Department of Computer Science, Rochester Institute of Technology

Performance Evaluation of $k - SAT$ Solvers
Applied to Graph Arrowing

Christopher Wood
Advisor: Professor Stanisław Radziszowski

April 7, 2013

1 Introduction

In his 1972 seminal paper entitled, "Reducibility Among Combinatorial Problems," Karp introduced a list of 21 NP-complete problems, including Boolean satisfiability, the maximum cut of a graph, and 0-1 integer programming [1]. The complexity of these problems was proven by deriving a polynomial-time reduction from CIRCUIT-SAT = $\{\langle C\rangle : C$ is a satisfiable Boolean combinational circuit $\}$, the first problem shown to be NP-complete by Cook in 1971 [2], starting the rush of complexity theory research.

The problem $3-SAT$, or more formally, $3-CNFSAT$, is a special case of satisfiability. It is a decision problem in which takes as input a 3-CNF Boolean formula and returns YES if the formula is satisfiable, and NO otherwise [3]. A 3-CNF formula, more formally known as a Boolean formula in 3-conjunctive normal form, is expressed as the Boolean AND of arbitrarily many clauses, where each clause is the Boolean OR of three literals, which is a Boolean variable or its negation. Such a Boolean formula is said to be satisfiable if and only if there exists an assignment of truth values to the variables such that substituting them into the literals of the formula will cause it to evaluate to true (or 1). Expressed as a formal language, we have that $3 - SAT = \{\langle\phi\rangle : \phi$ is satisfiable $\}$.

In 2002 Hans van Maaren of and John Franco initiated the public SAT competition in search of optimal performing SAT solvers judged by a variety of criteria and specializations, including their ability to demonstrate satisfiability and exhaustively prove unsatisfiability. In addition, since SAT is a problem that often arises in academia and the

industry, each of the candidate solvers are rigorously tested with massive application-specific, crafted, and random Boolean formulas as input. With three different solver specializations tested against three different types of inputs, and a first, second, and third place awarded to the candidates, a total of 27 possible trophies are awarded each year. In the most recent competition held in 2011, solvers were tested using CPU time and world-clock time as a basis for their results, thus expanding the trophy space to 54 slots. The next competition is slated to take place in 2013.

The SAT problem is particularly interesting when applied to graph arrowing. It can be shown how to reduce the question $G \overset{?}{\overset{\frown}{\rightarrow}} (3,3)^e$ to an equivalent $3-CNF$ formula ϕ_G such that $G \not\rightarrow (3,3)^e \Leftrightarrow \phi_G$. Intuitively, this is a very promising technique for determining if $G \rightarrow (3,3)^e$ for K_4-free graphs G.

The immediate application of this technique is to attack the upper bound of the Folkman number $F_e(3,3;4)$. In particular, to lower this bound, we will need to find a graph G on n vertices where $G \rightarrow (3,3)^e$. It has been conjectured that $G_{127} = G(127,3) = (\mathbb{Z}_{127}, E = \{(x,y)|x-y = \alpha^3 \mod 127\})$ is a prime candidate for witnessing an upper bound of 127 because of its denseness and large number of triangles. To determine whether G_{127} is indeed a witness we will decompose the problem of arrowing into problems on subgraphs H that witness $H \not\rightarrow (3,3)^e$, and then carefully extend H to encompass all of G. For each subgraph H will generate a corresponding 3-CNF formula ϕ_G by mapping the edges in $E(H)$ to variables in $\phi_H \in$ 3-SAT, and for edge adding the following clauses to ϕ_H:

$$(x+y+z) \wedge (\bar{x}+\bar{y}+\bar{z})$$

If H can be extended to encompass all of G and ϕ_G is shown to be unsatisfiable, then $G \rightarrow (3,3)^e$, and so $F_e(3,3;4) = 127$. Otherwise, if ϕ_G is satisfiable, then $G \not\rightarrow (3,3)^e$, disproving the conjecture made in by Radziszowski et al. [4].

Unfortunately, while this approach seems simple at face, the complexity of the formulas ϕ_H with $n \approx 85$ has proven to be very difficult for modern SAT solvers to handle. In this study, we will attempt to determine the structure of these formulas that makes them so difficult to solve, and as a result, discover the underlying cause for the phase transition that occurs at $n \approx 85$. We will also present a performance comparison for the popular SAT solvers entered into the SAT competition, including Minisat [5], zChaff [7, 6], and glucose [8].

