Beyond Traditional SAT Reasoning: QBF, Model Counting, and Solution Sampling

Ashish Sabharwal and Bart Selman

Cornell University

July, 2007

AAAI Conference
Vancouver, BC

Tutorial Roadmap

PART I: Automated Reasoning

The Quest for Machine Reasoning

Objective:

Develop foundations and technology to enable effective, practical, large-scale automated reasoning.

Machine Reasoning (1960-90s) → **Current reasoning technology**

Computational complexity of reasoning *appears* to severely limit real-world applications

Revisiting the challenge: Significant progress with new ideas / tools for dealing with **complexity (scale-up), uncertainty, and multi-agent reasoning**

General Automated Reasoning

Problem instance → Model Generator (Encoder) → General Inference Engine → Solution

Domain-specific
e.g. logistics, chess, planning, scheduling, ...

Generic
applicable to all domains
within range of modeling language

Research objective
Better reasoning and modeling technology

Impact
Faster solutions in several domains

Reasoning Complexity

Simple Example:

Knowledge Base

Variables (binary)
X1 = email_ received
X2 = in_ meeting
X3 = urgent
X4 = respond_to_email

X5 = near_deadline
X6 = postpone

X7 = air_ticket_info_request
X8 = travel_ request
X9 = info_request

Rules:
1. X1 & (not X2) & X3 ⇒ X4
2. X2 ⇒ not X4
3. X5 ⇒ X3 or X6
4. X7 ⇒ X8
5. X8 ⇒ X9
6. X8 ⇒ X5
7. X6 ⇒ not X9

- **EXPONENTIAL COMPLEXITY: INHERENT**
  A^N worst case
  N= No. of Variables/Objects A= Object states
- **TIME/SPACE**
  ↑Granularity → ↑ Object states
- **Current implementations trade time with soundness**

Question: Given: X1= true; X2 = false; X7=true.
What is X4 = ?

Answer Development: Inference Chain

Search for rules to apply

Step 1: X7 ⇒ X8 (rule 4)
Step 2: X8 ⇒ X5 (rule 6)
Step 3: X5 ⇒ X3 or X6 (rule 3)

M

Case A: X6 = true
Step 4: X6 ⇒ not X9
Step 5: X9 ⇒ not X8
Step 6: Contradiction
Backtrack to M

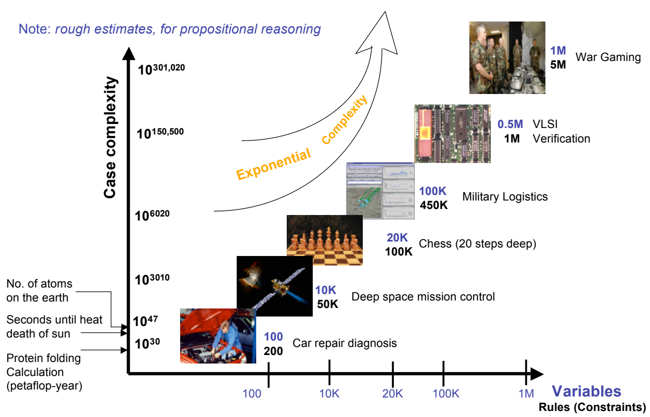
Case B: X3 = true
X1 & (not X2) & X3 ⇒ X4
Step 7: X4 = true (Rule 1)

Check Contradictions

For N variables: 2^N cases drive complexity!

Exponential Complexity Growth: The Challenge of Complex Domains

Note: *rough estimates, for propositional reasoning*



[Credit: Kumar, DARPA; Cited in Computer World magazine]

Tutorial Roadmap

1. **Automated Reasoning**
   – The complexity challenge
   – State of the art in Boolean reasoning
   – Boolean logic, expressivity
2. QBF Reasoning
   – A new range of applications
   – Quantified Boolean logic
   – Solution techniques overview
   – Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. Model Counting
   – Connection with sampling
   – A new range of applications
   – Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   – Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

Progress in Last 15 Years

Focus: Combinatorial Search Spaces
Specifically, the Boolean satisfiability problem, SAT

Significant progress since the 1990's.

How much?

- Problem size: **We went from 100 variables, 200 constraints (early 90's) to 1,000,000 vars. and 5,000,000 constraints in 15 years**.

  Search space: from 10^15 to 10^300,000.
  [Aside: "one can encode quite a bit in 1M variables."]

- Tools: 50+ competitive SAT solvers available

Overview of the state of the art:
Plenary talk at IJCAI-05 (Selman); Discrete App. Math. article (Kautz-Selman '06)

9

How Large are the Problems?

A bounded model checking problem:

From "SATLIB":

http://www.satlib.org/benchm.html

SAT-encoded bounded model checking instances
(contributed by Ofer Shtrichman)

In Bounded Model Checking (BMC) [BCCZ99], a rather newly introduced problem in formal methods, the task is to check whether a given model M (typically a hardware design) satisfies a temporal property P in all paths with length less or equal to some bound k. The BMC problem can be efficiently reduced to a propositional satisfiability problem, and in fact if the property is in the form of an invariant (Invariants are the most common type of properties, and many other temporal properties can be reduced to their form. It has the form of 'it is always true that ... '.), it has a structure which is similar to many AI planning problems.

10

SAT Encoding

(automatically generated from problem specification)

The instance `bmc-ibm-6.cnf`, IBM LSU 1997:

```
p cnf 51639 368352
-1 7 0
-1 6 0
-1 5 0
-1 -4 0
-1 3 0
-1 2 0
-1 -8 0
-9 15 0
-9 14 0
-9 13 0
-9 -12 0
-9 11 0
-9 10 0
-9 -16 0
-17 23 0
-17 22 0
```

i.e., ((not x_1) or x_7)
((not x_1) or x_6)
etc.

x_1, x_2, x_3, etc. are our Boolean variables (to be set to True or False)

Should x_1 be set to False??

11

10 Pages Later:

```
185 -9 0
185 -1 0
177 169 161 153 145 137 129 121 113 105 97
 89 81 73 65 57 49 41
 33 25 17 9 1 -185 0
186 -187 0
186 -188 0
...
```

i.e., (x_{177} or x_{169} or x_{161} or x_{153} … x_{33} or x_{25} or x_{17} or x_9 or x_1 or (not x_{185}))

clauses / constraints are getting more interesting…

Note x_1 …

12

4,000 Pages Later:

10236 −10050 0
10236 −10051 0
10236 −10235 0
10008 10009 10010 10011 10012 10013 10014
10015 10016 10017 10018 10019 10020 10021
10022 10023 10024 10025 10026 10027 10028
10029 10030 10031 10032 10033 10034 10035
10036 10037 10086 10087 10088 10089 10090
10091 10092 10093 10094 10095 10096 10097
10098 10099 10100 10101 10102 10103 10104
10105 10106 10107 10108 −55 −54 53 −52 −51 50
10047 10048 10049 10050 10051 10235 −10236 0
10237 −10008 0
10237 −10009 0
10237 −10010 0

…

13

Finally, 15,000 Pages Later:

−7 260 0
7 −260 0
1072 1070 0
−15 −14 −13 −12 −11 −10 0
−15 −14 −13 −12 −11 10 0
−15 −14 −13 −12 11 −10 0
−15 −14 −13 −12 11 10 0
−7 −6 −5 −4 −3 −2 0
−7 −6 −5 −4 −3 2 0
−7 −6 −5 −4 3 −2 0
−7 −6 −5 −4 3 2 0
185 0

Search space of truth assignments: $2^{50000} \approx 3.160699437 \cdot 10^{15051}$

Current SAT solvers solve this instance in under 30 seconds!

14

SAT Solver Progress

Solvers have continually improved over time

Instance	Posit' 94	Grasp' 96	Sato' 98	Chaff' 01
ssa2670-136	40.66s	1.20s	0.95s	0.02s
bf1355-638	1805.21s	0.11s	0.04s	0.01s
pret150_25	>3000s	0.21s	0.09s	0.01s
dubois100	>3000s	11.85s	0.08s	0.01s
aim200-2_0-no-1	>3000s	0.01s	< 0.01s	< 0.01s
2dlx_..._bug005	>3000s	>3000s	>3000s	2.90s
c6288	>3000s	>3000s	>3000s	>3000s

Source: Marques-Silva 2002

15

How do SAT Solvers Keep Improving?

From academically interesting to practically relevant.

We now have regular SAT solver competitions.
(Germany '89, Dimacs '93, China '96, SAT-02, SAT-03, …, SAT-07)

E.g. at SAT-2006 (Seattle, Aug '06):
- 35+ solvers submitted, most of them open source
- 500+ industrial benchmarks
- 50,000+ benchmark instances available on the www

This constant improvement in SAT solvers is the key to making, e.g., SAT-based planning very successful.

16

Current Automated Reasoning Tools

Most-successful fully automated methods:
based on Boolean Satisfiability (SAT) / Propositional Reasoning

– Problems modeled as rules / constraints over Boolean variables
– "SAT solver" used as the inference engine

Applications: *single-agent search*

- AI planning
  SATPLAN-06, **fastest optimal planner**;
  ICAPS-06 competition (Kautz & Selman '06)

- Verification – hardware and software
  Major groups at Intel, IBM, Microsoft, and universities such as CMU, Cornell, and Princeton.
  SAT has become the dominant technology.

- Many other domains: Test pattern generation, Scheduling, Optimal Control, Protocol Design, Routers, Multi-agent systems, E-Commerce (E-auctions and electronic trading agents), etc.

