These experiments have also shown that, in general, both *same rise–fall* and *greedy* approaches obtain results very close to the optimum, at least for the library used. This happens despite the fact that the library contained gates with significantly different rise and fall delays. We believe this is due to the fact that even for circuits with a moderate number of levels (say 5), the imbalances in the individual rise and fall gate delays cancel out by the time the primary outputs are reached. Nevertheless, we were able to show for the first time that for the load-independent delay model in the presence of rise and fall delays, *same rise–fall* and *greedy* approaches work quite well, achieving exact results on 43 and 57 circuits, respectively, out of 73.

There are several interesting directions for future work. One obvious direction for future research is the search for an exact pseudopolynomial algorithm for general combinational circuits. Another interesting area of research is the application of this method to the technology mapping problem. This should represent a relatively simple extension, as long as the load-independent delay model is assumed. Note that a generalization of the algorithm to the load-dependent delay model is not likely, since it has been recently proved that the gate assignment problem under the load-dependent delay model is NP-complete in the strong sense [9].

The memory usage currently represents the most significant bottleneck faced by the algorithm. It limits the applicability of the implementation to circuits with tens of thousands of gates and libraries with very fine delay granularities. Additional work is needed on the data structures used in the manipulation of the tables.

References

[1] C. L. Berman, J. L. Carter, and K. F. Day, "The fanout problem: From theory to practice," in *Advanced Research in VLSI: Proc. 1989 Decennial Caltech Conf.*, C. L. Seitz, Ed. Cambridge, MA: MIT Press, Mar. 1989, pp. 69–99.

[2] P. Chan, "Algorithms for library-specific sizing of combinational logic," in *Proc. Design Automation Conf.*, 1990, pp. 353–356.

[3] O. Coudert, R. Haddad, and S. Manne, "New algorithms for gate sizing: A comparative study," in *Proc. Design Automation Conf.*, 1996, pp. 734–739.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979, Mathematical Sciences Series.

[5] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimum Elmore delay," in *Int. Symp. Circuits and Systems*, 1990, pp. 865–868.

[6] Y. Kukimoto, R. K. Brayton, and P. Sawkar, "Delay-optimal technology mapping by DAG covering," in *Proc. Design Automation Conf.*, 1998, pp. 348–351.

[7] W. N. Li, A. Lim, P. Agarwal, and S. Sahni, "On the circuit implementation problem," in *Proc. Design Automation Conf.*, 1992, pp. 478–483.

[8] J. Lillis, C. K. Cheng, and T. T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," in *Proc. International Conf. Computer Aided Design*, 1995, pp. 138–143.

[9] R. Murgai, "On the complexity of minimum-delay gate resizing/technology mapping under load-dependent delay model," in *Int. Workshop Logic Synthesis*, 1999.

[10] ——, "Performance optimization under rise and fall parameters," in *Proc. Int. Conf. Computer Aided Design*, 1999, pp. 185–190.

[11] A. L. Oliveira and R. Murgai, "An exact gate assignment algorithm for tree circuits under rise and fall delays," in *Proc. ACM/IEEE Int. Conf. Computer Aided Design*, 2000, pp. 451–457.

[12] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *Proc. Int. Conf. Computer Design*, Oct. 1992, pp. 328–333.

[13] K. J. Singh, "Performance optimization of digital circuits," Ph.D. dissertation, Univ. Calif., Berkeley, Dec. 1992.

[14] I. Sutherland, R. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*. San Francisco, CA: Morgan Kaufmann, 1999.

[15] H. Touati, "Performance-oriented technology mapping," Ph.D. dissertation, UCB/ERL M90/109, Univ. Calif., Berkeley, Nov. 1990.

[16] H. Vaishnav and M. Pedram, "Routability-driven fanout optimization," in *Proc.Design Automation Conf.*, 1993, pp. 230–235.

sub-SAT: A Formulation for Relaxed Boolean Satisfiability With Applications in Routing

Hui Xu, Rob A. Rutenbar, and Karem Sakallah

***Abstract*—Advances in methods for solving Boolean satisfiability (SAT) for large problems have motivated recent attempts to recast physical design problems as Boolean SAT problems. One persistent criticism of these approaches is their inability to supply partial solutions, i.e., to satisfy most but not all of the constraints cast in the SAT style. In this paper, we present a formulation for "subset satisfiable" Boolean SAT: we transform a "strict" SAT problem with N constraints into a new, "relaxed" SAT problem which is satisfiable just if not more than $k \ll N$ of these constraints cannot be satisfied in the original problem. We describe a transformation based on explicit thresholding and counting for the necessary SAT relaxation. Examples from field-programmable gate-array routing show how we can determine efficiently when we can satisfy "almost all" of our geometric constraints.**

***Index Terms*—Boolean satisfiability.**

I. Introduction

The last decade has seen striking advances in our ability to pose and solve large Boolean satisfiability (SAT) problems. Early work on binary decision diagrams (BDDs) [1] has led to a large set of decision diagram implementations and functional variants with different capabilities [2]–[8]. BDDs allow us to represent and manipulate arbitrary sets of Boolean equations; SAT comes as a by-product of the BDD data structure. In those applications where we only need to ascertain SAT and find—if available—a single solution, direct SAT solvers have been developed to perform the necessary search for a satisfying solution, or prove that no such solution exists. Several such solvers exist today (e.g., [9]–[13]) with the best of them capable of handling thousands of variables and millions of logical clauses (constraints). Though aimed at logic and verification applications (see [14] for a recent survey), the existence of these solvers makes it attractive to ask how well we might be able to translate geometric design problems into Boolean problems.

There have been several attempts in this direction. Early attempts used BDDs for representation and manipulation. Devadas showed a formulation for simple two-layer channel routing using a sequence of BDDs [15]. Wood developed a more general routing formulation based on BDDs, and also demonstrated how field-programmable gate-arrays (FPGAs) were a natural target for such an approach, given their discrete routing resources [16]. The advantage of the BDD formulation is that all routing alternatives for all nets are represented explicitly.

