Results on Extensions of the Satisfiability Problem

by

Huxley Bennett

B.S., University of Wisconsin - Madison (2010)

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Department of Computer Science

2012

This thesis entitled:
Results on Extensions of the Satisfiability Problem
written by Huxley Bennett
has been approved for the Department of Computer Science

_____
Sriram Sankaranarayanan

_____
Bor-Yuh Evan Chang

_____
Aaron Clauset

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Bennett, Huxley (Master of Science)

Results on Extensions of the Satisfiability Problem

Thesis directed by Assistant Professor of Computer Science Sriram Sankaranarayanan

The satisfiability problem (SAT) and its extensions have become indispensible tools in artificial intelligence, verification, and many other domains. Extensions of the problem such as model counting, quantified Boolean formulae (QBF), and MAX-SAT have similarly seen increased study and applications. This thesis provides a survey on SAT, and then presents novel results related to model counting and random QBF.

Chapter 3 gives a general technique for computing inclusion-exclusion sums more efficiently for the purpose of model counting. The main contribution is a subsumption technique which reduces computational overhead. Treating an inclusion-exclusion sum's computation as tree exploration, subsumption allows us to prune large subtrees. We also give a better worst-case upper bound on the algorithm's running time, improving it from exponential in the number of clauses to the number of variables in a CNF formula.

Chapter 5 describes a new phase transition in random QBF, along with related results on random QBF models. The clause-to-variable ratio phase transition identified in random k-SAT has been the subject of intense study on what makes a SAT instance intractable, and recent work has studied a similar transition in random QBF. Here we show that a satisfiability threshold exists around phase transitions arising from altering the fraction of existentially versus universally quantified variables in a formula. In chapter 6 we revisit work on generating trivially false formulas in several related random QBF models, giving precise bounds for how likely they are to occur.

Acknowledgements

I would like to thank the National Science Foundation (NSF) for partial funding through award CNS-1016994. Many thanks go to the CUPLV students for their help and for stimulating conversations. Thanks also go to the members of my committee, Sriram Sankaranarayanan, Evan Chang, and Aaron Clauset, for teaching excellent courses on computational complexity, compilers, and algorithms respectively, as well as for giving great advice.

Contents

Tables

Table

Figures

Figure

Chapter 1

Introduction

In the past twenty years, the Boolean propositional satisfiability problem (SAT) and its extensions have become invaluable tools in artificial intelligence and formal verification. The problem remains important in theoretical computer science as a canonical intractable problem, but researchers have made large advances in developing algorithms for solving real-world SAT instances.

The handbook of satisfiability [7] gives a thorough survey of the history of the SAT problem, theoretical concerns, practical solving and applications, and extensions of SAT including quantified Boolean formula (QBF), model counting, constraint satisfaction problems (CSPs), and satisfiability modulo theories (SMT). Here we provide context for the new results in this thesis with a survey on SAT, extensions of SAT, and related solving algorithms.

1.1 Computational Complexity

We give several brief definitions regarding computational complexity, following the notation in [4].

Definition 1. *We say that an algorithm requires DTIME($f(n)$) if it takes a deterministic Turing machine $O(f(n))$ steps to complete the algorithm in the worst-case given an input of size n. We define NTIME($f(n)$) and DSPACE($f(n)$) similarly for nondeterministic time and deterministic space respectively.*

Definition 2. *We define the complexity class P to be the set of decision problems which require polynomial time to decide in the worst case.*

$$P = \bigcup_{c=1}^{\infty} DTIME(n^c) \tag{1.1}$$

We define the classes NP and PSPACE similarly with NTIME and DSPACE in place of DTIME respectively.

1.2 The Satisfiability Problem

We start by introducing some terminology, following standard notation used in [7] and elsewhere. A **Boolean variable**, or just variable, x may take the values of 'true' or 'false'. A **literal** ℓ is a variable x or its logical negation $\neg x$. Let $v(\ell)$ denote the variable constituent of a literal. A **clause** C_i is a disjunction of k_i literals.

$$C_i \quad = \quad \bigvee_{j=1}^{k_i} \ell_{i,j} \tag{1.2}$$

Definition 3. *A k-**CNF** (k-conjunctive normal form) SAT formula φ with a set $X = \{x_1, \ldots, x_n\}$ of Boolean variables is a conjunction of m clauses where the length k_j of every clause C_j is fixed as k.*

$$\varphi \quad = \quad \bigwedge_{i=1}^{m} C_i \quad = \quad \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k} \ell_{i,j} \tag{1.3}$$

For reasons which we will discuss later we may alternatively represent a SAT formula as a set of sets, where the outer set is a set of clauses, and the inner sets are sets of literals. We will call this representation the **set representation** of a formula.

Example 1.

$$(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \rightarrow \{\{x_1, x_2\}, \{\neg x_2, x_3\}\} \tag{1.4}$$

A **partial assignment** to the variables in a formula is a mapping $f : X' \subseteq X \longrightarrow \{0, 1\}$. If $X' = X$ then we say that f is a **complete assignment**.

We say that a literal ℓ is **satisfied** by a partial assignment f if $\ell = x$ and $f(x) = 1$, or if $\ell = \neg x$ and $f(x) = 0$. We also extend this definition to clauses and CNF formulas. A clause is satisfied if at least one of its literals is satified, and a CNF formula is satisfied if all of its clauses are satisfied. A literal, clause, or formula may also be **falsified** if its satisfiability condition is broken, meaning that no total assignment extending the partial assignment could satisfy it. If a partial assignment is neither satisfying nor falsifying, we say that a literal (resp. clause, formula) is **unresolved**.

The CNF-SAT (or SAT) decision problem asks whether there exists a complete assignment to the variables of a CNF formula φ that satisfies it. That is, it asks whether φ is satisfiable. Because two opposite literals x and $\neg x$ make a clause trivially satisfiable (i.e. $\models x \vee \neg x$), and two copies of the same literal do not affect satisfiability (i.e. $(\ell \vee \ell) \Leftrightarrow \ell$), we assume without loss of generality that clauses do not contain any repeated variables.

1.2.1 CNF

Any propositional formula with connectives in $\{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$ may be reduced to an equisatisfiable CNF formula in polynomial time via Tseitin transformation [49, 9, 37], meaning that CNF formulas have the same expressivity as general Boolean functions and that such a representation may be obtained efficiently. One can also reduce an arbitrary propositional formula to CNF using DeMorgan's laws without introducing new variables, but may incur an exponential blow-up in formula size.

Additionally, any clause with k literals may be reduced to an equisatisfiable set of $k - 2$ clauses with 3 literals. By introducing auxiliary variables, we may "chain together" clauses. Therefore, 3-CNF-SAT (3-SAT) is often taken as the normal form of choice.

Example 2.

$$x_1 \vee x_3 \vee \neg x_6 \vee x_7 \vee \neg x_9 \rightarrow (x_1 \vee x_3 \vee y_1) \wedge (\neg y_1 \vee x_6 \vee y_2) \wedge (\neg y_2 \vee x_7 \vee \neg x_9)$$

1.2.2 UNSAT cores

In certain cases we are interested in more than just detecting that solution is unsatisfiable. We may be interested in specifically which constraints (clauses) cause the unsatisfiability of a SAT formula. Regarding the set representation of an unsatisfiable formula φ, a subformula $S \subseteq \varphi$ is called an **UNSAT core** if S is unsatisfiable and $\varphi \setminus S$ is satisfiable. An UNSAT core which removing a single clause causes to be satisfiable is called a **min-UNSAT core**.

Example 3. *Given the formula* $\varphi = x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee x_4)$*, the subformula* $x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3)$ *is an UNSAT core, while* $x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ *is a min-UNSAT core of* φ*.*

1.2.3 Blocking Clauses

After obtaining a satisfying assignment A to a SAT formula, we may decide that we would like another satisfying assignment, if one exists. The simplest method for doing this is to construct a new clause $\varphi \wedge C_b(A)$ where $C_b(A)$ is the negation of the previous satisfying assignment converted to a clause via DeMorgan's law. This clause is called a **blocking clause** because it "blocks" the previous satisfying assignment from being a model of the formula.

Example 4. *Given the formula* $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$ *and satisfying assignment* $\{x_1 = 1, x_2 = 0, x_3 = 0\}$ *we may negate the assignment and convert the negation into a clause using DeMorgan's law.*

$$\neg(x_1 \wedge \neg x_2 \wedge \neg x_3) \rightarrow \neg x_1 \vee x_2 \vee x_3 \tag{1.5}$$

This yields a new formula $\varphi' = \varphi \wedge C_b(\{x_1 = 1, x_2 = 0, x_3 = 0\}) = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$ *for which the previous satisfying assignment is no longer satisfying.*

1.2.4 History

Theorem 1 (Cook-Levin)**.** *3-SAT is NP-complete*

The Cook-Levin Theorem [15, 30], proved independently by Stephen Cook and Leonid Levin in the early 1970s, showed that SAT, and its restriction 3-SAT are NP-complete. This work gave rise to the $P \neq NP$ question which is still open today, and the largest outstanding question in computational complexity [14]. Shortly after Cook's proof, Karp [28] published his seminal paper which showed that 21 well-known problems are NP-complete. This showed that NP-completeness is a powerful notion describing a wide variety of important, intractable problems.

Complexity theorists starting with Karp have worked to establish the NP-completeness of problems by reduction from SAT. In the 40 years since Karp gave his 21 original NP-complete problems, thousands of additional problems have been shown to be NP-complete.

However, in the last 20 years, more pragmatic work has been interested in giving reductions in the opposite direction. By definition of NP-completeness, any problem in NP can be reduced to SAT in polynomial time; such a reduction is called a SAT **encoding**. The aim of the SAT solving research program is to develop efficient encodings and fast algorithms for SAT. Because many problems may be encoded in SAT, by concentrating our algorithmic research effort on SAT solving we will be able to solve these problems as well.

1.2.5 Complete Solvers

The basic setup for a tree-search based solver is to partially generate a complete binary tree of depth n, representing all possible assignments to the variables in X. Each non-leaf node is labeled with a variable $x \in X$, and its two outgoing edges are labeled with 'true' and 'false', indicating the two possible assignments to x. Note that different branches of the tree may have different variables at different depths. Additionally, each leaf node is labeled with 'true' or 'false', depending on whether the assignment to its ancestors satisfies φ. Satisfiability can be checked in linear time.

The naive tree-search based SAT solving algorithm simply performs a depth-first search in this tree until it has either found a leaf node labeled 'true' or exhausted all possibilities. One hallmark of tree-search solvers is that they are **complete**, meaning that they are guaranteed to find a solution or a refutation in finite time.

Algorithm 1: DPLL SAT-solving algorithm based on algorithm 3.4 in [19]

Input: A SAT formula φ
Output: A satisfying assignment or `UNSAT`
$(I, \varphi) = \text{Unit-Resolution}(\varphi)$
if $\varphi = \emptyset$ **then** `/* All clauses satisfied */`
  return I
if $\emptyset \in \varphi$ **then** `/* Formula falsified */`
  return `UNSAT`
Choose $x \in \varphi$ `/* Choose an unassigned variable in` φ `*/`
if $I = \textbf{DPLL}(\varphi | x = \textbf{true}) \neq$ `UNSAT` **then** `/* Reduce formula setting` $x = \text{true}$ `*/`
  return I
else if $I = \textbf{DPLL}(\varphi | x = \textbf{false}) \neq$ `UNSAT` **then** `/* Reduce formula setting` $x = \text{false}$ `*/`
  return I
else
  return `UNSAT`

1.2.5.1 DPLL

The vast majority of modern SAT solvers are based on the DPLL algorithm, shown in algorithm 1, which is named after the authors of [20, 21]. The predominant modern variant is CDCL (conflict-driven clause learning), introduced in [47].

The DPLL algorithm [20, 21] introduced two large improvements to the naive algorithm discussed above. The first observation is that one need not always reach a leaf (give a complete assignment) before concluding that a formula is falsified. If a formula is observed to be falsified at depth d (by a partial assignment to d variables) then a subtree with 2^{n-d} leaves may be pruned from the naive search, and the search may **backtrack** early.

The second improvement in DPLL is the observation that a unit clause dictates how that literal must be assigned. For a literal $\ell = x$ (resp. $\ell = \neg x$), x must be assigned to true (resp. false) or the clause becomes falsified. Making these forced assignments is called **unit propogation** or **Boolean constraint propagation** (BCP). Although invented in the early 1960s, the DPLL strategy still forms the core of the most important SAT solving technique today.

The algorithm given in 1 formalizes this technique. This algorithm is based on those given in [19, 46].

