Generic ILP versus Specialized 0-1 ILP: An Update

Fadi A. Aloul, Arathi Ramani, Igor L. Markov, Karem A. Sakallah

Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122

{faloul, ramania, imarkov, karem}@eecs.umich.edu

ABSTRACT

Optimized solvers for the Boolean Satisfiability (SAT) problem have many applications in areas such as hardware and software verification, FPGA routing, planning, etc. 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" 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. Another aspect of our comparison is evaluation on 0-1 ILP benchmarks that originate in Electronic Design Automation (EDA), but that cannot be directly solved by a 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 tend to outperform generic ILP techniques on Boolean optimization problems as well as on general EDA SAT problems.

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 [17, 18, 26] 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 [21], routing of field-programmable gate arrays [19], and automatic test-pattern generation [14]. 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 solve some Boolean optimization problems that were traditionally handled as instances of Integer Linear Programming (ILP) [3, 16, 23]. These so-called 0-1 ILP problems call for the minimization or maximization of a linear objective function $\mathbf{c}^T\mathbf{x}$ subject to a set of m linear constraints[1] $\mathbf{A}\mathbf{x} \le \mathbf{b}$ where $\mathbf{b},\mathbf{c} \in \mathbb{Z}^n$, $\mathbf{A} \in \mathbb{Z}^m \times \mathbb{Z}^n$, and $\mathbf{x} \in \{0,1\}^n$. These constraints are commonly referred to as *pseudo-Boolean* (PB) inequalities (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 $(x_1 \vee x_2 \vee \cdots \vee x_k)$ is equivalent to the PB constraint $x_1 + x_2 + \cdots + x_k \ge 1$. 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 [9], Max-SAT and Max-ONEs [6]. 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 application in routing and routability estimation [24], 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

[1] Greater-than-or-equal and equality constraints are easily accommodated by the equivalences $\mathbf{A}\mathbf{x} \ge \mathbf{b} \Leftrightarrow -\mathbf{A}\mathbf{x} \le -\mathbf{b}$ and $\mathbf{A}\mathbf{x} = \mathbf{b} \Leftrightarrow (\mathbf{A}\mathbf{x} \le \mathbf{b}) \wedge (\mathbf{A}\mathbf{x} \ge \mathbf{b})$.

0-7803-7607-2/02/$17.00 ©2002 IEEE

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, e.g. 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 can the currently-best generic ILP techniques 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 [11] advertises great ILP performance improvements in CPLEX 7.* over 6.*. On the other hand, the Chaff SAT solver [18], 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 [22], 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 in 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 in 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 power of modern CNF-SAT solvers can be attributed to a few key algorithmic advances and implementation optimizations to the basic Davis-Logemann-Loveland [7] 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 [17] and its tight integration with Boolean constraint propagation (BCP), non-chronological backtracking, and clause recording. Conflict diagnosis refers to analysis of the implication chains, initiated by elective variables assignments, that 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 non-chronological 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 [17, 18]. Recent experimental evidence [27], 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 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, 18]. A 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. [15] proposed and empirically evaluated combining random restarts with random backtracking. In this scheme, the diagnosis engine periodically backtracks non-chronologically 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 a SAT solver’s run time is spent in the BCP procedure [18]. 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 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 [18, 28]. In a SAT instance consisting of n k-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 [18]. The heuristic maintains two counters for every variable that are incremented if a positive (resp. 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:

$$a_1\dot{x}_1 + a_2\dot{x}_2 + \cdots + a_n\dot{x}_n \le b \tag{1}$$

where $a_i, b \in \mathbb{Z}^+$ and $\dot{x}_i$ denotes either x_i or $\overline{x}_i$. 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 $\overline{x}_i = 1 - x_i$. For example, $-3x_1 + 2x_2 - x_3 \ge -1$ is first transformed to the "$\le$" inequality $3x_1 - 2x_2 + x_3 \le 1$ which, upon substituting $x_2 = 1 - \overline{x}_2$ and re-arranging terms, yields $3x_1 + 2\overline{x}_2 + x_3 \le 3$.

