SAT: Disruption, Demise & Resurgence

Joao Marques-Silva

CDCL
The power of learning

PoS Workshop

IST, Lisbon, Portugal

July 8 2019

SAT: Disruption, Demise & Resurgence

Joao Marques-Silva

PoS Workshop

IST, Lisbon, Portugal

July 8 2019

CDCL

The power of learning

SAT: Disruption, Demise & Resurgence

Joao Marques-Silva

PoS Workshop

IST, Lisbon, Portugal

July 8 2019

How good are CDCL SAT solvers?

Demos

How good are CDCL SAT solvers?

Demos

- Sample SAT of solvers:
    1. POSIT: state of the art DPLL SAT solver in 1995
    2. GRASP: first CDCL SAT solver, state of the art 1995∼2000
    3. Minisat: CDCL SAT solver, state of the art until the late 00s
    4. Glucose: modern state of the art CDCL SAT solver
    5. ...

How good are CDCL SAT solvers?

Demos

- Sample SAT of solvers:
  1. POSIT: state of the art DPLL SAT solver in 1995
  2. GRASP: first CDCL SAT solver, state of the art 1995∼2000
  3. Minisat: CDCL SAT solver, state of the art until the late 00s
  4. Glucose: modern state of the art CDCL SAT solver
  5. ...
- Example 1: model checking example (from IBM)
- Example 2: cooperative path finding (CPF)

How good are SAT solvers? – an example

- Cooperative pathfinding (CPF)
  - N agents on some grid/graph
  - Start positions
  - Goal positions
  - Minimize makespan
  - Restricted planning problem

How good are SAT solvers? – an example

- Cooperative pathfinding (CPF)
  - *N* agents on some grid/graph
  - Start positions
  - Goal positions
  - Minimize makespan
  - Restricted planning problem
- Concrete example
  - Gaming grid
  - 1039 vertices
  - 1928 edges
  - 100 agents

How good are SAT solvers? – an example

- Cooperative pathfinding (CPF)
  - N agents on some grid/graph
  - Start positions
  - Goal positions
  - Minimize makespan
  - Restricted planning problem

- Concrete example
  - Gaming grid
  - 1039 vertices
  - 1928 edges
  - 100 agents

```
*** tracker: a pathfinding tool ***

Initialization ... CPU Time: 0.004711
Number of variables: 113315
Tentative makespan 1
Number of variables: 226630
Number of assumptions: 1
c Running SAT solver ... CPU Time: 0.718112
c Done running SAT solver ... CPU Time: 0.830099
No solution for makespan 1
Elapsed CPU Time: 0.830112
Tentative makespan 2
Number of variables: 339945
Number of assumptions: 1
c Running SAT solver ... CPU Time: 1.27113
c Done running SAT solver ... CPU Time: 1.27114
No solution for makespan 2
Elapsed CPU Time: 1.27114
...
...
Tentative makespan 24
Number of variables: 2832875
Number of assumptions: 1
c Running SAT solver ... CPU Time: 11.8653
c Done running SAT solver ... CPU Time: 11.8653
No solution for makespan 24
Elapsed CPU Time: 11.8653
Tentative makespan 25
Number of variables: 2946190
Number of assumptions: 1
c Running SAT solver ... CPU Time: 12.3491
c Done running SAT solver ... CPU Time: 16.6882
Solution found for makespan 25
Elapsed CPU Time: 16.6995
```

How good are SAT solvers? – an example

- Cooperative pathfinding (CPF)
  - *N* agents on some grid/graph
  - Start positions
  - Goal positions
  - Minimize makespan
  - Restricted planning problem
- Concrete example
  - Gaming grid
  - 1039 vertices
  - 1928 edges
  - 100 agents
  - Formula w/ 2946190 variables**!**

```
*** tracker: a pathfinding tool ***

Initialization ... CPU Time: 0.004711
Number of variables: 113315
Tentative makespan 1
Number of variables: 226630
Number of assumptions: 1
c Running SAT solver ... CPU Time: 0.718112
c Done running SAT solver ... CPU Time: 0.830099
No solution for makespan 1
Elapsed CPU Time: 0.830112
Tentative makespan 2
Number of variables: 339945
Number of assumptions: 1
c Running SAT solver ... CPU Time: 1.27113
c Done running SAT solver ... CPU Time: 1.27114
No solution for makespan 2
Elapsed CPU Time: 1.27114
...
...
Tentative makespan 24
Number of variables: 2832875
Number of assumptions: 1
c Running SAT solver ... CPU Time: 11.8653
c Done running SAT solver ... CPU Time: 11.8653
No solution for makespan 24
Elapsed CPU Time: 11.8653
Tentative makespan 25
Number of variables: 2946190
Number of assumptions: 1
c Running SAT solver ... CPU Time: 12.3491
c Done running SAT solver ... CPU Time: 16.6882
Solution found for makespan 25
Elapsed CPU Time: 16.6995
```

How good are SAT solvers? – an example

- Cooperative pathfinding (CPF)
  - N agents on some grid/graph
  - Start positions
  - Goal positions
  - Minimize makespan
  - Restricted planning problem

- Concrete example
  - Gaming grid
  - 1039 vertices
  - 1928 edges
  - 100 agents
  - Formula w/ 2946190 variables!

- **Note:** In the early 90s, SAT solvers could solve formulas with a few hundred variables!

```
*** tracker: a pathfinding tool ***

Initialization ... CPU Time: 0.004711
Number of variables: 113315
Tentative makespan 1
Number of variables: 226630
Number of assumptions: 1
c Running SAT solver ... CPU Time: 0.718112
c Done running SAT solver ... CPU Time: 0.830099
No solution for makespan 1
Elapsed CPU Time: 0.830112
Tentative makespan 2
Number of variables: 339945
Number of assumptions: 1
c Running SAT solver ... CPU Time: 1.27113
c Done running SAT solver ... CPU Time: 1.27114
No solution for makespan 2
Elapsed CPU Time: 1.27114
...
...
Tentative makespan 24
Number of variables: 2832875
Number of assumptions: 1
c Running SAT solver ... CPU Time: 11.8653
c Done running SAT solver ... CPU Time: 11.8653
No solution for makespan 24
Elapsed CPU Time: 11.8653
Tentative makespan 25
Number of variables: 2946190
Number of assumptions: 1
c Running SAT solver ... CPU Time: 12.3491
c Done running SAT solver ... CPU Time: 16.6882
Solution found for makespan 25
Elapsed CPU Time: 16.6995
```

Grasping the search space ...

- Number of seconds since the Big Bang: $\approx 10^{17}$

Grasping the search space …

- Number of seconds since the Big Bang: $\approx 10^{17}$
- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)

Grasping the search space ...

- Number of seconds since the Big Bang: $\approx 10^{17}$
- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)
- Search space with 15775 propositional variables (worst case):

Grasping the search space ...

- Number of seconds since the Big Bang: $\approx 10^{17}$

- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)

- Search space with 15775 propositional variables (worst case):
  - # of assignments to 15775 variables: $> 10^{4748}$ **!**
  - **Obs:** SAT solvers in the late 90s (but formula dependent)

Grasping the search space ...

- Number of seconds since the Big Bang: $\approx 10^{17}$
- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)
- Search space with 15775 propositional variables (worst case):
    - # of assignments to 15775 variables: $> 10^{4748}$ **!**
    - **Obs:** SAT solvers in the late 90s (but formula dependent)
- Search space with 2832875 propositional variables (worst case):

Grasping the search space ...

- Number of seconds since the Big Bang: $\approx 10^{17}$
- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)
- Search space with 15775 propositional variables (worst case):
  - # of assignments to 15775 variables: $> 10^{4748}$ **!**
  - **Obs:** SAT solvers in the late 90s (but formula dependent)
- Search space with 2832875 propositional variables (worst case):
  - # of assignments to $> 2.8 \times 10^6$ variables: $\gg 10^{840000}$ **!!**
  - **Obs:** SAT solvers at present (but formula dependent)

1 SAT Disruption

The CDCL SAT disruption

- CDCL SAT solving is a success story of Computer Science

The CDCL SAT disruption

- CDCL SAT solving is a success story of Computer Science
    - Conflict-Driven Clause Learning (CDCL)
    - (CDCL) SAT has impacted many different fields
    - Hundreds (thousands?) of practical applications

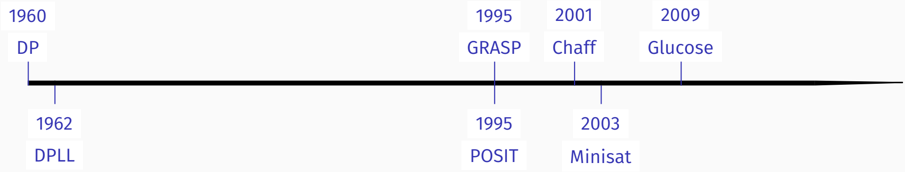
So, what is a CDCL SAT solver?

- Extend DPLL SAT solver with: [DP60, DLL62]
    - Clause learning & non-chronological backtracking [MS95, MSS96, MSS99]
    - Search restarts [GSC97, BMS00, Hua07, Bie08, LOM+18]
    - Lazy data structures
    - Conflict-guided branching
    - ...

So, what is a CDCL SAT solver?

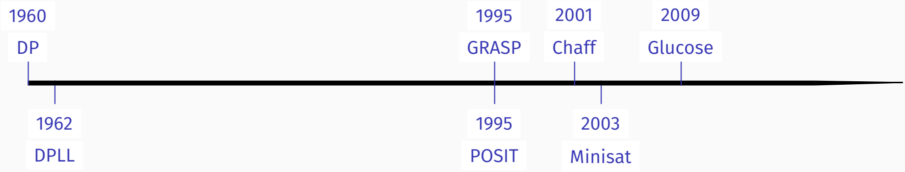
- Extend DPLL SAT solver with: [DP60, DLL62]
  - Clause learning & non-chronological backtracking [MS95, MSS96, MSS99]
    - Exploit UIPs [MS95, MSS99, ZMMM01, SSS12]
    - Minimize learned clauses [SB09, Gel09, LLX+17]
    - Opportunistically delete clauses [MSS96, MSS99, GN02, AS09]
  - Search restarts [GSC97, BMS00, Hua07, Bie08, LOM+18]
  - Lazy data structures
    - Watched literals [MMZ+01]
  - Conflict-guided branching
    - Lightweight branching heuristics [MMZ+01]
    - Phase saving [PD07]
  - ...

CDCL timeline – somewhat incomplete



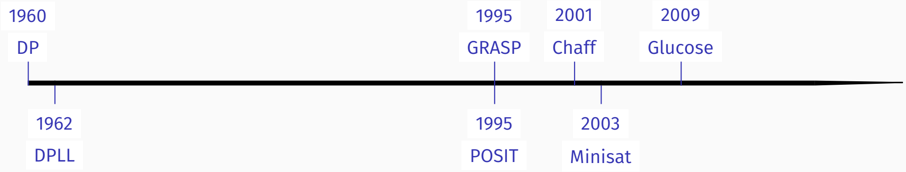
- DPLL (DP/DLL): backtracking search with unit propagation

CDCL timeline – somewhat incomplete



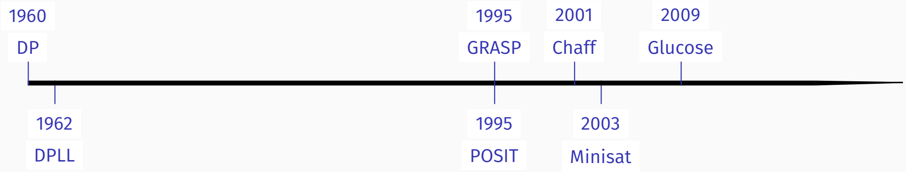
- DPLL (DP/DLL): backtracking search with unit propagation
- POSIT: efficient implementation of DPLL

CDCL timeline – somewhat incomplete



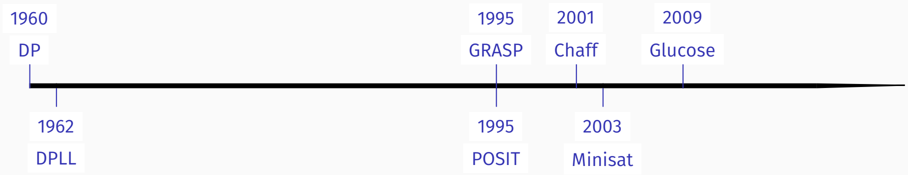
- DPLL (DP/DLL): backtracking search with unit propagation
- POSIT: efficient implementation of DPLL
- GRASP:
  1. Clause learning; UIPs, implication graphs, decision levels, antecedents, etc. [MS95, MSS96, MSS99]
  2. Integration of search restarts with clause learning [BMS00]

CDCL timeline – somewhat incomplete



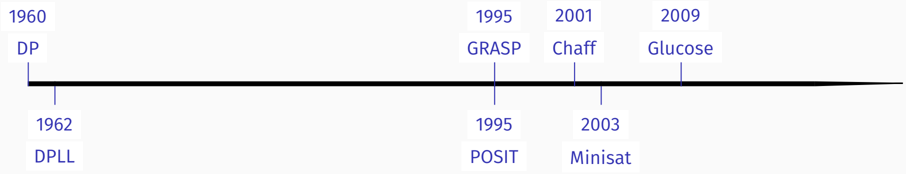
- DPLL (DP/DLL): backtracking search with unit propagation
- POSIT: efficient implementation of DPLL
- GRASP:
  1. Clause learning; UIPs, implication graphs, decision levels, antecedents, etc. [MS95, MSS96, MSS99]
  2. Integration of search restarts with clause learning [BMS00]
- Chaff:
  1. VSIDS, watched literals [MMZ+01]
  2. Always backtrack after conflict [ZMMM01]

CDCL timeline – somewhat incomplete



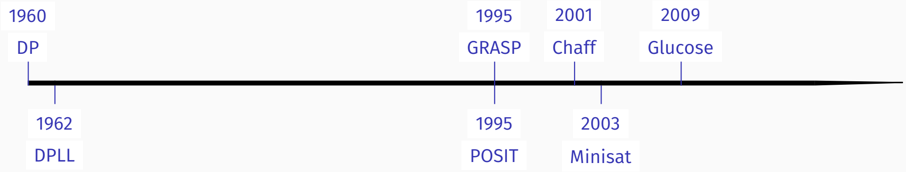
- DPLL (DP/DLL): backtracking search with unit propagation
- POSIT: efficient implementation of DPLL
- GRASP:
  1. Clause learning; UIPs, implication graphs, decision levels, antecedents, etc. [MS95, MSS96, MSS99]
  2. Integration of search restarts with clause learning [BMS00]
- Chaff:
  1. VSIDS, watched literals [MMZ+01]
  2. Always backtrack after conflict [ZMMM01]
- Minisat:
  1. Learned clause minimization [SB09]

CDCL timeline – somewhat incomplete



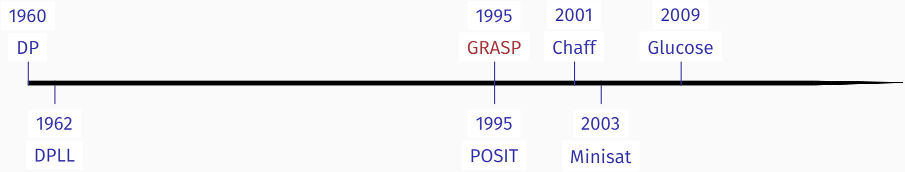
- DPLL (DP/DLL): backtracking search with unit propagation
- POSIT: efficient implementation of DPLL
- GRASP:
  1. Clause learning; UIPs, implication graphs, decision levels, antecedents, etc. [MS95, MSS96, MSS99]
  2. Integration of search restarts with clause learning [BMS00]
- Chaff:
  1. VSIDS, watched literals [MMZ+01]
  2. Always backtrack after conflict [ZMMM01]
- Minisat:
  1. Learned clause minimization [SB09]
- Glucose:
  1. LBD [AS09]

CDCL timeline – somewhat incomplete



- DPLL (DP/DLL): backtracking search with unit propagation
- POSIT: efficient implementation of DPLL
- GRASP:
  1. Clause learning; UIPs, implication graphs, decision levels, antecedents, etc. [MS95, MSS96, MSS99]
  2. Integration of search restarts with clause learning [BMS00]
- Chaff:
  1. VSIDS, watched literals [MMZ+01]
  2. Always backtrack after conflict [ZMMM01]
- Minisat:
  1. Learned clause minimization [SB09]
- Glucose:
  1. LBD [AS09]
- Berkmin, siege, picosat, lingeling, ...