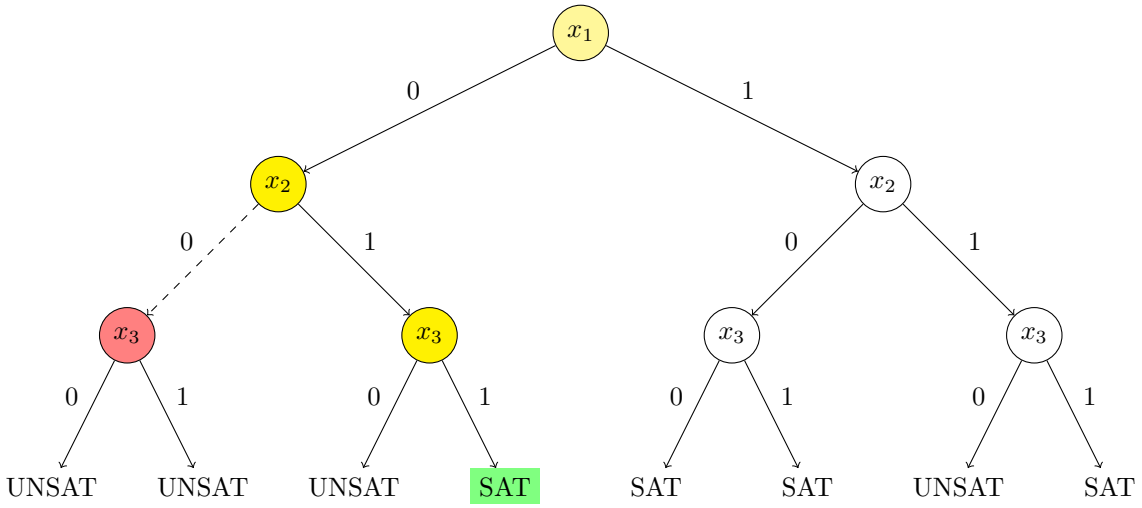


Figure 1.1: Example of DPLL search based on [19] for solving $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$. The light yellow indicates a decision node, while the darker yellow nodes indicate unit propagation, leading to the final solution in green. Because the assignment $x_2 = 1$ is forced by unit propagation, the left x_2 is not a decision node. However, if we "disabled" unit propagation, we would reach x_3, shown in red, at which point we would backtrack early without making an assignment.

1.2.5.2 CDCL

Additional techniques used in modern CDCL solvers [46] include:

- Non-chronological backtracking
- Lazy data structures
- Random restarts
- Variable selection heuristics
- Clause deletion heuristics

1.2.6 Incomplete Solvers

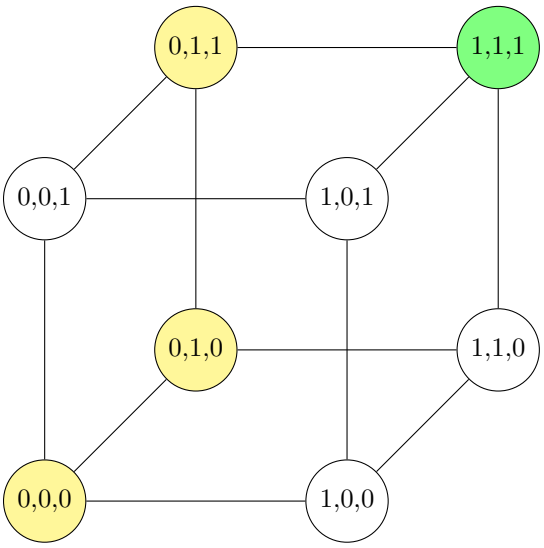
The main alternative paradigm for SAT solving is local search, and particularly one of its variants, stochastic local search (SLS). These algorithms are incomplete [29]. A greedy local search algorithm, GSAT, and its successor which mixes greedy local search with a random walk in the solution space, WalkSAT, were introduced in [44, 43] respectively.

Here we represent the SAT assignment state space as an n-regular undirected graph where the vertices represent complete assignments to the variables in φ, and each vertex has edges to vertices with assignments that are Hamming distance 1 away. For a SAT formula with n variables, this graph may be viewed as taking the set of vertices and edges from an n-dimensional hypercube.

1.2.6.1 WalkSAT

The predominant SLS algorithm is WalkSAT [43]. We start with a seed solution in this graph, and explore the state space by flipping one variable assignment at a time – that is, moving to an adjacent node in the graph. Because we explore nearby solutions, this is called local search. We will consider two possible ways to choose which edge to take:

(1) **Greedy moves** – Determine which variable, if flipped, would result in the highest net number (possibly negative) of clauses being satisfied.



SAT formula:
$(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge$
$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3)$

Iter.	Assignment
0	$(x_1^0, x_2^0, x_3^0) = (0, 0, 0)$
1	$(x_1^1, x_2^1, x_3^1) = (0, 1, 0)$
2	$(x_1^2, x_2^2, x_3^2) = (0, 1, 1)$
3	$(x_1^3, x_2^3, x_3^3) = (1, 1, 1)$

Figure 1.2: Example of a GSAT local search. The unsatisfying seed assignment (0,0,0) and intermediate assignments (0,1,0) and (0,1,1) are given in yellow, while the final (satisfying) assignment (1,1,1) is given in green. The notation x_i^j indicates the assignment of variable x_i at the jth iteration.

Algorithm 2: WalkSAT algorithm based on the algorithm presented in [29].

```
Input: CNF SAT formula φ, number of tries t, steps per try s, random choice parameter p* ∈ [0, 1]
Output: Satisfying assignment to φ or none-found
for i = 1 to t do
    σ := a random truth assignment to the variables in φ                    /* Seed assignment */
    for j = 1 to s do
        if σ is a model of φ then return σ
        C := a random unsatisfied clause of φ
        if ∃x_i ∈ C such that x_i := ¬x_i breaks no clauses then
            v := x
        else
            p := rand(0, 1)          /* Generate a random ''real'' number between 0 and 1 */
            if p < p* then                                                  /* Random move */
                v := a variable chosen at random in C
            else                                                            /* Greedy move */
                v := a variable in C which when flipped satisfies the most clauses
        Flip v in σ
```

(2) **Random moves** – Select a variable uniformly at random and flip it.

We may also discriminate between selecting variables from the whole formula, or just from a particular clause when considering which move to take.

The GSAT algorithm [44], a predecessor to WalkSAT, simply makes greedy moves with respect to the whole formula until a solution is found or the process times out.

The WalkSAT algorithm (algorithm 2) is slightly more complicated. At each decision point, it selects an falsified clause uniformly at random. With this clause it uses the following decision process for selecting a variable to flip:

- If a variable exists which, if flipped, does not falsify any other clauses flip it.
- Otherwise, with probability p select a variable in the clause to flip randomly, and with probability $1 - p$ select it greedily.

Chapter 2

Extensions of SAT

In this section we discuss several important extensions and restrictions of SAT. The new results in this thesis relate to model counting and QBF respectively, two problems which are conjectured to be strictly harder than SAT.

Table 2.1 shows a number of restrictions and extensions of SAT which result in problems with different complexity. It gives variations of the satisfiability problem in roughly increasing order of complexity.

2.1 2-SAT

The **2-SAT** problem restricts clauses in a CNF-SAT formula to have length at most 2. While k-SAT for $k \geq 3$ is known to be NP-complete, and therefore not known to have a polynomial time algorithm, 2-SAT is known not only to be in P, but the smaller class NL [3].

One algorithm for solving 2-SAT is in terms of graph exploration. A clause of length two is of the form $\ell_i \vee \ell_j$. This says that if one of the literals is false, then the other must be true. That is

$$(\ell_i \vee \ell_j) \Leftrightarrow (\neg\ell_i \Rightarrow \ell_j) \Leftrightarrow (\neg\ell_j \Rightarrow \ell_i) \tag{2.1}$$

Extracting these two implications from each clause in a 2-SAT formula φ allows us to form an **implication graph** $G = (V, E)$. In this graph, each vertex $v \in V$ is labeled with one of the $2n$ possible variable-assignment pairs in the formula. For each clause $\ell_i \vee \ell_j$ we add edges $e_{\neg\ell_i,\ell_j}$ and $e_{\neg\ell_j,\ell_i}$ to the graph. That is, $E = \{e_{\neg\ell_i,\ell_j} | \{\ell_i, \ell_j\} \in \varphi\}$. These edges denote implications, which translate to unit propagations:

Variation	Complexity	Applications
2-SAT	NL-complete	digraph path
XOR-SAT	$\oplus$L-complete	
Horn-SAT	P-complete	prolog
SAT/3-SAT/Circuit-SAT	NP-complete	model checking, reg. allocation
MAX-SAT	NP-hard	
k-QBF	Σ_i^P-complete/Π_i^P-complete	max-clique
$\oplus$SAT	$\oplus$P-complete	
MAJ-SAT	PP-complete	
#SAT/#2-SAT	#P-complete	model counting, model sampling
QBF	PSPACE-complete	game playing, planning

Table 2.1: A taxonomy of SAT variations.

in the clause $\ell_i \vee \ell_j$, if ℓ_i is false then ℓ_j must be true.

We then get an algorithm for 2-SAT by identifying cycles in the implication graph which contain nodes labeled with both a variable and its negation. This cycle says that given either assignment to a variable x_i, the opposite assignment will be implied resulting in a contradiction.

2.2 XOR-SAT

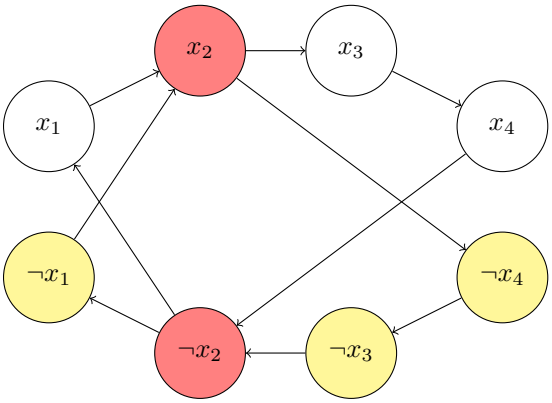
In the **XOR-SAT** problem, clauses contain the logical connective XOR $\oplus$ rather than OR $\vee$. A XOR-SAT clause $x_{i,1} \oplus \cdots \oplus x_{i,k} = b$ for some $b \in \{0, 1\}$ is simply a linear equation over the finite field with two elements $GF(2)$ (also denoted $\mathbb{Z}_2$ or $\mathbb{Z}/2\mathbb{Z}$). Because a XOR-SAT formula is simply a conjunction of linear equations, we may check for a solution in polynomial time using Gaussian elimination.

2.3 Model Counting

Counting the number of satisfying solutions (models) to a SAT formula is one natural extension of SAT. The SAT problem on its own asks "are there one or more satisfying solutions to this formula?" without discriminating between how many solutions more than one there might be.

Definition 4. *The **#SAT** problem asks how many solutions there are to a SAT formula. This is the canonical **#P**-complete problem.*

While many #P-complete problems correspond to the counting version of NP-complete decision prob-



$$2 - SAT formula:$$
$$x_1 \vee x_2$$
$$\neg x_1 \vee x_2$$
$$\neg x_2 \vee x_3$$
$$\neg x_3 \vee x_4$$
$$\neg x_4 \vee x_2$$

Figure 2.1: A 2-SAT formula and corresponding implication graph.

lems, surprisingly the counting version of some problems in P are also intractible. For example, 2-SAT is in P (more specifically, is NL-complete) and 3-SAT is NP-complete, however #2-SAT and #3-SAT are both #P-complete. In his seminal 1979 paper that introduced counting complexity, Valiant [50] also proved that computing the permanent of a matrix is #P-complete. This stands in contrast to computing the determinant, which can be done in polynomial time.

2.3.1 Sampling

One of the most important applications of counting is sampling. In fact counting, sampling, and enumerating solutions to #P-complete problems are all of about the same difficulty in the sense that given an oracle for one we can perform the other two. Sampling solutions is useful in practice. For example, suppose that we have modeled the correctness of a hardware system as a SAT formula, and wish to try out how different solutions to the constraints affect performance. One may then want to select solutions uniformly at random from the set of all possible solutions.

2.3.2 Counting via Blocking Clauses

A simple, naive algorithm for model counting use a SAT solving algorithm, "as-is". This algorithm simply counts the number of times it repeats the following two steps on a given CNF formula φ:

(1) Use a SAT solving algorithm to find a solution A to φ

(2) Add a blocking clause $C_b(A)$ to the formula to obtain a new formula $C_b(A)$

In practice this is a good way to obtain a small number of solutions to a formula. However, because a formula may have a number of solutions exponential in its number of variables n, this algorithm takes not only exponential time but also exponential space in the worst-case. Improved variants may block more than one satisfying assignment at a time by using blocking-style clauses shorter than n. However, the standard algorithm for model counting is a variant of DPLL. Chapter 3 discusses another basic algorithm, and improvements to it, which uses combinatorial techniques to count the number of models to a formula.

2.3.3 Counting via DPLL

The standard technique for solving #SAT uses a DPLL-based search [8, 26]. The technique is similar to the algorithm for SAT with some exceptions. The main difference is that for #SAT we must search the entire tree. This precludes the use of or reduces the effectiveness of several common optimizations to the DPLL algorithm. For example, when model counting we want to backtrack not just when a partial assignment falsifies a formula, but also when it satisfies it. When we identify a satisfying partial assignment of size d then we may add 2^{n-d} to our total solution count, and then backtrack.

2.4 Quantified Boolean Formulas (QBF)

The quantified Boolean formula (QBF) problem is a generalization of SAT which allows quantification. In addition to the logical connectives allowed in SAT formulas, we also allow QBFs Φ to contain the existential quantifier $\exists$ and the universal quantifier $\forall$. For much of the notation we follow [10].

We may α-rename subformulas of Φ to avoid the following two occurrences:

(1) Two quantifiers in Φ have the same variable x. For example, $\forall x(y \wedge x) \vee \exists x(y \vee x)$.

(2) A variable occurs free in one subexpression and bound in another. For example, $x \vee \forall x(y \wedge x)$.

Formulas without these cases are called **cleansed**.