The two choices for handling PB constraints in a SAT solver are 1) to convert them, in a pre-processing 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 [12]. 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 there is no other such formula with the same number of clauses but with fewer literals. This minimal formula can be derived by recursive application of Boole's expansion theorem [5, p. 36]. Let $\varphi(x_1, x_2, \cdots, x_n)$ denote the function of the PB constraint in (1). Expanding around x_i we get:

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

where φ_{x_i} and $\varphi_{\overline{x}_i}$ 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:

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

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

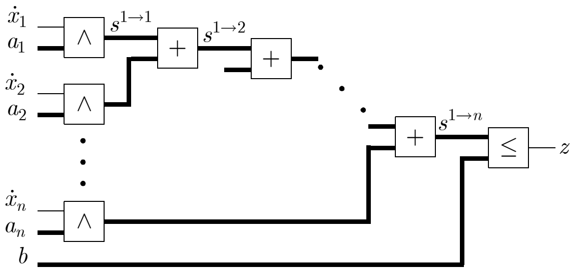

Figure 1: Circuit representation of (4). Single- and multi-bit signals are denoted, respectively, by thin and bold lines.

$$\begin{aligned}\varphi &= (3x_1 + 2\overline{x}_2 + x_3 \le 3) \\ &= (2\overline{x}_2 + x_3 \le 3)(\overline{x}_1 \vee (2\overline{x}_2 + x_3 \le 0)) \\ &= (1)(\overline{x}_1 \vee [(x_3 \le 0)(x_2 \vee (x_3 \le -2))]) \\ &= (\overline{x}_1 \vee [(\overline{x}_3)(x_2 \vee 0)]) \\ &= (\overline{x}_1 \vee [(\overline{x}_3)(x_2)]) \\ &= (\overline{x}_1 \vee \overline{x}_3)(\overline{x}_1 \vee x_2)\end{aligned}$$

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 [25]. 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 $s^{i\to i} \equiv a_i\dot{x}_i$, $s^{i\to j} \equiv \sum_{i\le k\le j} s^{k\to k}$, we can re-write (1) as follows:

$$\underbrace{\underbrace{\underbrace{(((s^{1\to 1} + s^{2\to 2})}_{s^{1\to 2}} + s^{3\to 3})}_{s^{1\to 3}} + \cdots + s^{n\to n})}_{s^{1\to n}} \le b \tag{4}$$

Schematically, (4) can be viewed as a multi-level prefix computation [13] "circuit" (see Figure 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,

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

where ψ is the *circuit consistency function*

$$\psi = (z \leftrightarrow s^{1\to n} \le b) \wedge \bigwedge_{2\le i\le n} (s^{1\to i} \leftrightarrow s^{1\to i-1} + s^{i\to i}) \wedge \bigwedge_{1\le i\le n} (s^{i\to i} \leftrightarrow a_i\dot{x}_i) \tag{6}$$

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 multi-bit 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 (e.g., 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 [24] and [25].

3.2 PB-SAT Algorithms

Even when conversion to CNF is feasible, it might be advantageous to process PB constraints directly within a 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 $\dot{x}_i = \overline{x}_i$, 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 $a_i > b - \text{LHS}$. 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 $3x_1 + 2\overline{x}_2 + x_3 \leq 3$, x_2 and x_3 are immediately implied to 1 and 0, respectively. In general, implications follow the template $\bigwedge_{\dot{x}_i \in True} \dot{x}_i \rightarrow \bigwedge_{\dot{x}_i \in Large} \bar{\dot{x}}_i$ where *True* is the set of true literals and *Large* is the set of literals whose coefficients exceed $(b - \text{LHS})$.

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. Alternatively, a near-minimal subset can be quickly found using the classic heuristic that packs starting from the largest coefficient towards the smallest.

