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

Hui Xu, Rob A. Rutenbar
Dept. of ECE, Carnegie Mellon University
Pittsburgh, Pennsylvania, 15213 USA
{huix,rutenbar}@ece.cmu.edu

Karem Sakallah
EECS Dept., The University of Michigan
Ann Arbor, Michigan, 48109 USA
karem@eecs.umich.edu

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<<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 FPGA routing show how we can determine efficiently when we can satisfy "almost all" of our geometric constraints.

Categories and Subject Descriptors

B.7.2 [Integrated Circuits]: Design aids—layout

General Terms

Algorithms

1. 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 [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; satisfiability comes as a by-product of the BDD data structure. In those applications where we only need to ascertain satisfiability 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.

ISPD'02, April 7-10, 2002, San Diego, California, USA.

There have been several attempts in this direction. Early attempts used BDDs for representation and manipulation. Devadas showed a formulation for simple 2-layer channel routing using a sequence of BDDs [15]. Wood developed a more general routing formulation based on BDDs, and also demonstrated how 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. 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 represent in its entirety. More recently, Schmiedle *et al.* formulated gridded maze routing in an entirely symbolic form, using BDDs to model maze wavefront expansion is 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 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. 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: Sec. 2 briefly motivates our approach in more detail. Sec. 3 presents details of a SAT-transformation based on explicit thresholding and counting of the relaxed constraints. Sec. 4 then shows experimental results from both FPGA routing problems, and a more general set of common SAT benchmark problems. Finally, Sec. 5 offers some concluding remarks.

2. 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. For example, 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 1 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).

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.

3. sub-SAT via Transformation and Counting

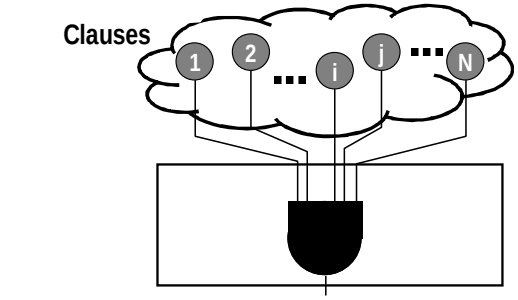
3.1 Basic Formulation

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 5 variables and 4 clauses is:

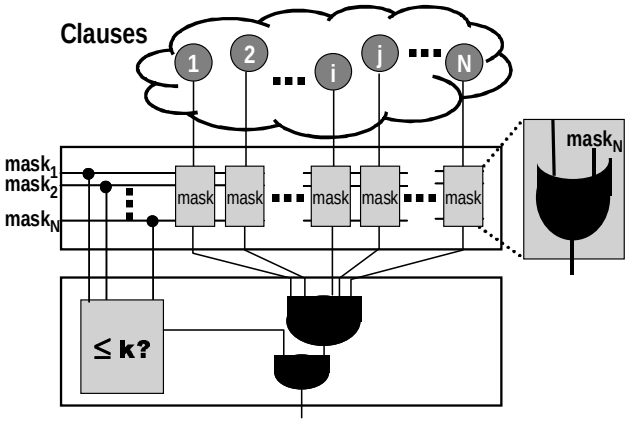
$$(\bar{X}_3 \vee X_4 \vee \bar{X}_5) \wedge (X_1 \vee \bar{X}_2) \wedge (X_1 \vee \bar{X}_3 \vee X_4) \wedge (X_2 \vee \bar{X}_5) \qquad (1)$$

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 con-



(a) Strict SAT formulation



(b) Relaxed (augmented CNF) SAT formulation

FIGURE 1. Formulating subset satisfiability by augmenting the original CNF form with masking and counting

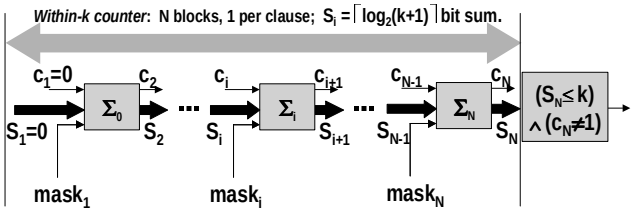


FIGURE 2. Implementing the threshold counter (at bottom-left of Fig. 1b) as a linear *within-k* structure

junction insensitive to whether these masked constraints are met or not.