Tutorial Roadmap

1. **Automated Reasoning**
   – The complexity challenge
   – State of the art in Boolean reasoning
   – Boolean logic, expressivity
2. QBF Reasoning
   – A new range of applications
   – Quantified Boolean logic
   – Solution techniques overview
   – Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. Model Counting
   – Connection with sampling
   – A new range of applications
   – Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   – Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

Boolean Logic

Defined over Boolean (binary) variables a, b, c, …
Each of these can be True (1, T) or False (0, F)

Variables connected together with logic operators: and, or, not (denoted $\neg$)
E.g. $((c \wedge \neg d) \vee f)$ is True iff
either c is True and d is False, or f is True

Fact: All other Boolean logic operators can be expressed with and, or, not
E.g. $(a \Rightarrow b)$ same as $(\neg a \text{ or } b)$

Boolean formula, e.g. F = (a or b) and $\neg$(a and (b or c))

(Truth) Assignment: any setting of the variables to True or False
Satisfying assignment: assignment where the formula evaluates to True
E.g. F has 3 satisfying assignments: (0,1,0), (0,1,1), (1,0,0)

Boolean Logic: Expressivity

All discrete single-agent search problems can be cast as a Boolean formula

Variables a, b, c, … often represent "states" of the system, "events", "actions", etc.
(more on this later, using Planning as an example)

Very general encoding language. E.g. can handle
- Numbers (k-bit binary representation)
- Floating-point numbers
- Arithmetic operators like +, x, exp(), log()
- …

SAT encodings (generated automatically from high level languages) routinely used in domains like planning, scheduling, verification, e-commerce, network design, …

Recall Example:

Variables
X1 = email_ received "event"
X2 = in_ meeting "state"
X3 = urgent
X4 = respond_to_email "action"
X5 = near_deadline
X6 = postpone

X7 = air_ticket_info_request
X8 = travel_ request
X9 = info_request

Rules:
1. X1 & (not X2) & X3 $\Rightarrow$ X4
2. X2 $\Rightarrow$ not X4 constraint
3. X5 $\Rightarrow$ X3 or X6
4. X7 $\Rightarrow$ X8
5. X8 $\Rightarrow$ X9
6. X8 $\Rightarrow$ X5
7. X6 $\Rightarrow$ not X9

Boolean Logic: Standard Representations

Each problem constraint typically specified as (a set of) clauses:

E.g. $\underbrace{\text{(a or b)}}, \underbrace{\text{(c or d or } \neg\text{f)}}, \underbrace{(\neg\text{a or c or d)}}, \ldots$

clauses (only "or", "not")

Formula in ***conjunctive normal form***, or CNF: a conjunction of clauses

E.g. F = (a or b) and ¬(a and (b or c)) changes to

F_{CNF} = (a or b) and (¬a or ¬b) and (b or ¬c)

Alternative [useful for QBF]: specify each constraint as a term (only "and", "not"):

E.g. (a and ¬d), (b and ¬a and f), (¬b and d and e), …

Formula in ***disjunctive normal form***, or DNF: a disjunction of terms

E.g. F_{DNF} = (¬a and b) or (a and ¬b and ¬c)

Boolean Satisfiability Testing

The Boolean Satisfiability Problem, or SAT:

Given a Boolean formula F,

- find a satisfying assignment for F
- or prove that no such assignment exists.

- A wide range of applications
- Relatively easy to test for small formulas (e.g. with a Truth Table)
- However, very quickly becomes hard to solve
  - Search space grows exponentially with formula size (more on this next)

SAT technology has been very successful in taming this exponential blow up!

PART II: QBF Reasoning

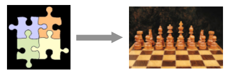
Tutorial Roadmap

1. Automated Reasoning
   - The complexity challenge
   - State of the art in Boolean reasoning
   - Boolean logic, expressivity
2. **QBF Reasoning**
   - A new range of applications
   - Quantified Boolean logic
   - Solution techniques overview
   - Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. Model Counting
   - Connection with sampling
   - A new range of applications
   - Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   - Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

The Next Challenge in Reasoning Technology

Multi-Agent Reasoning:
Quantified Boolean Formulae (QBF)

- Allow use of **Forall** and **Exists** quantifiers over Boolean variables
- QBF significantly more expressive than SAT: from ***single-person puzzles*** to ***competitive games***



New application domains:

- ***Unbounded length*** **planning and verification**
- **Multi-agent scenarios, strategic decision making**
- **Adversarial settings, contingency situations**
- **Incomplete / probabilistic information**

But, computationally *much* harder (formally PSPACE-complete rather than NP-complete)

Key challenge: Can we do for QBF what was done for SAT solving in the last decade?
Would open up a tremendous range of advanced automated reasoning capabilities!

25

SAT Reasoning vs. QBF Reasoning

	SAT Reasoning	QBF Reasoning
Scope of technology	Combinatorial search for optimal and near-optimal solutions	Combinatorial search for optimal and near-optimal solutions in multi-agent, uncertain, or hostile environments
Worst-case complexity	NP-complete (hard)	PSPACE-complete (harder)
Application areas	planning, scheduling, verification, model checking, …	adversarial planning, gaming, security protocols, contingency planning, …
Research status	From 200 vars in early '90s to 1M vars. Now a commercially viable technology.	From 200 vars in late 90's to 100K vars currently. Still rapidly moving.

26

The Need for QBF Reasoning

SAT technology, while very successful for single-agent search, is *not* suitable for adversarial reasoning.

Must model the adversary and incorporate his actions into reasoning

- SAT does not provide a framework for this

Two examples next:

1. Network planning: create a data/communication network between N nodes which is robust under failures during and after network creation
2. Logistics planning: achieve a transportation goal in uncertain environments

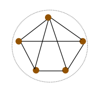
27

Adversarial Planning: Motivating Example

Network Planning Problem:

- Input: 5 nodes, 9 available edges that can be placed between any two nodes
- Goal: all nodes finally connected to each other (directly or indirectly)
- Requirement (A): final network must be robust against 2 node failures
- Requirement (B): network creation process must be robust against 1 node failure

E.g. a sample robust final configuration:
(uses only 8 edges)

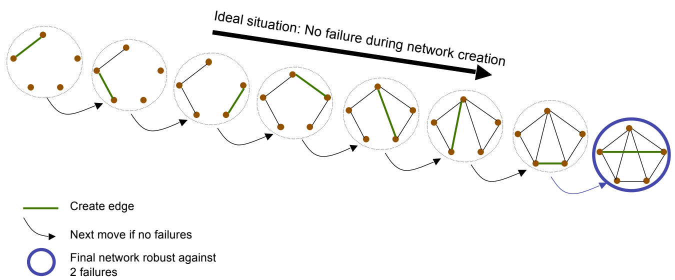


Side note: Mathematical structure of the problem:

1. (A) implies every node must have degree ≥ 3 (otherwise it can easily be "isolated")
2. At least one node must have degree ≥ 4 (follows from 1. and that not all 5 nodes can have odd degree in any graph)
3. Need at least 8 edges total (follows from 1. and 2.)
4. If one node fails during creation, the remaining 4 must be connected with 6 edges to satisfy (A)
5. Actually need 9 edges to guarantee construction (follows from 4. because a node may fail as soon as its degree becomes 3)

28

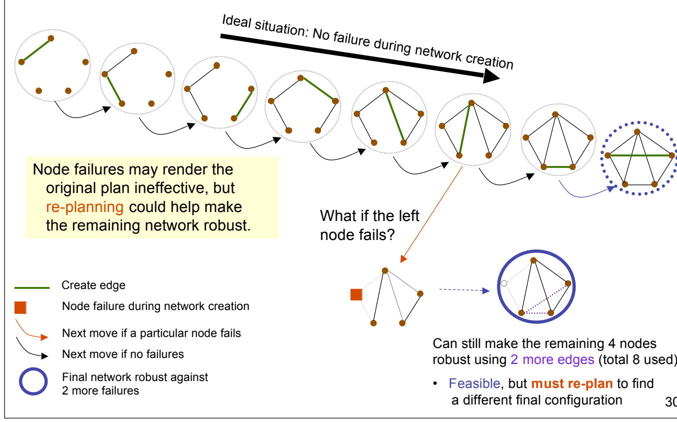
Example: A SAT-Based Sequential Plan



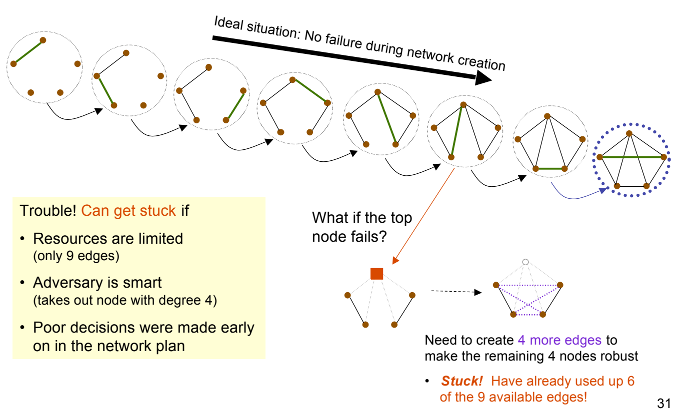
The plan goes smoothly and we end up with the target network, which is robust against *any 2 node failures*

29

Example: A SAT-Based Sequential Plan



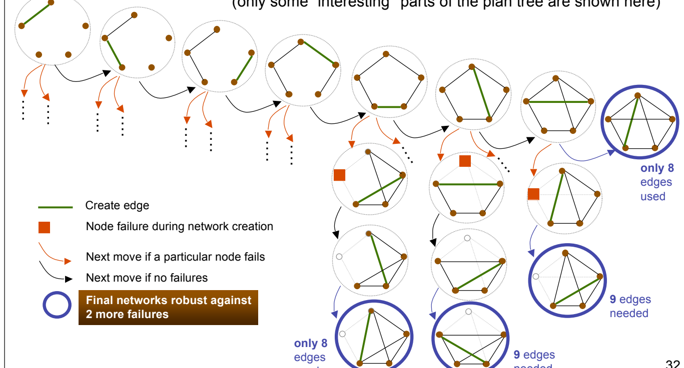
Example: A SAT-Based Sequential Plan



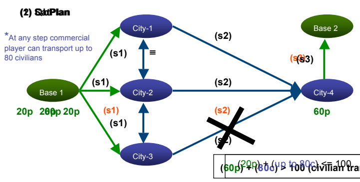
Example: A QBF-Based Contingency Plan

A QBF solver will return a robust **contingency plan** (a tree)

- Will consider all relevant failure modes and responses
  (only some "interesting" parts of the plan tree are shown here)