2 SAT Solver Algorithms

The algorithms used in state-of-the-art SAT solvers tend to be based on the famous DPLL backtracking algorithm proposed by David, Putnam, Logemann, and Loveland in [9]. The DPLL algorithm works by recursively splitting a CNF formula into smaller formulas by assigning truth values to individual variables and simplifying the resulting formula at each level of iteration. Simplification works by removing all clauses in ϕ that

Algorithm 1 DPLL Algorithm [9]

Require: ϕ
Ensure: $true$ or $false$
1: **if** ϕ is a tautology with the current assignment **then return** $true$
2: **end if**
3: **if** ϕ contains an empty clause or all literals in a clause are false (conflict) **then return** $false$
4: **end if**
5: **for all** unit clauses $c \in \phi$ **do**
6: $\quad \phi \leftarrow UnitPropagate(c, \phi)$
7: **end for**
8: **for all** pure literal $l \in \phi$ **do**
9: $\quad \phi \leftarrow PureLiteralElimination(l, \phi)$
10: **end for**
11: $l \leftarrow ChooseLiteral(\phi)$
12: **return** $DPLL(\phi \wedge l) \vee DPLL(\phi \wedge \neg l)$

become true under the new assignment and all literals which are false. This process continues recursively until the formula is deemed satisfiable or a conflict emerges. Satisfiability occurs when all variables have been assigned truth values and the resulting formula has non-empty clauses remaining, or when all clauses can be evaluted to true. If a conflict arises, the algorithm backtracks to the last assigned truth value, attempts to substitute the negated truth value, and then proceeds as normal. A formula ϕ is deemed unsatisfiabile if this algorithm exhaustively checks all possible truth value assignments without yielding a satisfiable formula.

The DPLL algorithm further enhances the backtracking search through the use of two additional procedures, unit propagation and pure literal elimination. Unit propagation works by assigning an appropriate truth value to all unit clauses (i.e. all clauses with only a single literal) and then propagating this truth value to the rest of the formula. For example, unit propagation of the formula $\phi = [(x_1) \wedge (x_1 \vee x_2 \vee x_3)]$ yields $\phi = [(x_2 \vee x_3)]$ by assigning a value of true to x_1. Pure literal elimination works by finding variables with have uniform polarity (i.e. always positive or negative). For all such literals, there always a exists a truth assignment such that the containing clauses will evaluate to true. Therefore, such clauses are delete from ϕ. For example, given the formula $\phi = [(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_4)]$, performing pure literal elimination yields $\phi = [(x_1 \vee x_2 \vee \neg x_4)]$ because we can assign $x_3 = false$ and eliminate the first two clauses. These two procedures are shown in the pseudocode description shown in Algorithm 1.

2.1 Advanced Heuristics

The Boolean Constraint Propagation (BCP) algorithm in [7] was a significant improvement on the traditional DPLL algorithm which seeks to minimize the number of times each clause is visited using *variable decisions* and *clause implications*. This procedure consists of modifying ϕ after assigning a variable, or deciding its truth value, by implying further assignments for unit clauses when detected. If a conflict emerges during propagation, then the algorithm must backtrack to the most recent literal decision, negate the value, and then continue down the search tree.

Generally speaking, most SAT solvers that employ this algorithm spend the majority of their time in this procedures. Chatt [7] introduced more advanced heuristics to optimize this part of the DPLL algorithm. Variable watching, which is an optimization that seeks to maintain some knowledge about which clauses recently become unit clauses after a variable assignment, helps reduce the number of clauses that are visited during the BCP procedure.

Another optimization is the Variable State Independent Decaying Sum (VSIDS) decision heuristic, which helps guide the solver when selecting a variable to assign at each layer in the search tree [7]. This procedure works by preprocessing ϕ at the beginning of the solve function to create a collection of counters for the number of times each literal appears in the formula. As the solver progresses, variables for decision are selected based on their count, and the solver will periodically reduce each count by a constant value. As a result, frequently occurring variables are selected first as they will propagate through ϕ more.

TODO: perform more literature survey and add more heurstics here!

