Chapter 1

INTRODUCTION

Propositional proof complexity is the study of the structure of proofs of mathematical statements expressed in a propositional or Boolean form. This thesis explores the algorithmic applications of this field, focusing on exact and approximation algorithms for combinatorial graph problems as well as on propositional reasoning systems used frequently in artificial intelligence and formal verification.

Science relies heavily on modeling systems and providing proofs of properties of interest. This motivates the study of the nature of proofs themselves and the use of computers to automatically generate them when possible. What do mathematical proofs look like? How are problems of interest represented in formats suitable for proving properties about them? What are the computational challenges involved in finding such proofs? Do short proofs even always exist when one's reasoning abilities are limited? How can our understanding of proof structures be used to improve combinatorial search algorithms? This work is a step towards answering these very natural and influential questions.

We concentrate on propositional statements. These are logical statements over a set of variables each of which can be either TRUE or FALSE. Suppose a propositional statement S is a tautology, i.e., it is TRUE for all possible combinations of values of the underlying variables. S is alternatively referred to as being valid. How can one provide a proof of the validity of S? Computationally, what does it mean to have such a proof?

Cook and Reckhow [39] introduced the formal notion of a proof system in order to study mathematical proofs from a computational perspective. They defined a propositional proof system to be an efficient algorithm A that takes as input a propositional statement S and a purported proof π of its validity in a certain pre-specified format. The crucial property of A is that for all invalid statements S, it rejects the pair (S, π) for all π, and for all valid statements S, it accepts the pair (S, π) for some proof π. This notion of proof systems can be alternatively formulated in terms of unsatisfiable formulas — those that are FALSE for all assignments to the variables.

They further observed that if there is no propositional proof system that admits short (polynomial in size) proofs of validity of all tautologies, i.e., if there exist computationally hard tautologies for every propositional proof system, then the complexity classes NP and co-NP are different, and hence P $\neq$ NP. This observation makes finding

tautological formulas (equivalently, unsatisfiable formulas) that are computationally difficult for various proof systems one of the central tasks of proof complexity research, with far reaching consequences to complexity theory and Computer Science in general. These hard formulas naturally yield a hierarchy of proof systems based on the sizes of proofs they admit. Tremendous amount of research has gone into understanding this hierarchical structure. A slightly outdated but interesting survey by Beame and Pitassi [20] nicely summarizes many of the results obtained along these lines in the last two decades.

As the most theoretical part of this work, we explore the proof complexity of a large class of structured formulas based on certain NP-complete combinatorial search problems on graphs. We prove that these formulas are hard for a very commonly studied proof system known as resolution. Algorithmically, this implies lower bounds on the running time of a class of algorithms for solving these problems exactly as well as approximately. These results complement the known approximation hardness results for these problems which build on the relatively sophisticated and recent machinery of probabilistically checkable proofs (PCPs) [9, 8].

On the more applied side which relates to automated reasoning systems, our focus is on propositional satisfiability algorithms, or *SAT solvers* as they are commonly known. Given a propositional formula F as input, the task of a SAT solver is to either find a variable assignment that satisfies F or declare F to be unsatisfiable. SAT solvers have evolved tremendously in the last decade, becoming general-purpose professional tools in areas as diverse as hardware verification [24, 112], automatic test pattern generation [74, 104], planning [70], scheduling [59], and group theory [114]. Annual SAT competitions have led to dozens of clever implementations [e.g. 13, 84, 113, 88, 57, 64], exploration of many new techniques [e.g. 80, 58, 84, 88], and creation of extensive benchmarks [63].