Another Example: Logistics Planning



- **Blue nodes are cities, green nodes are military bases**
- **Blue edges are commercial transports, green edges are military**
- **Green edges (transports) have a capacity of 60 people, blue edges have a capacity of 100 people**
- **operator: "transport" t(who, amount, from, to, step)**
- **parallel actions can be taken at each step**
- **Goal: Send 60 personal from Base-1 to Base-2 in at most 3 steps**

- **One player: military player, deterministic classic planning, SatPlan**
  - (1) Sat-Plan: **t(m, 60, base-1, city-3, 1), t(m, 60, city-3, city-4, 2), t(m, 60, city-4, base-2, 3)**
- **Two players: deterministic adversarial planning QB Plan**
  - Military Player (m) is "white player", Commercial Player (c) is "black player" (Chess analogy). Commercial player can move up to **80 civilians** between cities. Commercial moves can not be invalidated. Goal can be read as: "**send 60 personal from Base-1 to Base-2 in at most 3 steps whatever commercial needs (moves) are**"
  - If commercial player decides to move 80 civilians from city-3 to city-4 at the second step, we should replan (1). Indeed, the goal can not be achieved if we have already taken the first action of (1)
  - (2) QB-Plan: **t(m, 20, base-1, city-1, 1), t(m, 20, base-1, city-2, 1), t(m, 20, base-1, city-3, 1), t(m, 20, city-1, city-4, 2), t(m, 20, city-2, city-4, 2), t(m, 20, city-3, city-4, 2), t(m, 60, city-4, base-2, 3)**

Tutorial Roadmap

1. Automated Reasoning
   - The complexity challenge
   - State of the art in Boolean reasoning
   - Boolean logic, expressivity
2. **QBF Reasoning**
   - A new range of applications
   - Quantified Boolean logic
   - Solution techniques overview
   - Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. Model Counting
   - Connection with sampling
   - A new range of applications
   - Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   - Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

Quantified Boolean Logic

Boolean logic extended with "quantifiers" on the variables

- "there exists a value of x in {True,False}", represented by ∃x
- "for every value of y in {True,False}", represented by ∀y
- The rest of the Boolean formula structure similar to SAT, usually specified in CNF form

E.g. QBF formula F(v,w,x,y) =

$$\underbrace{\exists v\ \forall w\ \exists x\ \forall y}_{\text{Quantified Boolean variables}} : \underbrace{(\neg v \text{ or } w \text{ or } x) \text{ and } (v \text{ or } \neg w) \text{ and } (v \text{ or } y)}_{\text{constraints (as before)}}$$

Quantified Boolean variables — constraints (as before)

Quantified Boolean Logic: Semantics

F(v,w,x,y,z) = $\boxed{\exists v}\ \boxed{\forall w}\ \boxed{\exists x}\ \boxed{\forall y}$: (¬v or w or x) and (v or ¬w) and (v or y)

What does this QBF formula mean?

Semantic interpretation:

F is True iff
"There exists a value of v s.t.
for both values of w
there exists a value of x s.t.
for both values of y
(¬v or w or x) and (v or ¬w) and (v or y) is True"

Quantified Boolean Logic: Example

F(v,w,x,y,z) = ∃v ∀w ∃x ∀y : (¬v or w or x) and (v or ¬w) and (v or y)

Truth Table for F as a SAT formula

v	w	x	y	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Is F True as a QBF formula?

Without quantifiers (as SAT):
have many satisfying assignments
e.g. (v=0, w=0, x=0, y=1)

With quantifiers (as QBF):
many of these don't work
e.g. *no solution* with v=0

F does have a QBF solution
with v=1 and x set depending on w

QBF Modeling Examples

Example 1: a 4-move chess game

There exists a move of the white s.t.
for every move of the black
there exists a move of the white s.t.
for every move of the black
the white player wins

Example 2: contingency planning
for disaster relief

There exist preparatory steps s.t.
for every disaster scenario within limits
there exists a sequence of actions s.t.
necessary food and shelter can
be guaranteed within two days

Adversarial Uncertainty Modeled as QBF

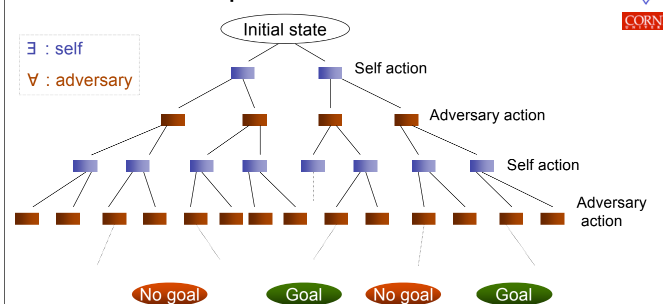
- Two agents: self and adversary
- Both have their own set of actions, rules, etc.
- Self performs actions at time steps 1, 3, 5, …, T
- Adversary performs actions at time steps 2, 4, 6, …, T-1

The following QBF formulation is True if and only if
self can achieve the goal *no matter what* actions adversary takes

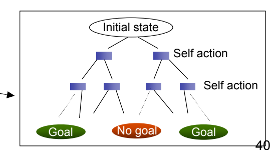
There exists a self action at step 1 s.t.
for every adversary action at step 2
there exists a self action at step 3 s.t.
for every adversary action at step 4
…
there exists a self action at step T s.t.
((initialState(time=1) and
self-respects-modeled-behavior(1,3,5,…,T) and goal(T))
OR (NOT adversary-respects-modeled-behavior(2,4,…,T-1)))

QBF Search Space

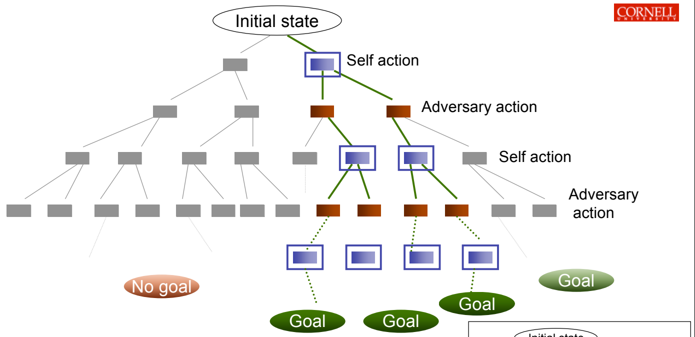
∃ : self
∀ : adversary



Recall traditional SAT-type search space

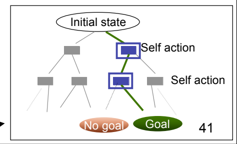


QBF Solution: A Policy or Strategy



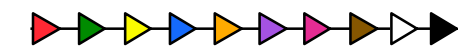
Contingency plan

- A **policy / strategy** of actions for self
- A subtree of the QBF search tree (contrast with a *linear sequence* of actions in SAT-based planning)



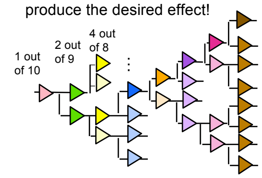
Exponential Complexity Growth

Planning (single-agent):
find the right sequence of actions

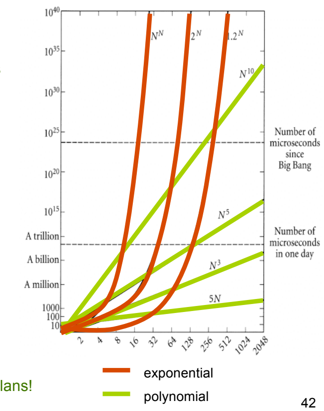


HARD: 10 actions, 10! = **3 x 10^6** possible plans

Contingency planning (multi-agent):
actions may or may not produce the desired effect!



REALLY HARD: 10 x 9^2 x 8^4 x 7^8 x … x 2^{256} =
10^{224} possible contingency plans!



Computational Complexity Hierarchy

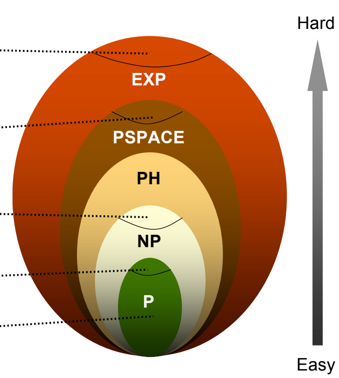
EXP-complete:
games like Go, …

PSPACE-complete:
QBF, adversarial planning, …

NP-complete:
SAT, scheduling, graph coloring, …

P-complete:
circuit-value, …

In P:
sorting, shortest path…



Note: widely believed hierarchy; know P≠EXP for sure

Tutorial Roadmap

1. Automated Reasoning
   - The complexity challenge
   - State of the art in Boolean reasoning
   - Boolean logic, expressivity
2. **QBF Reasoning**
   - A new range of applications
   - Quantified Boolean logic
   - Solution techniques overview
   - Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. Model Counting
   - Connection with sampling
   - A new range of applications
   - Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   - Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

QBF Solution Techniques

- DPLL-based: the dominant solution method
  E.g. Quaffle, QuBE, Semprop, Evaluate, Decide, QRSat
- Local search methods:
  E.g. WalkQSAT
- Skolemization based solvers:
  E.g. sKizzo
- q-resolution based:
  E.g. Quantor
- BDD based:
  E.g. QMRES, QBDD

45

Focus: DPLL-Based Methods for QBF

- Similar to DPLL-based SAT solvers, except for branching variables being labeled as existential or universal
- In usual "top-down" DPLL-based QBF solvers,
  - Branching variables must respect the quantification ordering
    i.e., variables in outer quantification levels are branched on first
  - Selection of branching variables from within a quantifier level done heuristically

46

DPLL-Based Methods for QBF

- For existential (or universal, resp.) branching variables
  - Success: sub-formula evaluates to True (False, resp.)
  - Failure : sub-formula evaluates to False (True, resp.)
- For an existential variable:
  - o If left branch is True, then success (subtree evaluates to True)
  - o Else if right branch is True, then success
  - o Else failure
  - On success, try the last universal not fully explored yet
  - On failure, try the last existential not fully explored yet
- For a universal variable:
  - o If left branch is False, then success (subtree evaluates to False)
  - o Else if right branch is False, then success
  - o Else failure
  - On success, try the last existential not fully explored yet
  - On failure, try the last universal not fully explored yet