Definition 5. *A **prenex normal form** formula Φ contains a block of quantifiers followed by a quantifier-free subformula called the **matrix**, where $Q_i \in \{\exists, \forall\}$.*

$$\Phi = Q_1 x_1 \cdots Q_n x_n \varphi \tag{2.2}$$

We may convert an arbitrary QBF into cleansed, prenex normal form in linear time [10]. Furthermore, we may apply the transformations described in the previous section to the matrix to obtain a 3-CNF formula in linear time. Therefore, we will define Q^*kCNF to be the set of QBFs in cleansed, prenex normal form and having a 3-CNF matrix containing no free variables. This will serve as a normal form.

Determining the satisfiability of a QBF formula is a PSPACE-complete problem. PSPACE is a complexity class known to contain, and conjectured to properly contain, NP.

2.5 MAX-SAT

Another optimization variant of the SAT problem, called **MAX-SAT** asks for the maximum number of clauses which are satisfiable simultaneously in a given CNF-SAT formula. In particular, if the formula is satisfiable, then the answer is trivially m, the number of clauses in the formula. Therefore an algorithm for MAX-SAT easily solves SAT, meaning that MAX-SAT is NP-hard.

Two further generalizations of MAX-SAT are **weighted MAX-SAT** and **partial MAX-SAT** [31]. Weighted MAX-SAT assigns a positive weight $w_i > 0$ to every clause C_i, representing the importance of that clause being true. The resulting optimization problem asks for a variable assignment which admits a set of satisfied clauses of maximal weight. The partial MAX-SAT problem divides clauses into two sets: hard constraints and soft constraints. The problem is then to satisfy the maximum number of soft constraints such that the hard constraints are all satisfied. That is, we treat the hard constraints as a normal SAT problem, and the soft constraints as a MAX-SAT problem for assignments satsifying the hard constraints. This is similar to linear programming where we seek to maximize a set of linear expressions subject to a set of inequalities.

Chapter 3

Inclusion-Exclusion

The inclusion-exclusion formula is well-known and widely used in combinatorics and probability. Our motivation in studying the inclusion-exclusion principle came from the observation that it can be used to solve hard counting problems. Specifically, we can use the inclusion-exclusion principle to solve #SAT as will be discussed in detail below. Other algorithms for solving #SAT are discussed in the previous chapter.

This chapter of the thesis draws heavily from work done jointly with and advised by Sriram Sankaranarayanan. A preliminary version of this work was presented as a poster and short paper [6] at SAT 2011, and an updated journal version is currently in preperation [41]. Specifically, this work presents two improvements to existing techniques. First, we give an improved characterization of Bonferroni inequalities where we can obtain provable upper and lower bounds on a larger class of subformulas. Second, we introduce a technique called **subsumption** which asymptotically improves the worst-case running time of the inclusion-exclusion based algorithm for solving #SAT, and (non-asymptotically) improves the average-case running time.

Work on improving Bonferroni inequalities, either in terms of generalization or reduced computation, is prevelant in the literature [22, 24].

To the best of our knowledge, Linial and Nisan in [32] first introduced the idea of using the inclusion-exclusion principle to count the models of a propositional formula, also showing that beyond depth $O(\sqrt{n})$ an inclusion-exclusion sum converges rapidly. This work was furthered by Iwama [27] and Lonzinskii [35], both of which take advantage of the structure of CNF SAT to analyze the average time complexity required for exactly computing the number of models of a random CNF formula. However, as [32] points out, the idea of using inclusion-exclusion to solve hard counting problems goes back to work in the early 1960s [40] in which

Ryser uses it in an algorithm for computing the permanent of a matrix before the problem's complexity was formally established by Valiant [50].

Suppose we have a collection of sets $\mathcal{A} = \{A_1, \ldots, A_n\}$. The inclusion-exclusion principle gives a formula for computing the cardinality of the union of this collection of sets: $|\cup_{i=1}^{n} A_i|$. The formula, expressed as an alternating sum, plays an important role in combinatorics and probability. Bonferroni inequalities generalize the inclusion-exclusion principle by showing that truncactions of the sum at odd (even) depths give upper (lower) bounds. The inclusion-exclusion sum includes a term for each element in the powerset of $\{A_1, \ldots, A_n\}$ other than the empty set and therefore requires exponentially many computations in the worst case.

Definition 6. *Given a family of sets $\{A_1, \ldots, A_n\}$, the **inclusion-exclusion** principle gives a formula for computing $|\cup_{i=1}^{n} A_i|$.*

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{S \subseteq [n], S \neq \emptyset} (-1)^{1+|S|} \left|\bigcap_{i \in S} A_i\right| \tag{3.1}$$

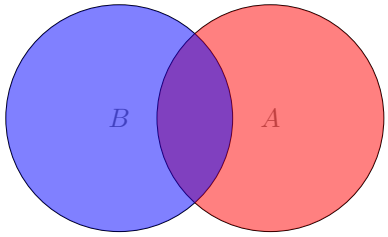
where the notation $[n]$ denotes the set $\{1, \ldots, n\}$.

This sum has $2^n - 1$ terms, one for each element in the power set of $[n]$ other than the empty set, meaning that the number of operations we will have to perform to compute the sum is exponential in n in the worst case. However, the inclusion-exclusion sum is *alternating*, a fact which plays a key role in the optimizations presented in [6]. The main result in our work shows how we can identify large numbers of terms with the same cardinality but different sign, and therefore conclude that their mutual contribution to the sum is zero.

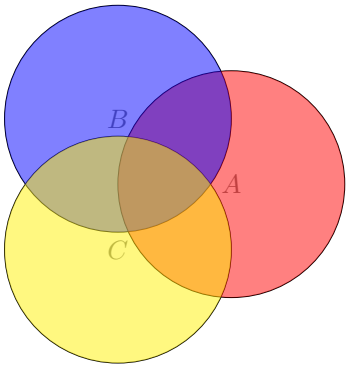
Definition 7. *We also get a corollary to equation 3.1 called the **Bonferroni inequalities** in which we may obtain an upper or lower bound on $|\cup_{i=1}^{n} A_i|$ by truncating the sum to sets of a given size k:*

$$\left|\bigcup_{i=1}^{n} A_i\right| \leq \sum_{S \in [n], 0 < |S| \leq k} (-1)^{1+|S|} \left|\bigcap_{i \in S} A_i\right| \quad \textit{for odd } k \tag{3.2}$$

$$\left|\bigcup_{i=1}^{n} A_i\right| \geq \sum_{S \in [n], 0 < |S| \leq k} (-1)^{1+|S|} \left|\bigcap_{i \in S} A_i\right| \quad \textit{for even } k \tag{3.3}$$



$$|A \cup B| = |A| + |B| - |A \cap B|$$



$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

Figure 3.1: Venn diagrams for two and three sets demonstrate the inclusion-exclusion principle visually.

3.1 Inclusion-Exclusion as Tree Exploration

The improvements discussed in [41] come from the observation that the inclusion-exclusion principle may be viewed as tree exploration. We use a tree as a device for organizing the terms in an inclusion-exclusion sum.

Definition 8. *Given a family $\mathcal{A} = \{A_1, \ldots, A_n\}$ of sets, the **inclusion-exclusion (I-E) tree** of $\mathcal{A}$ has a node for each subset $S \subseteq [n]$.*

We formally define an I-E tree using the following rules:

(1) The root node of an I-E tree is labeled with $[\emptyset]$. Each non-root node $S = [i_1, \ldots, i_d]$ at depth d in the inclusion-exclusion tree is labeled with an ordered, increasing sequence of d integers $1 \leq i_1 < \cdots < i_d \leq n$.

(2) Each node S has $n - i_d$ children $[S, i_d + 1], [S, i_d + 2], \ldots, [S, n]$, where the syntax $S' = [S, i_{d+1}]$ is shorthand for $[i_1, \ldots, i_d, i_{d+1}]$.

The tree shown in figure 3.2 gives the organizational layout for the tree representation of an inclusion-exclusion sum for a collection of four sets, with leaf nodes colored green for clarity.

Remark 1. *An I-E tree for a collection of n sets may be viewed as an organizational device for enumerating the elements in the power set of the first n integers. Because the I-E tree representation of any collection of*

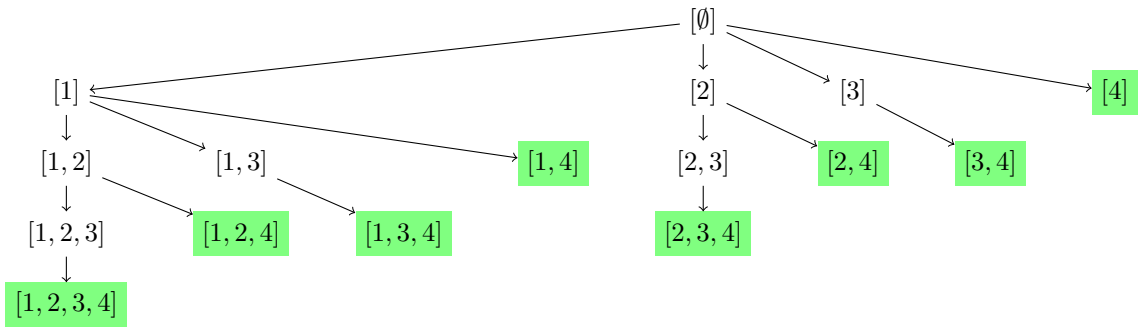


Figure 3.2: Inclusion-exclusion tree for a family $\mathcal{A} = \{A_1, A_2, A_3, A_4\}$ consisting of four sets.

n sets is the same, all of these trees are isomorphic, although the cardinalities of the sets that they represent differ.

There is a one-to-one correspondence between non-root nodes $S = [i_1, \ldots, i_d]$ in the tree and terms

$$t(S) = (-1)^{1+|S|} \left|\cap_{i \in S} A_i\right|$$

in the inclusion-exclusion sum 3.1. Therefore, if we set $t([\emptyset]) = 0$, we may compute the inclusion-exclusion sum by performing a search of the tree, summing the contributions of each node.

One powerful improvement to this search comes from the observation that nodes S with $\cap_{i \in S} A_i = \emptyset$ contribute nothing to the sum. Furthermore, because $\emptyset \cap B = \emptyset$ for any set B, no descendant of S will ever contribute to the overall sum. Therefore we may optimize our algorithm by pruning any subtrees whose roots $\cap_{i \in S} A_i = \emptyset$ from our traversal.

Algorithm 3 formally presents the basic I-E tree based technique for computing bounds on the cardinality of the union of a family of sets using a depth-first traversal of an I-E tree. Starting with the root, we perform a depth-first traversal of the tree by pushing the children of each non-empty node (other than the root) onto a stack, and then popping off the stack's top element. We sum the contributions of each non-empty node. Recall that for odd d algorithm 3 gives an upper bound, whereas for even d it gives a lower bound. Note that if we replaced the stack with a queue, then our algorithm works by performing a breadth-first traversal. Because the empty set pruning optimization works given either a depth-first or breadth-first traversal order, the reduced space complexity of a depth-first search is desirable, but both traversals are

Algorithm 3: Basic algorithm for computing the cardinality of the union of a family of sets via an I-E tree.

```
Input: Collection of sets A = {A_1, ..., A_n}, depth bound d
Output: Upper or lower bound on |∪_{i=1}^n A_i|
stack.push([∅]);
sum := 0;
while !(stack.isEmpty()) do
    S := stack.pop();
    v := t(S);
    if (|S| ≤ d) && (S = ∅ || v > 0) then
        stack.push(children(S));
        sum := sum + v
return sum;
```

sound.

For each node S in the I-E tree, let $\mathsf{subtree}(S)$ denote the **subtree** rooted at S, and let $\mathsf{subtree}^\dagger(S) = \mathsf{subtree}(S) \setminus \{S\}$ denote the **proper subtree** of S. We also extend the valuation function t to subtrees as the sum over all nodes in the subtree:

$$t(\mathsf{subtree}(S)) = \sum_{S' \in \mathsf{subtree}(S)} t(S')$$

$$t(\mathsf{subtree}^\dagger(S)) = t(\mathsf{subtree}(S)) - t(S)$$

The following lemma makes the key observation that subtrees of the I-E tree are also I-E trees. Let $S = [i_1, \ldots, i_d]$ be a non-root node of the I-E tree with $k > 0$ children $T_1 = [S, i_d + 1], \ldots, T_k = [S, i_d + k]$ where $n = i_d + k$. Recall that the contribution of the proper subtree rooted at S is given by

$$t(\mathsf{subtree}^\dagger(S)) = \sum_{j=1}^{k} t(\mathsf{subtree}(T_j)) \tag{3.4}$$

Lemma 1.

$$\left| t(\mathsf{subtree}^\dagger(S)) \right| = (-1)^d \left| \bigcup_{j=i_d+1}^{n} \mathsf{Intersect}(S) \cap A_j \right|$$

where $\mathsf{Intersect}(S) = \cap_{i \in S} A_i$.

Proof. The identity follows from applying the inclusion-exclusion principle to the LHS of the identity, and noting the bijection between terms in the inclusion-exclusion sum to nodes in $\mathsf{subtree}^\dagger(S)$. □

3.2 Subsumption

Suppose that for some collection of sets S and some set $A_j \notin S$ we have that $\mathsf{Intersect}(S) \subseteq A_j$. Then it follows that $\mathsf{Intersect}(S) = \mathsf{Intersect}(S) \cap A_j$. When computing inclusion-exclusion sums the situation may arise where two sets have the same cardinality for the reason described above, but their contributions to the sum will have opposite parity. Identifying the resulting cancellations in the inclusion-exclusion tree forms the idea behind subsumption.