Manuscript received May 31, 2002; revised November 25, 2002. This work was supported in part by the National Science Foundation under Contract CCR-9971142. This paper was recommended by Associate Editor C. J. Alpert.

H. Xu and R. A. Rutenbar are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: huix@ece.cmu.edu; rutenbar@ece.cmu.edu).

K. A. Sakallah is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109–2122 USA (e-mail: karem@umich.edu).

Digital Object Identifier 10.1109/TCAD.2003.811450

A satisfying assignment of the variables represents a solution for how to embed all nets concurrently. This is also a disadvantage of BDDs; the number of routing alternatives may be too large in total. More recently, Schmiedle *et al.* formulated gridded maze routing in an entirely symbolic form, using BDDs to model maze wavefront expansion in a style similar to finite-state machine-symbolic reachability analysis [17]. Sulimma showed a column-wise gridded channel router, also representing all routing options symbolically, on a per column basis [18]. The difficulty again is that BDDs limit these tools to problems of very small size.

To get around these problems, Nam *et al.* introduced a more efficient mapping from routing resources to Boolean variables [19], [20], and abandoned BDDs in favor of a direct SAT solver [11]. These made it possible to formulate routing for complete, though small, FPGAs as a single, solvable SAT problem. Variations on this theme, such as ECO-style routing, were also demonstrated [21]. Recent work [22] shows how carefully coupling a conventional router with a SAT formulation can handle FPGA problems of industrial scale, e.g., 10 000 nets.

However, even assuming that progress on SAT-solver engines continues apace, and that we can formulate large-scale geometric problems as solvable SAT problems, these approaches still suffer from two fundamental problems.

- **No quality metric**: Boolean SAT formulations are intrinsically binary. Boolean variables represent solution alternatives, Boolean formulas represent constraints. All satisfying solutions are equivalent. We have no cost mechanisms to favor one over another.
- **No partial solutions**: SAT formulations either satisfy or they do not; there is no middle ground. We might be happy with a solution that routes 999 out of 1000 nets, but in a SAT formulation, the unroutability of the entire problem means that a SAT solution simply returns "no," and not a useful 999-net partial solution.

In this paper, we attack the "no partial solutions" problem, expanding on an earlier version of this work we developed in [23]. Our strategy is to transform the original problem into an augmented, relaxed SAT problem which is satisfiable just if some small, arbitrary subset of the original constraints are violated, and all other constraints are met. We refer to this as subset-satisfiability, or sub-SAT for short. The idea is to permit users of SAT-based tools to set thresholds that yield incomplete, but useful, SAT solutions. The key is to create transformations that are general, yet yield SAT problems that we can still solve with current SAT engines.

The remainder of the paper is organized as follows. Section II briefly motivates our approach in more detail. Section III presents details of an SAT-transformation based on explicit thresholding and counting of the relaxed constraints. Section IV then shows experimental results from both FPGA routing problems, and a more general set of common SAT benchmark problems. Finally, Section V offers some concluding remarks.

II. Motivation and Approach

The routing example of the previous section is worth revisiting, since it provides a concise example of what we expect in a partial solution. Let us assume that our 1000-net problem is intrinsically unroutable, but that 999 of 1000 nets can be embedded. There are circumstances where a strict SAT formulation is desirable for this problem, i.e., when we are analyzing an over-congested routing region of moderate size, and asking the question "can we ever route this completely?" On the other hand, there are two strong arguments for why a partial solution is desirable.

- **More consistent with traditional routers**: If we actually seek to use SAT for detailed routing, a partial solution is more consistent with the behavior of traditional routers and flows. Today, detailed routing often progresses through a sequence of increasing levels of effort. If a few nets fail, there is always a "next" effort level that increases the aggressiveness of the search process, at the cost of some CPU time. If we terminate the router early, we always have some partial solution.
- **SAT style more abstract than geometric style**: The process of mapping from geometric to Boolean constraints necessarily abstracts the problem, and results in the loss of some fidelity to an unconstrained geometric solution. The fact that one net out of 1000 cannot be routed really means that our SAT abstraction of the geometric constraints cannot be solved. We might easily complete this last net with a more conventional router, which is willing to embed a more adventurous path than we would tolerate in the SAT abstraction. By returning partial solutions, we hope to encourage hybrid solutions that exploit the best characteristics of SAT (concurrent embedding of all nets) with the best of traditional routers (tenacious shape-level search for the last few paths).

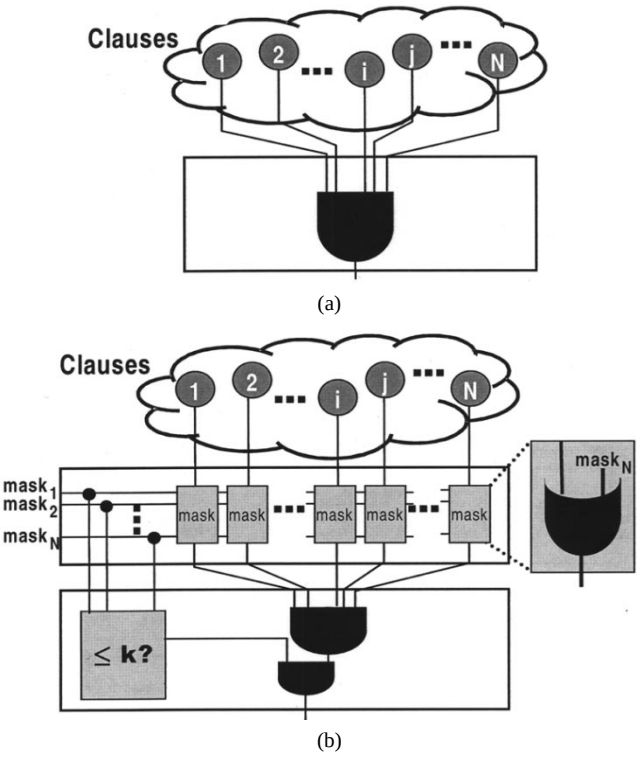

Fig. 1. Formulating subset SAT by augmenting the original CNF form with masking and counting. (a) Strict SAT formulation. (b) Relaxed (augmented CNF) SAT formulation.

