally, we should point out that this construction is very similar to those described in [29] and [30].

Note that some constraints in specific applications can be expressed efficiently by CNF. Basically, if a constraint can be expressed efficiently in terms of a Boolean circuit, then it can be expressed by CNF with only a linear-space overhead since circuit-SAT can be easily converted into CNF-SAT. On the other hand, if a constraint can be expressed by CNF efficiently, then it can be expressed by a Boolean circuit efficiently since a CNF can be viewed as a Boolean circuit.

3.2 PB-SAT Algorithms

Even when conversion to CNF is feasible, it might be advantageous to process PB constraints directly within an SAT solver. The required bookkeeping is fairly inexpensive, consisting mainly of updating the value of a PB constraint's left-hand side (LHS) to reflect the current truth assignment. Initially, set to 0, LHS is updated as follows:

- If $\dot{x}_i = x_i$, increment LHS by a_i when x_i is set to 1 and decrement it by a_i, when x_i is unassigned from 1; otherwise, leave LHS unchanged.
- If $\dot{x}_i = \bar{x}_i$, increment LHS by a_i when x_i is set to 0 and decrement it by a_i when x_i is unassigned from 0; otherwise, leave LHS unchanged.

Implications. Implications are triggered by a PB constraint for each literal $\dot{x}_i$ whose coefficient a_i satisfies $a_i > b - LHS$. Note that, unlike CNF clauses, a PB constraint can cause the simultaneous implication of several variables. For example, after setting x_1 to 1 in the constraint $3x_1 + 2\bar{x}_2 + x_3 \leq 3$, x_2 and x_3 are immediately implied to 1 and 0, respectively. In general, implications follow the template:

$$\bigcap_{\dot{x}_i \in True} \dot{x}_i \rightarrow \bigcap_{\dot{x}_i \in Large} \bar{\dot{x}}_i,$$

where *True* is the set of true literals and *Large* is the set of literals whose coefficients exceed $(b - LHS)$.

Conflicts. Conflicts are indicated when the current variable assignment causes LHS to exceed b. In this case, we need to choose a subset C of the true literals such that

$$\sum_{\dot{x}_i \in C} a_i > b$$

and return it as a *conflicting assignment* to the diagnosis engine. Ideally, it is desirable to find the smallest such subset, but this is an instance of the KNAPSACK NP-complete problem [12]. Alternatively, a near-minimal subset can be quickly found using the classic heuristic that packs starting from the largest coefficient toward the smallest.

4 The Pseudo-Boolean Solver (PBS)

PBS is a new SAT solver/optimizer written in C++.[2] PBS incorporates all of the modern CNF-SAT solver features described in Section 2. In addition, it handles PB constraints in both optimization and decision (that is, SAT) applications. It uses the "watched literal" data structure from Chaff [23] for CNF constraints and a structure similar to that in SATIRE [28] for PB constraints. Specifically, every PB constraint is represented internally by a record with the following fields:

- A list of the coefficients a_i and their respective literals $\dot{x}_i$. For efficiency, this list is sorted in the order of increasing coefficient values.
- The right-hand side b.
- LHS, which stores the value of the left-hand side under the current variable assignment.

In addition, PBS maintains, for each variable, a list of PB constraints in which the variable occurs positively and another in which it occurs negatively. These lists are used to facilitate the update (according to the rules described in Section 3.2) of the LHS of each relevant PB constraint whenever a variable is assigned or unassigned.

In optimization mode, PBS converts the objective function to a PB constraint with a sliding right-hand-side goal (see Section 1) and proceeds to solve a sequence of SAT instances that differ only in the value of that goal. To illustrate, assume a maximization scenario, denote the sequence of SAT instances by $I_0, I_1, I_2, \ldots$, and let $\hat{g}_i$ be the goal for the ith instance. If the instance is satisfiable, substituting its solution in the objective function constraint should yield a new goal value, $g_i \geq \hat{g}_i$. The goal, for instance $(I_i + 1)$, is now set to $(g_i + 1)$ and the process is repeated. The goal reached in the last satisfiable instance is returned by PBS as the optimal value of the objective function. We refer to this approach as the *linear* search scheme.

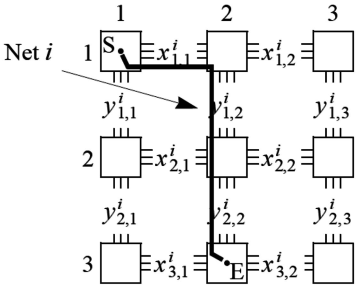


Fig. 2. A 3×3 global routing grid with 12 intercell routing channels. Horizontal and vertical channels are labeled, respectively, with x and y net-to-channel assignment variables. For example, the highlighted 2-pin connection from S in cell (1, 1) to E in cell (3, 2), is specified by $x^i_{1,1} = y^i_{1,2} = y^i_{2,2} = 1$ and 0 for the nine remaining channel variables.