47

Learning Techniques in QBF

- Can adapt clause learning techniques from SAT
- Existential "player" tries to satisfy the formula
  - Prune based on partial assignments that are known to falsify the formula and thus can't help the existential player
  - E.g. add a CNF clause when a sub-formula is found to be unsatisfiable
  - Conflict clause learning
  - Uses implication graph analysis similar to SAT
- Universal "player" tries to falsify the formula
  - Prune based on partial assignments that are known to satisfy the formula and thus can't help the universal player
  - E.g. add a DNF term (cube) when a sub-formula is found to be satisfiable
  - Solution learning
  - When satisfiable due to previously added DNF terms, uses implication graph analysis; when satisfiable due to all CNF clauses being satisfied, uses a covering analysis to find a small set of True literals covering clauses

48

Preprocessing for QBF

- Preprocessing the input often results in a significant reduction in the QBF solution cost --- much more so than for SAT
- Has played a key role in the success of the winning QBF solvers in the 2006 competition [Samulowitz et al. '06]
- E.g. binary clause reasoning / hyper-binary resolution
- Simplification steps performed at the beginning and sometimes also dynamically during the search
  - Typically too costly to be done dynamically in SAT solvers
  - But pay off well in QBF solvers

49

Eliminating Variables with the Deepest Quantification

- Consider **∃w ∀x ∃y ∀z . (w ∨ x ∨ y ∨ z)**
- Fix any truth values of w, x, and y
- Since (w ∨ x ∨ y ∨ z) has to be True for both z=True and z=False, it must be that (w ∨ x ∨ y) itself is True
  ⇒ **Can simplify to ∃w ∀x ∃y . (w ∨ x ∨ y)** without changing semantics
- Note: cannot proceed to similarly remove x from this clause because the value of y may depend on x (e.g. suppose w=F. When x=T then y may need to be F to help satisfy other constraints.)

In general,

If a variable of a CNF clause with the deepest quantification is universal, can "delete" this variable from the clause

If a variable in a DNF term with the deepest quantification is existential, can "delete" this variable from the term

50

Unit Propagation

- Unit propagation on CNF clauses sets existential variables,
  on DNF terms sets universal variables
- Elimination of variables with the deepest quantification results in stronger unit propagation
- E.g. again consider ∃w ∀x ∃y ∀z . (w ∨ x ∨ y ∨ z)

  When w=F and x=F,
  - No SAT-style unit propagation from (w ∨ x ∨ y ∨ z)
  - However, as a QBF clause, can first remove z to obtain (w ∨ x ∨ y). Unit propagation now sets y=T

51

Challenge #1

- Most QBF benchmarks have only 2-3 quantifier levels
  - Might as well translate into SAT (it often works well!)
  - Early QBF solvers focused on such instances
  - *Benchmarks with many quantifier levels are often the hardest*
- Practical issues in both modeling and solving become much more apparent with many quantifier levels

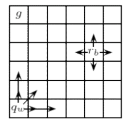
Can QBF solvers be made to scale well with 10+ quantifier alternations?

52

Challenge #2

QBF solvers are extremely sensitive to encoding!

– Especially with many quantifier levels, e.g., evader-pursuer chess instances [Madhusudan et al. 2003]



Instance (N, steps)		Model X [Madhusudan et al. '03]			Model A [Ansotegui et al. '05]		Model B [Ansotegui et al. '05]	
		QuBEJ	Semprop	Quaffle	Best other solver	Cond-Quaffle	Best other solver	Cond-Quaffle
4	7	2030	>2030	>2030	7497	3	0.03	0.03
4	9	--	--	--	--	28	0.06	0.04
8	7	--	--	--	--	800	5	5

Can we design generic QBF modeling techniques that are simple and efficient for solvers?

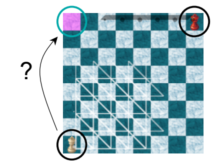
53

Challenge #3

For QBF, traditional encodings hinder unit propagation

– E.g. unsatisfiable "reachability" queries
– A SAT solver would have simply unit propagated
– Most QBF solvers need 1000's of backtracks and relatively complex mechanisms like learning to achieve simple propagation

	Best solver with only unit propagation	Best solver (Qbf-Cornell) with learning
conf-r1	2.5	0.2
conf-r5	8603	5.4
conf-r6	>21600	7.1



q-unsat: *too few steps for White*

Can we achieve effective propagation across quantifiers?

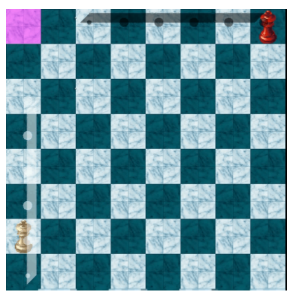
54

Example: Lack of Effective Propagation (in Traditional QBF Solvers)

Question:
Can White reach the pink square without being captured?

Impossible! White has one too few available moves

[click image for video]



This instance should ideally be easy *even with many additional (irrelevant) pieces!*
Unfortunately, all CNF-based QBF solvers scale exponentially ☹

Good news: Duaffle based on dual CNF-DNF encoding resolves this issue

Challenge #4

QBF solvers suffer from the "illegal search space issue" [Ansotegui-Gomes-Selman 2005]

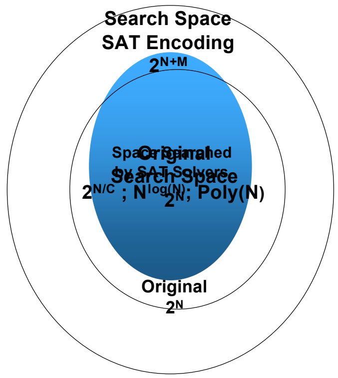
- Auxiliary variables needed for conversion into CNF form
- Can push solver into large irrelevant parts of search space
- *Bottleneck*: detecting clause violation is easy (local check) but detecting that all residual clauses can be easily satisfied [no matter what the universal vars are] is much harder esp. with learning (global check)

- Note: negligible impact on SAT solvers due to effective propagation

- Solution A: CondQuaffle [Ansotegui et al. '05]
  – Pass "flags" to the solver, which detect this event and trigger backtracking
- Solution B: Duaffle [Sabharwal et al. '06]
  – Solver based on dual CNF-DNF encoding simply avoids this issue
- Solution C: Restricted quantification [Benedetti et al. '07]
  – Adds constraints under which quantification applies

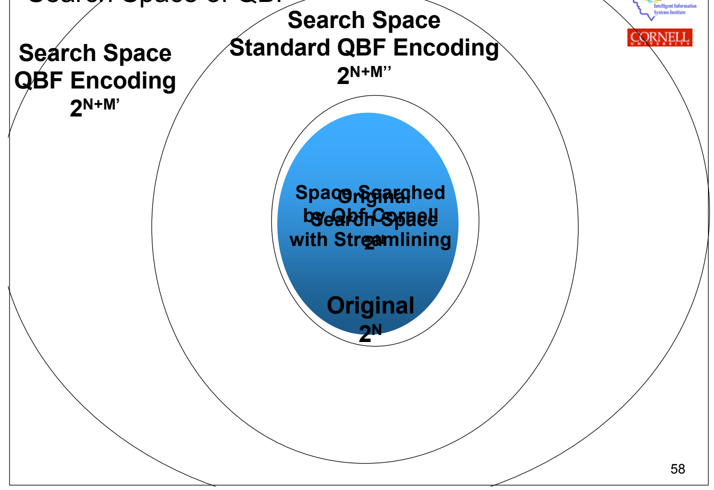
56

Intuition for Illegal Search Space: Search Space for SAT Approaches



In practice, for many real-world applications, polytime scaling.

Search Space of QBF



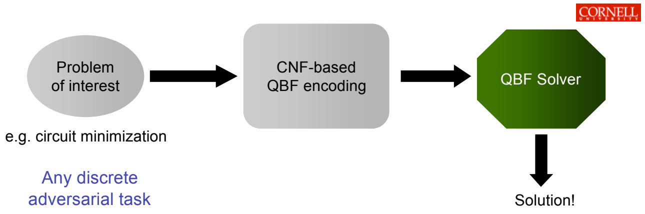
Tutorial Roadmap

1. Automated Reasoning
   - The complexity challenge
   - State of the art in Boolean reasoning
   - Boolean logic, expressivity
2. **QBF Reasoning**
   - A new range of applications
   - Quantified Boolean logic
   - Solution techniques overview
   - Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. Model Counting
   - Connection with sampling
   - A new range of applications
   - Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   - Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

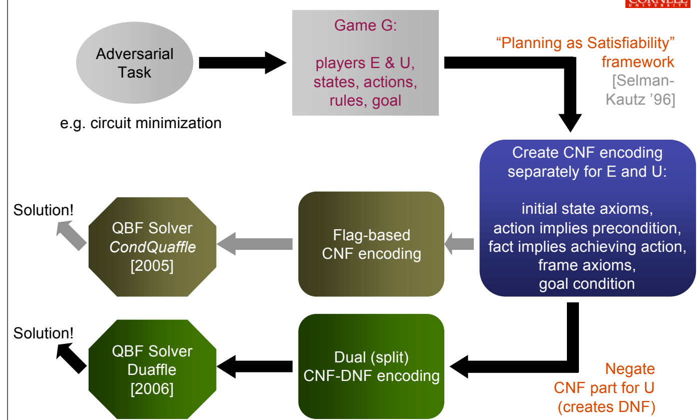
Modeling Problems as QBF

- In principle, traditional QBF encodings similar to SAT encodings
  - Create propositional variables capturing problem variables
  - Create a set of constraints
  - Conjoin (AND) these constraints together: obtain a CNF
  - Add "appropriate" quantification for variables
- In practice, can often be much harder / more tedious than for SAT
  - E.g. in many "game-like" scenarios, must ensure that
    1. If existential agent violates constraints, formula falsified
       : easy, some clause violation
    2. If universal agent violates constraints, formula satisfied
       : harder, *all* clauses must be satisfied,
       could use auxiliary variables for cascading effect