Our principal focus is on partial solutions that are "almost" satisfiable: all but a very few constraints are satisfied. We assume that the user gives a threshold k for how many constraints may be violated, and k is very small in relation to the total number of constraints. We transform the initial SAT problem into a new SAT problem which, if satisfiable, yields a suitably complete partial solution to the original task.

III. sub-SAT via Transformation and Counting

A. Basic Formulation

We assume our SAT problem is formulated in standard conjunctive normal form (CNF), where each constraint is a disjunction of literals (either true or complemented instances of Boolean variables). In this form, the constraints are commonly referred to as clauses, though we shall use the two terms interchangeably. A simple example with five variables and four clauses is

$$(\bar{X}_3 \vee X_4 \vee \bar{X}_5) \wedge (X_1 \vee \bar{X}_2) \wedge (X_1 \vee \bar{X}_3 \vee X_4) \wedge (X_2 \vee \bar{X}_5). \quad (1)$$

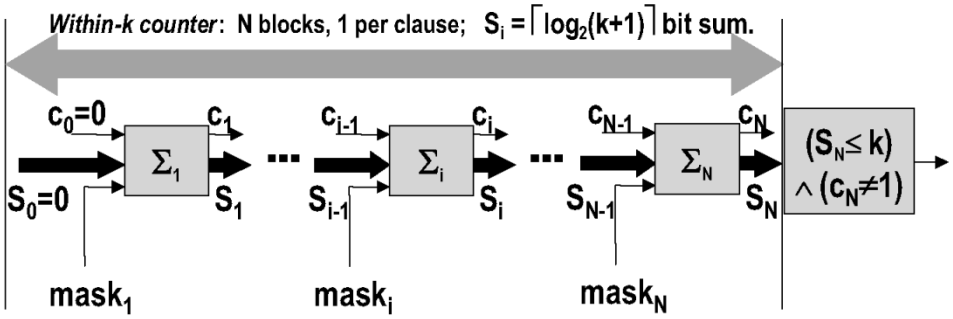

Fig. 2. Implementing the threshold counter [at bottom-left of Fig. 1(b)] as a linear *within-**k*** structure.

Assume a general CNF formula is written over M variables $\{x_1, x_2, \ldots, x_M\}$ and has N total clauses.

Fig. 1 illustrates our strategy for supporting partial solutions, and makes precise our notion of a relaxed SAT problem. In a conventional strict formulation, we must find an assignment of each variable that renders each CNF clause identically 1. It is helpful to think of the CNF formula in terms of digital logic: we must satisfy each clause, and the "hardware" to check that all clauses are satisfied is a single N-input AND gate. To relax this, we allow some arbitrary subset of not more than k constraints to "opt out" of the conjunction. In terms of logic, we replace the single AND with new logic that allows us to mask a subset of the constraints, rendering the final conjunction insensitive to whether these masked constraints are met or not.

As shown in Fig. 1(b), we can override a single constraint by ORing in the masking bit for that constraint. Setting this bit automatically forces the clause to be satisfied, and creates a value we can later count. We refer to these as masked constraints. A separate piece of logic determines if the number of masked constraints is not more than an arbitrary threshold k. The new relaxed problem is satisfied just if the nonmasked constraints are all satisfied, and the number of masked (potentially unsatisfied) constraints is not more than k.

Fig. 2 shows the logic to handle the counting task. Since we are assuming $k \ll N$ opt-out clauses, we do not need a large adder structure. We adopt a variant of the "within-k" linear counter circuit from [1]. Each Σ_i block is essentially an incrementer circuit, operating on a $\lceil \log_2(k+1) \rceil$-bit value S_i. Each block tallies the next mask bit. Some extra logic checks to ensure that we do not overflow our small counter; this is the c_i value that propagates stage to stage, and ensures our final count is accurately less than or equal to k. Each stage computes the following:

$$c_i = \begin{cases} 1, & \text{if } \left(S_{i-1} = 2^{\lceil log_2(k+1) \rceil}\right) \text{ and } \mathrm{mask}_i = 1 \\ c_{i-1}, & \text{else} \end{cases}$$
$$S_i = \begin{cases} (S_{i-1}+1), & \text{if} \left(S_{i-1} < 2^{\lceil log_2(k+1) \rceil}\right) \text{ and } \mathrm{mask}_i = 1 \\ S_{i-1}, & \text{else.} \end{cases} \quad (2)$$

We implement each of these incrementer structures as additional clauses in the CNF formula. All the logic functions are transformed into a product-of-sums form. For example:

$$\begin{aligned} a = b \quad &\Rightarrow (a + \bar{b})(\bar{a} + b) \\ \text{if } a, \text{ then } b \quad &\Rightarrow (\bar{a} + b). \end{aligned} \quad (3)$$

Then, we can write them as a CNF file. These additional mask variables, counter variables, and the clauses to make this logic, augment the initial CNF representation. The original CNF formula is relaxed in the sense that it may admit more solutions than the original strict form, but the price is a larger transformed CNF formula. Let $s = \lceil \log_2(k+1) \rceil$;

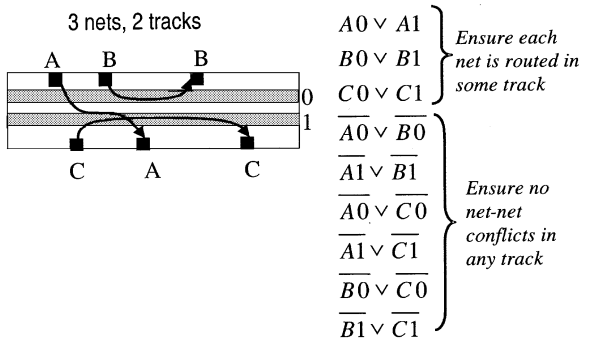

Fig. 3. Small unsatisfiable routing example and its associated strict CNF representation.

some careful accounting shows that our augmentation adds the following to the CNF form in the general case.