As shown in Fig. 1b, 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 non-masked 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 $k<<N$ 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:

$$c_i = \begin{cases} 1 & \text{if } S_i = 2^{\lceil log_2(k+1) \rceil} \text{ and } mask_i = 1 \\ c_{i-1} & \text{else} \end{cases}$$

$$S_i = \begin{cases} (S_{i-1}+1) & \text{if } (S_i < 2^{\lceil log_2(k+1) \rceil}) \text{ and } mask_i = 1 \\ S_{i-1} & \text{else} \end{cases} \quad (2)$$

We implement each of these incrementer structures as additional clauses in the CNF formula. 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 $s = \lceil log_2(k+1) \rceil$; 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 $O(Nlogk)$ variables and $O(Nlog^2k)$ clauses in our augmentation of the original *M*-variable, *N*-clause CNF formula. Since our principal interest is in "almost" satisfiable problems, we expect $k<<N$. Nevertheless, the relaxation threshold *k* still determines the size of the resulting augmented SAT formula.

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 satisfiability), this is a significant advantage: we can use the *same* SAT engine. Alternative formulations such as *maximum satisfiability* (MAX-SAT [23], and partial variants [24]), assign weights to the constraints and seek the maximum-weight feasible subset [25]. Unfortunately, progress in this area has yet to yield solvers with the same maturity, generality and capacity as Boolean SAT.

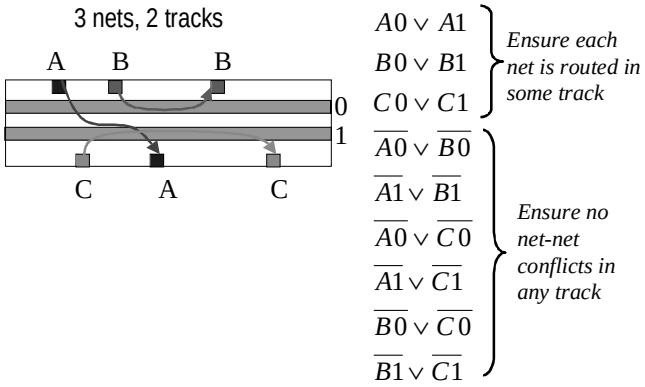


FIGURE 3. A small unsatisfiable routing example and its associated strict CNF representation.

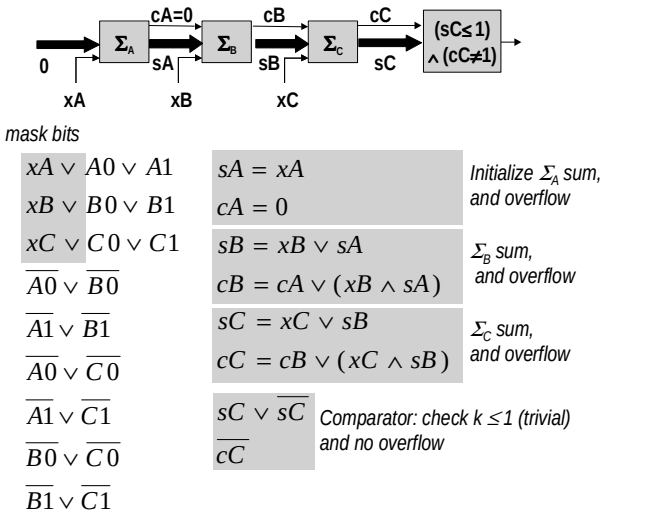
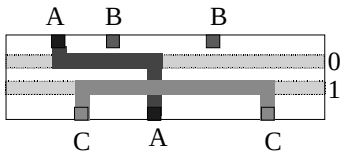


FIGURE 4. Augmented CNF when above routing problem is relaxed to allow k=1 unrouted nets



$\{xA, A0, A1, sA, cA\} = \{0,1,0,0,0\}$
$\{xB, B0, B1, sB, cB\} = \{1,0,0,1,0\}$
$\{xC, C0, C1, sC, cC\} = \{0,0,1,1,0\}$

FIGURE 5. SAT solution for augmented CNF yields a partial routing with exactly k=1 unrouted nets

3.2 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 *A*0 is set to assign net A to track 0; $\overline{A0}$ means 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 1 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 3 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 resulting of satisfying this relaxed problem: we find a solution with exactly one unrouted net.