CDCL timeline – somewhat incomplete



- DPLL (DP/DLL): backtracking search with unit propagation
- POSIT: efficient implementation of DPLL
- **GRASP**:
  1. Clause learning; UIPs, implication graphs, decision levels, antecedents, etc. [MS95, MSS96, MSS99]
  2. Integration of search restarts with clause learning [BMS00]
- Chaff:
  1. VSIDS, watched literals [MMZ+01]
  2. Always backtrack after conflict [ZMMM01]
- Minisat:
  1. Learned clause minimization [SB09]
- Glucose:
  1. LBD [AS09]
- Berkmin, siege, picosat, lingeling, ...

GRASP – a somewhat unknown story

Efficient Generation of Test Patterns Using Boolean Difference

Tracy Larrabee
Computer Science Department
Stanford University
Stanford, CA 94305

Abstract

Most automatic test pattern generation systems for combinational circuits generate a test for a given fault by directly searching a data structure representing the circuit to be tested. This paper describes a new system that divides the problem into two parts: First, it constructs a formula expressing the Boolean difference between the unfaulted and faulted circuits. Second, it applies a Boolean satisfiability algorithm to the resulting formula. The new system can incorporate any of the heuristics used by structural search techniques. It is not only quite general, but is able to test or prove untestable every fault in the popular Brglez-Fujiwara[1] test benchmark.

ITC1989

GRASP – a somewhat unknown story

Timing Analysis and Delay-Fault Test Generation using Path-Recursive Functions*

Patrick C. McGeer Alexander Saldanha Paul R. Stephan Robert K. Brayton
Alberto L. Sangiovanni-Vincentelli

University of California - Berkeley CA

Abstract

Functional analysis of paths through combinational logic circuits has recently emerged as a critical problem in timing analysis and various forms of test generation. In this paper, we introduce an efficient method for generating the functional forms of path analysis problems. We demonstrate that the resulting function is linear in the size of the circuit. The functions are then tested for satisfiability either using a Boolean network satisfiability algorithm suggested in [5] or through the construction of BDD's [1]. The effectiveness of the proposed approach is shown for timing analysis and robust path delay-fault test generation. This method also holds promise for both static and dynamic hazard analysis, and for test generation using all other delay-fault models, τ-irredundant fault models, and stuck-open fault models.

ICCAD 1991

GRASP – a somewhat unknown story

4

IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN, VOL. 11, NO. 1, JANUARY 1992

Test Pattern Generation Using Boolean Satisfiability

Tracy Larrabee, *Member, IEEE*

Abstract—This article describes the Boolean satisfiability method for generating test patterns for single stuck-at faults in combinational circuits. This new method generates test patterns in two steps: First, it constructs a formula expressing the *Boolean difference* between the unfaulted and faulted circuits. Second, it applies a *Boolean satisfiability* algorithm to the resulting formula. This approach differs from previous methods now in use, which search the circuit structure directly instead of constructing a formula from it. The new method is general and effective: it allows for the addition of heuristics used by structural search methods, and it has produced excellent results on popular test pattern generation benchmarks.

IEEE TCAD 1992

GRASP – a somewhat unknown story

EECS 579: Digital System Testing

Instructor: John P. Hayes

Coverage
This course examines the theory and practice of fault analysis, test generation, and design for testability for digital circuits and systems.

Lab
A term project or paper is required which is tailored to individual student interests, and typically involve one of the following:
A. Programming a test generation or simulation algorithm covered in the course
B. In-depth literature survey of some advanced topic
C. Individual research into some special topic or problem
D. Experiments with commercial test and simulation CAD hardware or software
All projects require a written report and an oral presentation to the class at the end of the term. Item D is subject to availability of the hardware and software needed.

Textbook(s)

UofM Spring'92

GRASP – a somewhat unknown story

EECS 579: Digital System Testing

Instructor: John P. Hayes

Coverage

This course examines the theory and practice of fault analysis, test generation, and design for testability for digital circuits and systems.

Lab

A term project or paper is required which is tailored to individual student interests, and typically involve one of the following:

A. Programming a test generation or simulation algorithm covered in the course.

B. In-depth literature survey of some advanced topic

C. Individual research into some special topic or problem

D. Experiments with commercial test and simulation CAD hardware or softwa

All projects require a written report and an oral presentation to the class at the e

Item D is subject to availability of the hardware and software needed.

Textbook(s)

UofM Spring'92

Larrabee's SAT algorithm **didn't** work!

GRASP – a somewhat unknown story

Dynamic Search-Space Pruning Techniques in Path Sensitization

João P. Marques Silva and Karem A. Sakallah
Department of Electrical Engineering and Computer Science
University of Michigan

Abstract— A powerful combinational path sensitization engine is required for the efficient implementation of tools for test pattern generation, timing analysis, and delay fault testing. Path sensitization can be posed as a search, in the n-dimensional Boolean space, for a consistent assignment of logic values to the circuit nodes which also satisfies a given condition. In this paper we propose and demonstrate the effectiveness of several new techniques for search-space pruning for test pattern generation. In particular, we present linear-time algorithms for dynamically identifying unique sensitization points and for dynamically maintaining reduced head line sets. In addition, we present two powerful mechanisms that drastically reduce the number of backtracks: failure-driven assertions and dependency-directed backtracking. Both mechanisms can be viewed as a form of learning while searching and have analogs in other application domains. These search pruning methods have been implemented in a generic path sensitization engine called LEAP. A test pattern generator, TG-LEAP, that uses this engine was also developed. We present experimental results that compare the effectiveness of our proposed search pruning strategies to those of PODEM, FAN, and SOCRATES. In particular, we show that LEAP is very efficient in identifying undetectable faults and in generating tests for difficult faults.

DAC 1994

GRASP – a somewhat unknown story

Dynamic Search-Space Pruning Techniques in Path Sensitization

João P. Marques Silva and Karem A. Sakallah
Department of Electrical Engineering and Computer Science
University of Michigan

Abstract— A powerful combinational path sensitization engine is required for the efficient implementation of tools for test pattern generation, timing analysis, and delay fault testing. Path sensitization can be posed as a search, in the n-dimensional Boolean space, for a consistent assignment of logic values to the circuit nodes which also satisfies a given condition. In this paper we propose and demonstrate the effectiveness of several new techniques for search-space pruning for test pattern generation. In particular, we present linear-time algorithms for dynamically identifying unique sensitization points and for dynamically maintaining reduced head line sets. In addition, we present two powerful mechanisms that drastically reduce the number of backtracks: failure-driven assertions and dependency-directed backtracking. Both mechanisms can be viewed as a form of learning while searching and have analogs in other application domains. These search pruning methods have been implemented in a generic path sensitization engine called LEAP. A test pattern generator, TG-LEAP, that uses this engine was also developed. We present experimental results that compare the effectiveness of our proposed search pruning strategies to those of PODEM, FAN, and SOCRATES. In particular, we show that LEAP is very efficient in identifying undetectable faults and in generating tests for difficult faults.

UIP's inspired on **USP**'s!

DAC 1994

GRASP – a somewhat unknown story

SEARCH ALGORITHMS FOR SATISFIABILITY PROBLEMS IN COMBINATIONAL SWITCHING CIRCUITS

by

João Paulo Marques da Silva

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
1995

GRASP – a somewhat unknown story

SEARCH ALGORITHMS FOR SATISFIABILITY PROBLEMS IN COMBINATIONAL SWITCHING CIRCUITS

by

João Paulo Marques da Silva

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
1995

Proposes modern clause learning!

GRASP – a somewhat unknown story

IMPROVEMENTS TO PROPOSITIONAL SATISFIABILITY SEARCH ALGORITHMS

JON WILLIAM FREEMAN

A DISSERTATION

in

COMPUTER AND INFORMATION SCIENCE

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy.

1995

GRASP – a somewhat unknown story

IMPROVEMENTS TO PROPOSITIONAL SATISFIABILITY SEARCH ALGORITHMS

JON WILLIAM FREEMAN

A DISSERTATION

in

COMPUTER AND INFORMATION SCIEN

Claims clause learning will **not** work!

Presented to the Faculties of the University of Pennsylvania in Partial Fulfi

Requirements for the Degree of Doctor of Philosophy.

1995

GRASP – a somewhat unknown story

GRASP—A New Search Algorithm for Satisfiability

João P. Marques Silva
Cadence European Laboratories
IST/INESC
1000 Lisboa, Portugal

Karem A. Sakallah
Department of EECS
University of Michigan
Ann Arbor, Michigan 48109-2122

Abstract

This paper introduces GRASP (Generic seaRch Algorithm for the Satisfiability Problem), an integrated algorithmic framework for SAT that unifies several previously proposed search-pruning techniques and facilitates identification of additional ones. GRASP is premised on the inevitability of conflicts during search and its most distinguishing feature is the augmentation of basic backtracking search with a powerful conflict analysis procedure. Analyzing conflicts to determine their causes enables GRASP to backtrack non-chronologically to earlier levels in the search tree, potentially pruning large portions of the search space. In addition, by "recording" the causes of conflicts, GRASP can recognize and preempt the occurrence of similar conflicts later on in the search. Finally, straightforward bookkeeping of the causality chains leading up to conflicts allows GRASP to identify assignments that are necessary for a solution to be found. Experimental results obtained from a large number of benchmarks, including many from the field of test pattern generation, indicate that application of the proposed conflict analysis techniques to SAT algorithms can be extremely effective for a large number of representative classes of SAT instances.

ICCAD 1996

GRASP – a somewhat unknown story

506 IEEE TRANSACTIONS ON COMPUTERS, VOL. 48, NO. 5, MAY 1999

GRASP: A Search Algorithm for Propositional Satisfiability

João P. Marques-Silva, *Member, IEEE,* and Karem A. Sakallah, *Fellow, IEEE*

Abstract—This paper introduces GRASP (Generic seaRch Algorithm for the Satisfiability Problem), a new search algorithm for Propositional Satisfiability (SAT). GRASP incorporates several search-pruning techniques that proved to be quite powerful on a wide variety of SAT problems. Some of these techniques are specific to SAT, whereas others are similar in spirit to approaches in other fields of Artificial Intelligence. GRASP is premised on the inevitability of conflicts during the search and its most distinguishing feature is the augmentation of basic backtracking search with a powerful conflict analysis procedure. Analyzing conflicts to determine their causes enables GRASP to backtrack nonchronologically to earlier levels in the search tree, potentially pruning large portions of the search space. In addition, by "recording" the causes of conflicts, GRASP can recognize and preempt the occurrence of similar conflicts later on in the search. Finally, straightforward bookkeeping of the causality chains leading up to conflicts allows GRASP to identify assignments that are necessary for a solution to be found. Experimental results obtained from a large number of benchmarks indicate that application of the proposed conflict analysis techniques to SAT algorithms can be extremely effective for a large number of representative classes of SAT instances.

Index Terms—Satisfiability, search algorithms, conflict diagnosis, conflict-directed nonchronological backtracking, conflict-based equivalence, failure-driven assertions, unique implication points.

GRASP – a somewhat unknown story

Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability

CP 2000

Luís Baptista and João Marques-Silva

Department of Informatics, Technical University of Lisbon,
IST/INESC/CEL, Lisbon, Portugal
{lmtb,jpms}@algos.inesc.pt

Abstract. This paper addresses the interaction between randomization, with restart strategies, and learning, an often crucial technique for proving unsatisfiability. We use instances of SAT from the hardware verification domain to provide evidence that randomization can indeed be essential in solving real-world satisfiable instances of SAT. More interestingly, our results indicate that randomized restarts and learning may cooperate in proving both satisfiability and unsatisfiability. Finally, we utilize and expand the idea of algorithm portfolio design to propose an alternative approach for solving hard unsatisfiable instances of SAT.

GRASP – a somewhat unknown story

Using CSP Look-Back Techniques to Solve Real-World SAT Instances

Roberto J. Bayardo Jr.
The University of Texas at Austin
Department of Computer Sciences (C0500)
Austin, TX 78712 USA
bayardo@cs.utexas.edu
http://www.cs.utexas.edu/users/bayardo

Robert C. Schrag
Information Extraction and Transport, Inc.
1730 North Lynn Street, Suite 502
Arlington, VA 22209 USA
schrag@iet.com
http://www.iet.com/users/schrag

Abstract

We report on the performance of an enhanced version of the "Davis-Putnam" (DP) proof procedure for propositional satisfiability (SAT) on large instances derived from real-world problems in planning, scheduling, and circuit diagnosis and synthesis. Our results show that incorporating CSP look-back techniques -- especially the relatively new technique of relevance-bounded learning -- renders easy many problems which otherwise are beyond DP's reach. Frequently they make DP, a systematic algorithm, perform as well or better than stochastic SAT algorithms such as GSAT or WSAT. We recommend that such techniques be included as options in implementations of DP, just as they are in systematic algorithms for the more general constraint satisfaction problem.