- **Variables**: We add $N \cdot (s + 2)$ new variables. For each of the N clauses in the formula, we create one stage of our within-k counter with an s-bit sum, 1 mask bit, and 1 c_i bit.
- **Clauses**: We add $(s+2)+(N-1)(s+3+s(s+3))+(s+1)$ new clauses. The first term is for the first within-k block, the second term accounts for the remaining $N-1$ counter blocks, and the final term is the final comparator at the right of Fig. 2.

Hence, we add $O(N \log k)$ variables and $O(N \log^2 k)$ clauses in our augmentation of the original M-variable, N-clause CNF formula. Since our principal interest is in "almost" satisfiable problems, we expect $k \ll N$. Nevertheless, the relaxation threshold k still determines the size of the resulting augmented SAT formula. In addition, we usually have more constraints from the application. For example, which clauses could be ignored, which clauses must be lumped together as a group if we choose to ignore them, etc. Then, the number of mask bits and extra clauses will be much smaller. Therefore, in practice, the above complexity bound is a worst case, pessimistic one.

The main advantage of a transformation of this type is that the relaxation is just another—bigger—SAT problem. Given progress on Boolean SAT (CNF SAT), this is a significant advantage: we can use the same SAT engine. Alternative formulations such as maximum SAT (MAX-SAT [24], and partial variants [25]), assign weights to the constraints and seek the maximum-weight feasible subset [26]. Unfortunately, progress in this area has yet to yield solvers with the same maturity, generality, and capacity as Boolean SAT.

B. A Simple Example

A small example helps clarify the details of our augmented CNF, and illustrates some subtle details. Fig. 3 shows a trivial routing problem that cannot be completed. Three nets contend for two tracks, and only two can embed. Variable $A0$ is set to assign net A to track 0; $\overline{A0}$ means

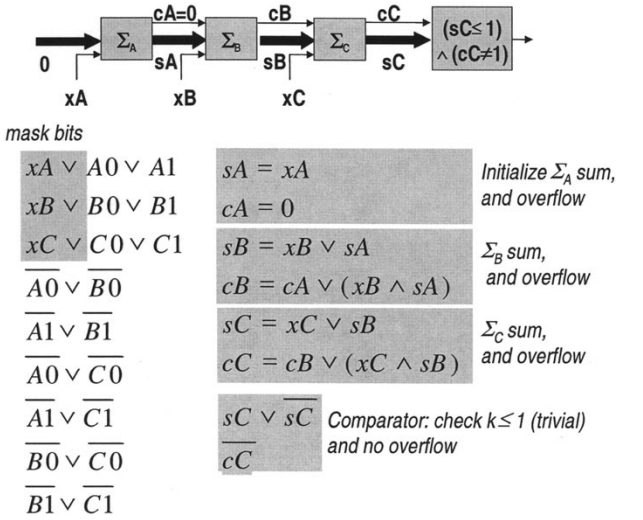

Fig. 4. Augmented CNF when above routing problem is relaxed to allow $\mathbf{k = 1}$ unrouted nets.

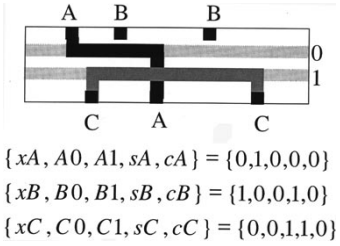

Fig. 5. SAT solution for augmented CNF yields a partial routing with exactly $\mathbf{k = 1}$ unrouted nets.

this net is not in track 0, and so on. Let us set a relaxation threshold $k = 1$, and see if we can solve this problem with only one unrouted net. Fig. 4 shows the augmented form of the CNF formula that results from this relaxation, with new variables and clauses highlighted, and a simple representation of the within-k counter also shown. One critical point to note is that, although in the worst case a relaxation might add $O(N \log_2 k)$ variables, the number we really need is application dependent. It is not the case that all clauses are equivalent, in the sense that any of them can be relaxed. Complex problems generate a large, nonhomogeneous set of clauses; we may be able to relax some but not others, to obtain a usable partial solution. This is shown above, where it is clear that we only need to mask the three clauses that ensure that each net is assigned to some track. (We can actually mask all of these clauses and obtain the same result, but it is not necessary to do so.) Fig. 5 shows the results of satisfying this relaxed problem: we find a solution with exactly one unrouted net.

IV. sub-SAT: Experimental Results

We describe two sets of experiments. The first focuses on SAT-based FPGA routing, which was the original motivation for this paper. The second looks at how our sub-SAT formulation applies to a general set of SAT benchmark tasks. All of our results are based on the Chaff SAT engine from Princeton [13]. Experiments were run on a 1-GHz Pentium III running LINUX.

A. FPGA Routing Results

Table I shows results from a set of standard FPGA layout benchmarks from [27]. The CNF forms for the benchmarks represent two different SAT routing formulations.