Encoding: The Traditional Approach



Encoding: A Game-Based Approach



From Adversarial Tasks To Games

Example #1:

Circuit Minimization: *Given a circuit C, is there a smaller circuit computing the same function as C?*

- Related QBF benchmarks: adder circuits, sorting networks
- A game with 2 turns
  - Moves : First, E commits to a circuit C_E; second, U produces an input p and computations of C_E, C on p.
  - Rules : C_E must be a legal circuit smaller than C; U must correctly compute C_E(p) and C(p).
  - Goal : E wins if C_E(p) = C(p) no matter how U chooses p
- “E wins” iff there *is* a smaller circuit

From Adversarial Tasks To Games

Example #2:

The Chromatic Number Problem: *Given a graph G and a positive number k, does G have chromatic number k?*

- Chromatic number: minimum number of colors needed to color G so that every two adjacent vertices get different colors
- A game with 2 turns
  - Moves : First, E produces a coloring S of G; second, U produces a coloring T of G
  - Rules : S must be a legal k-coloring of G; T must be a legal (k-1)-coloring of G
  - Goal : E wins if S is valid and T is not
- “E wins” iff graph G has chromatic number k

From Games to Formulas

Use the "planning as satisfiability" framework [Kautz-Selman '96]

– I : Initial conditions
– Tr_E : Rules for legal transitions/moves of E
– Tr_U : Rules for legal transitions/moves of U
– G_E : Goal of E (negation of goal of U)

CNF clauses

Two alternative formulations of the QBF Matrix

$M_1 = I \wedge Tr_E \wedge (Tr_U \rightarrow G_E)$ Fits circuit minimization, chromatic number problem, etc.

Fits games like chess, etc. $M_2 = Tr_U \rightarrow (I \wedge Tr_E \wedge G_E)$

On Normal Forms for Formulas

- Expressions like $Tr_U \rightarrow (I \wedge Tr_E \wedge G_E)$ need to be converted to standard forms for formulas, like CNF
- Should we stick to the CNF format for QBF?

  At least many good reasons to use the CNF format for SAT:

  – Fairly "natural" representation: Many problems are a *conjunction* of several "simple" constraints
  – Efficient pruning of unsat. parts of the search space using violated clauses
  – Simplicity: A clear uniform standard that facilitates clever techniques (e.g. watched literals, implication graph, …)

  However, CNF form for QBF does appear to lead to illegal search space issues and to hinder unit propagation across quantifiers.

  For QBF, no a priori reason to prefer CNF over DNF: equally simple, etc.

  Dual CNF-DNF forms quite advantageous [Sabharwal et al. '06, Zhang '06]

The Dual Encoding

Two alternative formulations of the dual QBF matrix

$M'_1 = \underbrace{(I \wedge Tr_E)}_{\text{CNF}} \wedge \underbrace{(\neg Tr_U \vee \neg G_U)}_{\text{DNF}}$

DNF (negation of CNF clauses)

$M'_2 = \underbrace{(I \wedge Tr_E \wedge G_E)}_{\text{CNF}} \vee \underbrace{\neg Tr_U}_{\text{DNF}}$

In contrast with [Zhang, AAAI '06]: split, non-redundant

Variables : state vars $S^1, S^2, \ldots, S^{k+1}$
action vars $A^1, A^2, \ldots, A^k$

$\exists S^1\ \exists A^1 \exists S^2\ \forall A^2 \forall S^3\ \exists A^3 \exists S^4\ \forall A^k \forall S^{k+1}\quad \mathbf{M'_i}$ $i \in \{1,2\}$

The Dual Encoding: Example

- Chess: White as E, Black as U
- Tr_E: Transition axioms for E: CNF clauses

  e.g. ¬ Move(Wking, sqA, sqB, step 5) ∨ Loc(Wking, sqA, 5)
- Tr_U: Transition axioms for U: DNF terms (negated "traditional" axiom clauses)

  e.g. Move(Bking, sqA, sqB, step 5) ∧ ¬ Loc(Bking, sqA, 5)

Dual Input Format: Example

```
c Dual QBF format
c 100 variables
c 25 CNF clauses, 32 DNF terms
c
p cnfdnf and 100 25 32
c
c Quantifiers
e 1 2 5 9 23 56 … 0
a 6 7 21 22 … 0
…
…
c CNF clauses
-4 -7 8 12 0
9 5 -55 0
…
…
c DNF terms
43 -61 -2 0
4 1 -100 0
…
…
```

- Straightforward extension of QDIMACS format
- Specifies quantification, CNF clauses, DNF terms
- Flag for choosing between formulations M'_1 (connective $\wedge$) and M'_2 (connective $\vee$)
- Existential player: CNF
- Universal player: DNF

QBF Solver Duaffle

- Extends QBF solver Quaffle [Zhang-Malik '02] ("dual-Quaffle")
  - Already has support for DNF terms (cubes)
  - However, its DNF terms logically imply the CNF part
- Exploits the CNF-DNF format
  ⇒ simpler and more succinct encoding mechanism
- DNF and CNF parts are "independent"
  ⇒ requires variation in propagation method, backtrack policy
  (e.g. what to do if CNF part is falsified but DNF part is undecided?)
- Incorporates features of successful SAT/QBF solvers
  (e.g. clever data structures, dynamic decision heuristic, clause and cube learning, fast backjumping, …)

Where Does QBF Reasoning Stand?

We have come a long way since the first QBF solvers several years ago

- From 200 variable problems to 100,000 variable problems
- From 2-3 quantifier alternations to 10+ quantifiers

- New techniques for modeling and solving
- A better understanding of issues like
  propagation across quantifiers and illegal search space
- Many more benchmarks and test suites
- Regular QBF competitions and evaluations

QBF Summary

QBF Reasoning: a promising new automated reasoning technology!

On the road to a whole new range of applications:

- **Strategic decision making**
- **Performance guarantees in complex multi-agent scenarios**
- **Secure communication and data networks in hostile environments**
- **Robust logistics planning in adversarial settings**
- **Large scale contingency planning**
- **Provably robust and secure software and hardware**

PART III: Model Counting

Tutorial Roadmap

1. Automated Reasoning
   - The complexity challenge
   - State of the art in Boolean reasoning
   - Boolean logic, expressivity
2. QBF Reasoning
   - A new range of applications
   - Quantified Boolean logic
   - Solution techniques overview
   - Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. **Model Counting**
   - Connection with sampling
   - A new range of applications
   - Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   - Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

74

Model Counting vs. Solution Sampling

[model ≡ solution ≡ satisfying assignment]

Model Counting (#SAT): Given a CNF formula F,
how many satisfying assignments does F have?

- Must continue searching after one solution is found
- With N variables, can have anywhere from 0 to 2^N solutions
- Will denote the model count by #F or M(F) or simply M

Solution Sampling: Given a CNF formula F,
produce a *uniform sample* from the solution set of F

- SAT solver heuristics designed to quickly narrow down to certain parts of the search space where it's "easy" to find solutions
- Resulting solution typically far from a uniform sample
- Other techniques (e.g. MCMC) have their own drawbacks

75

Counting and Sampling: Inter-related

From sampling to counting

- [Jerrum et al. '86] Fix a variable x. Compute fractions M(x+) and M(x-) of solutions, count one side (either x+ or x-), scale up appropriately
- [Wei-Selman '05] ApproxCount: the above strategy made practical using local search sampling
- [Gomes et al. '07] SampleCount: the above with (probabilistic) correctness guarantees

From counting to sampling

- Brute-force: compute M, the number of solutions; choose k in {1, 2, …, M} uniformly at random; output the k^{th} solution (requires solution enumeration in addition to counting)
- Another approach: compute M. Fix a variable x. Compute M(x+). Let p = M(x+) / M. Set x to True with prob. p, and to False with prob. 1-p, obtain F'. Recurse on F' until all variables have been set.

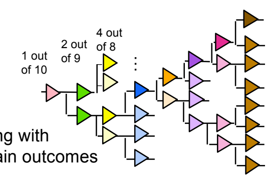
76

Why Model Counting?

Efficient model counting techniques will extend the reach of SAT to a whole new range of applications

- Probabilistic reasoning / uncertainty
  e.g. Markov logic networks [Richardson-Domingos '06]
- Multi-agent / adversarial reasoning (bounded length)

[Roth'96, Littman et al.'01, Park '02, Sang et al.'04, Darwiche'05, Domingos'06]



77

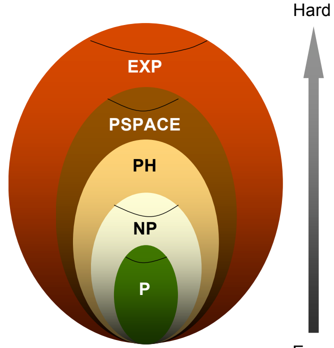
The Challenge of Model Counting

- In theory
  - Model counting is #P-complete
    (believed to be much harder than NP-complete problems)
  - E.g. #P-complete even for 2CNF-SAT and Horn-SAT
    (recall: satisfiability testing for these is in P)
- Practical issues
  - Often finding even a single solution is quite difficult!
  - Typically have huge search spaces
    - E.g. $2^{1000} \approx$ **10^{300}** truth assignments for a 1000 variable formula
  - Solutions often sprinkled unevenly throughout this space
    - E.g. with **10^{60} solutions**, the chance of hitting a solution at random is **10^{-240}**

78

Computational Complexity of Counting

- #P doesn't quite fit directly in the hierarchy --- not a decision problem
- But $P^{\#P}$ contains all of PH, the polynomial time hierarchy
- Hence, in theory, again much harder than SAT



79

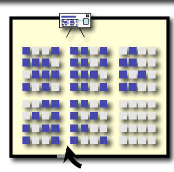
Tutorial Roadmap

1. Automated Reasoning
   - The complexity challenge
   - State of the art in Boolean reasoning
   - Boolean logic, expressivity
2. QBF Reasoning
   - A new range of applications
   - Quantified Boolean logic
   - Solution techniques overview
   - Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. **Model Counting**
   - Connection with sampling
   - A new range of applications
   - Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   - Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

80

How Might One Count?