5 Experimental Results

In this section, we report the results of an empirical evaluation of PBS and several other leading-edge solvers on a set of Boolean satisfiability and Boolean optimization benchmarks.

5.1 Benchmarks

We evaluated the various algorithms on three sets of benchmarks. For SAT, we used a set of difficult global routing instances that involve both CNF and PB counting constraints. For Boolean optimization, we chose Max-ONEs and Max-SAT instances from a variety of CNF families.

Global Routing. A set of difficult satisfiable global routing benchmarks was introduced in [1]. Each instance in this family entails the routing of a random set of n two-pin connections (nets) over a two-dimensional grid of cells (see Fig. 2). An r-by-c grid has $m = r(c-1) + c(r-1)$ intercell routing channels. The maximum number of routes that can pass through any channel is referred to as the channel capacity and denoted by C. For each net, a set of m Boolean variables (one per channel) is used to indicate how the net is routed through the grid. In addition, for each channel, a set of C variables per net is introduced to indicate how the net is routed through the channel (that is, the net's "track" assignment in the channel). Thus, the CNF formulation of these instances requires a total of $(1 + C)mn$ variables and consists of two sets of constraints: route definition constraints to express the possible routes that each net can take and capacity constraints to insure that no more than C nets are routed in each channel. These two sets are similar, respectively, to the connectivity and exclusivity constraints for SAT-based FPGA routing [24].

A quick calculation shows that the number of CNF clauses needed to express channel capacity constraints in the above formulation is $(n^2C + nC^2)m$. Using PB modeling, this can be reduced to just m PB inequalities (one per channel) of the form:

$$ch^1 + ch^2 + \ldots + ch^n \leq C,$$

where ch^i denotes the variable that associates net i with channel ch. The PB formulation also eliminates the need for the extra track assignment variables, bringing down the total number of variables to just mn.

[2] The PBS version used in this paper is v3.0, which is based on the SAT engine in the minimal SAT solver MiniSAT [22]. The original version of PBS v2.0 was based on a slower chaff-like in-house-implemented SAT engine.

In the experimental results reported below, the global routing instances are modeled using CNF clauses for route definition and PB inequalities for channel capacity. In addition, we also report on a CNF-only formulation derived by converting the PB capacity constraints using the linear transformation described in Section 3.1.

Max-ONEs. Max-ONEs instances are easily constructed by adding a single PB constraint $x_1 + x_2 + \ldots + x_n \geq b$ to any satisfiable CNF instance, where the goal b is monotonically increased until the instance becomes unsatisfiable. We constructed such instances for representative members from the DIMACS [10], Bejing [14], quasi-group [31], and sat-planning [14] benchmark families.

Max-SAT. Given an unsatisfiable CNF-SAT instance with m clauses, $C_1, C_2, \ldots, C_m$, a Max-SAT instance is constructed by introducing m auxiliary variables $y_1, y_2, \ldots, y_m$, m additional predicates $y_i \leftrightarrow C_i$, and a single objective function PB constraint $y_1 + y_2 + \ldots + y_m \geq b$. Each added predicate $y_i \leftrightarrow C_i$ introduces $|C_i|$ binary clauses and a single $(1 + |C_i|)$-literal clause, where $|C_i|$ is the number of literals in clause C_i. We constructed Max-SAT instances for representative unsatisfiable DIMACS and FPGA switch-box routing (*chnl*) [1] benchmarks.

5.2 Experimental Setup

We conducted several experiments to compare the performance of the new PBS solver (v3.0) against the following:

- the 0-1 ILP solvers OPBDP [4] and SATIRE [28],
- the generic commercial ILP solver CPLEX 7.0 [15], and
- the CNF-SAT solver zChaff [23].

Chaff was used only in the global routing SAT comparisons. All experiments were conducted on a Sun Blade 1000 workstation running SunOS 5.8 and equipped with 512 Mbytes of RAM. We used the default settings for zChaff, OPBDP, and CPLEX and the DLCS decision heuristic for SATIRE. PBS was configured to use the default features in zChaff (that is, all of the features described in Section 2 except for clause deletion, random restarts, and backtracking). A time-out limit of 1,000 seconds was used for each run.

5.3 Results for Global Routing Benchmarks

Table 1 lists the results of solving satisfiable and unsatisfiable global routing instances. The ith routing instance on an $x \times x$ grid with channel capacity y is named *grout-x.y−i*. For each instance, the table indicates the number of nets, the instance size (number of variables |V|, CNF clauses |C|, and PB constraints |PB|) for the hybrid CNF+PB, as well as for the pure CNF formulations, the runtimes of PBS, SATIRE, OPBDP, CPLEX, and zChaff, and the ratio of PBS's runtime to that of the other solvers. The pure CNF formulation was tested only on zChaff.