- **2-Pin**: This is the original SAT formulation of [19]. Following global routing, each multipin net is decomposed into a set of two-pin nets for detailed routing. Each unique mapping of a two-pin net onto an atomic FPGA routing resource (e.g., a wire segment) is represented by a unique vector of Boolean variables. Routing is formulated as a SAT problem that ensures that: 1) each net is connected by some legal path through FPGA resources and 2) no routing resource is used by more than one net. These constraints are called connectivity and exclusivity, respectively. Multipin nets are embedded by relaxing the exclusivity constraint between pairs of two-pin nets decomposed from the same multipin source net. All nets are routed concurrently.
- **RCS**: This is the improved SAT formulation of [20]. Following global routing, each multipin net is decomposed into a Steiner tree of two-pin connections. Then, each two-pin connection is routed in several different ways, ignoring all other nets. This generates a portfolio of admissible path solutions for each connection. Each such path alternative is mapped to a unique vector of Boolean variables. Routing is formulated as an SAT problem that ensures that: 1) each two-pin connection chooses one of its admissible path alternatives and 2) no pairs of electrically distinct nets choose paths that overlap. These constraints are called liveness and exclusivity, respectively.

The two-pin formulation yields a more fine-grain routing, in that any net can choose any path through FPGA resources. However, the corresponding CNF form is not only somewhat larger, but its associated exclusivity constraints are much harder to satisfy, which limits its scalability. The RCS formulation is geometrically less flexible, but scales to larger routing problems; it has many fewer hard exclusivity constraints. Together these are a good set of benchmarks for testing a sub-SAT solution.

The first column of Table I lists the benchmarks. The naming convention encodes the formulation style ("gr" for global routing before detailed routing; "rcs" or "2pin" for formulation), and the number of tracks per channel in the FPGA (e.g., "w7" means seven tracks). Track count per channel is relevant here since the way we make each of these benchmarks unroutable is to reduce the available wiring resources until the SAT detailed router fails. Columns 3 and 4 show the size of strict form of the problem, where we tolerate no unrouted nets. In this form, an unroutable solution returns no partial routing information at all. Columns 5 and 6 show the size of the relaxed form of the problem, and column 7 shows the relaxation threshold k, the number of unrouted nets we are willing to tolerate. We run our experiment from a minimum k. If we find a partial solution, we stop here. If we prove there is no partial solution with threshold k, we resolve with $\mathrm{k} + 1$. We also set a time out limit for each solution. If we run out of time for a partial solution with threshold k, we terminate it and try to find a partial solution with threshold $\mathrm{k} + 1$. Then, for each benchmark, column 7 gives the smallest number of unrouted nets that produces a relaxed SAT problem that we can solve in not more than 2000 s. Better solutions, with fewer unrouted nets may be possible here, but we terminated search before either finding them, or proving that there is no such less-relaxed solution. We label this in the table: for rows tagged with a "*" in column 1, better solutions may be possible if we run longer and for rows without the "*" we have proven there is no less-relaxed solution. Finally, columns 8 and 9 show CPU times for the strict and relaxed forms of the problem.

As an example of the dynamics of this process, Fig. 6 shows, for one benchmark from Table I, the runtime fluctuation as we change the sub-SAT threshold k. We increase k from 0 (the strict form). For the first three points, we find no partial answer. Then, at the point where $\mathrm{k} = 3$, we find our best partial solution. The running time increases as we approach this point. Normally, we would stop here, as per the procedure for the experiment in Table I. However, for the purpose of illustration,

TABLE I
sub-SAT RESULTS FROM TWO FPGA SAT-BASED ROUTING FORMULATIONS

FPGA Benchmark	Nets	Strict CNF Formulation (k=0)		Relaxed CNF Formulation		Relaxation Threshold: k=nets unrouted	SAT Time: (Sec)	
		Variables	Clauses	Variables	Clauses		Strict (k=0)	Relaxed
9symml_gr_2pin_w5.cnf	79	2604	32450	2841	33079	1	0.36	0.85
alu2_gr_2pin_w7.cnf	153	3882	84209	4341	85430	1	9.32	203.56
apex7_gr_2pin_w4.cnf	126	1322	10940	1826	12822	2	0.08	0.27
C499_gr_2pin_w5.cnf	115	2070	19908	2530	21625	3	0.74	80.3
C880_gr_2pin_w6.cnf*	234	4623	62711	5559	66213	3	50.26	1036.66
example2_gr_2pin_w5.cnf	205	3603	36334	4423	39411	2	1.33	17.93
term1_gr_2pin_w3.cnf	88	746	3517	1186	5614	7	<0.01	1.71
too_large_gr_2pin_w6.cnf*	186	3972	52678	4716	55460	3	0.59	833.98
9symml_gr_rcs_w5.cnf	79	1295	24309	1532	24938	1	0.08	0.03
alu2_gr_rcs_w7.cnf	153	3570	73478	4029	74699	1	15.16	0.07
apex7_gr_rcs_w4.cnf	126	1200	9416	1704	11298	2	0.02	0.11
C499_gr_rcs_w5.cnf	115	1560	15777	2020	17494	3	0.09	165.9
C880_gr_rcs_w6.cnf*	234	3936	53018	5106	58619	4	230.51	358.39
example2_gr_rcs_w5.cnf	205	2220	23144	3040	23211	2	0.8	2.56
term1_gr_rcs_w3.cnf	88	606	2518	1046	4615	7	<0.01	0.44
too_large_gr_rcs_w6.cnf*	186	3114	43251	3858	46033	3	1.52	203.61
vda_gr_rcs_w7.cnf*	225	5054	102047	6404	109898	13	118.84	1190.07
k2fix_gr_rcs_w9.cnf*	404	11313	305160	13737	319276	11	>2000	1828.17

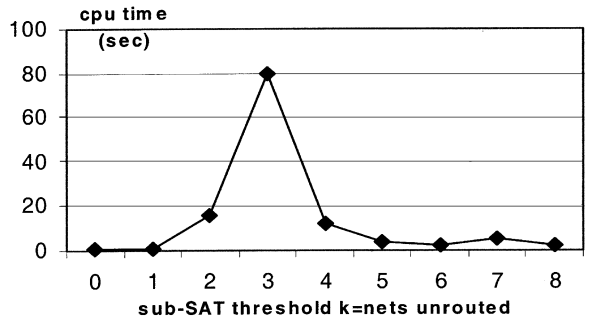

Fig. 6. Time fluctuation of C499_gr_2pin_w5.cnf.

we will continue to increase k. As we do so, we find the time to obtain a partial answer decreases substantially. This means that it is easier to find a partial solution with less quality (more clauses ignored). In this example, the best partial solution point $(\mathrm{k} = 3)$ requires the most effort. This is empirically true for many, but not all, of our benchmarks.