4 THE PSEUDO-BOOLEAN SOLVER PBS

PBS is a new SAT solver/optimizer, written in C++, that incorporates all of the modern CNF-SAT solver features described in Section 2. In addition, it handles PB constraints in both optimization and decision (i.e., SAT) applications. It uses the "watched literal" data structure from Chaff [18] for CNF constraints, and a structure similar to that in SATIRE [23] for PB constraints. Specifically, every PB constraint is represented internally by a record with the following fields:

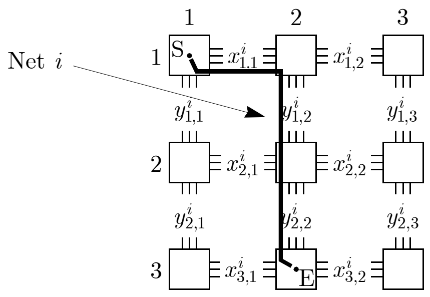

Figure 2: A 3-by-3 global routing grid with 12 inter-cell routing channels. Horizontal and vertical channels are labeled, respectively, with x and y net-to-channel assignment variables. For example, the highlighted 2-pin connection from S in cell (1,1) to E in cell (3,2), is specified by $x^i_{1,1} =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

- 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, . . . 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.

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 Figure 2). An r-by-c grid has $m = r(c - 1) + c(r - 1)$ inter-cell 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 (i.e., 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 [19].

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

$$ch^1 + ch^2 + \cdots + ch^n \leq C$$

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.

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 $x_1 + x_2 + \cdots + x_n \geq b$ 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 [8], Bejing [10], quasi-group [26], and sat-planning [10] benchmark families.

Max-SAT. Given an unsatisfiable CNF-SAT instance with m clauses $C_1, C_2, \cdots, C_m$, a Max-SAT instance is constructed by introducing m auxiliary variables $y_1, y_2, \cdots, y_m$, m additional predicates $y_i \leftrightarrow C_i$, and a single objective function PB constraint $y_1 + y_2 + \cdots + y_m \geq b$. 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 [1] benchmarks.

5.2 Experimental Setup

We conducted several experiments to compare the performance of the new PBS solver against:

- The 0-1 ILP solvers OPBDP [4] and SATIRE [23]
- The generic commercial ILP solver CPLEX 7.0 [11]
- The CNF-SAT solver Chaff [18]
- The Chaff-based Max-SAT solver sub-SAT [24]

Chaff was used only in the global routing SAT comparisons, and sub-SAT was only used in the Max-SAT comparisons. Except for the CPLEX runs, all experiments were done on a Pentium-II 333MHz workstation running Linux and equipped with 512 MB of RAM. The CPLEX experiments were conducted on a 440 MHz UltraSPARC workstation with a 2MB cache running SunOS 5.8. We used the default settings for Chaff, OPBDP, and CPLEX, and the DLCS decision heuristic for SATIRE. PBS was configured to use all of the features described in Section 2 except for clause deletion and random backtracking. A time-out limit of 5,000 seconds was used for each run.

Table 1: Run time results for various global routing instances.

Instance		Instance Size					Time, sec.					PBS Speedup			
		CNF + PB			CNF Only		CNF + PB				CNF				
Name	**Nets**	**|V|**	**|C|**	**|PB|**	**|V|**	**|C|**	**PBS**	**SATIRE**	**OPBDP**	**CPLEX**	**Chaff**	**SATIRE**	**OPBDP**	**CPLEX**	**Chaff**
grout-3.3-1	**18**	216	572	12	864	3692	1.72	0.41	4.51	**0.05**	3.25	0.24	2.62	0.03	1.9
grout-3.3-2	**22**	264	700	12	1056	4540	0.33	0.96	4.65	**0.05**	1.15	2.9	14.1	0.16	3.5
grout-3.3-3	**20**	240	636	12	960	4116	0.09	1.1	6.65	**0.05**	1.63	12	74	0.58	18
grout-3.3-4	**19**	228	604	12	912	3904	1.29	0.2	4.73	**0.05**	1.30	0.16	3.67	0.04	1.01
grout-3.3-5	**20**	240	634	12	960	4114	0.84	0.35	6.88	**0.04**	1.50	0.42	8.19	0.05	1.8
grout-4.3-1	**28**	672	2004	24	2688	11844	3.46	109.7	5000	**0.29**	254	32	>1445	0.08	73
grout-4.3-2	**27**	648	1928	24	2592	11408	1.92	32.13	5000	**0.81**	203	17	>2604	0.42	106
grout-4.3-3	**27**	648	1930	24	2592	11410	5.52	319.47	5000	**1.52**	145	58	>906	0.28	26
grout-4.3-4	**29**	696	2072	24	2784	12272	16.3	3772	5000	**0.31**	74.4	231	>307	0.02	4.6
grout-4.3-5	**30**	720	2144	24	2880	12704	2.06	567.12	5000	**0.47**	274	275	>2427	0.23	133
grout-4.3-6	**26**	624	1860	24	2496	10980	27	5000	5000	**1.04**	117	>185	>185	0.04	4.3
grout-4.3-7	**28**	672	2006	24	2688	11846	55	5000	5000	**0.23**	743	>91	>91	0.01	13.5
grout-4.3-8	**18**	432	1280	24	1728	7520	2.9	177.8	5000	**0.38**	60	61	>1724	0.13	21
grout-4.3-9	**35**	840	2502	24	3360	14862	58	5000	5000	**0.29**	271	>86	>86	0.01	4.7
grout-4.3-10	**35**	840	2504	24	3360	14864	7.4	5000	5000	**0.21**	159	>676	>676	0.03	21.5