Different learning: no UIPs, no decision levels, etc.

AAAI 1997

SATO: An Efficient Propositional Prover

CADE 1997

Hantao Zhang*

Department of Computer Science
The University of Iowa
Iowa City, IA 52242-1419, USA
hzhang@cs.uiowa.edu

Rehash of GRASP, without UIPs

SATO (Satisfiability Testing Optimized) is a propositional prover based on the Davis-Putnam method [3], which is is one of the major practical methods for the satisfiability (SAT) problem of propositional logic. The first report of SATO appeared in [12]. Since then, we constantly add new techniques into SATO to make it more efficient [14, 13].

GRASP – a somewhat unknown story

Symbolic Model Checking without BDDs*

Armin Biere[1], Alessandro Cimatti[2], Edmund Clarke[1], and Yunshan Zhu[1]

[1] Computer Science Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A
{Armin.Biere,Edmund.Clarke,Yunshan.Zhu}@cs.cmu.edu
[2] Istituto per la Ricerca Scientifica e Tecnologica (IRST)
via Sommarive 18, 38055 Povo (TN), Italy
cimatti@irst.itc.it

Abstract. Symbolic Model Checking [3, 14] has proven to be a powerful technique for the verification of reactive systems. BDDs [2] have traditionally been used as a symbolic representation of the system. In this paper we show how boolean decision procedures, like Stålmarck's Method [16] or the Davis & Putnam Procedure [7], can replace BDDs. This new technique avoids the space blow up of BDDs, generates counterexamples much faster, and sometimes speeds up the verification. In addition, it produces counterexamples of minimal length. We introduce a *bounded model checking* procedure for LTL which reduces model checking to propositional satisfiability. We show that bounded LTL model checking can be done without a tableau construction. We have implemented a model checker **BMC**, based on bounded model checking, and preliminary results are presented.

TACAS 1999

GRASP – a somewhat unknown story

33.1

Chaff: Engineering an Efficient SAT Solver

Matthew W. Moskewicz
Department of EECS
UC Berkeley
moskewcz@alumni.princeton.edu

Conor F. Madigan
Department of EECS
MIT
cmadigan@mit.edu

Ying Zhao, Lintao Zhang, Sharad Malik
Department of Electrical Engineering
Princeton University
{yingzhao, lintaoz, sharad}@ee.princeton.edu

ABSTRACT

Boolean Satisfiability is probably the most studied of combinatorial optimization/search problems. Significant effort has been devoted to trying to provide practical solutions to this problem for problem instances encountered in a range of applications in Electronic Design Automation (EDA), as well as in Artificial Intelligence (AI). This study has culminated in the development of several SAT packages, both proprietary and in the public domain (e.g. GRASP, SATO) which find significant use in both research and industry. Most existing complete solvers are variants of the Davis-Putnam (DP) search algorithm. In this paper we describe the development of a new complete solver, Chaff, which achieves significant performance gains through careful engineering of all aspects of the search – especially a particularly efficient implementation of Boolean constraint propagation (BCP) and a novel low overhead decision strategy. Chaff has been able to obtain one to two orders of magnitude performance improvement on difficult SAT benchmarks in comparison with other solvers (DP or otherwise), including GRASP and SATO.

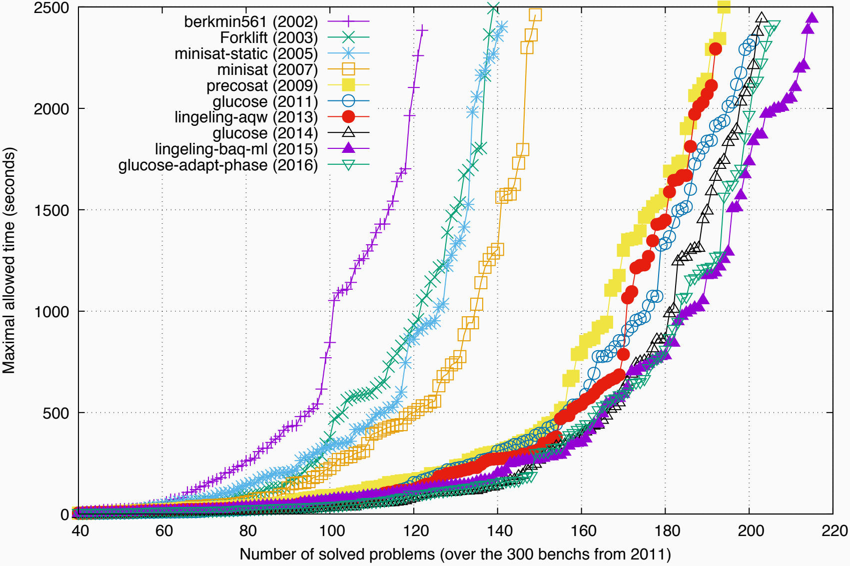
Chaff showed how to make GRASP-like clause learning scale in practice!

DAC 2001

2 SAT Demise?

CDCL SAT solver (continued?) improvement

[Source: Simon 2015]



Is there a problem with SAT?

- Dwindling number of papers on SAT solving, e.g. at the SAT conference

Is there a problem with SAT?

- Dwindling number of papers on SAT solving, e.g. at the SAT conference
  - No major performance breakthrough in close to two decades...

Is there a problem with SAT?

- Dwindling number of papers on SAT solving, e.g. at the SAT conference
  - No major performance breakthrough in close to two decades...
- Unclear the net gain over large range of benchmarks
  - Are SAT solvers being tuned to specific benchmarks?
  - What to do with preprocessing/inprocessing, e.g. when using SAT solvers as oracles?

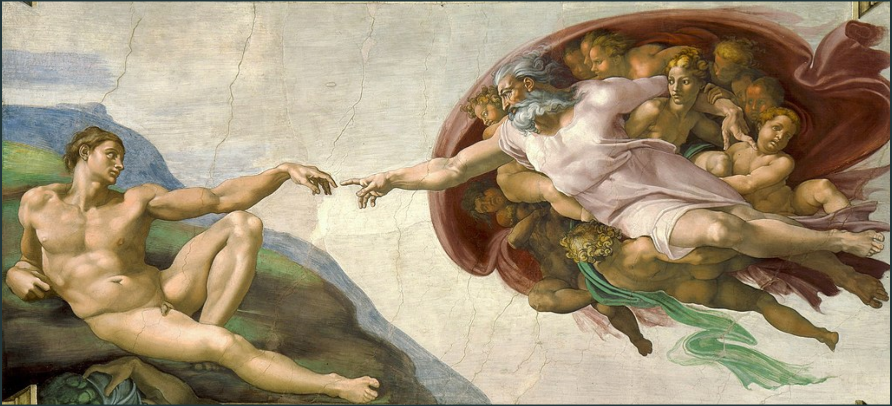
Is there a problem with SAT?

- Dwindling number of papers on SAT solving, e.g. at the SAT conference
    - No major performance breakthrough in close to two decades...
- Unclear the net gain over large range of benchmarks
    - Are SAT solvers being tuned to specific benchmarks?
    - What to do with preprocessing/inprocessing, e.g. when using SAT solvers as oracles?
- General perception among some researchers ...

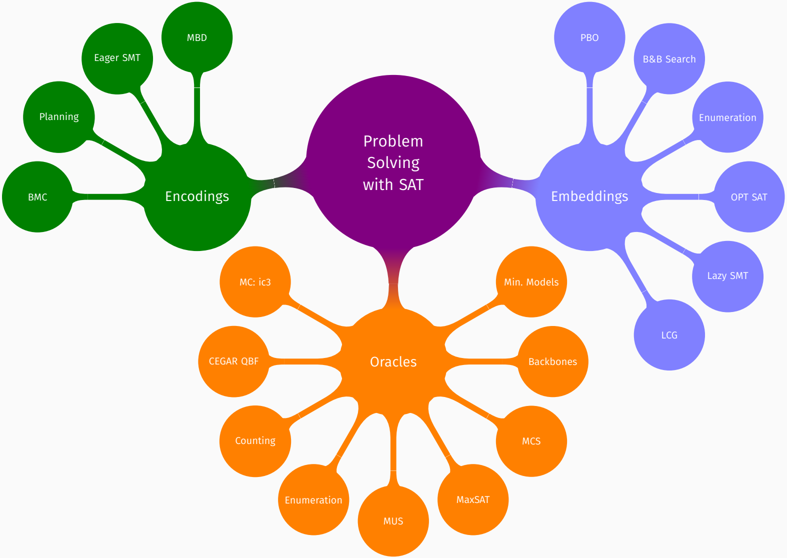
Is there a problem with SAT?

- Dwindling number of papers on SAT solving, e.g. at the SAT conference
  - No major performance breakthrough in close to two decades...
- Unclear the net gain over large range of benchmarks
  - Are SAT solvers being tuned to specific benchmarks?
  - What to do with preprocessing/inprocessing, e.g. when using SAT solvers as oracles?
- General perception among some researchers ...
- **Q: Is there a point in SAT research at present?**

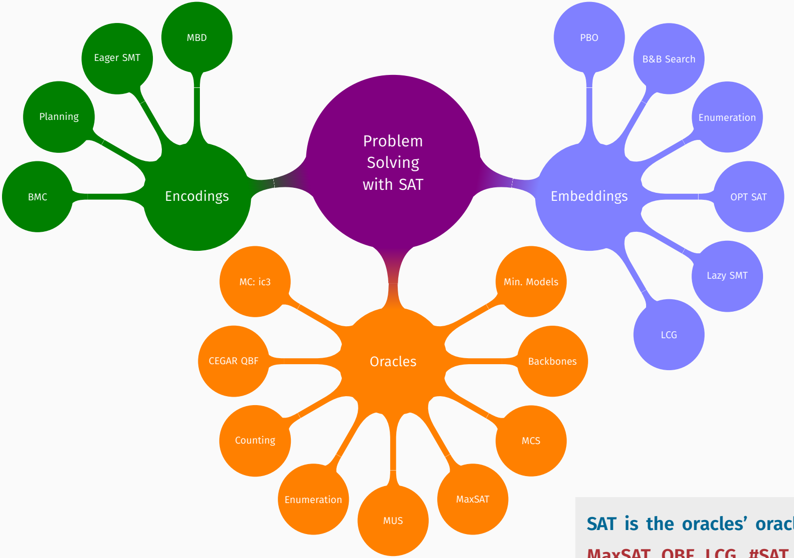
3 SAT Resurgence



CDCL SAT is ubiquitous in problem solving



CDCL SAT is ubiquitous in problem solving



SAT is the oracles' oracle:
MaxSAT, QBF, LCG, #SAT, SMT, ASP, FOL, ...

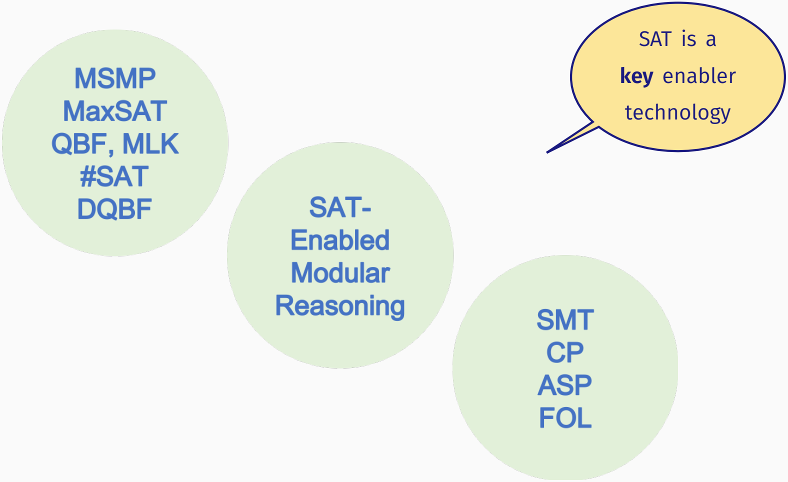
Age of SAT-enabled modular reasoning

MSMP
MaxSAT
QBF, MLK
#SAT
DQBF

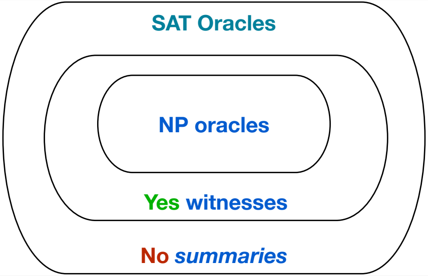
SAT-
Enabled
Modular
Reasoning

SMT
CP
ASP
FOL

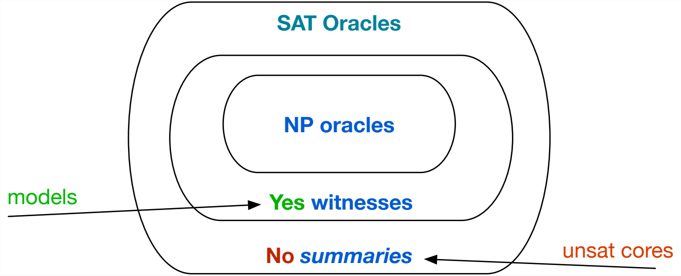
Age of SAT-enabled modular reasoning



So what are SAT oracles?



So what are SAT oracles?



The power of the (SAT) oracle

- **Q:** How to solve the FSAT problem?

  FSAT: Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle

The power of the (SAT) oracle

- **Q:** How to solve the FSAT problem?

  FSAT: Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle

  - A possible algorithm:
    1. Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \text{var}(\mathcal{F})$, in order
    2. $i \leftarrow 1$ and $\mathcal{F}_i \triangleq \mathcal{F}$
    3. Call NP oracle on $\mathcal{F}_i \wedge (x_i)$
    4. If answer is **yes**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (x_i)$
    5. If answer is **no**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (\neg x_i)$
    6. $i \leftarrow i + 1$
    7. If $i \leq n$, then repeat from 3.

The power of the (SAT) oracle

- **Q:** How to solve the FSAT problem?

  FSAT: Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle

  - A possible algorithm:
    1. Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \mathrm{var}(\mathcal{F})$, in order
    2. $i \leftarrow 1$ and $\mathcal{F}_i \triangleq \mathcal{F}$
    3. Call NP oracle on $\mathcal{F}_i \wedge (x_i)$
    4. If answer is **yes**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (x_i)$
    5. If answer is **no**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (\neg x_i)$
    6. $i \leftarrow i + 1$
    7. If $i \leq n$, then repeat from 3.
  - Algorithm needs $|\mathrm{var}(\mathcal{F})|$ calls to an NP oracle

The power of the (SAT) oracle

- **Q:** How to solve the FSAT problem?