We can generalize the above observation as follows.

Definition 9. *A node S is **subsumed** by its children children$(S) = T_1 = [S, A_{i_d}], \ldots, T_k = [S, A_n]$ if*

$$\mathsf{Intersect}(S) \subseteq \cup_{j=i_d+1}^{n} A_j$$

Remark 2. *An elementary case of subsumption occurs when $\mathsf{Intersect}(S) = \mathsf{Intersect}(T_j)$ for a single child T_j of S.*

The main result of this section is the following:

Theorem 2. *If S is a node in an I-E tree that is subsumed by its children then $t(\mathsf{subtree}(S)) = 0$.*

Proof. By lemma 1 we have that

$$t(\mathsf{subtree}^{\dagger}(S)) = \left| \bigcup_{j=i_d+1}^{n} \mathsf{Intersect}(S) \cap A_j \right|$$

Combining this with the definition of subsumption and noting that S has the opposite sign of $t(\mathsf{subtree}^{\dagger}(S))$ we have that

$$\begin{aligned}
t(\mathsf{subtree}(S)) &= t(S) + t(\mathsf{subtree}^{\dagger}(S)) \\
&= (-1)^{1+|S|} \left|\mathsf{Intersect}(S)\right| \; + \; (-1)^{|S|} \left| \textstyle\bigcup_{j=i_d+1}^{n} \left(\mathsf{Intersect}(S) \cap A_j\right) \right| \\
&= (-1)^{1+|S|} \left|\mathsf{Intersect}(S)\right| \; + \; (-1)^{|S|} \left| \mathsf{Intersect}(S) \cap \textstyle\bigcup_{j=i_d+1}^{n} (A_j) \right| \\
&= (-1)^{1+|S|} \left|\mathsf{Intersect}(S)\right| \; + \; (-1)^{|S|} \left|\mathsf{Intersect}(S)\right| \\
&= 0
\end{aligned}$$

□

Algorithm 4: Improved algorithm for computing the cardinality of the union of a family of sets via an I-E tree using subsumption.

Input: Collection of sets $\mathcal{A} = \{A_1, \ldots, A_n\}$, depth bound d
Output: Upper or lower bound on $|\cup_{i=1}^{n} A_i|$

```
stack.push([∅]);
sum := 0;
while !(stack.isEmpty()) do
    S := stack.pop();
    v := t(S);
    if (|S| ≤ d) && (S = ∅ || v > 0) && !(isSubsumed(S)) then
        stack.push(children(S));
        sum := sum + v
return sum;
```

In algorithm 3 we used one pruning technique for our tree exploration based on null intersections. Using theorem 2 we obtain a second pruning technique based on subsumption. This leads us to algorithm 4. Notice that using classical Bonferroni inequalities, the algorithm with $d < n$ would not give a guaranteed bound. We show in the next section that given extended Bonferroni inequalities the algorithm is in fact correct.

3.3 Extended Bonferroni Inequalities

In this section we show how characterizing an inclusion-exclusion sum as a tree leads to a natural generalization of Bonferroni inequalities.

Recall that the inclusion-exclusion sum over subsets of size at most d yields an upper bound on the overall sum if d is odd, and a lower bound if d is even. We give the following notation for this truncated sum.

$$B_d = \sum_{S \subseteq [n], 1 \leq |S| \leq d} (-1)^{1+|S|} \cdot |\mathsf{Intersect}(S)| \tag{3.5}$$

There is a natural correspondence between these partial sums B_d and subtrees of an I-E tree. Because the nodes at depth d in an I-E tree correspond to terms in the I-E sum which take the cardinality of an intersection of d sets, we get a alternative characterization of B_d. Given an I-E tree $\mathbf{T}$ let $\mathbf{T_d}$ be the subtree of $\mathbf{T}$ formed by removing all nodes at depth (strictly) greater than d, defined for $1 \leq d \leq n$. Then we have:

$$B_d = \sum_{S \in \mathbf{T_d}} t(S) \geq \left| \bigcup_{i=1}^{n} A_i \right| \quad \text{if } d \text{ is odd}$$
$$B_d = \sum_{S \in \mathbf{T_d}} t(S) \leq \left| \bigcup_{i=1}^{n} A_i \right| \quad \text{if } d \text{ is even}$$

Note that in this case, corresponding to classical Bonferroni inequalities, all "cutoffs" occur at the same depth d. However, we now show that treating the I-E sum as a tree leads to an extension of these inequalities.

Definition 10. *From an I-E tree* $\mathbf{T}$ *and a set of non-root nodes* $\mathcal{S} = \{S_1, \ldots, S_k\}$ *where* $\forall S_i \; S_i \in \mathbf{T} \wedge d(S_i) > 0 \wedge d(S_i) \mod 2 = 0$ *we obtain an* ***even depth cutoff tree*** $\mathbf{T}' = \{V \in \mathbf{T} | \forall S_i \; V \notin \mathsf{subtree}^{\dagger}(S_i)\}$*, where* $d(S_i)$ *indicates the depth of a node* S_i *in the tree.*

Conceptually, given an I-E tree $\mathbf{T}$ we may obtain an even depth cutoff tree $\mathbf{T}'$ by the following procedure:

(1) Select a cutoff frontier of non-root nodes $\mathcal{S}$ all at even depths in $\mathbf{T}$.

(2) Remove all nodes from $\mathbf{T}$ contained in the proper subtree $\mathsf{subtree}^{\dagger}(S_i)$ of every node $S_i \in \mathcal{S}$.

An **odd depth cutoff tree** is defined similarly, but with the restriction that $\forall S_i \; S_i \in \mathbf{T} \wedge d(S_i) \mod 2 = 1$.

Theorem 3. *Given a cutoff tree* $\mathbf{T}'$ *of an IE tree* $\mathbf{T}$

(1) If $\mathbf{T}'$ *is an odd depth cutoff tree then* $t(\mathbf{T}') \geq |\cup_{i=1}^{n} A_i|$

(2) If $\mathbf{T}'$ *is an even depth cutoff tree then* $t(\mathbf{T}') \leq |\cup_{i=1}^{n} A_i|$

Here we take advantage of the tree organization of the inclusion-exclusion sum. The proof of these inequalities uses the observation that the nodes contained in the proper subtree of a node correspond to the terms in a new inclusion-exclusion sum.

Proof. By lemma 1 we have that

$$t(\mathsf{subtree}^{\dagger}(S)) = (-1)^d \left| \bigcup_{j=i_d+1}^{n} \mathsf{Intersect}(S) \cap A_j \right|$$

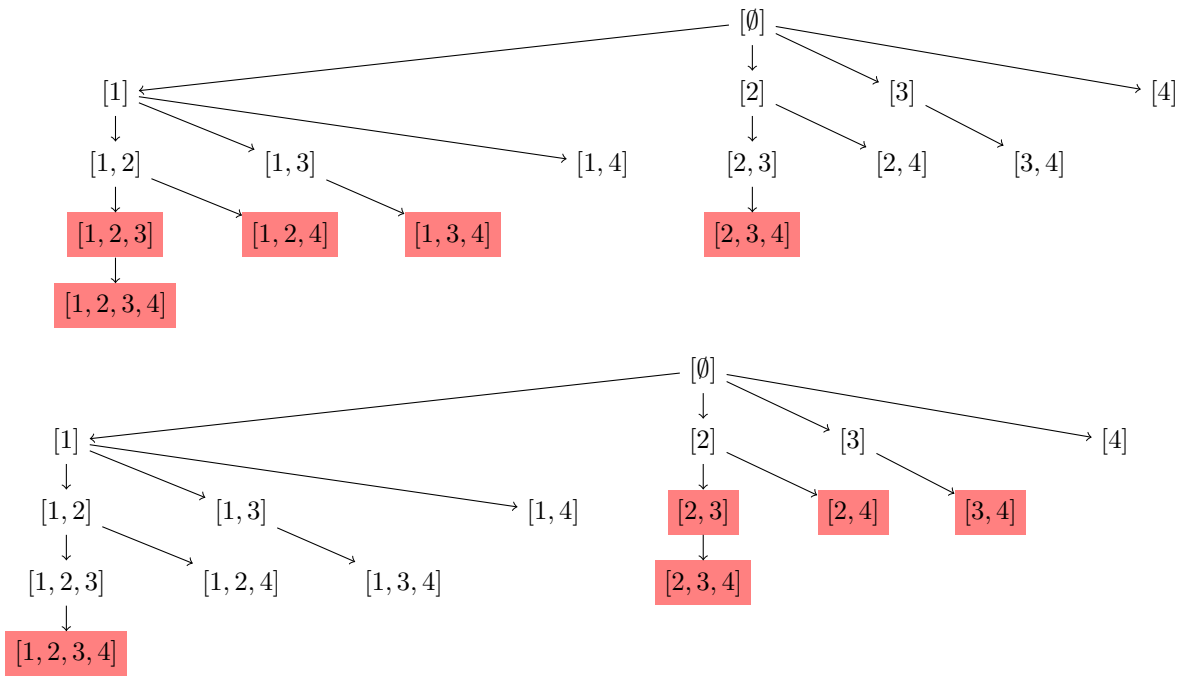


Figure 3.3: Cutoff tree corresponding to classical Bonferroni inequalities (top) and cutoff tree with cutoffs occurring at different heights (bottom). Pruned nodes appear in red. The sum of the terms in the tree on top yield a lower bound on the overall sum, while the terms in the lower tree yield an upper bound.

Therefore, if d is odd then $t(\mathsf{subtree}^\dagger(S)) \leq 0$ and if d is even then $t(\mathsf{subtree}^\dagger(S)) \geq 0$.

Let $\mathbf{T}'$ be an odd depth cutoff tree with cutoff frontier $\mathcal{S} = \{S_1, \ldots, S_k\}$. Without loss of generality we assume that no node in $\mathcal{S}$ is a descendant of another node in $\mathcal{S}$. Using the inclusion-exclusion formula we have

$$t(\mathbf{T}') = \left| \bigcup_{i=1}^{n} A_i \right| - \sum_{i=1}^{k} t(\mathsf{subtree}^\dagger(S_i))$$

A given cutoff node S_i occurs at an odd depth, so

$$\forall S_i \in \mathcal{S} \;\; t(\mathsf{subtree}^\dagger(S_i)) \leq 0 \quad \Longrightarrow \quad \sum_{i=1}^{k} t(\mathsf{subtree}^\dagger(S_i)) \leq 0 \quad \Longrightarrow \quad t(\mathbf{T}') \geq \left| \bigcup_{i=1}^{n} A_i \right|$$

The proof for even depth cutoff trees is similar. □

Corollary 1. *The odd (even) depth cutoff trees induced by subsumption give upper (lower) on the inclusion-exclusion sum represented by the original tree.*

3.4 Model Counting via Inclusion-Exclusion

We now show how the #SAT problem may be formulated in terms of an inclusion-exclusion sum.

Let φ be a k-CNF formula with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. Using the set representation of φ, we view a clause as a set of k literals: $C_i = \{\ell_{i,1}, \ldots, \ell_{i,k}\}$.

We will use the inclusion-exclusion principle to count the number N_u of falsifying solutions to φ. Note that given N_u we may easily compute the number of satisfying solutions to φ as $2^n - N_u$.

Let A_i denote the set of total assignments to $x_1, \ldots x_n$ that do not satisfy clause C_i. If C_i has k distinct literals then we note that $N_u(C_i) = |A_i| = 2^{n-k}$. This is because there is exactly one partial assignment to the k variables in C_i that does not satisfy C_i. Then the remaining $n - k$ variables may be assigned arbitrarily, giving a total of 2^{n-k} falsifying asssignments to C_i assuming that there are no repeated variables in C_i. This assumption is without loss of generality; we may assume that we have preprocessed the formula to remove any such repeats as described in the introduction.

Given a subset $S \subseteq \{C_1, \ldots, C_m\}$ of the clauses in φ, let $\text{lits}(S) = \cup_{C_i \in S} C_i$. That is, $\text{lits}(S)$ is the set of all literals appearing in one of the clauses in S.

Let N_u denote the number of falsifying assignments to the formula φ.

Definition 11. *Given two clauses C_i, C_j in φ we say that C_i **conflicts** with C_j if C_i contains a literal ℓ and C_j contains its negation $\neg\ell$. A set of clauses $S \subseteq \{C_1, \ldots, C_m\}$ is conflicting if there are two conflicting clauses in S.*

Using the base case counting scheme given for a single clause above together with the definition of conflicting clauses, we get the following counting scheme for a set of clauses S.

$$\bigcap_{C_i \in S} A_i = \begin{cases} 0 & \text{if } \exists j \text{ such that } \{x_j, \neg x_j\} \subseteq \text{lits}(S) \\ (-1)^{|S|+1} \cdot 2^{n-|\text{lits}(S)|} & \text{otherwise} \end{cases}$$

We may then represent the total number of falsifying solutions N_u to φ using the following inclusion-exclusion sum:

$$N_u = \sum_{S \subseteq \{C_1, \ldots, C_m\}, S \neq \emptyset} (-1)^{|S|+1} \cdot \bigcap_{C_i \in S} A_i \tag{3.6}$$

By then applying algorithm 3 we can compute the number of satisfying solutions to a given propositional formula φ.

The average-case time complexity of this algorithm was studied in papers by Lonzinskii [35] and Iwama [27]. The following theorem gives greatly improved time bounds over the naive algorithm by taking into account conflicts, but does not account for subsumptions.