4. 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 work. 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 750MHz Pentium III running LINUX.

4.1 FPGA Routing Results

Table 1 shows results from a set of standard FPGA layout benchmarks from [26]. 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 multi-pin net is decomposed into a set of 2-pin nets for detailed routing. Each unique mapping of a 2-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 (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. Multi-pin nets are embedded by relaxing the exclusivity constraint between pairs of 2-pin nets decomposed from the same multi-pin source net. All nets are routed concurrently.
- **RCS**: this is the improved SAT formulation of [20]. Following global routing, each multi-pin net is decomposed into a Steiner tree of 2-pin connections. Then each 2-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 a SAT problem that ensures (1) each 2-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 2-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 1 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 7 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 3,4 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 5,6 show the size of the relaxed form of the problem, and column 7 shows the relaxation

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	2762	40262	1	0.55	1.3.
9symml_gr_rcs_w5.cnf	79	1295	24309	1453	28194	1	0.13	0.04
alu2_gr_2pin_w7.cnf	153	3882	84209	4188	95855	1	14.6	341.
alu2_gr_rcs_w7.cnf	153	3570	73478	3876	84188	1	24.3	0.1
apex7_gr_2pin_w4.cnf	126	1322	10940	1574	14906	2	0.1	0.4
apex7_gr_rcs_w4.cnf	126	1200	9416	1452	13016	2	0.03	0.18
C499_gr_2pin_w5.cnf	115	2070	19908	2300	26118	3	1.13	137.
C499_gr_rcs_w5.cnf	115	1560	15777	1790	20457	3	0.13	276.
C880_gr_2pin_w6.cnf*	234	4623	62711	5091	76580	5	81.4	>2000
C880_gr_rcs_w6.cnf*	234	3936	53018	4404	64826	4	368.	575
example2_gr_2pin_w5.cnf	205	3603	36334	4013	47153	2	2.06	17.9
example2_gr_rcs_w5.cnf	205	2220	23144	2630	29804	2	1.26	2.56
term1_gr_2pin_w3.cnf	88	746	3517	922	5755	7	0.01	1.71
term1_gr_rcs_w3.cnf	88	606	2518	782	4336	7	<0.01	0.44
too_large_gr_2pin_w6.cnf*	186	3972	52678	4344	64594	3	1.79	834
too_large_gr_rcs_w6.cnf*	186	3114	43251	3486	52593	3	4.36	204
vda_gr_rcs_w7.cnf*	225	5054	102047	5504	117209	15	169.	1147
k2fix_gr_rcs_w9.cnf*	404	13176	345426	13984	384954	20	1137.	1131

TABLE 1. sub-SAT results from two FPGA SAT-based routing formulations

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
bf0432-007	1040	3668	1	1	0.19
bf1355-075	2180	6778	1	1	0.29
bf1355-638	2177	6768	1	1	0.58
bf2670-001	1393	3434	1	1	0.13
duboise50	150	400	1	1	0.02

TABLE 2. sub-SAT results for small unsatisfiable DIMACS benchmarks; most solve easily by relaxing just 1 constraint

threshold *k*, the number of unrouted nets we are willing to tolerate. For each benchmark, this column gives the smallest number of unrouted nets that produces a relaxed SAT problem that we can solve in not more than 2000 seconds. 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; for rows without the "*", we have proved there is no less-relaxed solution. Finally, columns 8,9 show CPU times for the strict and relaxed forms of the problem.

Sometimes strict unsatisfiability is easy to prove, and sometimes not. In either extreme, an unsatisfiable problem generates no partial routing solution. Our relaxed problems show the same wide variance in runtimes as the strict forms. Sometimes the relaxation solves quickly: this means its 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. All these relaxations yield useful, almost-routed 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.

