Proof Complexity and Algorithmic Analysis

Rahul Santhanam
University of Edinburgh

Plan of the Talk

- SAT Algorithms and Proof Complexity: Preliminaries
- Paradigms for Branching Algorithms, and Satisfiable Hard Instances
- Hard Instances for State of the Art Algorithms
- Alternative Paradigms and Future Work

Plan of the Talk

- *SAT Algorithms and Proof Complexity: Preliminaries*
- Paradigms for Branching Algorithms, and Satisfiable Hard Instances
- Hard Instances for State of the Art Algorithms
- Alternative Paradigms and Future Work

Satisfiability Algorithms

- Fundamental question: What is the worst-case time complexity of SAT?
- If P ≠ NP, complexity is super-polynomial
- If ETH (Exponential Time Hypothesis) holds, complexity is $2^{\Omega(n)}$
- More precise answer depends on which variant of SAT we consider

Satisfiability Variants

- k-SAT: Satisfiability of k-CNFs
- CNF SAT
- AC0 SAT: Satisfiability of constant-depth circuits
- Formula SAT: Satisfiability of de Morgan formulae
- Circuit SAT

Notation

- m: input size; n: number of variables
- Brute-force algorithm runs in time 2^npoly(m)
- We are interested in algorithms running in time $2^{n-f(n)}$poly(m), for f(n) asymptotically as large as possible
  - We call f the *savings* of the algorithm

State of the Art

- 3-SAT: ~ 1.308^n [H11]
- k-SAT: Savings $\Omega(n/k)$ [PPSZ98, S99]
- CNF SAT: Savings $\Omega(n/\log(m))$ [S05]
- Formula SAT: Savings $\Omega(n^3/m^2)$ [S10]
- AC0 SAT: Savings $\Omega(n/(\log(m/n))^{d-1})$ [IMP12]
- Circuit SAT: No non-trivial savings known

Can We Do Better?

- Are the analyses of known algorithms tight?
- What are worst-case instances of these algorithms?

More Generally...

- Can we formulate classes of algorithms including all those known to be useful, and show fundamental barriers on the power of these algorithms?
- Which properties of instances guarantee hardness for known algorithms?

Motivation

- Doing better would not just give us improved algorithms but also possibly lower bounds using Williams connection [W10,W11]
- On the other hand, theory of hard instances could help in formulating benchmarks for SAT solvers
- Moreover, there is a heuristic connection between explicit hard instances for known algorithms and circuit lower bounds

The Role of Proof Complexity

- Propositional proof complexity studies the hardness of explicit tautologies for various propositional proof systems, i.e., non-deterministic algorithms
- Hardness against a non-deterministic algorithm implies hardness against any deterministic algorithm which operates by *fixing* the non-deterministic choices
- Well-known example: Connection between branching (DLL) algorithms and tree resolution

Branching (DLL) Algorithms

- Iteratively apply
  - A *strategy* to choose which variable of the formula to set next
  - A *policy* to decide which value to set it to
  - Simplify the resulting formula according to rules which preserve satisfiability (unit propagation, pure literal elimination, subsumption)
- If at any point the formula evaluates to false, *backtrack* by choosing a different value for most recently set variable

Branching Algorithms and Tree Resolution

- If a CNF formula ϕ is unsatisfiable, a tree resolution proof of unsatisfiability can be derived from the recursion tree of any det branching algorithm run on ϕ. Size of proof is essentially size of recursion tree
- Iterative procedure
  - Associate, with each leaf of the recursion tree, an unsatisfied clause C of ϕ
  - Suppose we branch on a variable x at a node v of the recursion tree, and suppose clauses C_1 and C_2 have been associated with children of v. Then associate with v the clause C' obtained by resolving C_1 and C_2 on x

Hard Instances for Branching Algorithms

- First non-trivial algorithm for k-SAT [MS85], as well as several further improvements[S93, S96, K99] were branching algorithms
- [PI99] show that there are unsatisfiable k-CNFs which require tree resolution proofs of size $2^{n-\Omega(n/k^{1/8})}$. Hence these are also hard instances for branching algorithms
- Instances encode linear equations mod 2 with at most k variables per equation

Issues with Direct Use of Proof Complexity

- A resolution-based lower bound can only apply to unsatisfiable formulas, since satisfiable formulas trivially have small proofs
- Suppose we tweak the branching algorithm model by "timing out" after f(n) steps, for some f. Then running algorithm on unsat formula doesn't give unsatisfiability proof
- Alternatively, suppose the strategy rule is probabilistic. Again link with proof complexity breaks down

Plan of the Talk

- SAT Algorithms and Proof Complexity: Preliminaries
- *Paradigms for Branching Algorithms, and Satisfiable Hard Instances*
- Hard Instances for State of the Art Algorithms
- Alternative Paradigms and Future Work

Satisfiable Hard Instances

- [ABM03] showed that *random* k-CNFs of a certain density (below the conjectured satisfiability threshold) are hard for certain natural branching algorithms (Ordered DLL and variants)
- Analysis uses proof complexity
- Issues
  - Not clear for which algorithms precisely the lower bound holds
  - Lower bound is of form $2^{\Omega(n)}$: not quite tight enough for us