3 3-SAT Phase Transition

TODO: information about the ratio and why/where it occurs

Phase Transitions in the Regular Random 3-SAT Problem*

4 Experiments

In our experiments we denote the number of vertices removed from the graph G_{127} as m. The CNF formula ϕ_G for G_{127} experiences a peculiar phase transition when $m \approx 41$, making it seemingly intractable for modern SAT solvers to halt with an answer in a realistic amount of time. Table 1 shows how the Minisat solver time increases as m decreases from 46 to 43. The CPU time continues to increase as m decreases.

To understand the nature of the phase transition, we conducted the experiments described in the following sections.

Table 1: Performance of Minisat solver by removing N_R vertices from G_{127}

N_R	CPU Time	Restarts	Conflicts	Decisions	Propagations	Conflict Literals
46	1228.97s	28668	18968226	41661327	1999748077	930184526
45	4362.29s	66558	52952393	106164381	5779728314	2755169058
44	5576.51s	81915	62420966	126120884	6748259547	3254804365
43	21681.5s	245755	199022470	384078647	21687371442	10581492993

4.1 Experiment 1: Subgraph Reduction and Variable Assignment Propagation

Breaking the problem of $G_{127} \to (3,3)^e$ into small graphs H is a natural way to approach this problem. In addition, clause reduction and unbalancing through variable assignment may provider further simplifications to ϕ_H. Therefore, for this experiment, we have the following parameters:

- m - the number of vertices in $V \subset V(G)$ such that $H = G[V]$. The selection of these vertices will be both structured and unstructured. That is, we will experiment with removing structured groups of vertices, such as those contained within independent sets, as well as unstructure groups composed of randomly selected vertices.

- r - the number of variables $x_i \in \phi_H$ assigned truth values that are propogated through the rest of the formula.

- p - the number of levels of recursive propagation that occur after assigning r truth values.

Clause reduction will use the same unit propagation and pure literal elimination techniques presented in the DPLL algorithm as a preprocessing step for ϕ_H before it is run with a solver. The hardness of solving ϕ_H can be estimated by summing together the time for all individual 2^r truth value assignments. In particular, if ϕ_H^i is the resulting formula after assigning the ith variable configuration, and $T(\phi_H^i)$ is the time required to decide ϕ_H^i, then the hardness (estimated decision time) for ϕ_H is $\sum_{i=1}^{2^r} T(\phi_H^i)$.

TODO: run the experiments and drop data here for DIFFERENT solvers

References

[1] R. Karp, Reducibility among combinatorial problems, *Complexity of Computer Computations, (RE Miller and JM Thatcher, eds.)* (1972), 85-103.

[2] S. A. Cook, The Complexity of Theorem-Proving Procedures, *In Proceedings of the third annual ACM symposium on Theory of computing (STOC*

'71). ACM, New York, NY, USA (1971), 151-158. DOI=10.1145/800157.805047. http://doi.acm.org/10.1145/800157.805047

[3] T. H. Gormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, *MIT Press* **44** (1990), 97-138.

[4] S. P. Radziszowski, X. Xu, On the Most Wanted Folkman Graph, *Geocombinatiorics* **16** (4) (2007), 367-381.

[5] N. Sörensson, N. Eèn, Minisat v1.13 - A SAT Solver with Conflict-Clause Minimization, *SAT 2005* (2005) 53.

[6] Y. Mahajan, Z. Fu, S. Malik, Zchaff2004: An Efficient SAT Solver, *Theory and Applications of Satisfiability Testing*. Springer Berlin/Heidelberg (2005).

[7] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, *Proceedings of the 38th annual Design Automation Conference* ACM (2001).

[8] G. Audemard, L. Simon, GLUCOSE: a solver that predicts learnt clauses quality, *SAT Competition* (2009), 7-8.

[9] M. Davis, G. Logemann, and D. W. Loveland, A machine program for theorem-proving, *Communications of the ACM* **5(7)** (1962), 394-397.

[10] Y. Boufkhad, O. Dubois, Y. Interian, B. Selman. Regular Random k-SAT: Properties of Balanced Formulas, *Journal of Automated Reasoning* **35.1-3** (2005), 181-200.

[11] Random k-SAT: Two moments suffice to cross a sharp threshold