4.2 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 is the central repository for interesting/ difficult SAT benchmarks [27]. 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. These 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 2 shows a set of small DIMACS benchmarks that are each unsatisfiable in their original form. Benchmarks labeled "*4" are actually a family of 4 same-sized test cases; we show the longest time to solve any of these. 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 the input specification for how much to try to relax the problem—just one constraint in all these cases.

Table 3 shows a larger set of DIMACS benchmarks that are each synthetically constructed to be unsatisfiable. 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. We increase the threshold *k* until we find a satisfiable relaxation (or if we simply run over our 2000 second time limit, as in *jnh302*). For example, *jnh304* from Table 3 is unsolvable in strict form (*k*=0), and also by relaxing only 1 or 2 clauses. But it solves in roughly one minute if we can tolerate 3 clauses being masked.

5. Conclusions

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 for the first time pose 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.

DIMACS Benchmark	Strict (Original) CNF Form		Relaxed (Augmented) CNF Form		Relaxation Threshold k	sub-SAT Result	CPU Time (Sec)
	Vars	Clauses	Vars	Clauses			
jnh2	100	850	2650	7647	1	1	1.46
jnh3	100	850	2650	7647	1	unsat	92.9
			3500	13592	2	2	10.8
jnh4	100	850	2650	7647	1	1	9.56
jnh5	100	850	2650	7647	1	1	15.2
jnh6	100	850	2650	7647	1	1	9.05
jnh8	100	850	2650	7647	1	1	9.36
			3500	13592	2	2	43.9
jnh9	100	850	2650	7647	1	1	26.1
			3500	13592	2	2	17.6
jnh10	100	850	2650	7647	1	1	0.31
jnh11	100	850	2650	7647	1	1	2.41
jnh13	100	850	2650	7647	1	1	8.8
			3500	13592	2	2	9.71
jnh14	100	850	2650	7647	1	1	11.5
			3500	13592	2	2	10.1
jnh15	100	850	2650	7647	1	1	16.5
			3500	13592	2	2	19.6
jnh16	100	850	2650	7647	1	1	2.52
jnh18	100	850	2650	7647	1	1	5.98
jnh19	100	850	2650	7647	1	unsat	42.0
			3500	13592	2	2	18.0
jnh20	100	850	2650	7647	1	unsat	14.0
			3500	13592	2	2	50.8
jnh202	100	800	2500	7197	1	1	2.99
jnh203	100	800	2500	7197	1	1	33.3
jnh206	100	800	2500	7197	1	1	1.76
jnh208	100	800	2500	7197	1	1	14.3
jnh211	100	800	2500	7197	1	unsat	11.8
			3000	12792	2	2	11.9
jnh214	100	800	2500	7197	1	1	0.32
jnh215	100	800	2500	7197	1	1	5.82
jnh216	100	800	2500	7197	1	1	2.17
jnh219	100	800	2500	7197	1	1	21.2
jnh302	100	900	2800	8097	1	unsat	6.24
			3700	14392	2	unsat	54.6
			3700	14392	3	-	>2000
			4600	22485	4	4	75.8
jnh303	100	900	4600	22485	4	4	75.8
jnh304	100	900	3700	14392	2	unsat	65.1
			3700	14392	3	3	35.8
jnh305	100	900	2800	8097	1	unsat	17.6
			3700	14392	2	unsat	234.
			3700	14392	3	3	369.

TABLE 3. 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.

Acknowledgment

We thank Gi-Joon Nam of IBM for access to FPGA routing benchmarks, and Sharad Malik of Princeton for access to the Chaff SAT engine. This work was supported in part by the National Science Foundation under contract CCR-9971142.

References

[1] R.E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," *ACM Computing Surveys*, Vol. 24, No. 3, September 1992, pp. 293-318.

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

[3] R. Rudell, "Dynamic Variable Ordering for Binary Decision Diagrams," *Proc. ACM/IEEE Intl. Conf. on Computer-Aided Design*, Nov. 1993, pp. 42-47.

[4] S.-I. Minato. "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems." *Proc. 30th ACM/IEEE Design Automation Conference*, June 1993, pp. 272-277.

[5] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, F. Somenzi, "Algebraic Decision Diagrams and their Applications," *Proc. ACM/IEEE International Conference on CAD*, November 1993.