  FSAT: Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle

  - A possible algorithm:
    1. Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \text{var}(\mathcal{F})$, in order
    2. $i \leftarrow 1$ and $\mathcal{F}_i \triangleq \mathcal{F}$
    3. Call NP oracle on $\mathcal{F}_i \wedge (x_i)$
    4. If answer is **yes**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (x_i)$
    5. If answer is **no**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (\neg x_i)$
    6. $i \leftarrow i + 1$
    7. If $i \leq n$, then repeat from 3.
  - Algorithm needs $|\text{var}(\mathcal{F})|$ calls to an NP oracle
  - **Note:** Cannot solve FSAT with logarithmic number of NP oracle calls, unless $\text{P} = \text{NP}$ [GF93]

- FSAT is an example of a function problem

The power of the (SAT) oracle

- **Q:** How to solve the FSAT problem?

  FSAT: Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle

  - A possible algorithm:
    1. Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \mathrm{var}(\mathcal{F})$, in order
    2. $i \leftarrow 1$ and $\mathcal{F}_i \triangleq \mathcal{F}$
    3. Call NP oracle on $\mathcal{F}_i \wedge (x_i)$
    4. If answer is **yes**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (x_i)$
    5. If answer is **no**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (\neg x_i)$
    6. $i \leftarrow i + 1$
    7. If $i \leq n$, then repeat from 3.
  - Algorithm needs $|\mathrm{var}(\mathcal{F})|$ calls to an NP oracle
  - **Note:** Cannot solve FSAT with logarithmic number of NP oracle calls, unless $\mathrm{P} = \mathrm{NP}$ [GF93]

- FSAT is an example of a function problem
  - **Note:** FSAT can be solved with **one** SAT oracle call

Beyond decision problems

Answer	Problem Type

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems
Some solution	

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems
# solutions	

Beyond decision problems

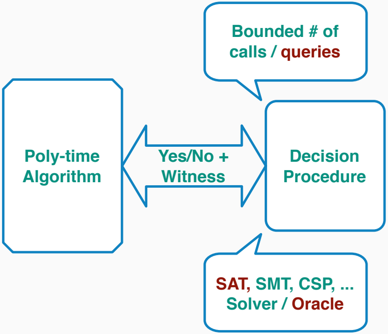
Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems
# solutions	Counting Problems

… and beyond NP – decision and function problems

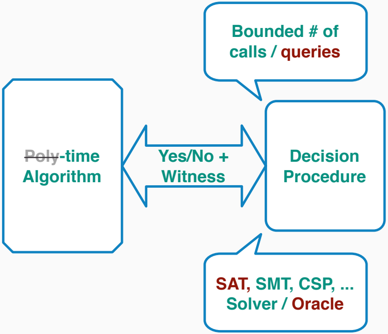
$$
\begin{tikzcd}
 & \vdots & \\
\Sigma_3^{\mathsf{p}} & & \Pi_3^{\mathsf{p}} \\
 & \Delta_3^{\mathsf{p}} \arrow[ul] \arrow[ur] & \\
\Sigma_2^{\mathsf{p}} \arrow[uu] \arrow[ur] & & \Pi_2^{\mathsf{p}} \arrow[uu] \arrow[ul] \\
 & \mathsf{P}^{\mathsf{NP}} = \Delta_2^{\mathsf{p}} \arrow[ul] \arrow[ur] & \\
\mathsf{NP} = \Sigma_1^{\mathsf{p}} \arrow[uu] \arrow[ur] & & \Pi_1^{\mathsf{p}} = \mathsf{coNP} \arrow[uu] \arrow[ul] \\
 & \Delta_0^{\mathsf{p}} = \Sigma_0^{\mathsf{p}} = \mathsf{P} = \Pi_0^{\mathsf{p}} = \Delta_1^{\mathsf{p}} \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

$$
\begin{tikzcd}
 & \vdots & \\
\mathsf{F}\Sigma_3^{\mathsf{p}} & & \mathsf{F}\Pi_3^{\mathsf{p}} \\
 & \mathsf{F}\Delta_3^{\mathsf{p}} \arrow[ul] \arrow[ur] & \\
\mathsf{F}\Sigma_2^{\mathsf{p}} \arrow[uu] \arrow[ur] & & \mathsf{F}\Pi_2^{\mathsf{p}} \arrow[uu] \arrow[ul] \\
 & \mathsf{FP}^{\mathsf{NP}} = \mathsf{F}\Delta_2^{\mathsf{p}} \arrow[ul] \arrow[ur] & \\
\mathsf{FNP} = \mathsf{F}\Sigma_1^{\mathsf{p}} \arrow[uu] \arrow[ur] & & \mathsf{F}\Pi_1^{\mathsf{p}} = \mathsf{coFNP} \arrow[uu] \arrow[ul] \\
 & \mathsf{F}\Delta_0^{\mathsf{p}} = \mathsf{F}\Sigma_0^{\mathsf{p}} = \mathsf{FP} = \mathsf{F}\Pi_0^{\mathsf{p}} = \mathsf{F}\Delta_1^{\mathsf{p}} \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

Oracle-based problem solving – simple scenario



Oracle-based problem solving – general setting



Many problems to solve – within $\mathsf{FP}^{\mathsf{NP}}$

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems

Many problems to solve – within $\mathrm{FP}^{\mathrm{NP}}$

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems

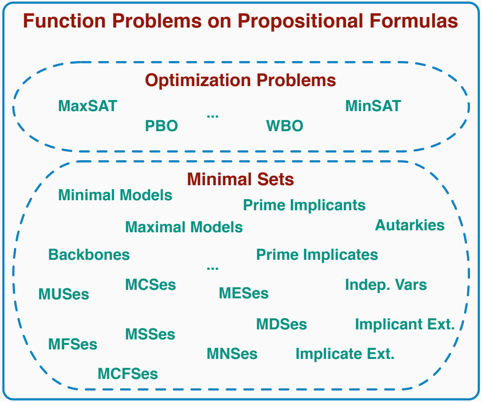
Function Problems on Propositional Formulas

MaxSAT **PBO** **...** **WBO** **MinSAT**

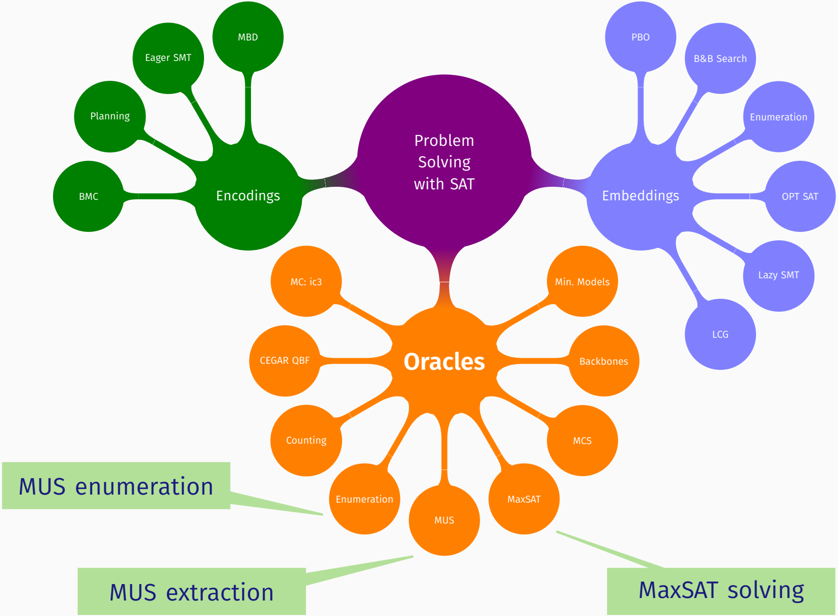
Minimal Models **Prime Implicants** **Maximal Models** **Autarkies** **Backbones** **Prime Implicates** **...** **MUSes** **MCSes** **MESes** **Indep. Vars** **MDSes** **Implicant Ext.** **MFSes** **MSSes** **MNSes** **Implicate Ext.** **MCFSes**

Many problems to solve – within $\mathrm{FP}^{\mathrm{NP}}$

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems



Selection of topics



Outline

Minimal Unsatisfiability

MUS Enumeration

Maximum Satisfiability

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?
- Minimal subset of constraints whose removal makes remaining constraints consistent?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?
- Minimal subset of constraints whose removal makes remaining constraints consistent?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?
- Minimal subset of constraints whose removal makes remaining constraints consistent?

- How to compute these **minimal** sets?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
… (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
… (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?
- Minimal subset of constraints whose removal makes remaining constraints consistent?
- How to compute these **minimal** sets?

Minimality matters!

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}(\vDash \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \vDash \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}(\vDash \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \vDash \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}(\vDash \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \vDash \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- Given $\mathcal{F}(\vDash \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \nvDash \bot$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \vDash \bot$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ is MSS

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}(\vDash \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \vDash \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- Given $\mathcal{F}(\vDash \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \nvDash \bot$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \vDash \bot$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ is MSS

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}(\vDash \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \vDash \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- Given $\mathcal{F}(\vDash \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \nvDash \bot$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \vDash \bot$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ is MSS

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- MUSes and MCSes are (subset-)**minimal** sets
- MUSes and minimal hitting sets of MCSes and vice-versa [Rei87, BS05]
  - Easy to see **why**

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}(\vDash \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \vDash \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- Given $\mathcal{F}(\vDash \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \nvDash \bot$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \vDash \bot$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ is MSS

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- MUSes and MCSes are (subset-)**minimal** sets
- MUSes and minimal hitting sets of MCSes and vice-versa [Rei87, BS05]
  - Easy to see **why**
- How to compute MUSes & MCSes **efficiently** with SAT oracles?

Why it matters?

- Analysis of over-constrained systems
  - Model-based diagnosis [Rei87]
    - Software fault localization
    - Spreadsheet debugging
    - Debugging relational specifications (e.g. Alloy)
    - Type error debugging
    - Axiom pinpointing in description logics
    - ...
  - Model checking of software & hardware systems
  - Inconsistency measurement
  - Minimal models; MinCost SAT; ...
  - ...
- Find minimal relaxations to recover consistency
  - But also minimum relaxations to recover consistency, eg. **MaxSAT**
- **Find minimal explanations of inconsistency**
  - But also minimum explanations of inconsistency, eg. Smallest MUS

Why it matters?

- Analysis of over-constrained systems
  - Model-based diagnosis [Rei87]
    - Software fault localization
    - Spreadsheet debugging
    - Debugging relational specifications (e.g. Alloy)
    - Type error debugging
    - Axiom pinpointing in description logics
    - ...
  - Model checking of software & hardware systems
  - Inconsistency measurement
  - Minimal models; MinCost SAT; ...
  - ...

Enumeration required!

- Find minimal relaxations to recover consistency
  - But also minimum relaxations to recover consistency, eg. **MaxSAT**
- **Find minimal explanations of inconsistency**
  - But also minimum explanations of inconsistency, eg. Smallest MUS

Deletion-based algorithm

Input : Set $\mathcal{F}$
Output: Minimal subset $\mathcal{M}$
begin
  $\mathcal{M} \leftarrow \mathcal{F}$
  foreach $c \in \mathcal{M}$ **do**
    if $\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\}$ // If $\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$, then $c \notin$ MUS
  return $\mathcal{M}$ // Final $\mathcal{M}$ is MUS
end

- Number of oracles calls: $\mathcal{O}(m)$ [CD91, BDTW93]

Deletion-based algorithm

Input : Set $\mathcal{F}$
Output: Minimal subset $\mathcal{M}$
begin
  $\mathcal{M} \leftarrow \mathcal{F}$
  foreach $c \in \mathcal{M}$ **do**
    if $\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\}$ // Remove c from $\mathcal{M}$
  return $\mathcal{M}$ // Final $\mathcal{M}$ is MUS
end

Monotonicity implicit & essential!

- Number of oracles calls: $\mathcal{O}(m)$

[CD91, BDTW93]

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathsf{SAT}(\mathcal{M} \setminus \{c\})$	Outcome

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg \mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3
$c_4..c_7$	$c_5..c_7$	0	Keep c_4

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\text{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3
$c_4..c_7$	$c_5..c_7$	0	Keep c_4
$c_4..c_7$	$c_4 c_6 c_7$	0	Keep c_5

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3
$c_4..c_7$	$c_5..c_7$	0	Keep c_4
$c_4..c_7$	$c_4 c_6 c_7$	0	Keep c_5
$c_4..c_7$	$c_4 c_5 c_7$	0	Keep c_6

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3
$c_4..c_7$	$c_5..c_7$	0	Keep c_4
$c_4..c_7$	$c_4 c_6 c_7$	0	Keep c_5
$c_4..c_7$	$c_4 c_5 c_7$	0	Keep c_6
$c_4..c_7$	$c_4..c_6$	1	Drop c_7

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\text{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3
$c_4..c_7$	$c_5..c_7$	0	Keep c_4
$c_4..c_7$	$c_4 c_6 c_7$	0	Keep c_5
$c_4..c_7$	$c_4 c_5 c_7$	0	Keep c_6
$c_4..c_7$	$c_4..c_6$	1	Drop c_7

- MUS: $\{c_4, c_5, c_6\}$

Many MUS algorithms

- Formula $\mathcal{F}$ with m clauses k the size of largest minimal subset

Algorithm	Oracle Calls	Reference
Insertion-based	$\mathcal{O}(k\,m)$	[dSNP88, vMW08]
MCS_MUS	$\mathcal{O}(k\,m)$	[BK15]
Deletion-based	$\mathcal{O}(m)$	[CD91, BDTW93]
Linear insertion	$\mathcal{O}(m)$	[MSL11, BLM12]
Dichotomic	$\mathcal{O}(k\,\log(m))$	[HLSB06]
QuickXplain	$\mathcal{O}(k + k\,\log(\frac{m}{k}))$	[Jun04]
Progression	$\mathcal{O}(k\,\log(1 + \frac{m}{k}))$	[MJB13]

- **Note:** Lower bound in $\mathrm{FP}^{\mathrm{NP}}_{||}$ and upper bound in $\mathrm{FP}^{\mathrm{NP}}$ [CT95]
- Oracle calls correspond to testing unsatisfiability with SAT solver
- Practical optimizations: clause set trimming; clause set refinement; redundancy removal; (recursive) model rotation

Outline

Minimal Unsatisfiability

MUS Enumeration

Maximum Satisfiability

How to enumerate MUSes?

How to enumerate MUSes?

1. Standard solution:
   Exploit HS duality between MCSes and MUSes [Rei87, LS08]

   MCSes are MHSes of MUSes and vice-versa

   - Enumerate **all** MCSes and then enumerate **all** MHSes of the MCSes, i.e. compute all the MUSes
   - Problematic if too many MCSes, and we want the MUSes
   - And, often we want to enumerate the MUSes

How to enumerate MUSes?