Theorem 4 (Lonzinskii, 1992)**.** *Given a random[1] k-SAT formula φ with n variables and m clauses the average time complexity of computing the exact number of models of φ using algorithm 3 is*

$$O(m^{b+2} \cdot n)$$

where

[1] Variables are selected with uniform probability for each clause, and negated with probability 1/2, meaning repeated clauses are allowed, but repeated variables within a clause are disallowed.

	No Subsumptions	Subsumptions	Perc. Change
Av. Time:	44.848	29.826	-0.335
Av. Nodes Explored	5437962	3649815	-0.329
Av. # Conflicts:	4440964	2887100	-0.35
Av. # Subsumptions:	0	19981	N/A

Table 3.1: Inclusion-exclusion based model counting with and without subsumption pruning. This table shows averages over 50 trials on randomly generated SAT instances with 30 variables, 50 clauses, and 4 literals per clause.

$$b < \frac{\ln m}{n}\left(\frac{2n^2}{k^2} - \frac{3}{8}\right)$$

Lemma 2. *The maximum depth necessary to explore in an inclusion-exclusion model counting tree before encountering a conflict or subsumption is* $\min n+1, m$.

Proof. Because the entire inclusion-exclusion tree with no pruning has depth m, the m bound is trivial. Furthermore, by the pigeonhole principle, because the formula contains n variables, only n clauses may introduce new, non-conflicting literals. Therefore a clause sequence $S = [C_{i_1}, \ldots, C_{i_{n+1}}]$ is either conflicting or a subsumption occurs, meaning that $\text{lits}(C_{i_{\ell+1}}) \subseteq \text{lits}(C_{i_1}, \ldots, C_{i_\ell})$ for some $\ell \leq n$. □

3.5 Experiments

Our experiments show that subsumptions greatly improve the average-case performance of inclusion-exclusion based model counting. Table 3.1 shows that for random SAT instances generated with 30 variables, 50 clauses, and 4 literals per clause that the subsumption-based inclusion-exclusion technique yields an over 30% speed-up. The plot in figure 3.4 shows that the subsumption-enhanced method converges approximately and completely at lower depths.

Experiments comparing inclusion-exclusion based model counting to DPLL based model counting were mixed. DPLL based model counting seems to be faster in many likely parameter settings, however random instances with large clauses and low clause-to-variable ratios were solved much more quickly by the inclusion-exclusion algorithm. Unfortunately instances with long clauses are not conducive to subsumptions occurring (conflicts are very likely) so our improved algorithm did not yield a substantial speed-up.

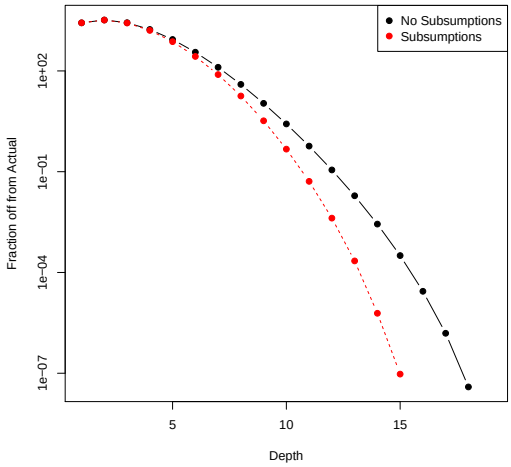


Figure 3.4: Plot showing the convergence of subsumption-enhanced inclusion-exclusion versus normal ineclusion-exclusion based model counting based on averages over 50 trials. Subsumption enhanced inclusion-exclusion converges more rapidly, and fully converges at a lower depth.

3.6 Future work

One aim for future work is to perform average-case complexity analysis on algorithm 4 as Lozinskii and Iwama did for algorithm 3. Doing this for subsumptions is more difficult than for conflicts because subsumption is an ordered property – whether a given clause is subsumed depends on at what depth it occurs at in the tree.

Chapter 4

Phase Transitions

One key question in constraint solving is understanding what makes SAT and related problems difficult. Many industrial SAT instances are solvable very quickly, such as those encountered in model checking [13] and planning, while certain random instances seem to take a very long time to solve. Phase transitions, the idea of which is taken from statistical physics, give insight into this question.

This chapter focuses on models for generating random SAT and QBF instances, while the next chapter focuses on phase transitions occuring in formulas generated using these models. We start with a brief survey of both of these topics.

There are two phenomena that constitute phase transitions:

(1) **Satisfiability thresholds** – when some input parameter $\alpha < \alpha^* - \epsilon$ then random instances are satisfiable with high probability (w.h.p.), whereas when $\alpha > \alpha^* + \epsilon$ they are unsatisfiable with w.h.p. When $\alpha = \alpha^*$ roughly half of the instances are satisfiable.

(2) **Hardness thresholds** – when some input parameter $\alpha \in \{\alpha^* - \epsilon, \alpha^* + \epsilon\}$ then random instances require exponentially more time than when $\alpha \notin \{\alpha^* - \epsilon, \alpha^* + \epsilon\}$.

4.1 Random SAT Models

One important part of generating random problem instances is to rigorously define a random model. In the case of random SAT we wish to generate m clauses, each containing k random literals. This family of models, where the number of literals per clause k is fixed are called fixed clause length (FCL) random SAT

models. However, this notion is ambiguous in how it treats repeated variables and repeated literals within clauses, as well as repeated clauses within a formula. There are two common definitions of random k-SAT, both disallowing repeated variables within a clause. A model presented in [45] by Selman et al. allows repeated clauses and one in [1] does not.

We let n denote the number of variables, m the number of clauses, and k the number of literals per clause in a SAT formula.

Definition 12 (Selman Random SAT)**.** *A Selman random (n, m, k)-SAT instance is a SAT formula φ where m clauses are generated by repeatedly selecting k variables uniformly at random, and negating each with probability $\frac{1}{2}$.*

Definition 13 (Achlioptas Random SAT)**.** *A random (n, m, k)-SAT instance is a SAT formula φ where m clauses are chosen at random, without replacement, from the $2^k\binom{n}{k}$ possible clauses.*

The important difference between the two random SAT definitions is that Selman's model allows repeated clauses, while Achlioptas' model does not. However, which of these models we pick is somewhat irrelevant. As stated by Selman et al. in [45], "this method of generation allows duplicate clauses ... However, as n gets large, duplicates will become rare because we generally select only a linear number of clauses." The following lemma makes this precise in terms of its effect on the satisfiability threshold.

Let r_k^s be the k-SAT critical clause-to-variable ratio in the Selman random model, and r_k^a the critical ratio in the Achlioptas random model.

Lemma 3. *$r_k^s = r_k^a$ for $k \geq 3$*

Proof. We will show that the expected number of identical pairs of clauses $C(n, m, k)$ in a $F_s(n, m, k)$ instance is zero around the phase transition.

The probability that a pair of randomly selected clause are identical is $p = \frac{1}{2^k\binom{n}{k}}$, while the number of such pairs in a given formula is $N = \binom{m}{2}$. Therefore, by the linearity of expectation, we have

$$C(n, m, k) = N \cdot p = \frac{\binom{m}{2}}{2^k\binom{n}{k}}$$

Fix $k \geq 3$. Then by Equation 4.1 we have that $r_k = O(2^k)$, so that the clause-to-variable ratio phase transition for k-SAT will occur at $m = O(2^k \cdot n) = O(n)$. Furthermore, we have that $\binom{m}{2} = O(m^2)$ and that $\binom{n}{k} = O(n^k)$. Combining this with the previous expression we get

$$\frac{\binom{m}{2}}{2^k \binom{n}{k}} = \frac{O(m^2)}{2^k \binom{n}{k}} = \frac{O(n^2)}{O(n^k)}$$

We conclude that for fixed $k \geq 3$, $m = O(2^k \cdot n)$

$$\lim_{n \to \infty} C(n, m, k) = \lim_{n \to \infty} \frac{O(n^2)}{O(n^k)} = 0$$

□

Conceptually Lemma 3 shows that because the phase transition occurs when $m = O(n)$, the number of possible clauses drastically exceeds the number of selected clauses. This means that for large n there is a very low chance of repeated clauses occuring in a formula in the Selman model. In the limit, the probability is zero meaning that the value of r_k is robust – it is not affected based on whether we allow repeated clauses or not.

4.2 The Clause-to-Variable Ratio Phase Transition

Phase transitions in satisfiability problems have interested both the applied and theoretical parts of the SAT community in the past three decades. Random SAT provides insight into which classes of formulas are easy to solve, and which are hard. From a practical standpoint, this gives SAT-solver designers heuristics for what types of formulas will be hard to solve, and good benchmarks for testing. Theorists use the SAT phase transition as evidence for the intractability of hard instances of NP-complete problems.

Following [1], we introduce the notation $F_k(n, m)$ to denote an Achlioptas random (n, m, k)-SAT instance.

Conjecture 1 (Satisfiability Threshold Conjecture)**.** *For every $k \geq 3$ there exists a constant r_k such that*

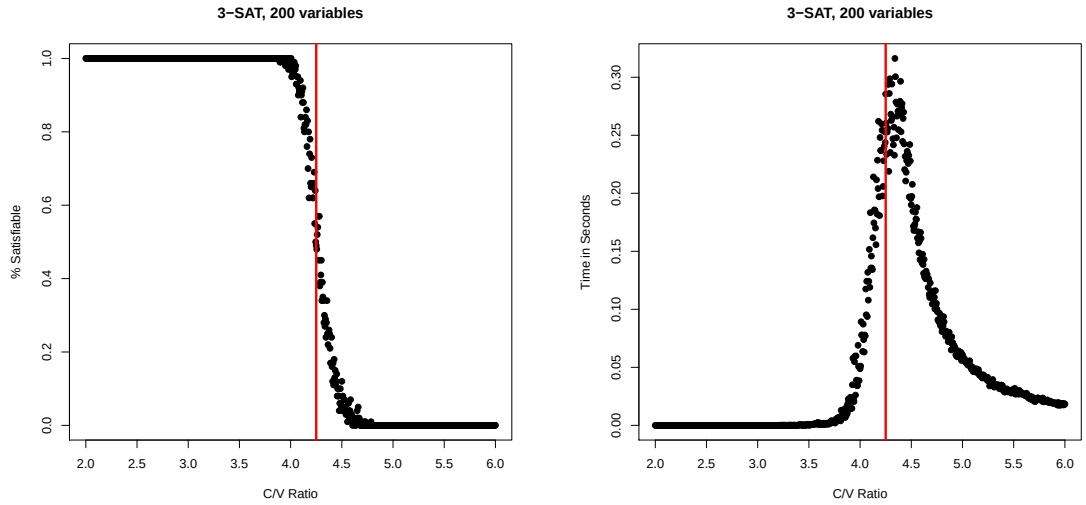


Figure 4.1: Scatter plots showing the clause-to-variable ratio phase transition through the sharp satisfiability threshold (left) and exponential hardness threshold (right) for randomly generated 3-SAT instances with 200 variables. The vertical red line at $c/v = 4.25$ shows the conjectured location of the phase transition [45]. Each data point represents 100 trials.

$$\lim_{n\to\infty} Pr[F_k(n, r\cdot n) \text{ is } SAT] = \begin{cases} 1 & \text{if } r < r_k \\ 0 & \text{if } r > r_k \end{cases}$$

Theorem 5 (Threshold bounds)**.** *There exist sequences* $\delta_k, \epsilon_k \to 0$ *such that for all* $k \geq 3$

$$2^k \ln 2 - (k+1)\frac{\ln 2}{2} - 1 - \delta_k \leq r_k \leq 2^k \ln 2 - \frac{1+\ln 2}{2} + \epsilon_k \tag{4.1}$$

The upper bound is due to independent work by Dubois and Boufkhad, and Kirousis et al., while the lower bound is due to Achlioptas and Peres [1].

4.3 2+p SAT

Another important random model in (2+p)-SAT. This model is meant to "interpolate" between tractable (polynomial time solvable) 2-SAT formulas and intractable 3-SAT formulas by mixing clauses of length 2 and 3. More formally, to generate (2+p)-SAT formula with m clauses and $p \in [0, 1]$, we generate a 2-SAT formula with pm clauses, and a 3-SAT formula with $(1-p)m$ clauses, and take the disjunction of

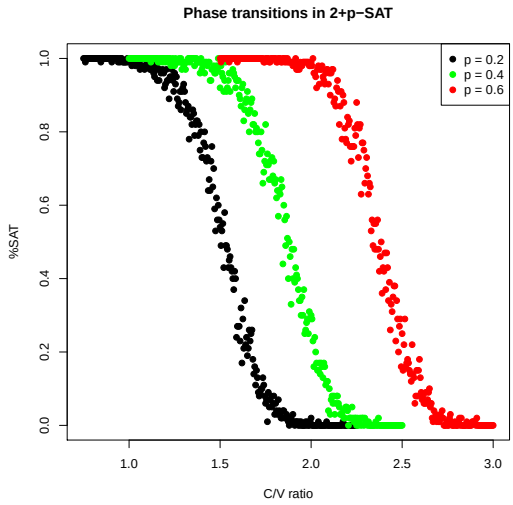


Figure 4.2: Scatter plot showing phase transitions in 2+p-SAT formulae with $p = 0.2, p = 0.4$ and $p = 0.6$. All trials were performed with 200 variables. Each data point represents 100 trials.

the two.