Although the sub-SAT transformation dramatically change the structure of the original strict form of the problem, our relaxed problems show the same wide variance in runtime as the strict forms. Sometimes the relaxation solves quickly: this means it is easy to find and omit a few nets and embed the rest. Sometimes the relaxation is much more time-consuming: the base problem was so significantly over-constrained that it was easy to prove the lack of any solution. But, the problem of routing with all but a handful of nets is actually a hard problem, with few viable solutions; it just takes longer to find such a solution. Unfortunately, we cannot conclude that sub-SAT relaxation of a problem always renders it easier or quicken to solve. However, in contrast with the strict form, satisfiable relaxations yield practical, partial solutions.

As with other experiences attacking SAT problems, the CNF problem size (variables, clauses) is a weak predictor of the overall difficulty in satisfying the problem. In our sub-SAT formulation, we increase the size in proportion to the total number of nets (which is much less than the number of variables or clauses, as shown in Fig. 4), and our total tolerance for unrouted nets (the threshold k). The augmented CNF forms are larger, but they are not always correspondingly harder to solve. This is consistent with the qualitative discussion of difficulty in the previous paragraph. We think this bodes well for the ultimate practicality of our approach.

Finally, it is worth noting that in our work to date, we have not directly addressed the new "auxiliary" problem created by the sub-SAT transformation: how to determine the optimal of k, the relaxation threshold. At least for the routing problem, users have empirical guidelines for the maximum amount of infeasibility that they can tolerate; this is $\mathrm{k}_{\max}$. If $\mathrm{k}_{\max}$ is very small, we might just search sequentially from 1 to $\mathrm{k}_{\max}$. If not, binary search is the obvious solution, perhaps also with some sensible discretization on the resolution of k so as to minimize the overall time to find the best k. In any case, this is a good subject for further work on sub-SAT methods in general.

B. sub-SAT Results for Other Standard Benchmarks

There is a large and vigorous SAT community today, pursuing a variety of solution strategies and applications. The DIMACS benchmark suite at Rutgers University (new Brunswick, NJ) is the central repository for interesting/difficult SAT benchmarks [28]. For completeness, we show just a few examples of applying our sub-SAT formulation to some of these benchmarks.

To relax these test cases, we have assumed that all clauses are equal targets for being masked. This is likely too optimistic, but lacking additional domain-specific information, this is the best assumption we can make. Note that for our routing benchmarks, knowledge of the clause structure of the CNF allows us to mask only those constraints that we know may be omitted, and still yield a usable partial solution. Our interest in attempting these test cases is to offer some insight into how "near" these various problems are to being satisfiable—what is the smallest perturbation (number of clauses opted out) that renders them satisfiable?

Table II shows a set of DIMACS benchmarks that are each unsatisfiable in their original form. Benchmarks labeled "$*4$" are actually a family of four test cases; we show the size or size range of strict forms in columns 2 and 3. We show the longest time to solve any of these in the last column. However, relaxing just a single constraint (i.e., allowing the sub-SAT formulation to opt-out one arbitrary clause in the CNF form) renders all of these quickly satisfiable. The fourth column shows

TABLE II
sub-SAT RESULTS FOR SMALL UNSATISFIABLE DIMACS BENCHMARKS. MOST SOLVE EASILY BY RELAXING JUST ONE CONSTRAINT

DIMACS Benchmark	Strict CNF Form		Relaxation Threshold k	sub-SAT Result	CPU Time (Sec)
	Variables	Clauses			
aim-100-1_6 * 4	100	160	1	1	<0.01
aim-100-2_0 * 4	100	200	1	1	<0.01
aim-200-1_6 * 4	200	320	1	1	0.01
aim-200-2_0 * 4	200	400	1	1	<0.01
aim-50-1_6 * 4	50	80	1	1	<0.01
aim-50-2_0 * 4	50	100	1	1	<0.01
bf * 4	1040-2180	3668-6778	1	1	4.46
dubois * 12	60-150	160-400	1	1	0.01
hole * 5	42-110	133-561	1	1	0.09
pret60 * 4	60	160	1	1	0.01
pret150 * 4	150	400	1	1	0.01
ssa0432-003	435	1027	1	1	0.44
ssa2670 * 2	986-1359	2315-3321	1	1	1.77
ssa6288-047	10410	34238	1	1	55.7

TABLE III
sub-SAT RESULTS FOR A LARGER DIMACS BENCHMARK SERIES. THESE ARE AGAIN ALL UNSATISFIABLE IN THEIR ORIGINAL FORM, BUT RELAX EASILY BY MASKING ONLY A FEW CLAUSES

DIMACS Bench-mark	Strict (Original) CNF Form		Relaxed (Augmented) CNF Form		Relax-ation Thre-shold k	sub-SAT Result	Avg CPU Time (Sec)
	Vars	Clauses	Vars	Clauses			
jnh2, jnh4 jnh5, jnh6 jnh10, jnh11 jnh16, jnh18	100	850	2650	7647	1	1	4.11
jnh3, jnh8 jnh9, jnh13 jnh14, jnh15 jnh19, jnh20	100	850	2650	7647	1	unsat	18.17
			3500	13592	2	2	15.43
jnh202, jnh203 jnh206, jnh208 jnh214, jnh215 jnh216, jnh219	100	800	2500	7197	1	1	7.36
jnh211	100	800	2500	7197	1	unsat	8.53
			3000	12792	2	2	12.69
jnh302	100	900	2800	8097	1	unsat	4.5
			3700	14392	2	unsat	39.16
			3700	14392	3	unsat	990.
			4600	22485	4	4	55.41
jnh303, jnh304 jnh305, jnh307 jnh310	100	900	2800	8097	1	unsat	11.32
			3700	14392	2	unsat	158.1
			3700	14392	3	3	148.3
jnh306	100	900	2800	8097	1	1	21.68
jnh308, jnh309	100	900	2800	8097	1	unsat	11.55
			3700	14392	2	2	9.08

the input specification for how much to try to relax the problem—just one constraint in all these cases.