1. Standard solution:
   Exploit HS duality between MCSes and MUSes [Rei87, LS08]

   MCSes are MHSes of MUSes and vice-versa

   - Enumerate **all** MCSes and then enumerate **all** MHSes of the MCSes, i.e. compute all the MUSes
   - Problematic if too many MCSes, and we want the MUSes
   - And, often we want to enumerate the MUSes

2. Exploit recent advances in 2QBF solving

How to enumerate MUSes?

1. Standard solution:
   Exploit HS duality between MCSes and MUSes [Rei87, LS08]

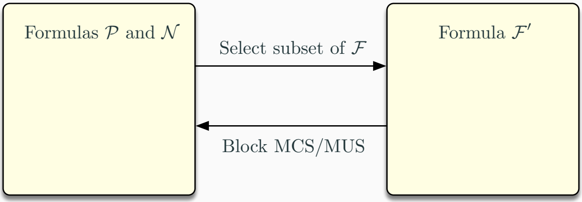
   MCSes are MHSes of MUSes and vice-versa

   - Enumerate **all** MCSes and then enumerate **all** MHSes of the MCSes, i.e. compute all the MUSes
   - Problematic if too many MCSes, and we want the MUSes
   - And, often we want to enumerate the MUSes

2. Exploit recent advances in 2QBF solving

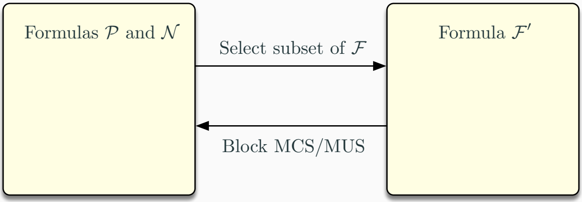
3. Implicit hitting set dualization [LPMM16]
   - Most effective if MUSes provided to user on-demand
   - Also used in prime enumeration, propositional abduction, logic synthesis, SMUS, quantification & XAI

How to enumerate MUSes, preferably?



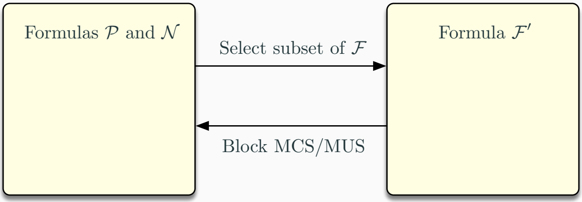
1. Keep sets representing computed MUSes (set $\mathcal{N}$) and MCSes (set $\mathcal{P}$)

How to enumerate MUSes, preferably?



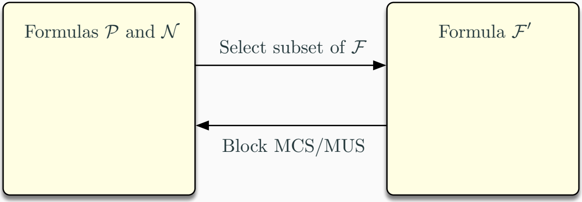
1. Keep sets representing computed MUSes (set $\mathcal{N}$) and MCSes (set $\mathcal{P}$)
2. Compute minimal hitting set (MHS) H of $\mathcal{N}$, subject to $\mathcal{P}$
    - Must **not** repeat MUSes
    - Must **not** repeat MCSes
    - Maximize clauses picked, i.e. prefer to check satisfiability on as **many** clauses as possible
    - If unsatisfiable: no more MUSes/MCSes to enumerate

How to enumerate MUSes, preferably?



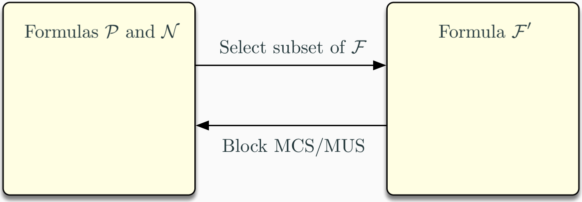
1. Keep sets representing computed MUSes (set $\mathcal{N}$) and MCSes (set $\mathcal{P}$)
2. Compute minimal hitting set (MHS) H of $\mathcal{N}$, subject to $\mathcal{P}$
   - Must **not** repeat MUSes
   - Must **not** repeat MCSes
   - Maximize clauses picked, i.e. prefer to check satisfiability on as **many** clauses as possible
   - If unsatisfiable: no more MUSes/MCSes to enumerate
3. Target set: $\mathcal{F}'$, i.e. $\mathcal{F}$ minus clauses from H

How to enumerate MUSes, preferably?



1. Keep sets representing computed MUSes (set $\mathcal{N}$) and MCSes (set $\mathcal{P}$)
2. Compute minimal hitting set (MHS) H of $\mathcal{N}$, subject to $\mathcal{P}$
   - Must **not** repeat MUSes
   - Must **not** repeat MCSes
   - Maximize clauses picked, i.e. prefer to check satisfiability on as **many** clauses as possible
   - If unsatisfiable: no more MUSes/MCSes to enumerate
3. Target set: $\mathcal{F}'$, i.e. $\mathcal{F}$ minus clauses from H
4. Run SAT oracle on $\mathcal{F}'$
   - If $\mathcal{F}'$ unsatisfiable: **extract new MUS**
   - Otherwise, H **is** already an **MCS of $\mathcal{F}$**

How to enumerate MUSes, preferably?



1. Keep sets representing computed MUSes (set $\mathcal{N}$) and MCSes (set $\mathcal{P}$)
2. Compute minimal hitting set (MHS) H of $\mathcal{N}$, subject to $\mathcal{P}$
    - Must **not** repeat MUSes
    - Must **not** repeat MCSes
    - Maximize clauses picked, i.e. prefer to check satisfiability on as **many** clauses as possible
    - If unsatisfiable: no more MUSes/MCSes to enumerate
3. Target set: $\mathcal{F}'$, i.e. $\mathcal{F}$ minus clauses from H
4. Run SAT oracle on $\mathcal{F}'$
    - If $\mathcal{F}'$ unsatisfiable: **extract new MUS**
    - Otherwise, H **is** already an **MCS of $\mathcal{F}$**
5. Repeat loop

MARCO/eMUS algorithm

Input: CNF formula $\mathcal{F}$

begin

  $I \leftarrow \{p_i \mid c_i \in \mathcal{F}\}$

  $(\mathcal{P}, \mathcal{N}) \leftarrow (\emptyset, \emptyset)$

  while true **do**

    $(st, H) \leftarrow \mathrm{MinHittingSet}(\mathcal{N}, \mathcal{P})$

    if not st **then return**

    $\mathcal{F}' \leftarrow \{c_i \mid p_i \in I \land p_i \notin H\}$

    if not $\mathrm{SAT}(\mathcal{F}')$ **then**

      $\mathcal{M} \leftarrow \mathrm{ComputeMUS}(\mathcal{F}')$

      ReportMUS $(\mathcal{M})$

      $\mathcal{N} \leftarrow \mathcal{N} \cup \{\neg p_i \mid c_i \in \mathcal{M}\}$

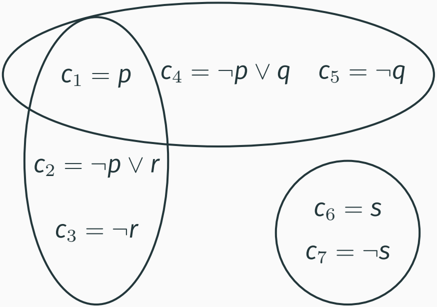
    else

      $\mathcal{P} \leftarrow \mathcal{P} \cup \{p_i \mid p_i \in H\}$

end

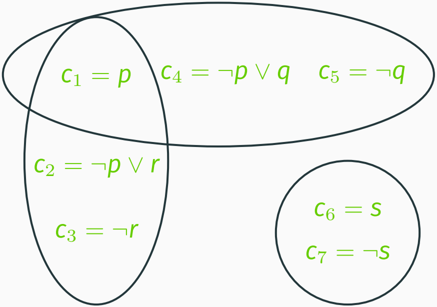
An example

MinHS ($\mathcal{N}$) $p_1p_2p_3p_4p_5p_6p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



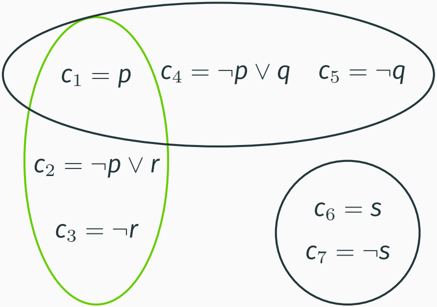
An example

MinHS ($\mathcal{N}$)	$\mathcal{F}'$	MUS/MCS
$p_1p_2p_3p_4p_5p_6p_7$	S/U	
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



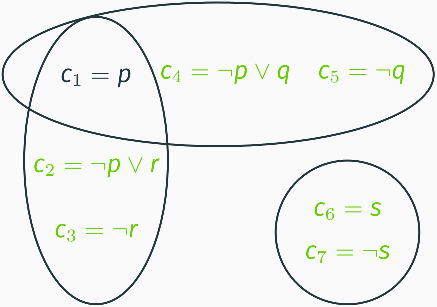
An example

MinHS ($\mathcal{N}$)	$\mathcal{F}'$	MUS/MCS
$p_1 p_2 p_3 p_4 p_5 p_6 p_7$	S/U	
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



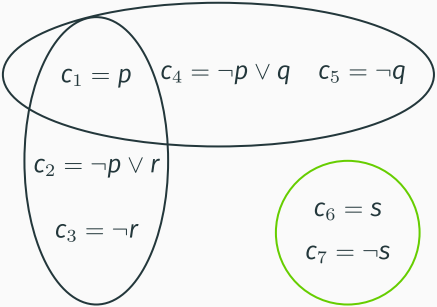
An example

MinHS ($\mathcal{N}$) $p_1 p_2 p_3 p_4 p_5 p_6 p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



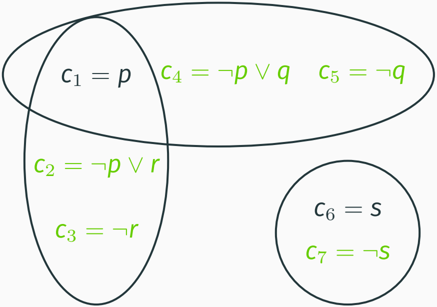
An example

MinHS ($\mathcal{N}$) $p_1 p_2 p_3 p_4 p_5 p_6 p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



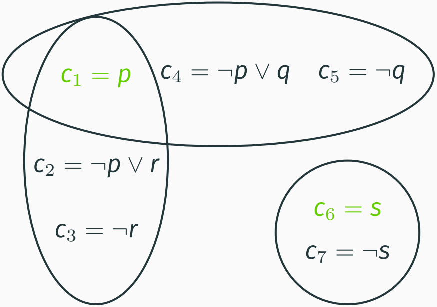
An example

MinHS ($\mathcal{N}$) $p_1 p_2 p_3 p_4 p_5 p_6 p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



An example

MinHS ($\mathcal{N}$) $p_1p_2p_3p_4p_5p_6p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



Outline

Minimal Unsatisfiability

MUS Enumeration

Maximum Satisfiability

Recap MaxSAT

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable

Recap MaxSAT

$\boxed{x_6 \vee x_2}$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\boxed{\neg x_4 \vee x_5}$
$\boxed{x_7 \vee x_5}$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula

Recap MaxSAT

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\boxed{\neg x_1}$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$\boxed{x_7 \vee x_5}$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest MCSes

Recap MaxSAT

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest MCSes
  - **Note**: Clauses can have weights & there can be hard clauses

Recap MaxSAT

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest **cost** MCSes
  - **Note**: Clauses can have weights & there can be hard clauses

Recap MaxSAT

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest **cost** MCSes
  - **Note**: Clauses can have weights & there can be hard clauses
- **Many** practical applications [SZGN17]

MaxSAT problem(s)

		Hard Clauses?	
		No	Yes
Weights?	No		
	Yes		

MaxSAT problem(s)

		Hard Clauses?	
		No	Yes
Weights?	No	Plain	Partial
	Yes	Weighted	Weighted Partial

MaxSAT problem(s)

		Hard Clauses?	
		No	Yes
Weights?	No	Plain	Partial
	Yes	Weighted	Weighted Partial

- Must satisfy hard clauses, if any
- Compute set of satisfied soft clauses with maximum cost
  - Without weights, cost of each falsified soft clause is 1
- **Or**, compute set of falsified soft clauses with minimum cost (s.t. hard & remaining soft clauses are satisfied)

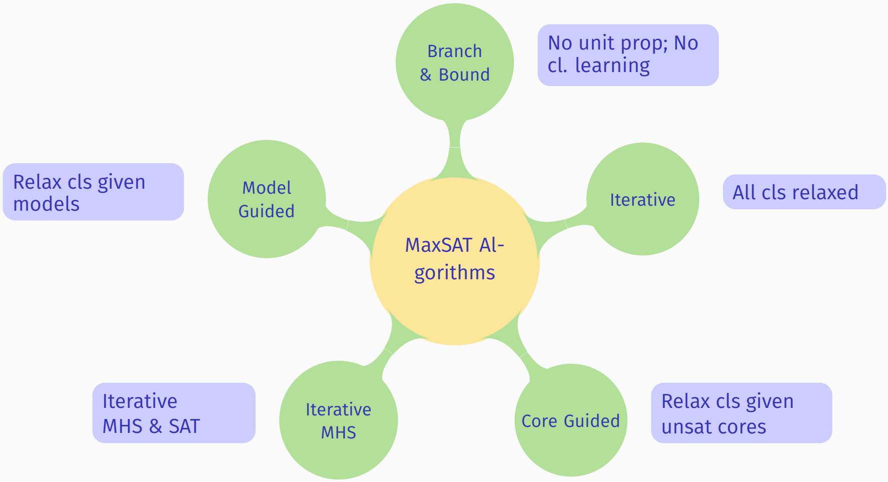
MaxSAT problem(s)