The following two important results related to (2+p)=SAT appear in [2]. Let $g_p(n, r)$ be the probability that $F_{2+p}(n, rn)$ is satisfiable.

Theorem 6 (AKKK1)**.** *For every* $p \in [0, 1]$ *there exists* $r_p(n)$ *such that for every* $\epsilon > 0$

$$\lim_{n\to\infty} g_p(n, r_p(n) - \epsilon) = 1 \quad \textit{and} \quad \lim_{n\to\infty} g_p(n, r_p(n) - \epsilon) = 0$$

This theorem asserts the *sharpness* of the transition from satisfiable to unsatisfiable of any family of (2+p) random formulas in the sense that formulas generated with $r < r_p(n) - \epsilon$ are satisfiable w.h.p, and formulas generated with $r > r_p(n) + \epsilon$ are unsatisfiable w.h.p. Figure 4.2 demonstrates this property experimentally; sharp phase transitions are shown for (2+p)-SAT instances with several values of p. As p increases, the phase transition shifts to the right.

We may consider partitioning a given (2+p)-SAT formula φ into 2-clauses $\varphi(2)$ and 3-clauses $\varphi(3)$. Let p^* be the largest value of p such that a $F_{2+p}(n, rn)$ random instance φ is almost surely satisfiable if and only if $\varphi(2)$ is satisfiable. A priori it is not even clear that $p > 0$, but in fact $p \geq 0.4$. The theorem given below also gives an upper bound.

Theorem 7 (AKKK2)**.** $0.4 \geq p^* \geq 0.695$

To the best of our knowledge, the upper bound first appeared in [2] while the lower bound was known from a variety of sources. It is conjectured that $p^* = 0.413$.

Chapter 5

Existential-Universal Phase Transition

5.1 Introduction

In the past ten years a sequence of papers has aimed to extend the study of random SAT into the harder quantified Boolean formula (QBF) problem domain. This work has mainly focused on adapting study of the clause-to-variable phase transition into QBF. The main contribution discussed in this section of the thesis is a new phase transition which arises from altering the fraction of variables in a quantified formula which are existentially versus universally quantified. We give strong experimental data confirming the existence of the phase transition given a variety of different parameter settings. This section of the thesis is an improved version of our submitted work [5].

5.1.1 Related Work

Work on random QBF started with a paper by Cadoli et al. [11]. In this paper, the authors adapt the fixed clause length (FCL) random SAT model to QBF, and give a number of experimental results showing phase transitions arising from varying the ratio of clauses to variables per quantifier block. They also tweak their random model to account for trivially unsatisfiable instances arising when a formula has a clause containing all universally quantified variables.

Shortly afterward, Gent and Walsh [25] wrote a follow-up paper in which they point out a "flaw" in the Cadoli-Giovanardi-Schaerf (Cadoli FCL) random QBF model stemming from the fact that a large fraction of generated instances were trivially unsatisfiable. They proposed two models to fix this flaw, the first of which we discuss in detail. They also describe new experimental results for the presence of a phase

transition in 2-QBF arising from increasing the clause to existentially quantified variable ratio in each of their models, and try to estimate the location of the phase transition formally using a general notion of constrainedness.

In [12], Chen and Interian gave a new model for random QBF generation. The key difference in their model is that the number of variables from each quantifier block appearing in a clause is fixed. They argue that this model is less ad-hoc, and more conducive to mathematical analysis.

In a recent pair of papers [17, 18] Creignou et al. use the Chen-Interian model to study (1,2)-CNF random formulas in detail by exploiting their close connection to 2-SAT. Additionally, they view their problem as an interpolation as well – between P and coNP-complete. Another paper by Creignou and coauthors [16] studies phase transitions in quantified XOR formulas.

A number of papers [11, 38, 17, 18] discuss the effect that the number of universal variables has on the satisfiability and run time of a QBF. The paper by Cadoli et al. [11] discusses several experiments they ran attempting to solve QBFs with a different ratio of universal to existential variables. A paper by Rintanen [38] disusses randomly generated Σ_3-SAT formulas. The paper explores how the fraction of satisfiable formulas and their solution time change as a function of both the clause-to-variable ratio and fraction of universal variables used in generating the formulas. Creignou et al. [17, 18] show that the critical clause-to-existential-variable ratio in their work depends on the ratio of the total number of existential versus universal variables. Although each of these papers implicitly observes the effect of altering the fraction of universal versus existential variables in a QBF, none of them recognize that varying this fraction leads to a sharp satisfiability threshold or examine the effect of altering this parameter in isolation.

5.1.2 Preliminaries

A quantified conjunctive normal form (QCNF) formula is a QBF formula with a closed CNF propositional formula φ (CNF matrix). Any QBF formula may be expressed in this form.

$$F = Q_1 x_1 \cdots Q_n x_n \quad \varphi(x_1, \ldots, x_n) \tag{5.1}$$

As discussed in the introduction, versions of QBF with a fixed number of quantifier alternations are complete for levels of the polynomial hierarchy, while QBF with an unbounded number of quantifier alternations is complete for PSPACE [4].

Throughout this section, we will be considering Π_2-SAT formulas, meaning the quantifiers consist of a block of universals followed by a block of existentials. We let n be the total number of variables in F, m the number of clauses in φ, and k the (fixed) number of literals per clause in φ. We also introduce n_u, n_e, with $n = n_u + n_e$ to represent the number of universally and existentially quantified variables in F.

That is, we will consider formulas of the following form:

$$F = \forall x_1, \ldots, \forall x_{n_u} \exists x_{n_u+1} \ldots, \exists x_n \varphi \tag{5.2}$$

5.2 The Existential-Universal Phase Transition

For all of the experiments in this section, we used the DepQBF solver [33, 34], the top solver in the main division of the QBFEVAL'10 [36]. The experiments were all run on a shared server with sixteen quad-core Xeon X7350 CPUs running at 2.93GHz, and 32GB of memory. The random instances were generated using Python scripts[1] .

In each experiment, the maximum number of clauses corresponds to under 40% of the critical clause-to-variable ratio for the corresponding k-SAT problem. Adding more clauses and increasing the fraction of universally quantified variables are two separate ways to add constrainedness to a problem. We keep the number of clauses below this point to ensure that when a small fraction of variables are universally quantified that the formulas will be satisfiable with high probability.

5.2.1 Analysis of Experimental Data

The data given in the graphs in Figure 5.1 shows the existential-universal phase transition given a number of parameter settings using Gent-Walsh model A [25], which we take as our standard model, and two additional plots using the pure and Cadoli FCL random models. In each case, increasing the number of universally quantified variables by a small number causes the fraction of randomly generated satisfiable

[1] `https://csel.cs.colorado.edu/~bennethd/randgen.html`

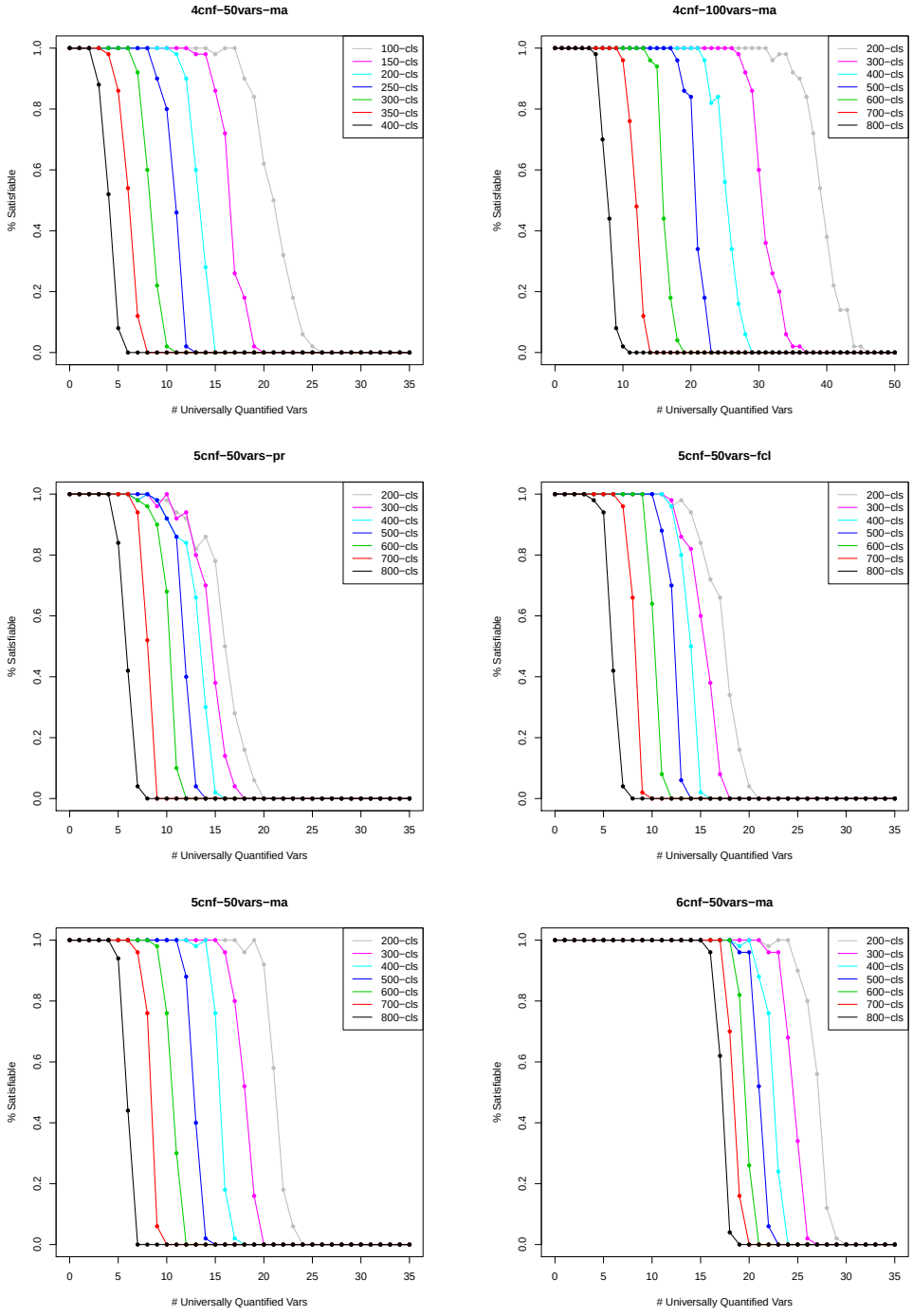


Figure 5.1: Plots showing satisfiability thresholds in Gent-Walsh Model A (ma), Cadoli FCL (fcl), and pure (pr) random models for a variety of parameter settings. Row 1: Satisfiability thresholds for $k = 4, n = 50$ and $k = 4, n = 100$, Row 2: Satisfiability thresholds for $k = 5, n = 50$ in the pure and fixed-clause-length random QBF models, Row 3: Satisfiability thresholds for $k = 5, n = 50$ and $k = 6, n = 50$.

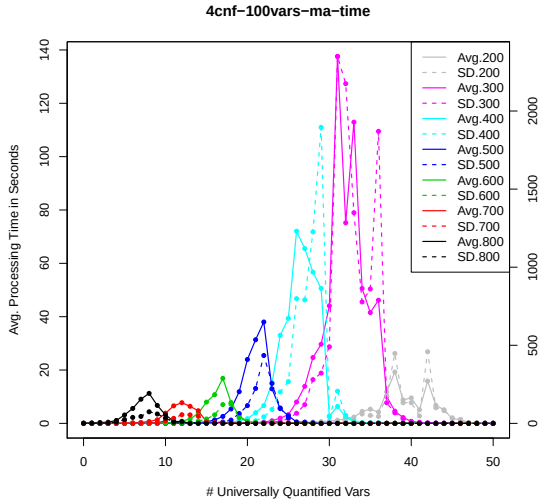


Figure 5.2: Averages and standard deviations of solution times for $k = 4, n = 100$. The scales on the left and right correspond to the average and standard deviations of solution times respectively.

instances to go from 1 to 0. Table 5.1 gives the estimated location of the phase transition for each of these models and clause settings. Interestingly, while the phase transition in the pure and FCL models occurs at significantly lower n_u values for 200 clauses, it is barely affected for 800 clauses. We explain this below using our bounds from the previous section.

The existential-universal phase transition also shows another common trait of phase transitions – an easy-hard-easy pattern centered around the critical value. This trend is show in Figure 5.2 where we use time as a proxy for the solution complexity of a given instance. Instances with a number of universal variables either slightly below or above the critical value are easy to solve, but instances at the critical value show an exponential blow-up in required computation time.

An issue with determining the exact location of the phase transitions in our experiments is the coarseness of the varying parameter, n_u. Because solving QBF is much more resource intensive than solving SAT, we were generally able to solve instances with no more than 100 variables consistently. In [17, 18] the authors are able to solve instances with a much higher number of variables because they use shorter clauses and very few universal variables. Because our experiments use so few variables, increasing n_u by 1 causes a pertubation of several percent in the input size. We compute the approximate phase transition locations

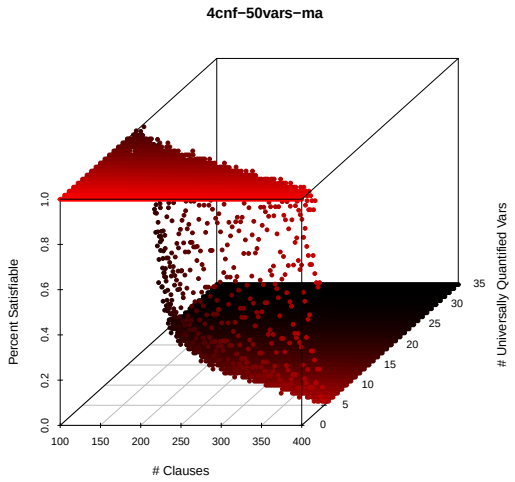


Figure 5.3: Plot showing the effect of clause-to-variable ratio and fraction of universal variables on QBF satisfiability.

shown in table 5.1 by interpolating between the two data points straddling the 50%-satisfiable mark.