5.3 Results for Global Routing Benchmarks

Table 1 lists the results of solving fifteen global routing instances. The ith routing instance on an x-by-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 run times of PBS, SATIRE, OPBDP, CPLEX, and Chaff, and the ratio of PBS's run time to that of the other solvers. The pure CNF formulation was tested only on Chaff.

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 four-to-five fold, Chaff manages in some cases to match PBS's run time within a factor of two or three.

Compared with the two other 0-1 ILP solvers, PBS comes out ahead: it solves all fifteen instances whereas SATIRE solves only eleven, and OPBDP just five. 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 all instances, in some cases with a substantial margin.

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 (i.e., optimal) number of 1s in the solution, the run times of each of the solvers, and PBS's speedup ratio. In this set of experiments, PBS outperforms all other solvers including CPLEX. The only exception is the jnh1 instance which both PBS and CPLEX solve in a fraction of a second.

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 (i.e., the satisfiable instance created by adding auxiliary variables and clauses as described earlier) and the minimum (i.e., optimal) number of original unsatisfiable clauses (#UnSAT). The remaining columns show the run times of the various solvers and PBS's speedup ratio.

In order to speed up the search process for all solvers, WalkSAT [20] was executed for 10 tries as a pre-processing step (with negligible run time), 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 most cases. The only exception this time was that sub-SAT was significantly faster on the FPGA routing benchmarks. This should not be surprising since sub-SAT was designed, mainly, for FPGA routing applications. We conjecture, further, that modeling channel capacity constraints using PB inequalities rather than CNF clauses, might give PBS a performance edge even in these cases.

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 [8]. The results of those tests are reported in Table 4. In this case, PBS outperforms CPLEX with an even

Table 2: Results of the Max-ONEs experiment

Benchmark Family	Satisfiable Instance				Time, sec.				PBS Speedup		
	Name	|V|	|C|	Max-ONEs	PBS	SATIRE	OPBDP	CPLEX	SATIRE	OPBDP	CPLEX
DIMACS [8]	**aim-50-1_6-yes1-1**	50	80	29	**0.01**	0.01	0.02	0.11	1	2	11
	aim-100-1_6-yes1-1	100	160	43	**0.01**	0.02	7.19	41	2	719	4100
	aim-200-2_0-yes1_1	200	400	96	**0.01**	0.06	5000	5000	6	>500K	>500K
	ii8b1	336	2068	275	**4.69**	3180	56.2	8.85	678	12	1.89
	jnh1	100	850	55	0.32	2.2	**0.12**	85.56	6.88	0.38	267.4
	par8-1	350	1149	79	**0.01**	0.06	0.05	1.03	6	5	103
Bejing [10]	**3blocks**	283	9690	63	**4.83**	49.53	4494	5000	10.3	930	1035
QG [26]	**qg7-09**	729	22060	81	**0.1**	5.41	9.8	17.92	54.1	98	179.2
	qg6-09	729	21844	81	**0.21**	5.56	45	564.65	26.5	214	2689
Satplan-sat [10]	**bw_a**	459	4675	73	**0.03**	0.43	0.21	0.51	14.3	7	17
	bw_b	1087	13772	136	**0.58**	6.39	17.86	66.16	11	31	114
	bw_c	3016	50457	272	**24.37**	315.5	5000	986.2	13	>205	40.5

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 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 problems 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.