Clearly, the size of instances, in terms of both variables and clauses, increases significantly for the CNF-only formulation. Pure CNF formulations, thus, are likely to run out of memory for more realistic routing grid sizes, leaving the hybrid CNF+PB formulations as the only viable alternative for this type of SAT problem. Still, it is remarkable that, even when problem size increases fourfold to fivefold, zChaff manages, in some cases, to outperform PBS.

Compared with the two other 0-1 ILP solvers, PBS comes out ahead: It solves all 15 instances, whereas SATIRE solves only 10 and OPBDP, just 5. This can be easily attributed to PBS's incorporation of the latest algorithmic and implementation features of modern CNF-SAT solvers. Compared with CPLEX, on the other hand, PBS does quite poorly; CPLEX beats PBS on most instances, in some cases with a substantial margin.

Finally, we compare PBS with the older version of PBS (v2.0). Clearly, the underlying SAT solver can have a great impact on the efficiency of the 0-1 ILP solver.

5.4 Results for Max-ONEs Benchmarks

The results of the Max-ONEs experiment are listed in Table 2. For each tested instance, the table indicates the instance size (number of variables |V| and clauses |C|), the maximum (that is, optimal) number of 1s in the solution, the runtimes of each of the solvers, and the PBS's speedup ratio. In this set of experiments, PBS outperforms all other solvers, including CPLEX. The only exceptions are the *bw_a* and *bw_b* instances in which SATIRE beats PBS by a minor margin.

5.5 Results for Max-SAT Benchmarks

The results of the Max-SAT experiment are shown in Table 3. For each unsatisfiable instance, the table lists the instance name and size (number of variables |V| and clauses |C|), the size of the corresponding companion satisfiable instance (that is, the satisfiable instance created by adding auxiliary variables and clauses as described earlier), and the minimum (that is, optimal) number of original unsatisfiable clauses (#UnSAT). The remaining columns show the runtimes of the various solvers and PBS's speedup ratio.

In order to speed up the search process for all solvers, WalkSAT [25] was executed for 10 tries as a preprocessing step (with negligible runtime) and the number of unsatisfied clauses it found was used as the initial solution for the optimization runs. It turned out that WalkSAT was able to identify the optimal solution for all tested instances. Thus, only a single run was required for each solver to prove the optimality of that solution. Again, PBS outperformed all other solvers in all cases, except for the pigeon hole instances, which have similar characteristics to the *grout* instances.

5.6 Summary

The above results suggest that combining PB modeling with state-of-the-art SAT algorithms gives PBS a definite performance advantage against other solvers in both optimization and SAT applications. The only anomaly is the unexpectedly good showing of CPLEX on the global routing SAT benchmarks. Unfortunately, lacking knowledge of CPLEX's algorithms, it is difficult to explain why it performs so well on these benchmarks. To better understand its behavior, we tested it on a variety of easy SAT instances from the DIMACS set [10]. The results of those tests are reported in Table 4. In this case, PBS outperforms CPLEX with an even higher margin than that of CPLEX over PBS in the global routing experiments. The only exception is the *hole7* instance, which has similar characteristics to the *grout* instances. This leads us to conjecture that CPLEX incorporates algorithms that