		Hard Clauses?	
		No	Yes
Weights?	No	Plain	Partial
	Yes	Weighted	Weighted Partial

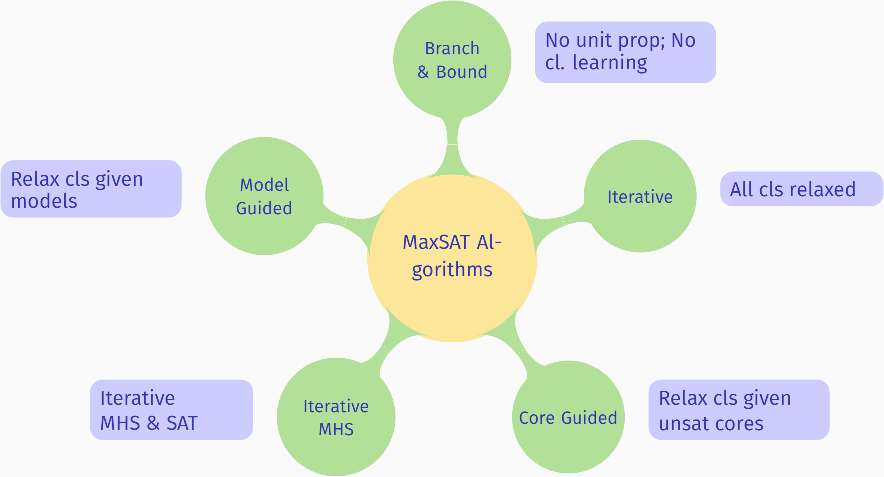
- Must satisfy hard clauses, if any
- Compute set of satisfied soft clauses with maximum cost
  - Without weights, cost of each falsified soft clause is 1
- **Or**, compute set of falsified soft clauses with minimum cost (s.t. hard & remaining soft clauses are satisfied)

- **Note**: goal is to compute **set** of satisfied (or falsified) clauses; **not** just the cost **!**

Many MaxSAT approaches



Many MaxSAT approaches



- For practical (**industrial**) instances: **core-guided** & **iterative MHS** approaches are the most effective [MaxSAT14]

Outline

Minimal Unsatisfiability

MUS Enumeration

Maximum Satisfiability

- Iterative SAT Solving
- Core-Guided Algorithms
- Minimum Hitting Sets

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

Example CNF formula

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$ $\quad$ $\neg x_6 \vee x_2 \vee r_2$ $\quad$ $\neg x_2 \vee x_1 \vee r_3$ $\quad$ $\neg x_1 \vee r_4$

$\neg x_6 \vee x_8 \vee r_5$ $\quad$ $x_6 \vee \neg x_8 \vee r_6$ $\quad$ $x_2 \vee x_4 \vee r_7$ $\quad$ $\neg x_4 \vee x_5 \vee r_8$

$x_7 \vee x_5 \vee r_9$ $\quad$ $\neg x_7 \vee x_5 \vee r_{10}$ $\quad$ $\neg x_5 \vee x_3 \vee r_{11}$ $\quad$ $\neg x_3 \vee r_{12}$

$\sum_{i=1}^{12} r_i \leq 12$

Relax all clauses; Set $UB = 12 + 1$

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$	$\neg x_6 \vee x_2 \vee r_2$	$\neg x_2 \vee x_1 \vee r_3$	$\neg x_1 \vee r_4$
$\neg x_6 \vee x_8 \vee r_5$	$x_6 \vee \neg x_8 \vee r_6$	$x_2 \vee x_4 \vee r_7$	$\neg x_4 \vee x_5 \vee r_8$
$x_7 \vee x_5 \vee r_9$	$\neg x_7 \vee x_5 \vee r_{10}$	$\neg x_5 \vee x_3 \vee r_{11}$	$\neg x_3 \vee r_{12}$

$\sum_{i=1}^{12} r_i \leq 12$

Formula is SAT; E.g. all $x_i = 0$ and $r_1 = r_7 = r_9 = 1$ (i.e. cost $= 3$)

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$ $\qquad \neg x_6 \vee x_2 \vee r_2$ $\qquad \neg x_2 \vee x_1 \vee r_3$ $\qquad \neg x_1 \vee r_4$

$\neg x_6 \vee x_8 \vee r_5$ $\qquad x_6 \vee \neg x_8 \vee r_6$ $\qquad x_2 \vee x_4 \vee r_7$ $\qquad \neg x_4 \vee x_5 \vee r_8$

$x_7 \vee x_5 \vee r_9$ $\qquad \neg x_7 \vee x_5 \vee r_{10}$ $\qquad \neg x_5 \vee x_3 \vee r_{11}$ $\qquad \neg x_3 \vee r_{12}$

$\sum_{i=1}^{12} r_i \leq 2$

Refine $UB = 3$

Basic MaxSAT with iterative SAT solving

$x_6 \lor x_2 \lor r_1$	$\neg x_6 \lor x_2 \lor r_2$	$\neg x_2 \lor x_1 \lor r_3$	$\neg x_1 \lor r_4$
$\neg x_6 \lor x_8 \lor r_5$	$x_6 \lor \neg x_8 \lor r_6$	$x_2 \lor x_4 \lor r_7$	$\neg x_4 \lor x_5 \lor r_8$
$x_7 \lor x_5 \lor r_9$	$\neg x_7 \lor x_5 \lor r_{10}$	$\neg x_5 \lor x_3 \lor r_{11}$	$\neg x_3 \lor r_{12}$

$\sum_{i=1}^{12} r_i \leq 2$

Formula is SAT; E.g. $x_1 = x_2 = 1$; $x_3 = \ldots = x_8 = 0$ and $r_4 = r_9 = 1$ (i.e. cost $= 2$)

Basic MaxSAT with iterative SAT solving

$x_6 \lor x_2 \lor r_1$ $\quad$ $\neg x_6 \lor x_2 \lor r_2$ $\quad$ $\neg x_2 \lor x_1 \lor r_3$ $\quad$ $\neg x_1 \lor r_4$

$\neg x_6 \lor x_8 \lor r_5$ $\quad$ $x_6 \lor \neg x_8 \lor r_6$ $\quad$ $x_2 \lor x_4 \lor r_7$ $\quad$ $\neg x_4 \lor x_5 \lor r_8$

$x_7 \lor x_5 \lor r_9$ $\quad$ $\neg x_7 \lor x_5 \lor r_{10}$ $\quad$ $\neg x_5 \lor x_3 \lor r_{11}$ $\quad$ $\neg x_3 \lor r_{12}$

$\sum_{i=1}^{12} r_i \leq 1$

Refine $UB = 2$

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$ $\quad \neg x_6 \vee x_2 \vee r_2$ $\quad \neg x_2 \vee x_1 \vee r_3$ $\quad \neg x_1 \vee r_4$

$\neg x_6 \vee x_8 \vee r_5$ $\quad x_6 \vee \neg x_8 \vee r_6$ $\quad x_2 \vee x_4 \vee r_7$ $\quad \neg x_4 \vee x_5 \vee r_8$

$x_7 \vee x_5 \vee r_9$ $\quad \neg x_7 \vee x_5 \vee r_{10}$ $\quad \neg x_5 \vee x_3 \vee r_{11}$ $\quad \neg x_3 \vee r_{12}$

$\sum_{i=1}^{12} r_i \leq 1$

Formula is UNSAT; terminate

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$ $\quad$ $\neg x_6 \vee x_2 \vee r_2$ $\quad$ $\neg x_2 \vee x_1 \vee r_3$ $\quad$ $\neg x_1 \vee r_4$

$\neg x_6 \vee x_8 \vee r_5$ $\quad$ $x_6 \vee \neg x_8 \vee r_6$ $\quad$ $x_2 \vee x_4 \vee r_7$ $\quad$ $\neg x_4 \vee x_5 \vee r_8$

$x_7 \vee x_5 \vee r_9$ $\quad$ $\neg x_7 \vee x_5 \vee r_{10}$ $\quad$ $\neg x_5 \vee x_3 \vee r_{11}$ $\quad$ $\neg x_3 \vee r_{12}$

$\sum_{i=1}^{12} r_i \leq 1$

MaxSAT solution is last satisfied UB: $UB = 2$

Basic MaxSAT with iterative SAT solving

$x_6 \lor x_2 \lor r_1$ $\quad$ $\neg x_6 \lor x_2 \lor r_2$ $\quad$ $\neg x_2 \lor x_1 \lor r_3$ $\quad$ $\neg x_1 \lor r_4$

$\neg x_6 \lor x_8 \lor r_5$ $\quad$ $x_6 \lor \neg x_8 \lor r_6$ $\quad$ $x_2 \lor x_4 \lor r_7$ $\quad$ $\neg x_4 \lor x_5 \lor r_8$

$x_7 \lor x_5 \lor r_9$ $\quad$ $\neg x_7 \lor x_5 \lor r_{10}$ $\quad$ $\neg x_5 \lor x_3 \lor r_{11}$ $\quad$ $\neg x_3 \lor r_{12}$

$$\sum_{i=1}^{12} r_i \leq 1$$

MaxSAT solution is last satisfied UB: $UB = 2$

AtMost$k$/PB constraints over **all** relaxation variables

All (possibly many) soft clauses relaxed

Outline

Minimal Unsatisfiability

MUS Enumeration

Maximum Satisfiability

- Iterative SAT Solving
- Core-Guided Algorithms
- Minimum Hitting Sets

MSU3 core-guided algorithm

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

Example CNF formula

MSU3 core-guided algorithm

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

Formula is UNSAT; $\mathrm{OPT} \leq |\varphi| - 1$; Get unsat core

MSU3 core-guided algorithm

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1 \vee r_1$	$\neg x_1 \vee r_2$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4 \vee r_3$	$\neg x_4 \vee x_5 \vee r_4$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3 \vee r_5$	$\neg x_3 \vee r_6$

$\sum_{i=1}^{6} r_i \leq 1$

Add relaxation variables and AtMost$k$, $k = 1$, constraint

MSU3 core-guided algorithm

$x_6 \vee x_2$ $\neg x_6 \vee x_2$ $\neg x_2 \vee x_1 \vee r_1$ $\neg x_1 \vee r_2$

$\neg x_6 \vee x_8$ $x_6 \vee \neg x_8$ $x_2 \vee x_4 \vee r_3$ $\neg x_4 \vee x_5 \vee r_4$

$x_7 \vee x_5$ $\neg x_7 \vee x_5$ $\neg x_5 \vee x_3 \vee r_5$ $\neg x_3 \vee r_6$

$\sum_{i=1}^{6} r_i \leq 1$

Formula is (again) UNSAT; OPT $\leq |\varphi| - 2$; Get unsat core

MSU3 core-guided algorithm

$x_6 \vee x_2 \vee r_7$ $\quad \neg x_6 \vee x_2 \vee r_8$ $\quad \neg x_2 \vee x_1 \vee r_1$ $\quad \neg x_1 \vee r_2$

$\neg x_6 \vee x_8$ $\quad x_6 \vee \neg x_8$ $\quad x_2 \vee x_4 \vee r_3$ $\quad \neg x_4 \vee x_5 \vee r_4$

$x_7 \vee x_5 \vee r_9$ $\quad \neg x_7 \vee x_5 \vee r_{10}$ $\quad \neg x_5 \vee x_3 \vee r_5$ $\quad \neg x_3 \vee r_6$

$\sum_{i=1}^{10} r_i \leq 2$

Add new relaxation variables and update AtMost$k$, k=2, constraint

MSU3 core-guided algorithm

$x_6 \vee x_2 \vee r_7$	$\neg x_6 \vee x_2 \vee r_8$	$\neg x_2 \vee x_1 \vee r_1$	$\neg x_1 \vee r_2$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4 \vee r_3$	$\neg x_4 \vee x_5 \vee r_4$
$x_7 \vee x_5 \vee r_9$	$\neg x_7 \vee x_5 \vee r_{10}$	$\neg x_5 \vee x_3 \vee r_5$	$\neg x_3 \vee r_6$

$\sum_{i=1}^{10} r_i \leq 2$

Instance is now SAT

MSU3 core-guided algorithm

$x_6 \vee x_2 \vee r_7$	$\neg x_6 \vee x_2 \vee r_8$	$\neg x_2 \vee x_1 \vee r_1$	$\neg x_1 \vee r_2$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4 \vee r_3$	$\neg x_4 \vee x_5 \vee r_4$
$x_7 \vee x_5 \vee r_9$	$\neg x_7 \vee x_5 \vee r_{10}$	$\neg x_5 \vee x_3 \vee r_5$	$\neg x_3 \vee r_6$

$\sum_{i=1}^{10} r_i \leq 2$

MaxSAT solution is $|\varphi| - \mathcal{I} = 12 - 2 = 10$

MSU3 core-guided algorithm

$x_6 \vee x_2 \vee r_7$	$\neg x_6 \vee x_2 \vee r_8$	$\neg x_2 \vee x_1 \vee r_1$	$\neg x_1 \vee r_2$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4 \vee r_3$	$\neg x_4 \vee x_5 \vee r_4$
$x_7 \vee x_5 \vee r_9$	$\neg x_7 \vee x_5 \vee r_{10}$	$\neg x_5 \vee x_3 \vee r_5$	$\neg x_3 \vee r_6$

$\sum_{i=1}^{10} r_i \leq 2$

MaxSAT solution is $|\varphi| - \mathcal{I} = 12 - 2 = 10$

AtMost$k$/PB constraints used

Relaxed soft clauses become **hard**

MSU3 core-guided algorithm

$x_6 \vee x_2 \vee r_7$ $\quad\neg x_6 \vee x_2 \vee r_8$ $\quad\neg x_2 \vee x_1 \vee r_1$ $\quad\neg x_1 \vee r_2$

$\boxed{\neg x_6 \vee x_8 \quad x_6 \vee \neg x_8}$ $\quad x_2 \vee x_4 \vee r_3$ $\quad\neg x_4 \vee x_5 \vee r_4$

$x_7 \vee x_5 \vee r_9$ $\quad\neg x_7 \vee x_5 \vee r_{10}$ $\quad\neg x_5 \vee x_3 \vee r_5$ $\quad\neg x_3 \vee r_6$

$\sum_{i=1}^{10} r_i \leq 2$

MaxSAT solution is $|\varphi| - \mathcal{I} = 12 - 2 = 10$

AtMost$k$/PB constraints used

Some clauses not relaxed

Relaxed soft clauses become **hard**

Outline

Minimal Unsatisfiability

MUS Enumeration

Maximum Satisfiability

- Iterative SAT Solving
- Core-Guided Algorithms
- Minimum Hitting Sets

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \emptyset$$

- Find MHS of $\mathcal{K}$:

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \emptyset$$

- Find MHS of $\mathcal{K}$: $\emptyset$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \emptyset$$

- Find MHS of $\mathcal{K}$: $\emptyset$
- SAT($\mathcal{F} \setminus \emptyset$)?