6 CONCLUSIONS

In this work we studied discrete optimization and decision problems related to Boolean satisfiability that can be tackled with (i) generic ILP solvers, and (ii) specialized 0-1 ILP solvers. We showed that the trade-offs between these methods are sensitive to the current state-of-the-art, which considerably changed over the last several years. Additionally, our work further pushes the performance envelope of 0-1 ILP techniques. We implemented a new 0-1 ILP solver, PBS, that uses the latest advances in Boolean satisfiability and compared it against four previously existing implementations, including the leading generic commercial ILP solver CPLEX [11], the currently-fastest CNF-SAT solver Chaff [18] and two specialized 0-1 ILP solvers [4, 23].

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 Max-Clique problems, respectively, as well as applications to min-wirelength routing. We also experimented with 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:

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

While 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 pseudo-Boolean constraints, in applications such as routing, can be much more compact than pure CNF and generally lead to faster run times.

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

Table 3: Results of the Max-SAT experiment

Unsatisfiable Instance			Satisfiable Instance		#UnSAT	Time, sec.					PBS Speedup			
Name	**|V|**	**|C|**	**|V|**	**|C|**		**PBS**	**SATIRE**	**OPBDP**	**CPLEX**	**SubSAT**	**SATIRE**	**OPBDP**	**CPLEX**	**Sub-SAT**
aim-100-1_6-no-1	100	160	260	640	1	**0.01**	0.02	5000	8.7	0.02	2	>500K	870	2.0
bf0432-007	1040	3668	4708	13242	1	**0.35**	7.88	5000	5000	0.86	22	>14K	>14K	2.5
dubois30	90	240	330	960	1	**0.02**	0.5	5000	5000	0.05	25	>250K	>250K	2.5
hole7	56	204	260	652	1	0.24	11.92	22.54	**0.01**	1.8	50	94	0.04	7.5
jnh14	100	850	950	5013	2	**3.43**	291.27	5000	461.3	26.2	85	>1458	134	7.6
jnh211	100	800	900	4688	2	**2.94**	196.25	5000	154.3	27.2	67	>1701	52	9.3
jnh307	100	900	1000	5265	3	**30.44**	5000	5000	5000	146.2	>164	>164	>164	4.8
jnh308	100	900	1000	5310	2	**6.51**	600.16	5000	5000	70.2	92	>768	>768	10.8
jnh8	100	850	950	4997	2	**2.69**	271.16	5000	193.5	11	101	>1858	72	4.1
jnh9	100	850	950	5006	2	**4.36**	349.7	5000	624.7	29.4	80	>1147	143	6.7
pret150_25	150	400	551	1601	1	**0.02**	0.02	5000	5000	385	1.0	>250K	>250K	19K
ssa0432-003	435	1027	1462	3391	1	**0.01**	0.25	5000	4.52	0.09	25	>500K	452	9
fpga3_4	24	44	68	140	2	**0.01**	0.4	0.01	0.18	**0.01**	40	1.0	18	1.0
fpga4_5	40	90	130	290	2	0.1	11.97	1.46	3.91	**0.03**	120	15	39	0.30
fpga5_6	60	162	222	522	2	7.04	485.27	91.12	220.8	**0.09**	69	13	31	0.01
fpga6_7	84	266	350	854	2	319	5000	5000	5000	**1.42**	>15.7	>15.7	>15.7	0.00

Table 4: PBS vs. CPLEX on DIMACS SAT Instances

Instance	SAT/UNS	Time, sec.		PBS
		PBS	CPLEX	Speedup
aim-50-1_6-no-1	U	**0.01**	0.24	24
aim-50-1_6-yes1-1	S	**0.01**	0.36	36
bf0432-007	U	**0.38**	5000	>13K
dubois20	U	**0.02**	5000	>250K
hole7	U	3.73	**0.01**	0.002
ii8c1	S	**0.01**	1.22	122
jnh1	S	**0.03**	27.91	930
par8-1-c	S	**0.01**	0.14	14
pret60_25	U	**0.03**	5000	>166K
ssa0432-003	U	**0.02**	9.47	474
ssa7552-038	S	**0.01**	64.2	6420

synthesis and other areas of design automation. In addition, we plan to enhance the optimization capabilities of PBS by incorporating lower/upper bound estimations of the value of the objective function to further prune the search space.