5.2.2 Interpretation of Results

Although we have presented strong experimental evidence demonstrating the existence of the existential-universal phase transition, there are several points remaining to be addressed:

a. The reason for the location of the phase transition.

b. The reason for the sharpness of the phase transition.

c. The relationship of the location of the phase transition to other parameters.

In addition to the upper bound on the location of the existential-universal phase transition that we measure from c-conflicts, we also obtain an easy upper bound when a QBF's CNF matrix has above the critical number of clauses for the corresponding SAT problem. This is because changing the quantification of variables from existential to universal may make a satisfiable formula unsatisfiable, but never the other way around.

We can also use this idea to get a lower bound on the phase transition's location by computing k_e,

Problem			Number of clauses m						
Model	n	k	100	150	200	250	300	350	400
ma	50	4	21.0	16.5	13.3	10.9	8.3	6.1	4.0
Model	n	k	200	300	400	500	600	700	800
ma	100	4	39.3	30.4	25.3	20.7	15.9	11.9	7.8
ma	50	5	21.1	18.1	15.4	12.8	10.6	8.4	5.9
fcl	50	5	17.5	15.5	14.0	12.3	10.3	8.3	5.8
pr	50	5	16.0	14.6	13.4	11.8	10.3	8.0	5.8

Table 5.1: This table shows the estimated of the phase transition in random formulas generated in a number of different models and parameter settings. Tested models include pure (pr), Cadoli fixed clause length (FCL), and Gent-Walsh model A (ma). All experiments were performed 50 times with each combination of parameters n_u and m. The entries give the expected location of the phase transition in terms of the number of universal variables.

the expected number of existential variables per clause. The idea is that if the existential variables in the formula alone can satisfy the formula, then the larger formula is satisfiable. In that case we may think of the universal-existential phase transition as an extension of the clause-to-variable ratio transition for k_e-SAT.

Key insight in understanding the SAT clause-to-variable ratio phase transition comes from the linear relationship between number of clauses and number of variables in a SAT formula, and that this relationship scales to different problem sizes. The projection of the phase transition shown in figure 5.3 suggests that there is no linear relationship between the clause-to-variable ratio m/n and the fraction of universally quantified variables n_u/n, and hence none between m and n_u.

Attempting to address each of these issues further will be an important goal for future work.

Chapter 6

Bounds on Trivially False Formula Generation

The model used for studying random QBF is crucial. Three of the first papers on random QBF [11, 25, 12] motivate and introduce new random models as a key part of their work. The importance of using a robust random moBecause the entire inclusion-exclusion tree with no pruning has depth m, the m bound is trivial. del motivates the bounds given in this section, which are relevant for the class of random model that we use in our experiments.

We call a random QBF formula **pure** if its CNF matrix is generated without regard to how each variable is quantified. Note that this is the same as the Cadoli model [11], except in that model clauses with no existential literals are discarded and regenerated. Gent-Walsh Model A [25] discards clauses with zero or one existential literals. We will call a random QBF generation model where the number of universal variables per clause may vary a **global** random model.

Algorithm 5 shows a basic algorithm for generating qunatified Boolean formulas whose kernels are in conjunctive normal form (QCNFs).

Pure random QBF has a simple definition, which makes it conducive to formal analysis. However, it is flawed in the sense that a single clause with no existential literals trivially falsifies a formula. Gent and Walsh [25] note that allowing clauses with a single existential literal may also cause trivial falsification. In that case, assume we have a pair of clauses, one containing an existentially quantified literal, and the other containing its negation. When all of the remaining literals in both clauses are universally quantified, with none in common between the two clauses, this pair of clauses acts like a pair of conflicting unit clauses in

Algorithm 5: Algorithm for generating pure random QCNF formulas

Input: Vector giving number of variables per quantifier block $n_1, \ldots, n_r$ with $n = \sum_i n_i$, number of clauses m, length of clauses k
Output: A quantified CNF formula φ
$\varphi := \emptyset$;
while $m > 0$ **do**
  $c = \text{choose}(n, k)$; `/* Choose` k `of the` n `total variables uniformly at random */`
  $\text{negVars}(c, 1/2)$; `/* Negate variables in` c `with probability 1/2 */`
  $\varphi := \varphi \cup c$;
  $m := m - 1$;
return φ;

SAT and resolves to false. The observation by Gent and Walsh motivates this section of the thesis. We derive precise bounds for the likelihood of a vast generalization of these local conflicts arising from instances generated using a pure random model.

The birthday paradox [51] asks what the fewest number of people in a room must be for there to be an over 50% chance for two people to have the same birthday. It is perhaps surprising that only 23 people are necessary for this to occur. Furthermore, if we ask how the number of people present necessary would scale based on a year with n days, the answer is $O(\sqrt{n})$. Noting the similarity between the probability of generating pairs of conflicting literals and the birthday paradox, Gent and Walsh write, "As there are only $2n$ different unit clauses, we expect to generate complementary unit clauses when [the number of generated clauses is] $\approx \sqrt{2n}$, just as we expect to find two people with the same birthday in a group of about $\sqrt{365}$ people."

Although computing exact bounds for this simple "birthday paradox" is straightforward, more complicated variants are difficult. Inspired by this connection between generating trivially false formulas and the birthday paradox and Schmuland's demonstration of how to use Poisson approximations to solve harder forms of the birthday paradox [42], we vastly generalize Gent and Walsh's observation about generating local conflicts, and present rigorous bounds on the probability with which they occur.

6.1 Definitions

We introduce two probability distributions which will be useful [23]:

Definition 14. *A **Poisson distribution** $f(x; \lambda)$ gives the approximate probability that x occurrences of an*

event E will happen. The input parameter $\lambda = N \cdot p$ equals the expected number of events, where N is the number of events and p is the probability of an event occurring.

$$\Pr(N_E = x) = f(x; \lambda) = \frac{\lambda^x \cdot e^{-\lambda}}{x!} \tag{6.1}$$

Note that a Poisson distribution gives the *approximate* probability that E occurs x times, denoted $N_E = x$. In practice the approximation is good for large N, small p, and moderate λ [39]. A standard way to compute the probability that E occurs at least once is to compute the probability that E occurs zero times, and complement it.

$$\Pr(x \in [1, \infty)) = 1 - \Pr(0; \lambda) = 1 - \frac{\lambda^0 \cdot e^{-\lambda}}{0!} = 1 - e^{-\lambda}$$

Definition 15. *Suppose we have a set of elements J from which we draw elements uniformly at random without replacement, and a subset of elements $X \subseteq J$ which if drawn are "successes". A* ***hypergeometric distribution*** *$H(x; J, X, j)$ gives the probability of x successes when drawing j elements without replacement from J.*

$$\Pr(\text{x successes}) = H(x; J, X, j) = \frac{\binom{X}{x}\binom{J-X}{j-x}}{\binom{J}{j}} \tag{6.2}$$

Note that when $x = 0$ or $x = j$ one of the terms in the numerator drops out.

We will use hypergeometric distributions to represent the probability of selecting a certain number c of existential variables when generating a clause:

$$\frac{\binom{n_e}{c}\binom{n_u}{k-c}}{\binom{n}{k}} \tag{6.3}$$

Given a set of c variables, there are 2^c different sets of literals of length c containing the c variables or their negations. We call each of these sets a **negation assignment**.

Example 5. *The set of variables $\{x_1, x_2\}$ has four negation assignments: $\{x_1, x_2\}, \{\neg x_1, x_2\}, \{x_1, \neg x_2\}, \{\neg x_1, \neg x_2\}$.*

$$
\begin{array}{ccc}
x_{e,1} \vee x_{e,2} \vee x_{e,3} \cdots \vee x_{e,c} & \vee & \ell_{u,1} \vee \cdots \vee \ell_{u,k-c} \\
\neg x_{e,1} \vee x_{e,2} \vee x_{e,3} \cdots \vee x_{e,c} & \vee & \ell_{u,k-c+1} \vee \cdots \vee \ell_{u,2(k-c)} \\
x_{e,1} \vee \neg x_{e,2} \vee x_{e,3} \cdots \vee x_{e,c} & \vee & \ell_{u,2(k-c)+1} \vee \cdots \vee \ell_{u,3(k-c)} \\
\neg x_{e,1} \vee \neg x_{e,2} \vee x_{e,3} \cdots \vee x_{e,c} & \vee & \ell_{u,3(k-c)+1} \vee \cdots \vee \ell_{u,4(k-c)} \\
 & \cdots & \\
\underbrace{\neg x_{e,1} \vee \neg x_{e,2} \vee \neg x_{e,3} \cdots \vee \neg x_{e,c}}_{\text{Existential variables in the } c\text{-conflict}} & \vee & \underbrace{\ell_{u,(2^c-1)(k-c)} \vee \cdots \vee \ell_{u,2^c(k-c)}}_{\text{Universal literals}}
\end{array}
$$

Figure 6.1: 2^c clauses involved in a c-conflict in a QBF with clauses of length k. x_e denotes an existentially quantified variable while ℓ_u denotes a universally quantified literal. The same c existential variables are used in each clause. The universal literals may not be distinct but reuses have the same polarity.

Definition 16. *A c-conflict for $c \geq 0$ over c existential variables $X = \{x_{i_1}, \ldots, x_{i_c}\}$ is a set S of 2^c clauses where:*

(1) Each clause in S contains a different one of the 2^c negation assignments to the variables in X.

(2) If a clause $C \in S$ contains existential variables then they are in S.

(3) Universal variables occurring in more than one clause have the same polarity.

The third condition generalizes an observation for $c = 1$ in [25], which requires that a universal variable occur in at most one of the 2^c c-conflict clauses. c-conflicts are a special class of min-UNSAT-cores. Recall that a min-UNSAT-core is an unsatisfiable set of clauses such that if any one clause is removed then the set of remaining clauses becomes satisfiable.

Example 6.

In the clauses shown in the example below existential variables are given first, followed by universal variables.

(1) In the QBF formula

$$\forall x_1 \forall x_2 \forall x_3 \exists x_4 \exists x_5 \quad (x_1 \vee \neg x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_2)$$

the clause $(x_1 \vee \neg x_2 \vee x_3)$ forms a 0-conflict.

(2) In the QBF formula

$$\forall x_1 \forall x_2 \forall x_3 \exists x_4 \exists x_5 \quad (x_1 \vee x_3 \vee x_5) \wedge (x_4 \vee x_1 \vee x_3) \wedge (\neg x_4 \vee x_2 \vee x_3)$$

the clauses $(x_4 \vee x_1 \vee x_3)$ *and* $(\neg x_4 \vee x_2 \vee x_3)$ *form a* 1*-conflict.*

(3) In the SAT formula

$$\exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 \quad (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_4 \vee x_5) \wedge (\neg x_4 \vee x_5) \wedge (x_4 \vee \neg x_5) \wedge (\neg x_4 \vee \neg x_5)$$

the clauses $(x_4 \vee x_5)$, $(\neg x_4 \vee x_5)$, $(x_4 \vee \neg x_5)$, $(\neg x_4 \vee \neg x_5)$ *form a* 2*-conflict.*

Lemma 4. *Any QBF formula* φ *containing a* c*-conflict* S *for* $c \geq 0$ *is unsatisfiable.*

Proof. Because any common universally quantified variables between the clauses in S have the same polarity, there exists a partial assignment to the universal variables which falsifies all universal literals occurring in clauses in S. Furthermore, any assignment to the c existential variables in S falsifies the remaining variables in one of the 2^c clauses, meaning that the formula is falsified. □

Several well-known types of UNSAT cores fall under the umbrella of c-conflicts.

a. When $c = 0$ in a QBF formula, S represents a single clause with all universally quantified variables.

b. When $c = 1$ in a SAT formula, S represents a pair of conflicting unit clauses

c. When $c = 1$ in a QBF formula, S represents a "flawed" pair of clauses as described in [25]

6.2 Local Conflict Bounds

The main contribution of this section is to give bounds for how likely c-conflicts are to occur, assuming that clauses for random QBF formulas are sampled with replacement.

Lemma 5. *The probability* p *of a* k*-conflict occurring in an* (n, m, k)*-SAT formula is approximately* $p \sim 1 - e^{-\lambda}$*, where* $\lambda = \binom{m}{2^k} \frac{(2^k-1)!}{(2^k\binom{n}{k})^{(2^k-1)}}$

Proof. Because we want to compute the probability of one or more k-conflicts occurring, we complement the probability that no conflicts occur, giving $p = 1 - e^{-\lambda}$.

After selecting any given first clause we must pick the remaining $2^k - 1$ clauses, each with probability $1/2^k\binom{n}{k}$. There are $(2^k - 1)!$ orders in which to do so.