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

$$\mathcal{K} = \emptyset$$

- Find MHS of $\mathcal{K}$: $\emptyset$
- SAT($\mathcal{F} \setminus \emptyset$)? No

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$ $\quad c_2 = \neg x_6 \vee x_2$ $\quad c_3 = \neg x_2 \vee x_1$ $\quad c_4 = \neg x_1$

$c_5 = \neg x_6 \vee x_8$ $\quad c_6 = x_6 \vee \neg x_8$ $\quad c_7 = x_2 \vee x_4$ $\quad c_8 = \neg x_4 \vee x_5$

$c_9 = x_7 \vee x_5$ $\quad c_{10} = \neg x_7 \vee x_5$ $\quad c_{11} = \neg x_5 \vee x_3$ $\quad c_{12} = \neg x_3$

$$\mathcal{K} = \emptyset$$

- Find MHS of $\mathcal{K}$: $\emptyset$
- SAT($\mathcal{F} \setminus \emptyset$)? No
- Core of $\mathcal{F}$: $\{c_1, c_2, c_3, c_4\}$

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$ $\quad$ $c_2 = \neg x_6 \vee x_2$ $\quad$ $c_3 = \neg x_2 \vee x_1$ $\quad$ $c_4 = \neg x_1$

$c_5 = \neg x_6 \vee x_8$ $\quad$ $c_6 = x_6 \vee \neg x_8$ $\quad$ $c_7 = x_2 \vee x_4$ $\quad$ $c_8 = \neg x_4 \vee x_5$

$c_9 = x_7 \vee x_5$ $\quad$ $c_{10} = \neg x_7 \vee x_5$ $\quad$ $c_{11} = \neg x_5 \vee x_3$ $\quad$ $c_{12} = \neg x_3$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$: $\emptyset$
- SAT($\mathcal{F} \setminus \emptyset$)? No
- Core of $\mathcal{F}$: $\{c_1, c_2, c_3, c_4\}$. Update $\mathcal{K}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$:

MHS approach for MaxSAT

$$c_1 = x_6 \lor x_2 \qquad c_2 = \neg x_6 \lor x_2 \qquad c_3 = \neg x_2 \lor x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \lor x_8 \qquad c_6 = x_6 \lor \neg x_8 \qquad c_7 = x_2 \lor x_4 \qquad c_8 = \neg x_4 \lor x_5$$

$$c_9 = x_7 \lor x_5 \qquad c_{10} = \neg x_7 \lor x_5 \qquad c_{11} = \neg x_5 \lor x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1\}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \quad c_2 = \neg x_6 \vee x_2 \quad c_3 = \neg x_2 \vee x_1 \quad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \quad c_6 = x_6 \vee \neg x_8 \quad c_7 = x_2 \vee x_4 \quad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \quad c_{10} = \neg x_7 \vee x_5 \quad c_{11} = \neg x_5 \vee x_3 \quad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1\}$
- SAT($\mathcal{F} \setminus \{c_1\}$)?

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$ $\quad$ $c_2 = \neg x_6 \vee x_2$ $\quad$ $c_3 = \neg x_2 \vee x_1$ $\quad$ $c_4 = \neg x_1$

$c_5 = \neg x_6 \vee x_8$ $\quad$ $c_6 = x_6 \vee \neg x_8$ $\quad$ $c_7 = x_2 \vee x_4$ $\quad$ $c_8 = \neg x_4 \vee x_5$

$c_9 = x_7 \vee x_5$ $\quad$ $c_{10} = \neg x_7 \vee x_5$ $\quad$ $c_{11} = \neg x_5 \vee x_3$ $\quad$ $c_{12} = \neg x_3$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1\}$
- SAT($\mathcal{F} \setminus \{c_1\}$)? No

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1\}$
- SAT($\mathcal{F} \setminus \{c_1\}$)? No
- Core of $\mathcal{F}$: $\{c_9, c_{10}, c_{11}, c_{12}\}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1\}$
- SAT($\mathcal{F} \setminus \{c_1\}$)? No
- Core of $\mathcal{F}$: $\{c_9, c_{10}, c_{11}, c_{12}\}$. Update $\mathcal{K}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$:

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1, c_9\}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1, c_9\}$
- SAT($\mathcal{F} \setminus \{c_1, c_9\}$)?

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1, c_9\}$
- SAT($\mathcal{F} \setminus \{c_1, c_9\}$)? No

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$ $\quad c_2 = \neg x_6 \vee x_2$ $\quad c_3 = \neg x_2 \vee x_1$ $\quad c_4 = \neg x_1$

$c_5 = \neg x_6 \vee x_8$ $\quad c_6 = x_6 \vee \neg x_8$ $\quad c_7 = x_2 \vee x_4$ $\quad c_8 = \neg x_4 \vee x_5$

$c_9 = x_7 \vee x_5$ $\quad c_{10} = \neg x_7 \vee x_5$ $\quad c_{11} = \neg x_5 \vee x_3$ $\quad c_{12} = \neg x_3$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1, c_9\}$
- SAT($\mathcal{F} \setminus \{c_1, c_9\}$)? No
- Core of $\mathcal{F}$: $\{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1, c_9\}$
- SAT($\mathcal{F} \setminus \{c_1, c_9\}$)? No
- Core of $\mathcal{F}$: $\{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}$. Update $\mathcal{K}$

MHS approach for MaxSAT

$$c_1 = x_6 \lor x_2 \qquad c_2 = \neg x_6 \lor x_2 \qquad c_3 = \neg x_2 \lor x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \lor x_8 \qquad c_6 = x_6 \lor \neg x_8 \qquad c_7 = x_2 \lor x_4 \qquad c_8 = \neg x_4 \lor x_5$$

$$c_9 = x_7 \lor x_5 \qquad c_{10} = \neg x_7 \lor x_5 \qquad c_{11} = \neg x_5 \lor x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$:

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_4, c_9\}$

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$ $\quad$ $c_2 = \neg x_6 \vee x_2$ $\quad$ $c_3 = \neg x_2 \vee x_1$ $\quad$ $c_4 = \neg x_1$

$c_5 = \neg x_6 \vee x_8$ $\quad$ $c_6 = x_6 \vee \neg x_8$ $\quad$ $c_7 = x_2 \vee x_4$ $\quad$ $c_8 = \neg x_4 \vee x_5$

$c_9 = x_7 \vee x_5$ $\quad$ $c_{10} = \neg x_7 \vee x_5$ $\quad$ $c_{11} = \neg x_5 \vee x_3$ $\quad$ $c_{12} = \neg x_3$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_4, c_9\}$
- SAT($\mathcal{F} \setminus \{c_4, c_9\}$)?

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_4, c_9\}$
- SAT($\mathcal{F} \setminus \{c_4, c_9\}$)? Yes

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_4, c_9\}$
- SAT($\mathcal{F} \setminus \{c_4, c_9\}$)? Yes
- Terminate & return 2

MaxSAT solving with SAT oracles – a sample

- A sample of recent algorithms:

Algorithm	# Oracle Queries	Reference
Linear search SU	Exponential***	[BP10]
Binary search	Linear*	[FM06]
FM/WMSU1/WPM1	Exponential**	[FM06, MP08, MMSP09, ABL09, ABGL12]
WPM2	Exponential**	[ABL10, ABL13]
Bin-Core-Dis	Linear	[HMM11, MHM12]
Iterative MHS	Exponential	[DB11, DB13a, DB13b]

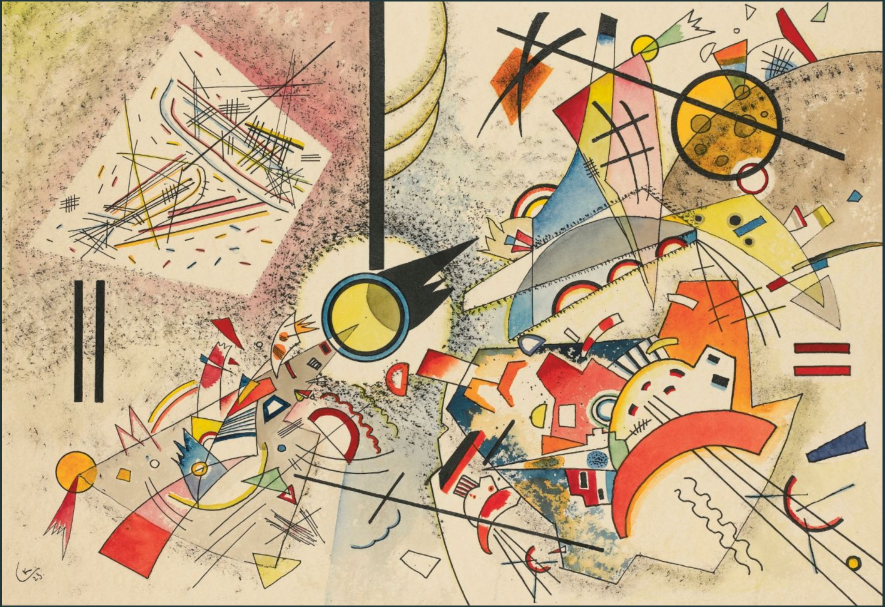
* $\mathcal{O}(\log m)$ queries with SAT oracle, for (partial) unweighted MaxSAT

** Weighted case; depends on computed cores

*** On # bits of problem instance (due to weights)

- But also additional recent work:
  - Progression [IMM+14]
  - Soft cardinality constraints (OLL) [MDM14, MIM14]
    - Recent implementation (RC2, using PySAT) won 2018 MaxSAT Evaluation
  - MaxSAT resolution [NB14]
  - ...

4 Sample of Applications



Flagship applications

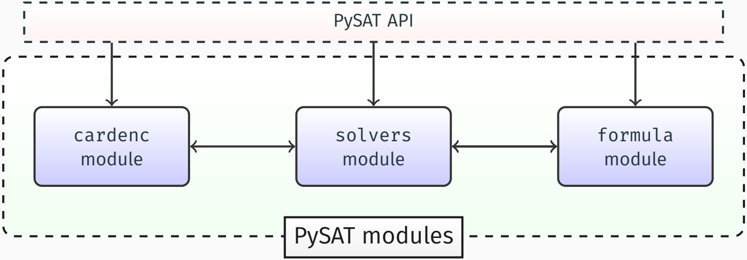
- Bounded (& unbounded) model checking
- Automated planning

- Software model checking
- Equivalence checking
- Package management
- Design debugging

- Haplotyping

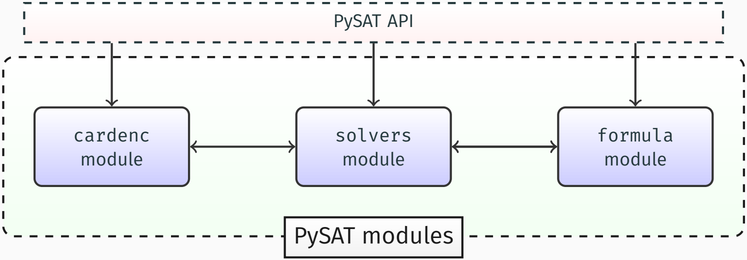
PySAT – SAT for all

[IMM18]



PySAT – SAT for all

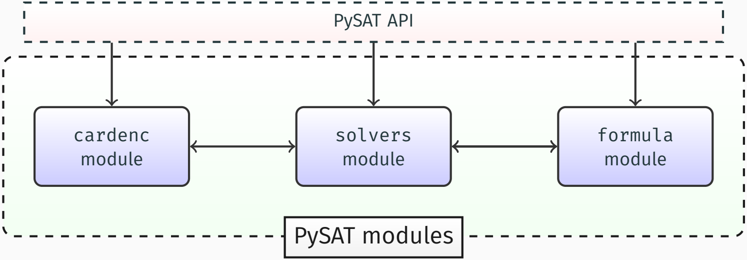
[IMM18]



- Open source, available on github
  - **URL**: `https://pysathq.github.io/`

PySAT – SAT for all

[IMM18]



- Open source, available on github
  - **URL**: `https://pysathq.github.io/`
- Comprehensive list of SAT solvers
- Comprehensive list of cardinality encodings
- Fairly comprehensive documentation
- Several use cases

Recent applications

- Two-level logic minimization with SAT [IPM15]
    - Reimplementation of Quine-McCluskey with SAT oracles

Recent applications

- Two-level logic minimization with SAT [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles
- Explainable decision sets [IPNM18]
  - Computation of smallest decision sets (rules)

Recent applications

- Two-level logic minimization with SAT [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles
- Explainable decision sets [IPNM18]
  - Computation of smallest decision sets (rules)
- Smallest (explainable) decision trees [NIPM18]
  - Computation of smallest decision trees

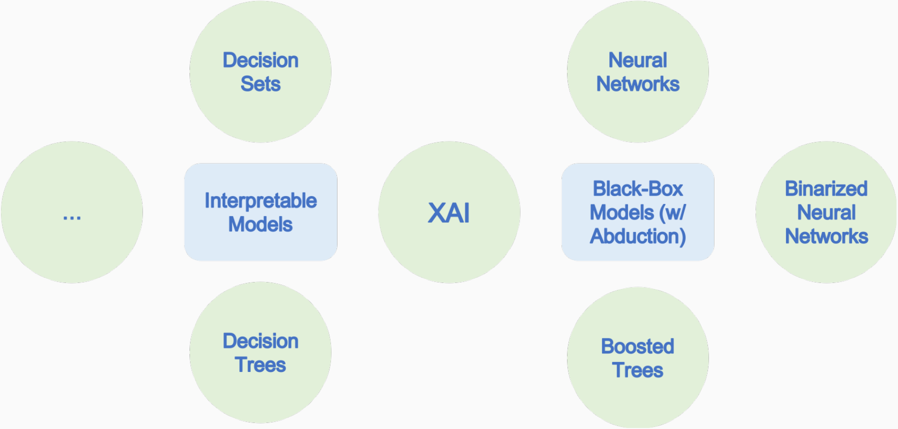
Recent applications

- Two-level logic minimization with SAT [IPM15]
    - Reimplementation of Quine-McCluskey with SAT oracles
- Explainable decision sets [IPNM18]
    - Computation of smallest decision sets (rules)
- Smallest (explainable) decision trees [NIPM18]
    - Computation of smallest decision trees
- Abduction-based explanations for ML models [INMS19]
    - On-demand extraction of explanations for any ML model
    - More applications in XAI (more later)

Recent applications

- Two-level logic minimization with SAT [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles
- Explainable decision sets [IPNM18]
  - Computation of smallest decision sets (rules)
- Smallest (explainable) decision trees [NIPM18]
  - Computation of smallest decision trees
- Abduction-based explanations for ML models [INMS19]
  - On-demand extraction of explanations for any ML model
  - More applications in XAI (more later)
- Lots of other applications, by us & by others

SAT (& SMT) meet(s) eXplainable AI



Smallest decision trees – encoding sizes in bytes

[NIPM18]

Model	Weather	Mouse	Cancer	Car	Income
CP'09*	27K	3.5M	92G	842M	354G

Smallest decision trees – encoding sizes in bytes

[NIPM18]

Model	Weather	Mouse	Cancer	Car	Income
CP’09*	27K	3.5M	92G	842M	354G
IJCAI’18	190K	1.2M	5.2M	4.1M	1.2G

Abduction-based explanations

[INMS19]

- Positive:
  - General approach, applicable to **any** ML model represented as a set of constraints
  - E.g. ability to explain predictions of NNs
- Negative:
  - NN sizes are fairly small, i.e. tens of neurons
  - Best results with ILP-based approach
    - SMT/SAT models currently ineffective
    - But, algorithms inspired SAT-based solutions

Abduction-based explanations

[INMS19]

- Positive:
  - General approach, applicable to **any** ML model represented as a set of constraints
  - E.g. ability to explain predictions of NNs
- Negative:
  - NN sizes are fairly small, i.e. tens of neurons
  - Best results with ILP-based approach
    - SMT/SAT models currently ineffective
    - But, algorithms inspired SAT-based solutions
- So, where is **SAT** used?