Myopic and Drunk Algorithms

- [AHI04] formulate two models of branching-type algorithms
  - Myopic algorithms: Here the policy and strategy are restricted to learn only $n^{1-\varepsilon}$ clauses at a time
  - Drunk algorithms: Here the strategy can be arbitrarily complex but the policy is to set a variable purely at random
- Myopic and drunk algorithms capture most algorithms considered in random SAT literature

Exploiting Myopia and Drunkenness

- Theorem [AHI04]: Strongly expanding systems of linear equations cannot be solved by myopic algorithms in time $2^{n^{1-\Omega(1)}}$
- Theorem [AHI04]: There are explicit k-CNFs ϕ_n such that any drunk algorithm takes time at least $2^{n-\Omega(n/k^{1/8})}$ on ϕ_n with all but exponentially small probability

Features of [AHI04]

- Formalizations of natural algorithmic models which refine branching paradigm
- Use of proof complexity results ([BSW01] for myopic lower bound, [PI99] for drunk lower bound)
- Near-tightness and genericity of result for drunk algorithm

The Priority Branching Tree Model

- Considered in [ABBIMP05]
- Information about variables is revealed one "piece" at a time – algorithm gets to see which clauses the variable occurs in. It can decide which piece to see based on variables it's seen so far and values it's set them to
- Theorem[ABBIMP05]: Expanding systems of linear equations cannot be solved by pBT's in time $2^{o(n)}$

Plan of the Talk

- SAT Algorithms and Proof Complexity: Preliminaries
- Paradigms for Branching Algorithms, and Satisfiable Hard Instances
- *Hard Instances for State of the Art Algorithms*
- Alternative Paradigms and Future Work

State of the Art Algorithms

- k-SAT: [PPZ97] and [PPSZ98] use randomized branching with restarts; [S99] uses local search
- CNF-SAT: [S05] uses a branching (on disjunctions) reduction to [PPSZ98]/[S99]
- Formula-SAT: [S10] uses deterministic branching
- AC0-SAT: [IMP12] uses randomized branching on disjunctions
- None of these algorithms seem to fall into paradigms discussed

Hard Instances for Formula-SAT and AC0-SAT Algorithms

- In both cases, analysis uses lower bound technique of random restrictions
- Hard instances can be defined based on block-wise parity, intuitively because Parity does not reduce under random restrictions
- Yet again, the hard instances are linear systems!

Hard Instances for [PPSZ98]?

- Open for a long time
- Recently, [CSTT13] constructed satisfiable k-CNFs on which PPSZ algorithm has expected running time at least $2^{n(1-O(\log^2(k)/k))}$
- PPSZ algorithm is randomized branching algorithm with restarts where
  - Strategy: Pick variable such that unitclause containing that variable can be derived from formula using o(n)-width bounded resolution. If no such variable exists, pick randomly
  - Policy: If unit clause exists, force variable, else set randomly

Hard Instances for PPSZ

- Hard instances are again systems of random sparse linear equations
- Resolution width approach cannot directly work, since linear systems have resolution width n/2+o(n), and we're aiming for a lower bound of form $2^{(1-\varepsilon)n}$
- However, proof does take advantage of expansion in a similar way to the proof complexity setting

Plan of the Talk

- SAT Algorithms and Proof Complexity: Preliminaries
- Paradigms for Branching Algorithms, and Satisfiable Hard Instances
- Hard Instances for State of the Art Algorithms
- *Alternative Paradigms and Future Work*

Alternative Paradigms

- Rather than a syntactic description of possible algorithms, we could aim for a semantic description, in the style of Razborov-Rudich proofs
- We consider three such paradigms
  - Witness compression
  - OPP (One-sided error probabilistic polynomial-time algorithms)
  - Decision tree-style algorithms for #SAT

Witness Compression

- Many SAT algorithms achieving savings s(n) can be interpreted as providing a deterministic or probabilistic reduction from SAT on n variables to Circuit SAT on ≈ n-s(n) variables
- [DH11] observes that this holds for local search algorithms. Also true for branching algorithms
- Can we give complexity-theoretic evidence against this in general?

OPP algorithms

- Defined by [PP10], who observe that most SAT algorithms with savings s(n) can be translated into probabilistic polynomial-time algorithms with probability at least $2^{s(n)-n}$ of success
- Note that witness compression implies OPP
- [PP10] show that if Circuit SAT has an OPP algorithm with $\Omega(n)$ savings, then ETH fails
- Still natural algorithms not known to be OPP, eg., FormulaSAT algorithm of [S10]

Hard Instances for #SAT algorithms

- Interesting fact: Best known upper bounds for SAT also hold for #SAT (for k-SAT, Formula SAT, AC0 SAT)
- Branching algorithms for #SAT tend to yield decision trees for input formula
- Tightness of analysis follows from known lower bounds for decision tree size of CNFs [MRW05]

Future Work

- Is there a nice model which encompasses branching algorithms and local search?
- Is there a nice model which captures branching algorithms, deterministic and randomized, with or without restarts?
- Can we show a general result about hardness of linear systems, inspired by methods from proof complexity?