Then because there are $\binom{m}{2^k}$ possible k-conflicts in a given formula, by the linearity of expectation we get

$$\lambda = \binom{m}{2^k} \frac{(2^k - 1)!}{(2^k\binom{n}{k})^{(2^k-1)}}$$

□

We now give a more general version for QBF. The third condition for c-conflicts stipulates that any universal variables occuring in more than one clause in the c-conflict must have the same negation. Becuase counting the number of ways that this can occur is difficult, we give upper and lower bounds instead. The upper bound corresponds to picking 2^c clauses whose existential variables form a c-conflict, but have arbitrary universal literals. The lower bound corresponds to no repeated universal variables between clauses.

Theorem 8. *The probability p of a c-conflict occurring in a pure random (n, m, k, n_u, n_e)-QBF formula is $1 - e^{-\lambda_1} \leq p \leq 1 - e^{-\lambda_2}$, where*

$$\lambda_1 = \tbinom{m}{2^c} \frac{\binom{n_e}{c}\binom{n_u}{k-c}}{\binom{n}{k}} \prod_{j=1}^{2^c-1} \frac{(2^c - j) \cdot \binom{n_u - j(k-c)}{k-c}}{2^c\binom{n}{k}}$$
$$\lambda_2 = \tbinom{m}{2^c} \frac{\binom{n_e}{c}\binom{n_u}{k-c}}{\binom{n}{k}} \prod_{j=1}^{2^c-1} \frac{(2^c - j) \cdot \binom{n_u}{k-c}}{2^c\binom{n}{k}}$$

Proof. We give the proof for the lower bound.

We will compute the probability of picking an initial clause with exactly c existential variables, followed by $2^c - 1$ additional clauses containing the other negations of the c variables that satisfy conditions (2) and (3).

Let p_n denote the probability that exactly c existential variables are chosen in a clause.

$$p_n = \frac{\binom{n_e}{c}\binom{n_u}{k-c}}{\binom{n}{k}}$$

Let p_e denote the probability that c given existential variables match the seed clause, with a different negation from the previous j clauses.

$$p_e(j) = \frac{2^c - j}{2^c \binom{n_e}{c}}$$

Let p_a denote the probability that $k - c$ given universal variables have not been selected in the j previous clauses.

$$p_a(j) = \frac{\binom{n_u - j(k-c)}{k-c}}{\binom{n_u}{k-c}}$$

Then the probability q of a given set of c clauses being a c-conflict is

$$q = p_n \prod_{j=1}^{2^c - 1} p_n p_e(j) p_a(j)$$

The p_n term on the outside comes from the seed clause for which only the number of existential variables matters. We then have

$$\begin{aligned} q &= \frac{\binom{n_e}{c}\binom{n_u}{k-c}}{\binom{n}{k}} \prod_{j=1}^{2^c-1} \frac{\binom{n_e}{c}\binom{n_u}{k-c}}{\binom{n}{k}} \cdot \frac{2^c - j}{2^c \binom{n_e}{c}} \cdot \frac{\binom{n_u - j(k-c)}{k-c}}{\binom{n_u}{k-c}} \\ &= \frac{\binom{n_e}{c}\binom{n_u}{k-c}}{\binom{n}{k}} \prod_{j=1}^{2^c-1} \frac{(2^c - j) \cdot \binom{n_u - j(k-c)}{k-c}}{2^c \binom{n}{k}} \end{aligned}$$

Because there are $\binom{m}{2^c}$ possible c-conflicts, the result follows.

The proof for the upper bound is similar.

□

Remark 3.

- *When $n_u = 0$ Theorem 8 reduces to Lemma 6.2.*
- *When $c = 0$ the upper and lower bounds in Theorem 8 coincide.*
- *These bounds hold for arbitrary QBF formulas. They are not dependent on the number of quantifier alternations, only on whether variables are existential or universal.*

We may also compute the exact probability that a formula has all universal literals without using a Poisson approximation:

n	k	n_u	c	200 cl		400 cl		600 cl		800 cl	
				lb	ub	lb	ub	lb	ub	lb	ub
50	5	10	0	0.024	0.024	0.046	0.046	0.069	0.069	0.091	0.091
50	5	10	1	0.0	0.004	0.001	0.016	0.003	0.035	0.004	0.061
50	5	15	0	0.247	0.247	0.433	0.433	0.573	0.573	0.678	0.678
50	5	15	1	0.034	0.135	0.131	0.44	0.271	0.729	0.429	0.902
50	5	20	0	0.769	0.769	0.946	0.946	0.988	0.988	0.997	0.997
50	5	20	1	0.444	0.79	0.905	0.998	0.995	1.0	1.0	1.0

Table 6.1: Lower and upper bounds on 0- and 1-conflict probabilities for $m = 200, 400, 600, 800$.

These bounds give us a framework for computing the probability of a c-conflict, a class of local conflicts which falsify clauses. In practice this formula will be useful for computing bounds for small values of c, which we will do in the experimental section to compare global random models.

Chapter 7

Conclusion and Future Work

This thesis has discussed new work in several areas related to the satisfiability problem and its extensions.

We have introduced a subsumption-based improvement to the inclusion-exclusion based algorithm for model counting which allows large subtrees to be pruned from the search space. By representing an inclusion-exclusion sum as a tree we gain a natural characterization of subsumption, and achieve an extension of elementary Bonferroni inequalities. Both of these contributions apply to general inclusion-exclusion sums. There are currently two primary outstanding goals for this work. We would like to perform average-case analysis on subsumptions. In terms of applications, we would like to find problem domains in which both our results and the inclusion-exclusion counting method in general are particularly effective.

Another section of work discussed random QBF. In this work, we described a phase transition resulting from alternating the fraction of existential versus universal variables in a formula, confirming experimentally the existence of a sharp satisfiability threshold. One aim for future work is to tighten theoretical bounds on where these phase transitions occur.

We also gave bounds on the probability of c-conflicts, a class of min-UNSAT-core, occurring in random SAT and QBF formulae given a number of input parameters. We applied these bounds to determining the location of phase transitions, and to identify when a given random model is applicable.

Bibliography

[1] D. Achlioptas. Random satisfiability. In Biere et al. [7], pages 245–270.

[2] D. Achlioptas, L. M. Kirousis, E. Kranakis, and D. Krizanc. Rigorous results for random (2+p)-SAT. Theor. Comput. Sci., 265(1-2):109–129, 2001.

[3] E. Allender, M. Bauland, N. Immerman, H. Schnoor, and H. Vollmer. The complexity of satisfiability problems: Refining schaefer's theorem. J. Comput. Syst. Sci., 75(4):245–254, 2009.

[4] S. Arora and B. Barak. Computational Complexity - A Modern Approach. Cambridge University Press, 2009.

[5] H. Bennett. The existential-universal phase transition in random qbf. Technical report, Department of Computer Science, University of Colorado, Boulder, Colorado, March 2012.

[6] H. Bennett and S. Sankaranarayanan. Model counting using the inclusion-exclusion principle. In K. A. Sakallah and L. Simon, editors, SAT, volume 6695 of Lecture Notes in Computer Science, pages 362–363. Springer, 2011.

[7] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. Handbook of Satisfiability, volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press, 2009.

[8] E. Birnbaum and E. L. Lozinskii. The good old davis-putnam procedure helps counting models. J. Artif. Intell. Res. (JAIR), 10:457–477, 1999.

[9] A. R. Bradley and Z. Manna. The calculus of computation - decision procedures with applications to verification. Springer, 2007.

[10] H. K. Büning and U. Bubeck. Theory of quantified boolean formulas. In Biere et al. [7], pages 735–760.

[11] M. Cadoli, A. Giovanardi, and M. Schaerf. Experimental Analysis of the Computational Cost of Evaluating Quantified Boolean Formulae. In M. Lenzerini, editor, AI*IA, volume 1321 of Lecture Notes in Computer Science, pages 207–218. Springer, 1997.

[12] H. Chen and Y. Interian. A Model for Generating Random Quantified Boolean Formulas. In L. P. Kaelbling and A. Saffiotti, editors, IJCAI, pages 66–71. Professional Book Center, 2005.

[13] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. ACM Trans. Program. Lang. Syst., 16(5):1512–1542, 1994.

[14] S. Cook. `http://www.claymath.org/millennium/P_vs_NP/pvsnp.pdf`.

[15] S. A. Cook. The complexity of theorem-proving procedures. In M. A. Harrison, R. B. Banerji, and J. D. Ullman, editors, STOC, pages 151–158. ACM, 1971.

[16] N. Creignou, H. Daudé, and U. Egly. Phase Transition for Random Quantified XOR-Formulas. J. Artif. Intell. Res. (JAIR), 29:1–18, 2007.

[17] N. Creignou, H. Daudé, U. Egly, and R. Rossignol. New Results on the Phase Transition for Random Quantified Boolean Formulas. In H. K. Büning and X. Zhao, editors, SAT, volume 4996 of Lecture Notes in Computer Science, pages 34–47. Springer, 2008.

[18] N. Creignou, H. Daudé, U. Egly, and R. Rossignol. (1, 2)-QSAT: A Good Candidate for Understanding Phase Transitions Mechanisms. In O. Kullmann, editor, SAT, volume 5584 of Lecture Notes in Computer Science, pages 363–376. Springer, 2009.

[19] A. Darwiche and K. Pipatsrisawat. Complete algorithms. In Biere et al. [7], pages 99–130.

[20] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. Commun. ACM, 5(7):394–397, 1962.

[21] M. Davis and H. Putnam. A computing procedure for quantification theory. J. ACM, 7(3):201–215, 1960.

[22] K. Dohmen. Improved Bonferroni Inequalities via Abstract Tubes: Inequalities and identities of the Inclusion-Exclusion Type. Lecture Notes in Mathematics. Springer–Verlag, 2003.

[23] M. Evans, N. Hastings, and B. Peacock. Statistical Distributions. A Wiley-Interscience publication. J. Wiley, 1993.

[24] J. Galambos. Bonferroni inequalities. The Annals of Probability, 5(4):pp. 577–581, 1977.

[25] I. P. Gent and T. Walsh. Beyond NP: the QSAT phase transition. In J. Hendler and D. Subramanian, editors, AAAI/IAAI, pages 648–653. AAAI Press / The MIT Press, 1999.

[26] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In Biere et al. [7], pages 633–654.

[27] K. Iwama. CNF-satisfiability test by counting and polynomial average time. SIAM Journal on Computing, 18(2):385–391, 1989.

[28] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, Complexity of Computer Computations, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[29] H. A. Kautz, A. Sabharwal, and B. Selman. Incomplete algorithms. In Biere et al. [7], pages 185–203.

[30] L. A. Levin. Universal enumeration problems. Problemy Peredači Informacii, 9(3):115–116, 1973.

[31] C. M. Li and F. Manyà. Maxsat, hard and soft constraints. In Biere et al. [7], pages 613–631.

[32] N. Linial and N. Nisan. Approximate inclusion-exclusion. Combinatorica, 10(4):349–365, 1990.

[33] F. Lonsing and A. Biere. `http://fmv.jku.at/depqbf`.

[34] F. Lonsing and A. Biere. Integrating Dependency Schemes in Search-Based QBF Solvers. In Strichman and Szeider [48], pages 158–171.

[35] E. L. Lozinskii. Counting propositional models. Information Processing Letters, 41:327–332, 1992.

[36] C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. The Seventh QBF Solvers Evaluation (QBFEVAL'10). In Strichman and Szeider [48], pages 237–250.

[37] S. D. Prestwich. Cnf encodings. In Biere et al. [7], pages 75–97.

[38] J. Rintanen. Improvements to the evaluation of quantified boolean formulae. In T. Dean, editor, IJCAI, pages 1192–1197. Morgan Kaufmann, 1999.

[39] S. Ross. A First Course in Probability. Prentice Hall, 2002.

[40] H. Ryser. Combinatorial mathematics. Carus mathematical monographs. Mathematical Association of America; distributed by Wiley New York, 1963.

[41] S. Sankaranarayanan and H. Bennett. Counting problems and the inclusion-exclusion principle. Technical report, Department of Computer Science, University of Colorado, Boulder, Colorado, March 2012.

[42] B. Schmuland. http://math.stackexchange.com/questions/25876/probability-of-3-people-in-a-room-of-30-having-the-same-birthday/25878#25878.

[43] B. Selman and H. A. Kautz. Domain-independent extensions to gsat: Solving large structured satisfiability problems. In IJCAI, pages 290–295, 1993.

[44] B. Selman, H. J. Levesque, and D. G. Mitchell. A new method for solving hard satisfiability problems. In W. R. Swartout, editor, AAAI, pages 440–446. AAAI Press / The MIT Press, 1992.

[45] B. Selman, D. G. Mitchell, and H. J. Levesque. Generating Hard Satisfiability Problems. Artif. Intell., 81(1-2):17–29, 1996.

[46] J. P. M. Silva, I. Lynce, and S. Malik. Conflict-driven clause learning sat solvers. In Biere et al. [7], pages 131–153.

[47] J. P. M. Silva and K. A. Sakallah. Grasp - a new search algorithm for satisfiability. In ICCAD, pages 220–227, 1996.

[48] O. Strichman and S. Szeider, editors. Theory and Applications of Satisfiability Testing - SAT 2010, 13th International C volume 6175 of Lecture Notes in Computer Science. Springer, 2010.

[49] G. S. Tseitin. On the complexity of derivation in propositional calculus. Studies in constructive mathematics and mathematical logic, 8(115-125):234–259, 1968.

[50] L. G. Valiant. The complexity of computing the permanent. Theor. Comput. Sci., 8:189–201, 1979.

[51] E. Weisstein. http://mathworld.wolfram.com/BirthdayProblem.html.