Researchers involved in the development and implementation of such solvers tackle the same underlying problem as those who work on propositional proof complexity — the propositional satisfiability problem. They have typically focused on specific techniques that are efficient in real world domains and have arguably interacted with the proof complexity community in a somewhat superficial way. One must grant that relatively straightforward correlations such as the equivalence between the basic backtracking SAT procedure called DPLL and a simple proof system called tree-like resolution as well as the relationship between more advanced SAT solvers using inequalities with a proof system called cutting planes have been regarded as common knowledge. However, a more in-depth connection between the ideas developed by the two communities is lacking. They have traditionally used very different approaches, rarely letting the concepts developed by one influence the techniques or focus of the other in any significant way.

The aim of the latter half of this work is to advance both of these fields – satisfiability algorithms and proof complexity – through independent work as well as

cross-fertilization of ideas. It explores the inherent strength of various propositional reasoning systems and uses theoretical insights to comprehend and improve the most widely implemented class of complete SAT solvers. It tries to achieve a balance between providing rigorous formal analysis and building systems to evaluate concepts. The results are a blend of theoretical complexity bounds, some of which explain observed behavior, and systems such as SymChaff built to demonstrate performance gains on real world problems.

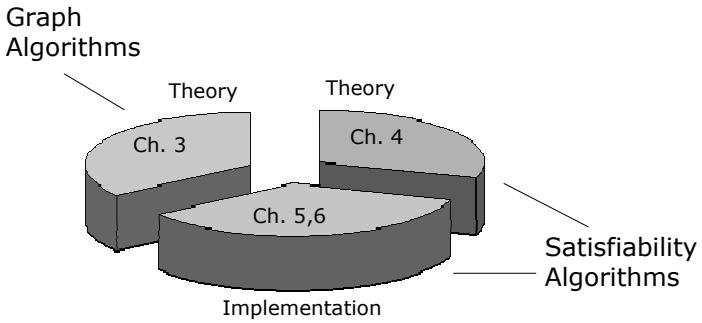

Figure 1.1: The three applications of proof complexity explored in this work

This thesis contains a broad spectrum of work from answering purely theoretical questions on one end to building systems that change the way we solve problems of human interest on the other (see Figure 1.1). A constant flow of ideas between the two extremes has far reaching benefits, as does research that addresses the middle ground. Realizing inherent strengths and limitations directs one's focus on concepts critical to good implementations and is the foundation of better systems. The work we will present in Chapters 5 and 6 is a testimony to this. On the other hand, systems whose development is motivated by real world applications provide a new avenue for utilizing the analysis techniques developed theoretically. The work in Chapter 4 stands in support of this.

The quest for short proofs continues amongst researchers for several reasons. One is pure mathematical curiosity in search for simplicity and elegance. Another is the far reaching effect on complexity theory that existence of short proofs might have. Yet another is computational efficiency demanded by the numerous real world applications that have come to depend on SAT solvers. The purpose of this thesis is to bring these motivations together in order to attain a better theoretical as well as practical understanding of the algorithmic applications of propositional proof complexity.

1.1 Theoretical Contributions

From a proof complexity perspective, the focus of this work is on the proof system called resolution. It is a very simple system with only one rule which applies to dis-

junctions of propositional variables and their negations: (a OR b) and ((NOT a) OR c) together imply (b OR c). Repeated application of this rule suffices to derive an empty disjunction if and only if the initial formula is unsatisfiable; such a derivation serves as a proof of unsatisfiability of the formula.

1.1.1 The Resolution Complexity of Structured Problems

In Chapter 3 we combine combinatorial and probabilistic techniques to show that propositional formulations of the membership in co-NP of almost all instances of some interesting co-NP complete graph problems, namely complements of Independent Set, Clique, and Vertex Cover, unconditionally require exponential size resolution proofs.

Resolution, although powerful enough to encompass most of the complete SAT solvers known today, does not admit short proofs of even simple counting-based formulas, most notably those encoding the pigeonhole principle: there is no one-one mapping from n pigeons to $(n-1)$ holes. Progress in the last decade has shown that almost all randomly chosen formulas are also difficult for resolution. However these random formulas are completely unstructured, unlike most real world problems. Are there large classes of hard but structured formulas that may be more representative of the instances encountered in practice? We give an affirmative answer to this.