Table III shows another set of DIMACS benchmarks that are each synthetically constructed to be unsatisfiable. The last column is the average CPU time of this set of benchmarks. Unlike many SAT problems, these are all fairly easy to prove unsatisfiability for. What is interesting to us is that they are also fairly easy to relax, and small relaxations render most of them solvable. Benchmarks with multiple result rows illustrate how we discover the smallest clause-masking perturbation that renders the problem satisfiable. In column 7 we show the sub-SAT result for a specific threshold k: the number is how many clauses were

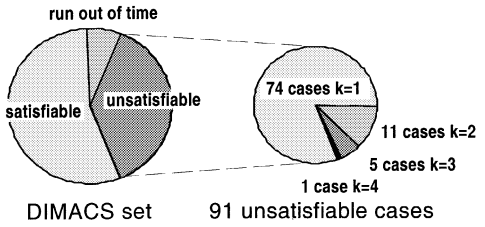

Fig. 7. Unsatisfiable DIMACS benchmarks solved by sub-SAT with threshold k.

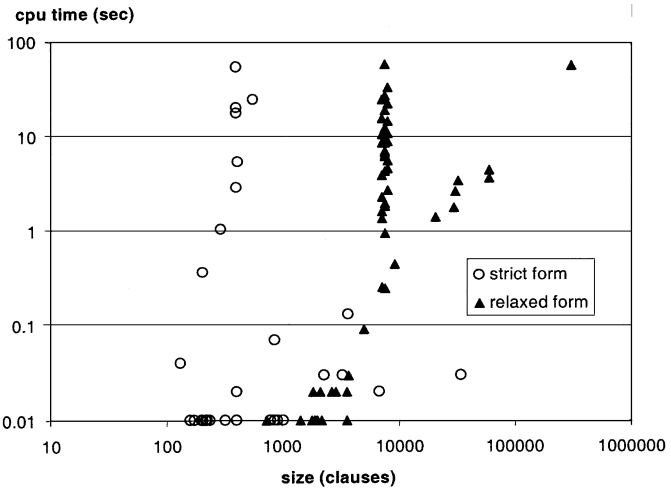

Fig. 8. Time–size scatter plot.

ignored in the partial solution, "unsat" means no partial solution. We increase the threshold k until we find a satisfiable relaxation. For example, $jnh302$ from Table III is unsolvable in strict form $(\mathrm{k} = 0)$, and by relaxing only 1, 2, or 3 clauses. But it solves it in roughly 50 s, if we can tolerate four clauses being masked.

Among all 91 of the unsatisfiable DIMACS cases we tested, 74 can be satisfied with only one constraint opting out. 11 cases have partial solutions with two constraints opting out. And all of the 91 cases are satisfiable with not more than four constraints opting out. This overall result is illustrated in Fig. 7.

Fig. 8 shows the time–size scatter plot of the strict forms versus relaxed forms with $\mathrm{k} = 1$ for the 91 unsatisfiable DIMACS benchmarks. We use the number of clauses to measure size. Both axes are in log scale. Circles represent the strict forms, and triangles represent the relaxed forms. We can see that the relaxed form problems are larger than the strict forms since they lie on the right side. However, the relaxed problems are not always harder than the strict forms. A number worth computing is the ratio of the total running times for each of these benchmark series [29]

$$R = \frac{\sum_{\text{benchmarks}} \text{cpu-time(relaxed form)}}{\sum_{\text{benchmarks}} \text{cpu-time(strict form)}}. \tag{4}$$

We note that the total cpu time requirement for all of the $\mathrm{k} = 1$ relaxed forms is about 3.4 times that of the strict forms. However, if we repeat this exercise and instead compute the ratio for the total size (in clauses) of the relaxed and strict forms, we find that the relaxed forms are roughly nine times larger than the strict forms. As is always the case with SAT problems, the time it takes to solve any individual problem is weakly correlated with its size. However, in aggregate, our relaxations are always larger, but not always correspondingly slower.

V. Conclusion

Recent progress on solvers for Boolean SAT motivates interest in casting geometric problems as Boolean problems. Several efforts have appeared to date, but a legitimate criticism of these approaches is the lack of any partial solution whenever an intrinsically unsatisfiable problem is encountered. We value the SAT solution style in large part because it allows us to express and solve a very large number of concurrent constraints. Returning to our opening example, when our goal is to route 1000 nets, it is frustrating if 999 are routable, but a SAT formulation returns "no" as its entire answer. In this paper, we showed how to transform the CNF description of an arbitrary SAT problem into a new, relaxed SAT problem that is satisfiable just if some number k of the clauses in the formulation are ignored. Although the transformed problem is relaxed (it admits more solutions), the CNF form of the problem is larger; our construction augments the CNF with new variables and clauses representing those parts of the problem that we may allow to "opt out" of the solution. The technique is general, but aimed at cases where the relaxation threshold k is much smaller than the total number of constraint clauses in the problem.