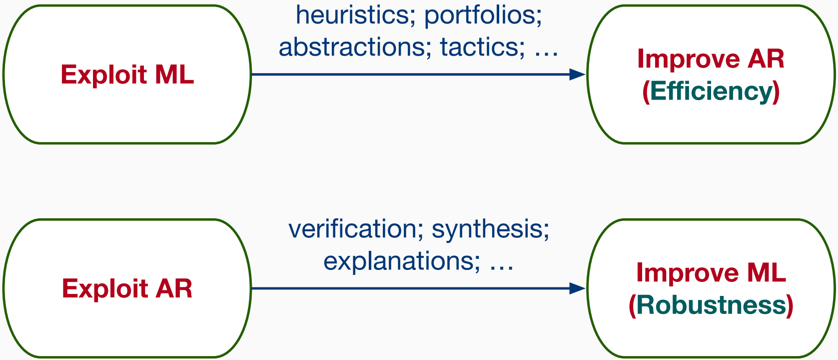
Abduction-based explanations

[INMS19]

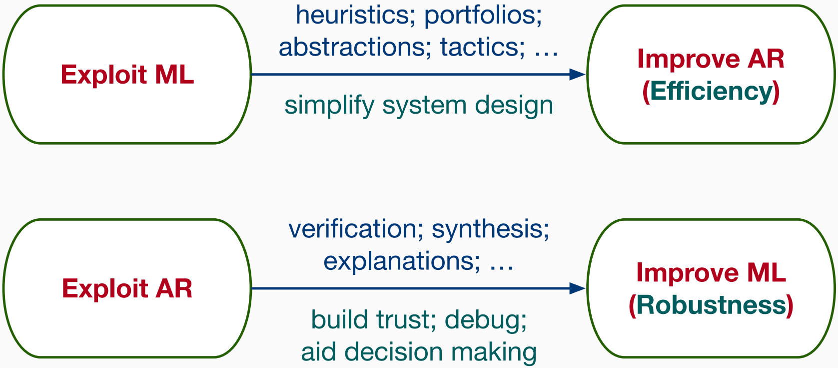
- Positive:
  - General approach, applicable to **any** ML model represented as a set of constraints
  - E.g. ability to explain predictions of NNs
- Negative:
  - NN sizes are fairly small, i.e. tens of neurons
  - Best results with ILP-based approach
    - SMT/SAT models currently ineffective
    - But, algorithms inspired SAT-based solutions
- So, where is **SAT** used?
  - Computing primes, with **SAT**-inspired algorithms
    - In general, oracle-based problem solving
  - Modeling NNs & boosted trees with **SMT**
  - Modeling BNNs with **SAT**

Machine learning vs. automated reasoning

Machine learning vs. automated reasoning



Machine learning vs. automated reasoning



5 SAT Future

Is there a future for SAT?

Is there a future for SAT? Yes!

Is there a future for SAT? Yes!

- Better solvers (always!) needed
    - Even if pace of improvement is modest

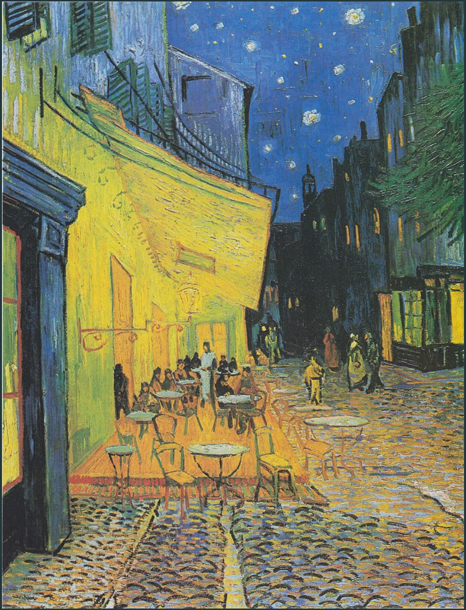
Is there a future for SAT? Yes!

- Better solvers (always!) needed
  - Even if pace of improvement is modest
- SAT-based problem solving is here to stay
  - And with high-profile applications, e.g. XAI

Is there a future for SAT? Yes!

- Better solvers (always!) needed
  - Even if pace of improvement is modest
- SAT-based problem solving is here to stay
  - And with high-profile applications, e.g. XAI
- Novel modular reasoning insights are the part of the future
  - A prediction:
    The future will see widespread uses of SAT-enabled modular reasoners

Questions?



References i

[ABGL12] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy.
Improving SAT-based weighted MaxSAT solvers.
In *CP*, pages 86–101, 2012.

[ABL09] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy.
Solving (weighted) partial MaxSAT through satisfiability testing.
In *SAT*, pages 427–440, 2009.

[ABL10] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy.
A new algorithm for weighted partial MaxSAT.
In *AAAI*, 2010.

[ABL13] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy.
SAT-based MaxSAT algorithms.
Artif. Intell., 196:77–105, 2013.

[AS09] Gilles Audemard and Laurent Simon.
Predicting learnt clauses quality in modern SAT solvers.
In *IJCAI*, pages 399–404, 2009.

References ii

[BDTW93] R. R. Bakker, F. Dikker, F. Tempelman, and P. M. Wognum.
Diagnosing and solving over-determined constraint satisfaction problems.
In *IJCAI*, pages 276–281, 1993.

[Bie08] Armin Biere.
PicoSAT essentials.
JSAT, 4(2-4):75–97, 2008.

[BK15] Fahiem Bacchus and George Katsirelos.
Using minimal correction sets to more efficiently compute minimal unsatisfiable sets.
In *CAV (2)*, volume 9207 of *Lecture Notes in Computer Science*, pages 70–86. Springer, 2015.

[BLM12] Anton Belov, Inês Lynce, and Joao Marques-Silva.
Towards efficient MUS extraction.
AI Commun., 25(2):97–116, 2012.

[BMS00] Luís Baptista and Joao Marques-Silva.
Using randomization and learning to solve hard real-world instances of satisfiability.
In *CP*, volume 1894 of *Lecture Notes in Computer Science*, pages 489–494. Springer, 2000.

[BP10] Daniel Le Berre and Anne Parrain.
The Sat4j library, release 2.2.
JSAT, 7(2-3):59–6, 2010.

[BS05] James Bailey and Peter J. Stuckey.
Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization.
In *PADL*, pages 174–186, 2005.

[CD91] John W. Chinneck and Erik W. Dravnieks.
Locating minimal infeasible constraint sets in linear programs.
INFORMS Journal on Computing, 3(2):157–168, 1991.

[CT95] Zhi-Zhong Chen and Seinosuke Toda.
The complexity of selecting maximal solutions.
Inf. Comput., 119(2):231–239, 1995.

[DB11] Jessica Davies and Fahiem Bacchus.
Solving MAXSAT by solving a sequence of simpler SAT instances.
In *CP*, pages 225–239, 2011.

[DB13a] Jessica Davies and Fahiem Bacchus.
Exploiting the power of MIP solvers in MAXSAT.
In *SAT*, pages 166–181, 2013.

[DB13b] Jessica Davies and Fahiem Bacchus.
Postponing optimization to speed up MAXSAT solving.
In *CP*, pages 247–262, 2013.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland.
A machine program for theorem-proving.
Commun. ACM, 5(7):394–397, 1962.

[DP60] Martin Davis and Hilary Putnam.
A computing procedure for quantification theory.
J. ACM, 7(3):201–215, 1960.

[dSNP88] J. L. de Siqueira N. and Jean-Francois Puget.
Explanation-based generalisation of failures.
In *ECAI*, pages 339–344, 1988.

[FM06] Zhaohui Fu and Sharad Malik.
On solving the partial MAX-SAT problem.
In *SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006.

[Gel09] Allen Van Gelder.
Improved conflict-clause minimization leads to improved propositional proof traces.
In *SAT*, pages 141–146, 2009.

[GF93] Georg Gottlob and Christian G. Fermüller.
Removing redundancy from a clause.
Artif. Intell., 61(2):263–289, 1993.

[GN02] Evguenii I. Goldberg and Yakov Novikov.
BerkMin: A fast and robust SAT-solver.
In *DATE*, pages 142–149. IEEE Computer Society, 2002.

[GSC97] Carla P. Gomes, Bart Selman, and Nuno Crato.
Heavy-tailed distributions in combinatorial search.
In *CP*, volume 1330 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 1997.

[HLSB06] Fred Hemery, Christophe Lecoutre, Lakhdar Sais, and Frédéric Boussemart.
Extracting MUCs from constraint networks.
In *ECAI*, pages 113–117, 2006.

[HMM11] Federico Heras, António Morgado, and Joao Marques-Silva.
Core-guided binary search algorithms for maximum satisfiability.
In *AAAI*. AAAI Press, 2011.

[Hua07] Jinbo Huang.
The effect of restarts on the efficiency of clause learning.
In *IJCAI*, pages 2318–2323, 2007.

[IMM+14] Alexey Ignatiev, António Morgado, Vasco M. Manquinho, Inês Lynce, and Joao Marques-Silva.
Progression in maximum satisfiability.
In *ECAI*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 453–458. IOS Press, 2014.

[IMM18] Alexey Ignatiev, António Morgado, and Joao Marques-Silva.
PySAT: A python toolkit for prototyping with SAT oracles.
In *SAT*, volume 10929 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018.

[INMS19] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva.
Abduction-based explanations for machine learning models.
In *AAAI*, 2019.

[IPM15] Alexey Ignatiev, Alessandro Previti, and Joao Marques-Silva.
SAT-based formula simplification.
In *SAT*, volume 9340 of *Lecture Notes in Computer Science*, pages 287–298. Springer, 2015.

[IPNM18] Alexey Ignatiev, Filipe Pereira, Nina Narodytska, and João Marques-Silva.
A SAT-based approach to learn explainable decision sets.
In *IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 627–645. Springer, 2018.

[Jun04] Ulrich Junker.
QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems.
In *AAAI*, pages 167–172, 2004.

[LLX+17] Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, and Zhipeng Lü.
An effective learnt clause minimization approach for CDCL SAT solvers.
In *IJCAI*, pages 703–711, 2017.

[LOM+18] Jia Hui Liang, Chanseok Oh, Minu Mathew, Ciza Thomas, Chunxiao Li, and Vijay Ganesh.
Machine learning-based restart policy for CDCL SAT solvers.
In *SAT*, pages 94–110, 2018.

[LPMM16] Mark H. Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques-Silva.
Fast, flexible MUS enumeration.
Constraints, 21(2):223–250, 2016.

[LS08] Mark H. Liffiton and Karem A. Sakallah.
Algorithms for computing minimal unsatisfiable subsets of constraints.
J. Autom. Reasoning, 40(1):1–33, 2008.

[MDM14] António Morgado, Carmine Dodaro, and Joao Marques-Silva.
Core-guided MaxSAT with soft cardinality constraints.
In *CP*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014.

[MHM12] António Morgado, Federico Heras, and João Marques-Silva.
Improvements to core-guided binary search for MaxSAT.
In *SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 284–297. Springer, 2012.

[MIM14] António Morgado, Alexey Ignatiev, and João Marques-Silva.
MSCG: robust core-guided MaxSAT solving.
JSAT, 9:129–134, 2014.

[MJB13] Joao Marques-Silva, Mikolás Janota, and Anton Belov.
Minimal sets over monotone predicates in boolean formulae.
In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 592–607. Springer, 2013.

[MMSP09] Vasco M. Manquinho, Joao Marques-Silva, and Jordi Planes.
Algorithms for weighted boolean optimization.
In *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
Chaff: Engineering an efficient SAT solver.
In *DAC*, pages 530–535. ACM, 2001.

[MP08] Joao Marques-Silva and Jordi Planes.
Algorithms for maximum satisfiability using unsatisfiable cores.
In *DATE*, pages 408–413. ACM, 2008.

References x

[MS95] J. Marques-Silva.
Search Algorithms for Satisfiability Problems in Combinational Switching Circuits.
PhD thesis, University of Michigan, May 1995.

[MSL11] Joao Marques-Silva and Inês Lynce.
On improving MUS extraction algorithms.
In *SAT*, volume 6695 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2011.

[MSS96] Joao Marques-Silva and Karem A. Sakallah.
GRASP - a new search algorithm for satisfiability.
In *ICCAD*, pages 220–227, 1996.

[MSS99] Joao Marques-Silva and Karem A. Sakallah.
GRASP: A search algorithm for propositional satisfiability.
IEEE Trans. Computers, 48(5):506–521, 1999.

[NB14] Nina Narodytska and Fahiem Bacchus.
Maximum satisfiability using core-guided maxsat resolution.
In *AAAI*, pages 2717–2723. AAAI Press, 2014.

[NIPM18] Nina Narodytska, Alexey Ignatiev, Filipe Pereira, and Joao Marques-Silva.
Learning optimal decision trees with SAT.
In *IJCAI*, pages 1362–1368, 2018.

[PD07] Knot Pipatsrisawat and Adnan Darwiche.
A lightweight component caching scheme for satisfiability solvers.
In *SAT*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, 2007.

[Rei87] Raymond Reiter.
A theory of diagnosis from first principles.
Artif. Intell., 32(1):57–95, 1987.

[SB09] Niklas Sörensson and Armin Biere.
Minimizing learned clauses.
In *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 237–243. Springer, 2009.

[SSS12] Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann.
Learning back-clauses in SAT.
In *SAT*, pages 498–499, 2012.

[SZGN17] Xujie Si, Xin Zhang, Radu Grigore, and Mayur Naik.
Maximum satisfiability in software analysis: Applications and techniques.
In *CAV*, pages 68–94, 2017.

[vMW08] Hans van Maaren and Siert Wieringa.
Finding guaranteed MUSes fast.
In *SAT*, pages 291–304, 2008.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik.
Efficient conflict driven learning in boolean satisfiability solver.
In *ICCAD*, pages 279–285. IEEE Computer Society, 2001.