TABLE 1
Runtime Results for Various Global Routing Instances

Instance			Instance Size					Time, sec.						PBS Speedup Over				
			CNF + PB			CNF Only		CNF + PB					CNF					
	Name	Nets	|V|	|C|	|PB|	|V|	|C|	PBS3	SATIRE	OPBDP	CPLEX	PBS2	Chaff	SATIRE	OPBDP	CPLEX	PBS2	Chaff
Satisfiable	**grout-3.3-1**	**18**	216	572	12	864	3692	**0.02**	0.13	1.83	0.03	5.88	0.29	6.5	91.5	1.5	294	14.5
	grout-3.3-2	**22**	264	700	12	1056	4540	**0.02**	0.31	2	0.03	14.4	0.29	15.5	100	1.5	719	14.5
	grout-3.3-3	**20**	240	636	12	960	4116	0.06	0.36	2.75	**0.03**	1.24	0.12	6	46	0.5	20.7	2.00
	grout-3.3-4	**19**	228	604	12	912	3904	0.05	0.07	1.96	**0.03**	5.23	0.2	1.4	39	0.6	105	4.00
	grout-3.3-5	**20**	240	634	12	960	4114	0.11	0.12	2.81	**0.02**	10.3	0.52	1.1	26	0.18	94	4.73
	grout-4.3-1	**28**	672	2004	24	2688	11844	38.5	37.8	>1K	**0.13**	1.77	5.3	0.98	>26	0	0	0.14
	grout-4.3-2	**27**	648	1928	24	2592	11408	351	8.28	>1K	**0.35**	0.59	1.21	0.02	>2.85	0	0	0.00
	grout-4.3-3	**27**	648	1930	24	2592	11410	**0.61**	145	>1K	0.63	1.07	7.86	237	>1639	1.03	1.8	12.9
	grout-4.3-4	**29**	696	2072	24	2784	12272	11.2	>1K	>1K	**0.16**	217	38.7	>89	>89	0.01	19.3	3.45
	grout-4.3-5	**30**	720	2144	24	2880	12704	1.6	267	>1K	**0.16**	1.86	16.6	167	>625	0.1	1.2	10.4
	grout-4.3-6	**26**	624	1860	24	2496	10980	4.24	>1K	>1K	**0.32**	427	78.3	>236	>236	0.08	101	18.5
	grout-4.3-7	**28**	672	2006	24	2688	11846	0.3	>1K	>1K	**0.14**	225	31.5	>3333	>3333	0.47	750	105
	grout-4.3-8	**18**	432	1280	24	1728	7520	2	70	>1K	**0.18**	0.95	1.79	35	>500	0.09	0.5	0.90
	grout-4.3-9	**35**	840	2502	24	3360	14862	0.63	>1K	>1K	**0.14**	1.43	162	>1587	>1587	0.22	2.3	257
	grout-4.3-10	**35**	840	2504	24	3360	14864	1.6	>1K	>1K	**0.14**	9.88	4.83	>625	>625	0.09	6.2	3.02
Unsatisfiable	**grout-3.3-1u**	**26**	312	828	12	1248	5388	**0.01**	0.61	3.63	0.03	2.53	0.26	61	363	3	253	26
	grout-3.3-2u	**28**	336	888	12	1344	5808	0.07	0.79	26.5	**0.03**	48.2	0.38	11.3	379	0.4	688	5.4
	grout-3.3-3u	**26**	312	824	12	1248	5384	0.07	0.18	27.9	**0.03**	36.1	0.05	2.6	399	0.4	516	0.7
	grout-3.3-4u	**28**	336	890	12	1344	5810	**0.01**	0.34	7	0.02	3.8	0.02	34	700	2	380	2
	grout-3.3-5u	**27**	324	858	12	1296	5598	0.09	2.95	43.8	**0.04**	5.95	4.96	32.8	487	0.4	66.1	55.1
	grout-4.3-1u	**84**	1008	3000	24	4032	17880	50.4	>1K	>1K	**0.11**	>1K	23.3	>19.8	>19.8	0.0	>19.8	0.5
	grout-4.3-2u	**78**	936	2790	24	3744	16590	0.12	>1K	>1K	**0.11**	5.36	1.73	>8333	>8333	0.9	44.7	14.4
	grout-4.3-3u	**88**	1056	3148	24	4224	18748	1.27	>1K	>1K	**0.23**	710	4.89	>787	>787	0.2	559	3.9
	grout-4.3-4u	**88**	1056	3144	24	4224	18744	**0.11**	>1K	>1K	0.12	17.4	0.2	>9091	>9091	1.1	158	1.8
	grout-4.3-5u	**88**	1056	3152	24	4224	18752	0.22	>1K	>1K	**0.11**	93.3	6.04	>4546	>4546	0.5	424	27.5
	grout-4.3-6u	**82**	984	2932	24	3936	17452	1.45	>1K	>1K	**0.16**	>1K	85	>690	>690	0.1	>690	58.6
	grout-4.3-7u	**88**	1056	3150	24	4224	18750	0.21	>1K	>1K	**0.15**	11.4	132	>4762	>4762	0.7	54.2	629
	grout-4.3-8u	**82**	984	2934	24	3936	17454	1.33	>1K	>1K	**0.11**	894	0.16	>752	>752	0.1	672	0.1
	grout-4.3-9u	**88**	1056	3144	24	4224	18744	2.05	>1K	>1K	**0.11**	267	122	>488	>488	0.1	130	59.5
	grout-4.3-10u	**102**	1224	3644	24	4896	21764	0.59	>1K	>1K	**0.12**	2.44	0.14	>1695	>1695	0.2	4.1	0.2

recognize and simplify certain structured problems (such as the pigeon hole and global routing instances), but not general structured EDA problems (such as the bridging fault *bf0432-007* and stuck-at-fault *ssa7552-038* instances).

In summary, generic ILP solvers, such as CPLEX, seem to be inadequate for solving Boolean optimization problems and the majority of Boolean satisfiability problems. Similarly, previous specialized 0-1 ILP solvers such as SATIRE and OPBDP perform poorly on almost all problems due to the limited SAT enhancements implemented in these solvers. The latest specialized 0-1 ILP solver, PBS, outperforms all of the presented solvers, except for some specific structured problems in which the commercial CPLEX solver wins. Furthermore, expressing constraints in CNF and PB allows for a significant reduction in memory and substantial speedup. It also allows for optimization problems to be efficiently solved using SAT-based techniques.