For SAT-based routing, this lets us pose, for the first time, and answer questions of the form "...can we route this layout with not more than k nets unconnected?" Our transformation strategy has the virtue that it creates a new problem that can be solved with the same SAT engines used for the original problem. Ongoing improvements in basic SAT engines will also improve our ability to solve sub-SAT problems.

Acknowledgment

The authors would like to thank G.-J. Nam of the Austin Research Labs, IBM Corporation, Austin, TX, for access to FPGA routing benchmarks, and S. Malik of Princeton University, Princeton, NJ, for access to the Chaff SAT engine.

References

[1] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, 1992.

[2] K. S. Brace, R. R. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. ACM/IEEE Design Automation Conf.*, Jan. 1990, pp. 40–45.

[3] R. Rudell, "Dynamic variable ordering for binary decision diagrams," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 42–47.

[4] S.-I. Minato, "Zero-suppressed BDD's for set manipulation in combinatorial problems," in *Proc. 30th ACM/IEEE Design Automation Conf.*, June 1993, pp. 272–277.

[5] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 188–191.

[6] A. Anuchitanukul, Z. Manna, and T. E. Uribe, "Differential BDDs," in *Computer Science Today, Lecture Notes in Computer Science*, J. van Leeuwen, Ed. Berlin, Germany: Springer-Verlag, Sept. 1995, vol. 1000, pp. 218–233.

[7] R. E. Bryant, "Binary decision diagrams and beyond: enabling technologies for formal verification," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, Nov. 1995, pp. 236–243.

[8] B. Yang, R. E. Bryant, D. R. O'Hallaron, A. Biere, O. Coudert, G. Janssen, R. K. Ranjan, and F. Somenzi, "A performance study of BDD-based model checking," in *Proc. 2nd Int. Conf. Formal Methods Computer-Aided Design*, Palo Alto, CA, Nov. 1998, pp. 255–289.

[9] B. Selman, H. Kautz, and B. Cohen, "Local search strategies for satisfiability testing," in *Proc. DIMACS Series Discrete Mathetmatics and Theoretical Computer Science*, vol. 26, 1996, pp. 521–532.

[10] H. Zhang, "SATO: an efficient propositional prover," in *International Conference on Automated Deduction (CADE'97), LNAI 1249*. Berlin, Germany: Springer-Verlag, 1997, pp. 272–275.

[11] J. P. Marques-Silva and K. A. Sakallah, "GRASP: a search algorithm for propositional satisfiability," *IEEE Trans. Comput.*, vol. 48, pp. 506–521, May 1999.

[12] J. P. Marques-Silva and L. G. e Silva, "Algorithms for satisfiability in combinational circuits based on backtrack search and recursive learning," in *Proc. 12th Symp. Integrated Circuits Syst. Design*, Sept.–Oct. 1999, pp. 192–195.

[13] Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient SAT solver," in *Proc ACM/IEEE 39th Design Automation Conf.*, Las Vegas, June 2001, pp. 530–535.

[14] M. Velev and R. E. Bryant, "Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors," in *Proc. ACM/IEEE Design Automation Conf.*, June 2001, pp. 226–231.

[15] S. Devadas, "Optimal layout via Boolean satisfiability," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, Nov. 1989, pp. 294–297.

[16] R. G. Wood and R. A. Rutenbar, "FPGA routing and routability estimation via Boolean satisfiability," *IEEE Trans. VLSI Syst.*, vol. 6, pp. 222–231, June 1998.

[17] F. Schmiedle, D. Unruh, and B. Becker, "Exact switchbox routing with search space reduction," in *Proc. ACM Int. Symp. Physical Design*, Apr. 2000, pp. 26–32.

[18] K. Sulimma and W. Kunz, "An exact algorithm for difficult detailed routing problems," in *Proc. ACM Int. Symp. Physical Design*, Apr. 2001, pp. 198–203.

[19] G.-J. Nam, K. Sakallah, and R. A. Rutenbar, "Satisfiability-based layout revisited: routing complex FPGA's via search-based Boolean SAT," in *Proc. ACM Int. Symp. FPGAs*, Feb. 1999, pp. 167–175.

[20] G.-J. Nam, F. Aloul, K. Sakallah, and R. A. Rutenbar, "A comparative study of two Boolean formulations of FPGA detailed routing constraints," in *Proc. ACM Int. Symp. Physical Design*, Apr. 2001, pp. 222–227.

[21] G.-J. Nam, K. Sakallah, and R. A. Rutenbar, "A Boolean satisfiability-based incremental rerouting approach with application to FPGAs," in *Proc. Design Automation Test Eur.*, Mar. 2001, pp. 560–564.

[22] G.-J. Nam, "A Boolean based layout approach and its application to fpga routing," Ph.D. dissertation, Dept. of Elect. Eng. Comp. Sci., Univ. of Michigan, Ann Arbor, MI, 2001.

[23] H. Xu, R. A. Rutenbar, and K. Sakallah, "sub-SAT: a formulation for relaxed Boolean satisfiability with applications in routing," in *Proc. Int. Symp. Physical Design*, 2002, pp. 182–187.

[24] P. Hansen and B. Jaumard, "Algorithms for the maximum satisfiability problem," *Computing*, vol. 44, pp. 279–303, 1990.

[25] B. Cha, K. Iwama, Y. Kambayashi, and S. Miyazaki, "Local search algorithms for partial MAXSAT," in *Proc. 14th Nat. Conf. Artificial Intell.*, 1997, pp. 263–268.

[26] J. N. Hooker, "Resolution and the integrality of satisfiability problems," *Math. Program.*, vol. 74, pp. 1–10, 1996.

[27] *http://www.eecg.toronto.edu/~lemieux/sega/sega.html* [Online]

[28] *http://dimacs.rutgers.edu/Challenges/* [Online]

[29] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design*. San Mateo, CA: Morgan Kaufmann, 1994, ch. 2.6, pp. 68–70.