How many people are present in the hall?



Problem characteristics:

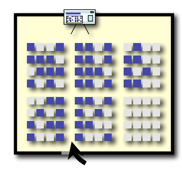
- Space naturally divided into rows, columns, sections, …
- Many seats empty
- Uneven distribution of people (e.g. more near door, aisles, front, etc.)

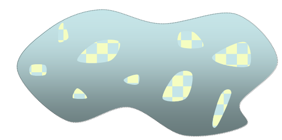
81

Counting People and Counting Solutions

Consider a formula F over N variables.

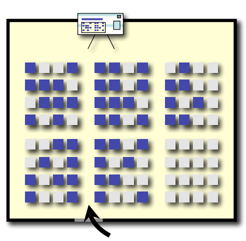
Auditorium	:	Boolean search space for F
Seats	:	2^N truth assignments
M occupied seats	:	M satisfying assignments of F
Selecting part of room	:	setting a variable to T/F or adding a constraint
A person walking out	:	adding additional constraint eliminating that satisfying assignment





82

How Might One Count?



: occupied seats (47)
: empty seats (49)

Various approaches:

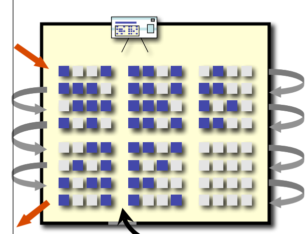
A. Exact model counting
   1. Brute force
   2. Branch-and-bound (DPLL)
   3. Conversion to normal forms

B. Count estimation
   1. Using solution sampling -- naïve
   2. Using solution sampling -- smarter

C. Estimation with guarantees
   1. XOR streamlining
   2. Using solution sampling

83

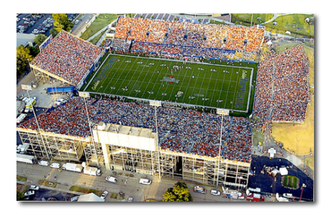
A.1 (exact): Brute-Force



Idea:
- Go through every seat
- If occupied, increment counter

Advantage:
- Simplicity, accuracy

Drawback:
- Scalability



For SAT: go through each truth assignment and check whether it satisfies F

84

A.1: Brute-Force Counting Example

Consider F = (a ∨ b) ∧ (c ∨ d) ∧ (¬d ∨ e)

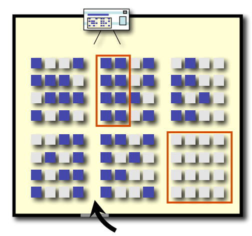
2^5 = 32 truth assignments to (a,b,c,d,e)

Enumerate all 32 assignments.
For each, test whether or not it satisfies F.

F has 12 satisfying assignments:

(0,1,0,1,1), (0,1,1,0,0), (0,1,1,0,1), (0,1,1,1,1),
(1,0,0,1,1), (1,0,1,0,0), (1,0,1,0,1), (1,0,1,1,1),
(1,1,0,1,1), (1,1,1,0,0), (1,1,1,0,1), (1,1,1,1,1),

A.2 (exact): Branch-and-Bound, DPLL-style



Framework used in DPLL-based systematic exact counters
e.g. Relsat [Bayardo-Pehoushek '00], Cachet [Sang et al. '04]

Idea:
- Split space into sections
  e.g. front/back, left/right/ctr, …
- Use smart detection of full/empty sections
- Add up all partial counts

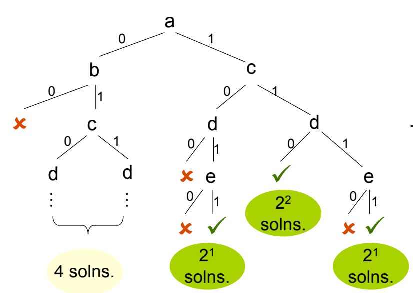
Advantage:
- Relatively faster, exact
- Works quite well on moderate-size problems in practice

Drawback:
- Still "accounts for" every single person present: need extremely fine granularity
- Scalability

A.2: DPLL-Style Exact Counting

- For an N variable formula, if the residual formula is satisfiable after fixing d variables, count 2^{N-d} as the model count for this branch and backtrack.

Again consider F = (a ∨ b) ∧ (c ∨ d) ∧ (¬d ∨ e)



Total 12 solutions

A.2: DPLL-Style Exact Counting

- For efficiency, divide the problem into independent components:
  G is a component of F if variables of G do not appear in F – G.

$$F = \underbrace{(a \vee b)}_{\substack{\text{Component \#1} \\ \text{model count = 3}}} \wedge \underbrace{(c \vee d) \wedge (\neg d \vee e)}_{\substack{\text{Component \#2} \\ \text{model count = 4}}}$$

Total model count = 4 x 3 = 12

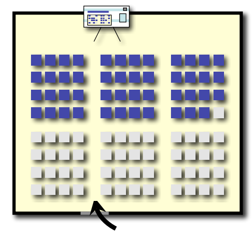
- Use "DFS" on F for component analysis (unique decomposition)
- Compute model count of each component
- Total count = product of component counts

- Components created dynamically/recursively as variables are set
- Component analysis pays off here much more than in SAT
  - Must traverse the whole search tree, not only till the first solution

A.2: Components, Caching, and Learning

- Save or cache the results obtained for sub-formulas of the original formula --- again, much more helpful than for SAT
- Component caching: record counts of component sub-formulas [Bacchus-Dalmao-Pitassi '03], [Formula caching: Majercik-Littman '98, Beame-Impagliazzo-Pitassi-Segerlind '03]
- Cachet [Sang et al. '04] efficiently combines two somewhat complementary techniques: component caching and clause learning
  - Save counts in a hash table
  - Periodically discard old entries (otherwise very space intensive)
- Also, new variable/value selection heuristics found to be more effective for model counting
  - E.g. VSADS [Sang-Beame-Kautz '05]

A.3 (exact): Conversion to Normal Forms



Framework used in DNNF-based systematic exact counter c2d [Darwiche '02]

Idea:
- Convert the CNF formula into another normal form
- Deduce count "easily" from this normal form

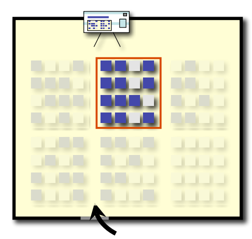
Advantage:
- Exact, normal form often yields other statistics as well in linear time

Drawback:
- Still "accounts for" every single person present: need extremely fine granularity
- Scalability issues
- May lead to exponential size normal form formula

Tutorial Roadmap

1. Automated Reasoning
   - The complexity challenge
   - State of the art in Boolean reasoning
   - Boolean logic, expressivity
2. QBF Reasoning
   - A new range of applications
   - Quantified Boolean logic
   - Solution techniques overview
   - Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. **Model Counting**
   - Connection with sampling
   - A new range of applications
   - Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   - Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

B.1 (estimation): Using Sampling -- Naïve

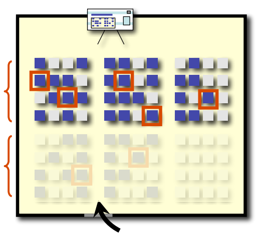


Idea:
- Randomly select a region
- Count within this region
- Scale up appropriately

Advantage:
- Quite fast

Drawback:
- Robustness: can easily under- or over-estimate
- Relies on near-uniform sampling, which itself is hard
- Scalability in sparse spaces: e.g. 10^{60} solutions out of 10^{300} means need region much larger than 10^{240} to "hit" any solutions

B.2 (estimation): Using Sampling -- Smarter



Framework used in approximate counters like ApproxCount [Wei-Selman '05]

Idea:
- Randomly sample k *occupied* seats
- Compute fraction in front & back
- Recursively count only front
- Scale with appropriate multiplier

Advantage:
- Quite fast

Drawback:
- Relies on uniform sampling of occupied seats -- not any easier than counting itself
- Robustness: often under- or over-estimates; no guarantees

93

B.2: ApproxCount

Idea goes back to Jerrum-Valiant-Vazirani ['86], made practical for SAT by Wei-Selman '05 using solution sampler SampleSat [Wei et al. '04]

- Let formula F have M solutions
- Select a variable x. Let $F|_{x=T}$ have M+ solutions and $F|_{x=F}$ have M- solutions (M+ + M- = M)
- Let p = M+ / M : fraction of solutions of F with x=T
- Solution count given by M = M+ . (1/p)

the "multiplier"

- Estimate M+ recursively by considering the simpler formula $F|_{x=T}$
- Estimate p using solution sampling:
  - obtain S samples, compute S+ and S-, compute est(p) = S+ / S
  - est(p) converges to p as S increases
- Estimated number of solutions: est(F) = est($F|_{x=T}$) / est(p)

94

B.2: ApproxCount

The quality of the estimate of M depends on various factors.