We should note that the search used in our experiments is *linear*, that is, it searches for the optimal solution in increments/decrements of 1. It is a possibility that a faster search scheme such as *binary* search could reduce this overhead. However, our experiments with binary search on PBS indicate that it actually performs worse than linear search. One reason for this may be that the starting values of objective function picked for the Max-ONEs experiments are very close to the actual optimal value, so the ordinary benefit of binary search is lost. Additionally, when using the linear search scheme, all conflict-induced clauses are

TABLE 2
Results of the Max-ONEs Experiment

Benchmark	**Satisfiable Instance**				**Time, sec.**				**PBS3 Speedup Over**		
Family	**Name**	**|V|**	**|C|**	**Max-ONEs**	**PBS3**	**SATIRE**	**OPBDP**	**CPLEX**	**SATIRE**	**OPBDP**	**CPLEX**
DIMACS [10]	**aim-50-1_6-yes1-1**	50	80	29	**0**	0.01	**0**	0.07	-	1	-
	aim-100-1_6-yes1-1	100	160	43	**0**	0.01	2.57	285	-	-	-
	aim-200-2_0-yes1_1	200	400	96	**0.03**	0.08	>1000	>1000	2.7	>33K	>33K
	ii8b1	336	2068	275	**6.07**	>1000	19.2	23	>165	3.2	3.8
	jnh1	100	850	55	**0.06**	0.67	**0.06**	21.4	11.2	1	356.7
	par8-1	350	1149	79	**0.03**	**0.03**	0.06	0.52	1	2.0	17.3
Bejing [14]	**3blocks**	283	9690	63	**1.4**	14.74	788	>1000	10.5	563	>714
QG [31]	**qg7-09**	729	22060	81	**0.16**	2.08	9.39	3.52	13	58.7	22
	qg6-09	729	21844	81	**0.23**	2.08	18.84	18.84	9	81.9	81.9
Satplan-sat [14]	**bw_a**	459	4675	73	0.15	**0.11**	0.35	0.37	0.7	2.3	2.5
	bw_b	1087	13772	136	2.1	**1.44**	6.2	2.98	0.7	3	1.4
	bw_c	3016	50457	272	**91**	162	>1000	419	1.8	>11	4.6

TABLE 3
Results of the Max-SAT Experiment

Unsatisfiable Instance			**Satisfiable Instance**		**#UnSAT**	**Time, sec.**				**PBS3 Speedup Over**		
Name	**|V|**	**|C|**	**|V|**	**|C|**		**PBS3**	**SATIRE**	**OPBDP**	**CPLEX**	**SATIRE**	**OPBDP**	**CPLEX**
aim-100-1_6-no-1	100	160	260	640	1	**0.001**	**0.001**	247	14.16	1.0	247K	14K
bf0432-007	1040	3668	4708	13242	1	**0.32**	2.26	331	>1000	7.1	1K	>3K
dubois30	90	240	330	960	1	**0.001**	0.16	>1000	>1000	160	>1M	>1M
hole7	56	204	260	652	1	14.72	946	19.3	**0.02**	64.3	1.3	0.0
jnh14	100	850	950	5013	2	**1.56**	82.4	>1000	258.46	52.8	>641	166
jnh211	100	800	900	4688	2	**2.1**	54	>1000	79.59	25.7	>476	37.9
jnh308	100	900	1000	5310	2	**3.01**	212	>1000	526.42	70.4	>332	175
jnh8	100	850	950	4997	2	**1.93**	79.4	>1000	88.4	41.1	>518	45.8
jnh9	100	850	950	5006	2	**1.35**	113	>1000	339.63	83.7	>741	251.6
pret150_25	150	400	551	1601	1	**0.02**	0.15	>1000	>1000	7.5	>50K	>50K
ssa0432-003	435	1027	1462	3391	1	**0.001**	0.07	1.45	8.14	70	1450	8140
chnl3_4	24	44	68	140	2	**0.01**	0.12	**0.01**	0.12	12	1	12
chnl4_5	40	90	130	290	2	**0.02**	3.55	0.46	2.86	178	23	143
chnl5_6	60	162	222	522	2	**0.55**	240	29.45	122.68	436	53.5	223
chnl6_7	84	266	350	854	2	**13.24**	>1000	>1000	>1000	>75.5	>75.5	>75.5

retained when performing successive searches, which may help the solver's performance. This difference in performance will have to be studied more closely to investigate trade-offs, for example, if the initial value of the objective is actually far from the optimal value, binary search may have more of a payoff.