More formally, we consider the problem of providing a resolution proof of the statement that a given graph with n vertices and average degree Δ does not contain an independent set of size k. For graphs chosen randomly from the distribution $\mathbb{G}(n,p)$, where $\Delta = np$, we show that such proofs asymptotically almost surely require size roughly exponential in n/Δ^6 for $k \leq n/3$. This, in particular, implies a $2^{\Omega(n)}$ lower bound for constant degree graphs. We deduce similar complexity results for the related vertex cover problem on random graphs.

This work was done jointly with Paul Beame and Russell Impagliazzo. It has been published in the proceedings of the 16^{th} Annual Conference on Computational Complexity (CCC), 2001 [16] and is currently under review for the journal Computational Complexity.

1.1.2 Hardness of Approximation

The complexity results described above translate into exponential lower bounds on the running time of a class of search algorithms for finding a maximum independent set or a minimum vertex cover. In fact, all resolution proofs studied in the above work turn out to be of exponential size even when one attempts to prove the non-existence of a much larger independent set or clique than the largest one, or a much smaller vertex cover than the smallest one. This in turn implies that a natural class of approximate optimization algorithms for these problems performs poorly on almost all problem instances.

In particular, we show that there is no resolution-based algorithm for approximating the maximum independent set size within a factor of $\Delta/(6 \log \Delta)$ or the minimum vertex cover size within a factor of 3/2. This latter result contrasts well with the commonly used factor of 2 approximation algorithm for the vertex cover problem.

We also give relatively simple algorithmic upper bounds for these problems and show them to be tight for the class of exhaustive backtracking techniques.

This work was done jointly with Paul Beame and Russell Impagliazzo and is currently under review for the journal Computational Complexity.

1.2 Proof Systems Underlying SAT Solvers

Our next contribution is in providing a formal understanding of the numerous satisfiability algorithms developed in the last decade. It is common knowledge that most of today's complete SAT solvers implement a subset of the resolution proof system. However, where exactly do they fit in the proof system hierarchy? How do they compare to refinements of resolution such as regular resolution? Why do certain techniques result in huge performance gains in practice while others have limited benefits? This work provides the first answers to some of these questions.

1.2.1 Clause Learning, Restarts, and Resolution

Chapter 4 develops an intuitive but formal understanding of the behavior of SAT solvers in practice. Using a new framework for rigorous analysis of techniques frequently used in solver implementations, we show that the use of a critical technique called clause learning makes a solver more powerful than many refinements of resolution, and with yet another technique – arbitrary restarts – and a slight modification, makes it as strong as resolution in its full generality.

Conflict-driven clause learning works by caching and reusing reasons of failure on subproblems. Random restarts help a solver avoid unnecessarily exploring a potentially huge but uninteresting search space as a result of a bad decision. These are two of the most important ideas that have lifted the scope of modern SAT solvers from experimental toy problems to large instances taken from real world challenges. Despite overwhelming empirical evidence, not much was known of the ultimate strengths and weaknesses of the two. We provide a formal explanation of the observed exponential speedups seen when using these techniques.

This presents the first precise characterization of clause learning as a proof system and begins the task of understanding its power by relating it to resolution. In particular, we show that with a new learning scheme called FirstNewCut, clause learning can provide exponentially shorter proofs than any proper refinement of general resolution satisfying a natural self-reduction property. These include regular and ordered resolution, which are already known to be much stronger than the ordinary DPLL procedure which captures most of the SAT solvers that do not incorporate clause

learning. We also show that a slight variant of clause learning with unlimited restarts is as powerful as general resolution itself.

This work was done jointly with Paul Beame and Henry Kautz. It has been published in the proceedings of the 18^{th} International Joint Conference on Artificial Intelligence (IJCAI), 2003 [18] as well as in the Journal of Artificial Intelligence Research (JAIR), 2004 [19].