ACKNOWLEDGMENTS

This work is funded by the DARPA/MARCO Gigascale Silicon Research Center, an Agere Systems/SRC Research fellowship, and a DAC graduate scholarship.

REFERENCES

[1] F. Aloul, A. Ramani, I. Markov, and K. Sakallah, "Solving Difficult SAT instances in the Presence of Symmetry," in *Proc. of the Design Automation Conference*, pp. 731-736, 2002.

[2] L. Baptista and J. P. Marques-Silva, "Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability," in *Proc. of the 6th International Conference on Principles and Practice of Constraint Programming (CP)*, 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.*

[5] F. M. Brown, "Boolean Reasoning," *Kluwer Academic Publishers*, 1990.

[6] N. Creignou, S. Kanna, and M. Sudan, "Complexity Classifications of Boolean Constraint Satisfaction Problems", *Society for Industrial and Applied Mathematics* (SIAM), 2001.

[7] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving," *Communications of the ACM*, 5(7), pp. 394-397, 1962.

[8] DIMACS Challenge benchmarks, *ftp://Dimacs.rutgers.EDU/pub/challenge/sat/benchmarks/cnf.*

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

[10] H. Hoos and T. Stützle, *http://www.satlib.org.*

[11] ILOG CPLEX, *http://www.ilog.com/products/cplex.*

[12] Z. Kohavi, "Switching and Finite Automata Theory," *Second ed. McGraw-Hill*, 1978.

[13] R. E. Ladner and R. E. Fischer, "Parallel Prefix Computation", *J. of the ACM* (JACM), 27(4), pp. 831-838, 1980.

[14] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. on CAD*, 11(1), pp. 4-15, 1992.

[15] I. Lynce, L. Baptista, and J. P. Marques-Silva, "Stochastic Systematic Search Algorithms for Satisfiability," in *the LICS Workshop on Theory and Apps of Satisfiability Testing*, 2001.

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

[17] J. P. Marques-Silva and K. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Trans. on Computers*, 48(5), pp. 506-521, 1999.

[18] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proc. of the Design Automation Conference*, pp. 530-535, 2001.

[19] G. Nam, F. Aloul, K. Sakallah, and R. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints," in *Proc. of the Int'l Symposium on Physical Design*, pp. 222-227, 2001.

[20] B. Selman, H. Kautz, and B. Cohen. "Noise strategies for local search," in *Proc. of the National Conference on Artificial Intelligence*, pp. 337-343, 1994.

[21] M. Velev and R. Bryant, "Effective use of Boolean Satisfiability Procedure in the Formal Verification of Superscalar and VLIW Mircroprocessors," in *Proc. of the Design Automation Conference*, pp. 226-231, 2001.

[22] J. Walsor, "Solving Linear Pseudo-Boolean Constraint Problems with Local Search," in *Proc. of the National Conference on Artificial Intelligence*, pp. 269-274, 1997.

[23] J. Whittemore, J. Kim, and K. Sakallah, "SATIRE: A New Incremental Satisfiability Engine," in *Proc. of the Design Automation Conference*, pp. 542-545, 2001.

[24] H. Xu, R. Rutenbar, K. Sakallah, "sub-SAT: A Formulation for Relaxed Boolean Satisfiability with Applications in Routing," in *Proc. of the Int'l Symposium on Physical Design*, 2002.

[25] J. P. Warners, "A linear-time transformation of linear inequalities into conjunctive normal form," in *Information Processing Letters*, 68(2), pp. 63-69, 1998.

[26] H. Zhang, "SATO: An Efficient Propositional Prover," *in Int'l Conference on Automated Deduction*, pp. 272-275, 1997.

[27] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik, "Efficient Conflict Driven Learning in a Boolean Satisfiability Solver," in *Proc. of the Int'l Conference on Computer-Aided Design*, pp. 279-285, 2001.

[28] H. Zhang, and M. Stickel, "An efficient algorithm for unit-propagation," in *Proc. of the Int'l Symposium on Artificial Intelligence and Mathematics*, pp. 166-169, 1996.