- Variable selection heuristic
  - If unit clause, apply unit propagation. Otherwise use solution samples:
  - E.g. pick the most "balanced" variable: S+ as close to S/2 as possible
  - Or pick the most "unbalanced" variable: S+ as close to 0 or S as possible
- Value selection heuristic
  - If S+ > S-, set x=F: leads to small multipliers $\Rightarrow$ more stability, fewer errors
- Sampling quality
  - If samples are biased and/or too few, can easily under-count or over-count
  - Note: effect of biased sampling does partially cancel out in the multipliers
  - SampleSat samples solutions quite well in practice
- Hybridization
  - Once enough variables are set, use Relsat/Cachet for exact residual count

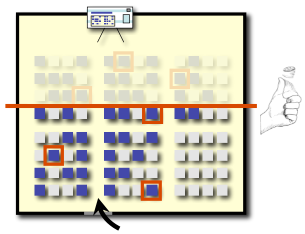
95

Tutorial Roadmap

1. Automated Reasoning
   - The complexity challenge
   - State of the art in Boolean reasoning
   - Boolean logic, expressivity
2. QBF Reasoning
   - A new range of applications
   - Quantified Boolean logic
   - Solution techniques overview
   - Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. **Model Counting**
   - Connection with sampling
   - A new range of applications
   - Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   - Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

96

C.1 (estimation with guarantees): Using Sampling for Counting



Idea:
- Identify a "balanced" row split or column split (roughly equal number of people on each side)
  - Use sampling for estimate
- Pick one side *at random*
- Count on that side recursively
- Multiply result by 2

This provably yields the true count on average!
- Even when an unbalanced row/column is picked accidentally for the split, e.g. even when samples are biased or insufficiently many
- Surprisingly good in practice, using SampleSat as the sampler

97

C.1: SampleCount

- Extends the strategy of ApproxCount
- But, provides concrete correctness guarantees without assuming anything about the quality of the solution samples
- Key ideas:
  - Use randomization (rather than heuristics) to fix variable values
    - Basic version: unbiased coin
    - Extended version: biased coin based on sample estimates
  - Use balanced vars as well as balanced var-pairs to boost quality
  - Use simple repetition to boost confidence and stability
  - Analyze correctness using Markov's inequality

Balanced var x	:	S/2 samples have x=T, S/2 have x=F
Balanced var-pair (x,y)	:	S/2 samples have x=y, S/2 have x≠y

98

Using Local Search for Counting

- Local search methods like WalkSat very efficient on certain problems beyond systematic (DPLL) SAT solvers
- Can be used to quickly provide solution samples by mixing in "Metropolis moves": SampleSat [Wei-Selman '05]
- However, as we saw, *no guarantees on the sample quality*

Can we exploit local search methods without guarantees to obtain model counts with guarantees?

99

Algorithm **SampleCount**

Input: Boolean formula F [Gomes-Hoffmann-Sabharwal-Selman '07]

1. Set numFixed = 0, slack = some constant (e.g. 2, 4, 7, …)
2. Repeat until F becomes feasible for exact counting
   a. Obtain s solution samples for F
   b. Identify the most balanced variable and variable-pair
   c. If x is more balanced than (x,y)
      then randomly set x to T or F (with prob. 1/2)
      else randomly replace x with y or –y (with prob. 1/2)
   d. Simplify F
   e. Increment numFixed

Output: model count $\geq 2^{\text{numFixed} - \text{slack}} \times$ exactCount(simplified F)
with confidence $(1 - 2^{-\text{slack}})$

Note: showing one iteration

100

Correctness Guarantee

Theorem: SampleCount with t iterations gives a correct lower bound with probability ≥ $(1 - 2^{-\text{slack} \times t})$

e.g. slack =2, t =4 ⇒ 99% correctness confidence

Key properties:

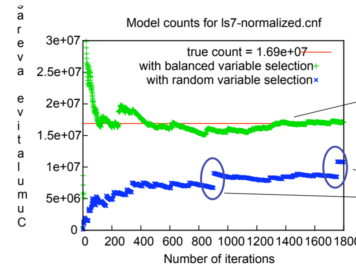
- Theorem holds irrespective of the quality of the sampler used
- Correctness confidence grows exponentially with *slack*

Proof sketch:

- Conditioned on numFixed, expected model count = true count
- Account for conditioning by averaging twice: E[E[X|Y]] = E[X]
- Markov's inequality gives Pr[error in a single run] ≤ $2^{-\text{slack}}$
- Using independence of runs, Pr[error over t runs] ≤ $2^{-\text{slack} \times t}$

101

Fast Convergence with Balanced Selection



Relatively quick stabilization

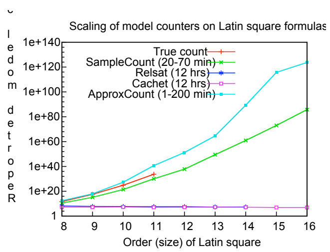
Extreme jumps, ≥ 100 × True count

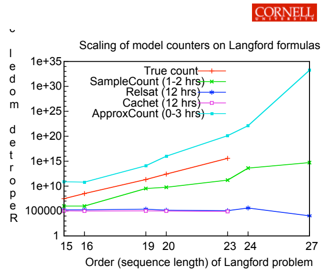
Balanced selection as well as equivalences between variable pairs are key to fast convergence to the true count in practice

- Each point is a single run of SampleCount *without* any slack or correctness guarantee.
- Both processes provably converge to the true count eventually.

102

Sample Results: Latin Square, Langford Problems





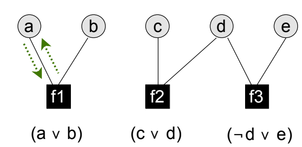
- SampleCount [-----] scales well, produces guaranteed lower bounds close to the true count [-----] within 1 hour
- Exact counters [-----, -----] do not scale well at all (12 hours)
- ApproxCount [-----] overestimates, does not provide guarantees

103

C.2 (estimation with guarantees): Using BP Techniques

- A variant of SampleCount where M+ / M is estimated using Belief Propagation (BP) techniques rather than sampling
  [Kroc-Sabharwal-Selman (in progress)]
- BP is a general iterative message-passing algorithm to compute marginal probabilities over "graphical models"
  - Convert F into a two-layer Bayesian network B
  - Variables of F become variable nodes of B
  - Clauses of F become function nodes of B

Iterative message passing



variable nodes

function nodes

104

C.2: Using BP Techniques

- For each variable x, use BP equations to estimate marginal prob.

  Pr [x=T | all function nodes evaluate to 1]

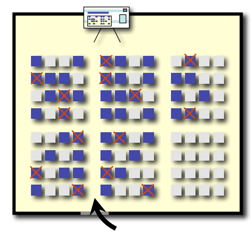
  Note: this is estimating precisely M+ / M !

- Using these values, apply the counting framework of SampleCount
- Challenge #1: Because of "loops" in formulas, BP equations may not converge to the desired value
  - Fortunately, SampleCount framework does not require any quality guarantees on the estimate for M+ / M
- Challenge #2: Iterative BP equations simply do not converge for many formulas of interest
  - Can add a "damping parameter" to BP equations to enforce convergence
  - Too detailed to describe here, but good results in practice!

105

C.3 (estimation with guarantees): Distributed Counting Using XORs

[Gomes-Sabharwal-Selman '06]



Idea (intuition):

In each round
- Everyone independently tosses a coin
- If heads ⇒ stays
  if tails ⇒ walks out

Repeat till only one person remains

Estimate: $2^{\#\text{(rounds)}}$

Does this work?
- On average, Yes!
- With M people present, need roughly $\log_2$ M rounds till only one person remains

106

XOR Streamlining: Making the Intuitive Idea Concrete

- How can we make each solution "flip" a coin?
  - Recall: solutions are implicitly "hidden" in the formula
  - Don't know anything about the solution space structure
- What if we don't hit a unique solution?
- How do we transform the average behavior into a robust method with provable correctness guarantees?

Somewhat surprisingly, all these issues can be resolved

107

XOR Constraints to the Rescue

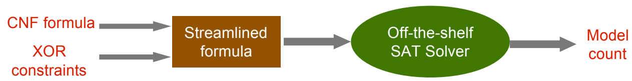
- Special constraints on Boolean variables
  - $a \oplus b \oplus c \oplus d = 1$:
    satisfied if an odd number of a,b,c,d are set to 1
    e.g. (a,b,c,d) = (1,1,1,0) satisfies it
    (1,1,1,1) does not
  - $b \oplus d \oplus e = 0$:
    satisfied if an even number of b,d,e are set to 1
  - These translate into a small set of CNF clauses (using auxiliary variables [Tseitin '68])
  - Used earlier in randomized reductions in Theoretical CS [Valiant-Vazirani '86]

108

Using XORs for Counting: MBound

Given a formula F

1. Add some XOR constraints to F to get F'
   (this eliminates some solutions of F)
2. Check whether F' is satisfiable
3. Conclude "something" about the model count of F



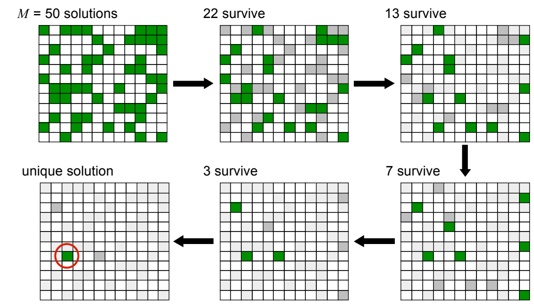
Key difference from previous methods:

- o The formula changes
- o The search method stays the same (SAT solver)

109

The Desired Effect

If each XOR cut the solution space roughly in half, would get down to a unique solution in roughly $\log_2 M$ steps



110

Which XOR Constraints to Use?

- A possibility: analyze structural properties of the instance so that F and F' are "related" in some controllable way
- MBound approach: keep it simple ---
  choose constraint X uniformly at random from all XOR constraints of size k (i.e. with k variables)

Good average behavior, some guarantees, lower bounds

Two crucial properties

1. For any k, for every truth assignment A,
   Pr [A satisfies X] = 0.5
2. When k=n/2, for every two truth assignments A and B,
   "A satisfies X" and "B satisfies X" are independent events
   (pairwise independence)

Provides low variation, stronger guarantees, upper bounds

111

Obtaining Correctness Guarantees

- For formula F with M models/solutions, should ideally add $\log_2 M$ XOR constraints
- Instead, suppose we add $s > \log_2 M + \alpha$ constraints

too many

slack factor

Fix any solution A.
Pr [A survives s XOR constraints] = $1/2^s < 1/(2^\alpha M)$

$\Rightarrow$ Exp [number of surviving solutions] $< M / (2^\alpha M) = 2^{-\alpha}$

$\Rightarrow$ Pr [no. of surviving solns. ≥ 1] $< 2^{-\alpha}$ (by **Markov's Ineq.**)

If we add "too many" XORs,
Pr [F remains satisfiable] is "low"

112

Obtaining Correctness Guarantees

If we add "too many" XORs,
Pr [F remains satisfiable] is "low"

contrapositive

If F is still satisfiable,
with high prob., must *not* have
added too many XORs
i.e., $s \leq \log_2 M + \alpha$

Theorem: If F is still satisfiable after s random XOR constraints, then F has $\geq 2^{s-\alpha}$ solutions with prob. $\geq (1 - 2^{-\alpha})$

e.g. $\alpha=4 \Rightarrow$ 93% correctness confidence

Observe: *confidence increases exponentially with α*

Boosting Correctness Guarantees

Simply repeat the whole process!