1.3 Building Faster SAT Solvers

The input to almost all SAT solvers of today is a formula in conjunctive normal form (CNF), i.e., a conjunction of disjunctions of variables or their negations. This shallow representation results in several algorithmic and implementation-related advantages and can, in most cases, be made fairly compact with the use of additional auxiliary variables.

On the other hand, one can easily argue that most real world problem instances given as input to a SAT solver in the CNF form originate from a more structured, high level problem description known to the problem designer. For example, the underlying high level structured object could be a circuit, a planning graph, or a finite state model for which one wants to prove a desired property.

In Chapters 5 and 6 we show how we can gain substantially by providing the solver extra information that relates variables to the high level object they originated from. This information can, for instance, be in the form of an ordering of variables based on the original structure (e.g. a depth-first traversal of a planning graph) or their semantics (e.g. variable x represents loading truck k with block q). Of course, a well-translated formula itself contains all this information, but in a hidden way. We argue that making, as is commonly done, a solver rediscover the structure that was clear to start with is unnecessary, if not totally wasteful.

1.3.1 Variable Ordering Using Domain Knowledge

Motivated by the theoretical work in Chapter 4, we propose in Chapter 5 a novel way of exploiting the underlying problem structure to guide clause learning algorithms toward faster solutions. The key idea is to generate a branching sequence for a CNF formula encoding a problem from the high level description of the problem such as a graph or a planning language specification. We show that this leads to exponential empirical speed-ups on the class of grid and randomized pebbling problems, as well as substantial improvements on certain ordering formulas.

Conflict-directed clause learning is known to dramatically increase the effectiveness of branch-and-bound SAT solvers. However, to realize its full power, a clause learner needs to choose a favorable order of variables to branch on. While several proposed formula-based static and dynamic ordering schemes help, they face the daunting task of recovering relevant structural information from the flat CNF formula representation

consisting only of disjunctive clauses. We show that domain knowledge can allow one to obtain this critical ordering information with much more ease. This is a case in point for using domain knowledge to boost performance of a SAT solver rather than using the solver as a pure blackbox tool in the traditional manner.

This work was done jointly with Paul Beame and Henry Kautz. It has been published in the proceedings of the 6^{th} International Conference on Theory and Applications of Satisfiability Testing (SAT), 2003 [99] as well as in the Journal of Artificial Intelligence Research (JAIR), 2004 [19].

1.3.2 Utilizing Structural Symmetry in Domains

In Chapter 6 we present a novel low-overhead framework for encoding and utilizing structural symmetry in SAT solvers. We use the notion of complete multi-class symmetry and demonstrate the efficacy of our technique through a new solver called SymChaff which achieves exponential speedup by using simple tags in the specification of problems from both theory and practice.

A natural feature of many application domains in which SAT solvers are used is the presence of symmetry or equivalence with respect to the underlying objects, such as that amongst all trucks at a certain location in logistics planning and all wires connecting two switch boxes in an FPGA circuit. Many of these problems turn out to have a concise description in what is called many-sorted first order logic. This description can be easily specified by the problem designer and almost as easily inferred automatically. SymChaff, an extension of the popular SAT solver zChaff, uses information obtained from the "sorts" in the first order logic constraints to create symmetry sets that are used to partition variables into classes and to maintain and utilize symmetry information dynamically.

The challenge in such work is to do it in a way that pushes the underlying proof system up in the hierarchy without incurring the significant cost that typically comes from large search spaces associated with complex proof systems. While most of the current SAT solvers implement subsets of resolution, SymChaff brings it up closer to symmetric resolution, a proof system known to be exponentially stronger than resolution [111, 76]. More critically, it achieves this in a time- and space-efficient manner.

This work has been published in the proceedings of the 20^{th} National Conference on Artificial Intelligence (AAAI), 2005 [98].