6 Conclusions

In this work, we studied discrete optimization and decision problems related to Boolean satisfiability that can be tackled with 1) generic ILP solvers and 2) specialized 0-1 ILP solvers. We showed that the trade-offs between these methods are sensitive to the current state of the art, which has considerably changed over the last several years. Noting the success of modern SAT solvers in handling extremely large instances (tens of thousands of variables and millions of clauses) despite the well-recognized worst-case complexity of CNF-SAT suggested that logic-based methods might be a viable alternative to general-purpose ILP techniques. This work further pushes the performance envelope of 0-1 ILP techniques by combining it with the best CNF-SAT methods. We implemented a new 0-1 ILP solver, PBS, which uses the latest advances in Boolean satisfiability. We showed how PBS can handle both decision and optimization problems. We compared PBS against four previously existing implementations, including the leading generic commercial ILP solver CPLEX [15], the CNF-SAT solver Chaff [23], and two specialized 0-1 ILP solvers [4], [28].

In particular, we evaluated PBS on instances of the Max-SAT and Max-ONEs optimization problems, whose significance is due to a reduction from the Max-Cut and

TABLE 4
PBS versus CPLEX on DIMACS SAT Instances

Instance	SAT/	Time, sec.		PBS3
	UNS	PBS3	CPLEX	Speedup
aim-50-1_6-no-1	U	**0**	0.16	-
aim-50-1_6-yes1-1	S	**0**	0.19	-
bf0432-007	U	**0.19**	>1000	>5263
dubois20	U	**0.01**	>1000	>100K
hole7	U	0.27	**0.03**	0.11
ii8c1	S	**0**	0.36	-
jnh1	S	**0**	3.17	-
par8-1-c	S	**0**	0.02	-
pret60_25	U	**0.01**	>1000	>100K
ssa0432-003	U	**0.01**	7.2	720
ssa7552-038	S	**0.01**	0.68	68

Max-Clique problems, respectively, as well as applications to min-wirelength routing. We also experimented with decision SAT problems that appear in global routing and other EDA domains.

Algorithmic and benchmarking contributions aside, our two most important suggestions for the EDA community are as follows:

- consider generic ILP solvers in the context of highly structured 0-1 constraint satisfaction (that is, Boolean SAT) problems and
- consider specialized 0-1 ILP techniques in the context of 0-1 optimization problems.

Although we did not find a universal "rule of thumb," we did identify relevant trends for most types of instances that we worked with. Additionally, tool developers may benefit from learning techniques used by their colleagues, although we do not know what techniques are currently used by the commercial tool, CPLEX, to solve ILP problems. Furthermore, encodings that utilize CNF and PB constraints in applications such as routing can be much more compact than pure CNF and generally lead to faster runtimes.

By significantly improving the efficiency of SAT-based applications, particularly routing, we hope to facilitate new uses of 0-1 techniques. Our on-going work in this direction includes embedding SAT-based routers into realistic algorithmic flows and benchmarking them against best known geometric algorithms.

Our progress on the Max-ONEs problem—a fundamental but unfortunately overlooked formulation—also opens new applications of SAT-based solvers. Our future work will study applications to Max-Clique, Max Independent Set, and Min Vertex Cover, which are fairly popular problems in logic synthesis and other areas of design automation. In addition, we are considering methods to further prune the search space by 1) enhancing the optimization capabilities of PBS by incorporating lower/upper bound estimations of the value of the objective function and 2) extending the conflict diagnosis engine to learning PB constraints rather than CNF constraints.

Since the publishing of this work at ICCAD 2002, 0-1 ILP has received a lot more exposure and many specialized 0-1 ILP solvers and potential applications have come to the forefront. Notably, Dixon and Ginsberg [11] extended the SAT solver RELSAT [5] to handle PB constraints and described a method for deriving PB constraints. Chai and Kuehlmann [7] described a watch-literal strategy that is applicable for BCP on PB constraints and presented a general algorithm for learning PB constraints based on cutting planes. The ideas presented by Chai and Kuehlmann were implemented in a tool called Galena [7], which was shown to be faster than PBS on several benchmarks. Nevertheless, one of the main contributions of this paper and the earlier version from ICCAD 2002 is extending the applicability of 0-1 ILP by developing better algorithms and making available a better solver. The introduction of PBS has sparked a new wave of 0-1 ILP solvers.

ACKNOWLEDGMENTS

This work was funded by the DARPA/MARCO Gigascale Silicon Research Center.

REFERENCES

[1] F Aloul, A Ramani, I Markov, and K Sakallah, "Solving Difficult SAT Instances in the Presence of Symmetry," *Proc. Design Automation Conf. (DAC '02),* pp. 731-736, 2002.