E.g. iterate 4 times independently with s constraints and $\alpha=2$.
Pr [F is satisfiable in every iteration] $\leq 1/4^4 < 0.004$

If F is satisfiable after adding s random XOR constraints in each of 4 iterations, then F has at least 2^{s-2} solutions with prob. ≥ 0.996.

Algorithm **Mbound** [lower bound]

Parameters: slack factor α, num iterations t, XOR length k
Input : Boolean formula F
Output: lower bound on model count of F

1. Repeat t times:
   a. Add s random XOR constrains of size k to F to get F'
   b. Check F' for satisfiability using a SAT solver
2. If F' is satisfiable in all t cases, output $2^{s-\alpha}$

Theorem: MBound returns a correct lower bound with probability at least $(1 - 2^{-\alpha t})$

Algorithm **Mbound** [upper bound]

Parameters: slack factor α, num iterations t
Input : Boolean formula F on n variables
Output: lower bound on model count of F

1. Repeat t times:
   a. Add s random XOR constrains of size n/2 to get F'
   b. Check F' for satisfiability using a SAT solver
2. If F' is unsatisfiable in all t cases, output $2^{s+\alpha}$

Theorem: MBound returns a correct upper bound with probability at least $(1 - 2^{-\alpha t})$

Analysis relies on pairwise independence, Chebychev's ineq.

Algorithm **Mbound** [hybrid]

Parameters: slack factor α, number of iterations t
Input : Boolean formula F on n variables
Output: lower bound on model count of F

1. Repeat t times:
   a. Add s random XOR constrains to F to get F'
   b. Count solutions of F' using an exact counter
2. Let m = **min count** for F' obtained over $\boxed{t \text{ iterations}}$
3. Output $\boxed{m.2^{s-\alpha}}$

Theorem: MBound-hybrid returns a correct lower bound with probability at least $(1 - 2^{-\alpha t})$

117

Summarizing MBound

- Can use any state-of-the-art SAT solver off the shelf
- Random XOR constraints independent of both the problem domain and the SAT solver used
- Very high provable correctness guarantees on reported bounds on the model count
  - May be boosted simply by repetition
  - Further boosted by "averaging within buckets, minimizing over buckets"
- Purely random XOR constraints are generally large
  - Not ideal for current SAT solvers
- In practice, must use relatively short XORs
  - Issue: Higher variation over different runs
  - Good news: lower bound correctness guarantees still hold
  - Better news: short XORs do work surprisingly well in practice (fairly low variation) [Gomes-Hoffmann-Sabharwal-Selman '07]

118

XORs for General CSPs

[Gomes-vanHoeve-Sabharwal-Selman; **AAAI-07 talk** on **Wed 5 PM**]

General constraint satisfaction problems (CSPs):
- Domains richer than {0,1}
- Constraints richer than "clauses"
  - *any* constraint, specified succinctly or even as a truth table
- E.g. $x_1 \in \{0,1,2,3,4\}$, $x_2 \in \{30, 31, \ldots, 50\}$,
  $x_3 \in$ {Mon, Tue, Wed, …}, …
  if (x_3 after Mon) then ($x_2 * x_1 \leq x_7$)

MBound can be extended to CSPs

Method 1: Link CSP variables to binary vars. ($x_i = k \Leftrightarrow y_{ik}$ = True)
   a. Individual domain filtering for XORs
   b. Global domain filtering using Gaussian elimination

Method 2: Use "generalized XORs" directly on CSP variables
   e.g. $x_1 + x_7 + x_{22} + x_{34} = r \pmod{d}$

Largest domain size

119

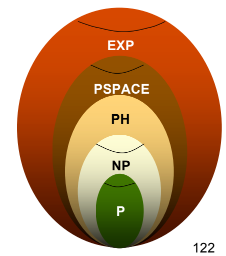
PART IV: Solution Sampling

Tutorial Roadmap

1. Automated Reasoning
   - The complexity challenge
   - State of the art in Boolean reasoning
   - Boolean logic, expressivity
2. QBF Reasoning
   - A new range of applications
   - Quantified Boolean logic
   - Solution techniques overview
   - Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. Model Counting
   - Connection with sampling
   - A new range of applications
   - Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. **Solution Sampling**
   - Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

Solution Sampling

- Problem: *Given a CNF formula F, generate a uniform sample from the set of solutions of F.*
- Worst-case computational complexity: **#P-hard**
  - Recall $P^{\#P}$ contains PH
  - NP is the first level out of an infinite number of levels in PH
- In practice, can often settle for "near"-uniform samples



Sampling Using Systematic Search #1

Enumeration-based solution sampling

- Compute the model count M of F (systematic search)
- Select k from {1, 2, …, M} uniformly at random
- Systematically scan the solutions again and output the k^{th} solution of F (solution enumeration)

✓ Purely uniform sampling
✓ Works well on small formulas (e.g. residual formulas in hybrid samplers)
✓

× Requires two runs of exact counters/enumerators like Relsat (modified)
× Scalability issues as in exact model counters

Sampling Using Systematic Search #2

"Decimation"-based solution sampling

- Arbitrarily select a variable x to assign value to
- Compute M, the model count of F
- Compute M+, the model count of $F|_{x=T}$
- With prob. M+/M, set value=T; otherwise set value=F
- Let F ← $F|_{x=value}$; Repeat the process

} decimation step

✓ Purely uniform sampling
✓ Works well on small formulas (e.g. hybrid samplers)
✓ Does not require solution enumeration ⇒ easier to use advanced techniques like component caching

× Requires 2N runs of exact counters
× Scalability issues as in exact model counters

Markov Chain Monte Carlo Sampling

MCMC-based Samplers

[Madras '02; Metropolis et al. '53; Kirkpatrick et al. '83]

- Based on a Markov chain simulation
  - Create a Markov chain with states $\{0,1\}^N$ whose stationary distribution is the uniform distribution on the set of satisfying assignments of F
- Purely-uniform samples *if* converges to stationary distribution

× Often takes exponential time to converge on hard combinatorial problems

× In fact, these techniques often cannot even find a single solution to hard satisfiability problems

✓ Newer work: using approximations based on factored probability distributions has yielded good results
E.g. Iterative Join Graph Propagation (IJGP)
[Dechter-Kask-Mateescu '02, Gogate-Dechter '06]

125

Sampling Using Local Search

WalkSat-based Sampling

[Selman-Kautz-Coen '93]

- Local search for SAT: repeatedly update current assignment (variable "flipping") based on local neighborhood information, until solution found

- **WalkSat**: Performs *focused local search* giving priority to variables from currently unsatisfied clauses
  - Mixes in freebie-, random-, and greedy-moves

× Efficient on many domains but far from ideal for uniform sampling
  - Quickly narrows down to certain parts of the search space which have "high attraction" for the local search heuristic
  - Further, it mostly outputs solutions that are on cluster boundaries

126

Sampling Using Local Search

- Walksat approach is made more suitable for sampling by mixing-in occasional simulated annealing (SA) moves: **SampleSat**
  [Wei-Erenrich-Selman '04]

- With prob. p, make a random walk move
  with prob. (1-p), make a fixed-temperature annealing move, i.e.
  - Choose a neighboring assignment B uniformly at random
  - If B has equal or more satisfied clauses, select B
  - Else select B with prob. $e^{-\Delta cost(B) / temperature}$
    (otherwise stay at current assignment and repeat)

Metropolis move

✓ Walksat moves help reach solution clusters with various probabilities

✓ SA ensures purely uniform sampling from within each cluster

× Quite efficient and successful, but has a known "band effect"
  - Walksat doesn't quite get to each cluster with probability proportional to cluster size

127

XorSample: Sampling using XORs

[Gomes-Sabharwal-Selman '06]

XOR constraints can also be used for near-uniform sampling

Given a formula F on n variables,
1. Add "a bit too many" random XORs of size k=n/2 to F to get F'
2. Check whether F' has exactly one solution
   o If so, output that solution as a sample

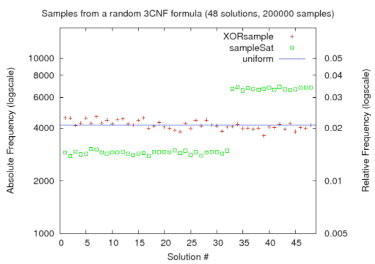
Correctness relies on pairwise independence

Hybrid variation: Add "a bit too few". Enumerate all solutions of F' and choose one uniformly at random (using an exact model counter+enumerator / pure sampler)
Correctness relies on "three-wise" independence

128

The “Band Effect”

XORSample does not have the “band effect” of SampleSat



E.g. a random 3-CNF formula

KL-divergence from uniform:

XORSample : 0.002
SampleSat : 0.085

Sampling disparity in SampleSat:

solns. 1-32 sampled ~2,900x each
solns. 33-48 sampled ~6,700x each

129

Recap

1. Automated Reasoning
   – The complexity challenge
   – State of the art in Boolean reasoning
   – Boolean logic, expressivity
2. QBF Reasoning
   – A new range of applications
   – Quantified Boolean logic
   – Solution techniques overview
   – Modeling
     1. Game-based framework
     2. Dual CNF-DNF approach
3. Model Counting
   – Connection with sampling
   – A new range of applications
   – Solution techniques
     1. Exact counting
     2. Estimation
     3. Bounds with correctness guarantees
4. Solution Sampling
   – Solution techniques
     1. Systematic search
     2. MCMC methods
     3. Local search
     4. Random Streamlining

130

Thank you for attending!

Tutorial slides: http://www.cs.cornell.edu/~sabhar/tutorials/AAAI07-BeyondSAT

A recent SAT (and beyond) survey chapter to appear in the Handbook of Knowledge Representation, available at Ashish’s webpage

Ashish Sabharwal : http://www.cs.cornell.edu/~sabhar

Bart Selman : http://www.cs.cornell.edu/selman