[6] A. Anuchitanukul, Z. Manna and T.E. Uribe, "Differential BDDs," In J. van Leeuwen, ed, *Computer Science Today, Lecture Notes in Computer Science*, Vol. 1000, pp. 218-233, Springer-Verlag, Sep. 1995.

[7] R.E. Bryant, "Binary decision diagrams and beyond: Enabling technologies for formal verification," *Proc. ACM/IEEE International Conf. on Computer-Aided Design (ICCAD)*, November 1995.

[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. Second International Conference on Formal Methods in Computer-Aided Design (FMCAD'98)*, Palo Alto, CA, November 1998, pp. 255-289.

[9] B. Selman, H. Kautz, B. Cohen, "Local Search Strategies for Satisfiability Testing," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 26, 1996, pp. 521-532.

[10] H. Zhang, "SATO: An Efficient Propositional Prover," International Conference on Automated Deduction (CADE'97), LNAI 1249, Springer-Verlag, 1997, pp. 272-275.

[11] J.P. Marques-Silva, and K.A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Transactions on Computers*, Vol. 48, No. 5, May 1999, pp. 506-521.

[12] J.P. Marques-Silva, and L.G. e Silva, "Algorithms for Satisfiability in Combinational Circuits Based on Backtrack Search and Recursive Learning," *Proc. 12th Symposium on Integrated Circuits and Systems Design (SBCCI '99)*,September – October 1999, pp. 192-195.

[13] Y. Zhao, L. Zhang, S. Malik, "Chaff: Engineering an Efficient SAT Solver", *Proc ACM/IEEE 39th Design Automation Conference*, Las Vegas, June 2001.

[14] M. Velev and R.E. Bryant, "Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors," *Proc. ACM/IEEE Design Automation Conf.*, June 2001.

[15] S. Devadas, "Optimal Layout Via Boolean Satisfiability," *Proc. ACM/IEEE Int'l Conference on CAD*, Nov. 1989, pp. 294-297.

[16] R. G. Wood and R. A. Rutenbar, "FPGA Routing and Routability Estimation Via Boolean Satisfiability," *IEEE Transactions on VLSI Systems*, June 1998, pp. 222 - 231.

[17] F.Schmiedle, D. Unruh and B. Becker: "Exact Switchbox Routing with Search Space Reduction," Proc. *ACM International Symposium on Physical Design*, April 2000, pp. 26-32.

[18] K. Sulimma and W. Kunz, "An Exact Algorithm for Difficult Detailed Routing Problems," *Proc. ACM Int'l Symposium on Physical Design (ISPD'01)*, April 2001.

[19] G.-J. Nam, K. Sakallah, and R.A. Rutenbar, "Satisfiability-Based Layout Revisited: Routing Complex FPGAs Via Search-Based Boolean SAT," *Proc. ACM Int'l Symposium on FPGAs*, Feb. 1999.

[20] G.-J. Nam, F. Aloul, K. Sakallah, and R.A. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints," *Proc. ACM Int'l Symposium on Physical Design (ISPD'01)*, April 2001.

[21] G.-J. Nam, K. Sakallah, and R.A. Rutenbar, "A Boolean Satisfiability-Based Incremental Rerouting Approach with Application to FPGAs, *Proc. Design Automation & Test Europe* (DATE'01), March 2001.

[22] G.-J. Nam, *A Boolean Based Layout Approach and its Application to FPGA Routing*, Ph.D. Thesis, Dept. of EECS, the University of Michigan, 2001.

[23] P. Hansen and B. Jaumard, "Algorithms for the maximum satisfiability problem," *Computing*, Vol. 44, 1990, pp. 279-303.

[24] B. Cha, K. Iwama, Y. Kambayashi and S. Miyazaki, "Local search algorithms for partial MAXSAT," *Proc. AAAI*, 1997, pp. 263-268.

[25] J. N. Hooker, "Resolution and the integrality of satisfiability problems," *Mathematical Programming*, Vol. 74, 1996, pp. 1-10.

[26] http://www.eecg.toronto.edu/~lemieux/sega/sega.html

[27] http://dimacs.rutgers.edu/Challenges/