[2] L. Baptista and J.P. Marques-Silva, "Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability," *Proc. Sixth Int'l Conf. Principles and Practice of Constraint Programming (CP '00),* 2000.

[3] P. Barth, "A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization," Technical Report MPI-I-95-2-003, Max-Planck-Institut Für Informatik, 1995.

[4] P. Barth, "OPBDP: A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization," http://www.mpi-sb.mpg.de/units/ag2/software/opbdp, 2007.

[5] R. Bayardo, Jr. and R. Schrag, "Using CSP Look-Back Techniques to Solve Real World SAT Instances," *Proc. Nat'l Conf. Artificial Intelligence,* pp. 203-208, 1997.

[6] F.M. Brown, *Boolean Reasoning.* Kluwer Academic, 1990.

[7] D. Chai and A. Kuehlmann, "A Fast Pseudo-Boolean Constraint Solver," *Proc. Design Automation Conf. (DAC '03),* pp. 830-835, 2003.

[8] N. Creignou, S. Kanna, and M. Sudan, *Complexity Classifications of Boolean Constraint Satisfaction Problems.* SIAM, 2001.

[9] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving," *Comm. ACM,* vol. 5, no. 7, pp. 394-397, 1962.

[10] DIMACS Challenge Benchmarks, ftp://Dimacs.rutgers.EDU/pub/challenge/sat/benchmarks/cnf, 2007.

[11] H. Dixon and M. Ginsberg, "Inference Methods for a Pseudo-Boolean Satisfiability Solver," *Proc. Nat'l Conf. Artificial Intelligence,* pp. 635-640, 2002.

[12] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[13] E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust SAT-Solver," *Proc. Design, Automation, and Test in Europe Conf. (DATE '02),* pp. 142-149, 2002.

[14] H. Hoos and T. Stützle, http://www.satlib.org, 2007.

[15] ILOG CPLEX, http://www.ilog.com/products/cplex, 2007.

[16] Z. Kohavi, *Switching and Finite Automata Theory,* second ed. McGraw-Hill, 1978.

[17] R. Ladner and R. Fischer, "Parallel Prefix Computation," *J. ACM,* vol. 27, no. 4, pp. 831-838, 1980.

[18] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. Computer-Aided Design,* vol. 11, no. 1, pp. 4-15, 1992.

[19] I. Lynce, L. Baptista, and J. Marques-Silva, "Stochastic Systematic Search Algorithms for Satisfiability," *Proc. LICS Workshop Theory and Applications of Satisfiability Testing,* 2001.

[20] V. Manquinho and J. Marques-Silva, "On Using Satisfiability-Based Pruning Techniques in Covering Algorithms," *Proc. Design Automation and Test Conf. in Europe,* pp. 356-363, 2000.

[21] J. Marques-Silva and K. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Trans. Computers,* vol. 48, no. 5, pp. 506-521, May 1999.

[22] N. Een and N. Sorensson, "An Extensible SAT-Solver," *Theory and Applications of Satisfiability Testing,* pp. 502-518, 2003.

[23] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," *Proc. Design Automation Conf. (DAC),* pp. 503-535, 2001.

[24] G. Nam, F. Aloul, K. Sakallah, and R. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints," *Proc. Int'l Symp. Physical Design (ISPD '01),* pp. 222-227, 2001.

[25] B. Selman, H. Kautz, and B. Cohen, "Noise Strategies for Local Search," *Proc. Nat'l Conf. Artificial Intelligence,* pp. 337-343, 1994.

[26] M. Velev and R. Bryant, "Effective Use of Boolean Satisfiability Procedure in the Formal Verification of Superscalar and VLIW Mircroprocessors," *Proc. Design Automation Conf. (DAC '01),* pp. 226-231, 2001.

[27] J. Walsor, "Solving Linear Pseudo-Boolean Constraint Problems with Local Search," *Proc. Nat'l Conf. Artificial Intelligence,* pp. 269-274, 1997.

[28] J. Whittemore, J. Kim, and K. Sakallah, "SATIRE: A New Incremental Satisfiability Engine," *Proc. Design Automation Conf. (DAC '01),* pp. 542-545, 2001.

[29] H. Xu, R. Rutenbar, and K. Sakallah, "Sub-SAT: A Formulation for Relaxed Boolean Satisfiability with Applications in Routing," *Proc. Int'l Symp. Physical Design (ISPD '02),* 2002.

[30] J. Warners, "A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form," *Information Processing Letters,* vol. 68, no. 2, pp. 63-69, 1998.

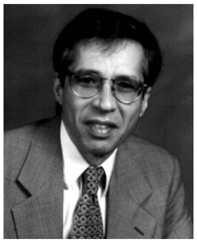
[31] H. Zhang, "SATO: An Efficient Propositional Prover," *Proc. Int'l Conf. Automated Deduction,* pp. 272-275, 1997.

[32] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik, "Efficient Conflict Driven Learning in a Boolean Satisfiability Solver," *Proc. Int'l Conf. Computer-Aided Design (ICCAD '01),* pp. 279-285, 2001.

[33] H. Zhang and M. Stickel, "An Efficient Algorithm for Unit-Propagation," *Proc. Int'l Symp. Artificial Intelligence and Math.,* pp. 166-169, 1996.

Fadi A. Aloul received the BS degree (summa cum laude) in electrical engineering from Lawrence Technological University, Southfield, Michigan, in 1997 and the MS and PhD degrees in computer science and engineering from the University of Michigan, Ann Arbor, in 1999 and 2003, respectively. He was a postdoctoral research fellow at the University of Michigan, Ann Arbor, during the summer of 2003. In the summer of 2005, he was a visiting researcher with the Advanced Technology Group, Synopsys, Portland, Oregon. He is currently an assistant professor of computer engineering at the American University of Sharjah (AUS), Sharjah, United Arab Emirates. His research interests include computer-aided design, combinatorial optimization, Boolean satisfiability, and computer security. He has published more than 45 refereed papers and has presented invited talks and tutorials at several industrial sites, universities, and conferences. He has also developed several tools for Boolean Satisfiability, including the pseudo-Boolean SAT solver and optimizer PBS. He has received a number of awards, including the Agere/SRC research fellowship, the GANN fellowship, and the LTU presidential scholarship. He has served on the technical program committees of many conferences. He was a local organizer of the 2006 IEEE International Conference on Computer Systems and Applications (AICCSA) and the AV chair of the 2003 International Workshop on Logic Synthesis (IWLS). He is a member of the IEEE, the ACM, and Tau Beta Pi. He is currently the GOLD chair of the IEEE UAE Section.

Arathi Ramani received the BS degree in computer engineering from Thadomal Shahani Engineering College, affiliated with the University of Mumbai, India, in 1999 and the MS and PhD degrees in computer engineering from the University of Michigan, Ann Arbor, in 2002 and 2005, respectively. Her research interests include algorithms for combinatorial optimization, hypergraph partitioning, and physical design of integrated circuits. Shee is currently working at Microsoft Corp., Redmond, Washington. She is a member of the IEEE.



Karem A. Sakallah received the BE degree in electrical engineering from the American University of Beirut, Beirut, Lebanon, in 1975 and the MSEE and PhD degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 1977 and 1981, respectively. In 1981, he was with the Department of Electrical Engineering at Carnegie Mellon University as a visiting assistant professor. From 1982 to 1988, he was with the Semiconductor Engineering Computer-Aided Design Group, Digital Equipment Corporation, Hudson, Massachusetts, where he headed the Analysis and Simulation Advanced Development Team. Since September 1988, he has been with the University of Michigan, Ann Arbor, as a professor of electrical engineering and computer science. From September 1994 to March 1995, he was with the Cadence Berkeley Laboratory, Berkeley, California, on a six-month sabbatical leave. His current research interests include computer-aided design with emphasis on logic and layout synthesis, Boolean satisfiability, discrete optimization, and hardware and software verification. He has authored or coauthored more than 200 papers and has presented seminars and tutorials at many professional meetings and various industrial sites. He was an associate editor of the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* from 1995 to 1997 and has served on the program committees of the International Conference on Computer-Aided Design, Design Automation Conference, and the International Conference of Computer Design, as well as numerous other workshops. He is currently an associate editor of the *IEEE Transactions on Computers*. He is a fellow of the IEEE and a member of the ACM and Sigma Xi.

Igor L. Markov received the MA degree in mathematics and the PhD degree in computer science from the University of California, Los Angeles, in 1994 and 2001, respectively. He is currently an associate professor in the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. His research interests include combinatorial optimization with applications to the design and verification of integrated circuits, as well as quantum logic circuits. He has published more than 100 refereed papers. In 2001, he was awarded the DAC Fellowship and received the IBM University Partnership Award. He received the 2004 IEEE CAS Donald O. Pederson paper-of-the-year award and the 2004 ACM SIGDA Outstanding New Faculty Award. He was the recipient of the best paper award at Design Automation and Test in Europe Conference (DATE) 2005 in the Circuit Test category, as well as the US National Science Foundation CAREER and the Synplicity Inc. Faculty Awards. He has served on the technical program committees at the Design Automation Conference, International Conference on Computer-Aided Design, DATE Conference, International Symposium on Physical Design, and several other IEEE conferences and symposia. He served as the general chair and the technical program committee chair of the International Workshop on System-Level Interconnect Prediction. He is a member of the ACM and American Mathematical Society (AMS) and is a senior member of the IEEE.