SAT: Propositional Satisfiability

22c:145 Artificial Intelligence
Russell & Norvig, Ch. 7.6

The Einstein Puzzle

Supposedly, Albert Einstein wrote this riddle, and said 98% of the world could not solve it.

- There are 5 houses in five different colors.
- In each house lives a person with a different nationality.
- These 5 owners drink a certain drink, smoke a certain brand of tobacco and keep a certain pet.
- No owners have the same pet, smoke the same tobacco, or drink the same drink.
- The question is: Who owns the fish?

Hints to Einstein Puzzle

- The Brit lives in the red house
- The Swede keeps dogs as pets
- The Dane drinks tea
- The green house is adjacent on the left of the white house
- The green house owner drinks coffee
- The person who smokes Pall Mall raises birds
- The owner of the yellow house smokes Dunhill
- The man living in the house right in the center drinks milk

Hints to Einstein Puzzle (cont)

- The Norwegian lives in the first house
- The man who smokes Blends lives next to the one who keeps cats
- The man who keeps horses lives next to the one who smokes Dunhill
- The owner who smokes Bluemaster drinks juice
- The German smokes Prince
- The Norwegian lives next to the blue house
- The man who smokes Blend has a neighbor who drinks water.

Specify Einstein Puzzle in SAT

- The houses are presented by 1, 2, 3, 4, 5.
- Definition of colors */
  - #define red 0
  - #define green 1
  - #define white 2
  - #define blue 3
  - #define yellow 4
- #define color(x,y) ((x)+5*(y))
- Answer: 3 9 15 17 21

Specify Einstein Puzzle in SAT

- The houses are presented by 1, 2, 3, 4, 5.
- Definition of nationality
  - #define brit 5
  - #define swede 6
  - #define dane 7
  - #define norwegian 8
  - #define german 9
- #define lives(x,y) ((x)+5*(y))
- Answer: 28 35 37 41 49

Specify Einstein Puzzle in SAT

- The houses are presented by 1, 2, 3, 4, 5.
- Definition of drinks
  - #define tea 10
  - #define coffee 11
  - #define water 12
  - #define juice 13
  - #define milk 14
- #define drinks(x,y) ((x)+5*(y))
- Answer: 52 59 61 70 73

Clauses in DIMACS Format

```
printf("p cnf 125 1000\n");  // actual clauses: 885

for (k = 0; k < 5; k++) {
  // every house has a color
  for (a = 1; a <= 5; a++) printf("%d ", color(a, k));
  printf("0\n");
  for (a = 1; a <= 5; a++) {
     for (b = 1; b < a; b++)     // a color can be used once
        printf("-%d -%d 0\n", color(a, k), color(b, k));
     for (b = 0; b < 5 ; b++) if (b != k)
        // a house can have only one color.
        printf("-%d -%d 0\n", color(a, k), color(a, b));
  }
```

Clauses in DIMACS Format

```
// The Brit lives in the red house
 for (a = 1; a <= 5; a++) {
   printf("-%d %d 0\n", lives(a, brit), color(a, red));
   printf("%d -%d 0\n", lives(a, brit), color(a, red));
 }

 // The Swede keeps dogs as pets
 for (a = 1; a <= 5; a++) {
   printf("-%d %d 0\n", lives(a, swede), pets(a, dog));
   printf("%d -%d 0\n", lives(a, swede), pets(a, dog));
 }
```

Clauses in DIMACS Format

```
// The man living in the house right in the center drinks milk
    printf("%d 0\n", drinks(3, milk));

// The Norwegian lives in the first house
   printf("%d 0\n", lives(1, norwegian));
```

Clauses in DIMACS Format

```
// The man who smokes Blends lives next to the one
// who keeps cats
   printf("-%d %d 0\n", smokes(1, Blends), pets(2, cat));
   printf("-%d %d 0\n", smokes(5, Blends), pets(4, cat));
   for (a = 2; a <= 4; a++) {
      printf("-%d %d %d 0\n",
        smokes(a, Blends),
        pets(a-1, cat), pets(a+1, cat));
   }
```

Sato's Result

```
------- SATO 3.2.1, 04/2000 on serv16.divms.uiowa.edu ------
c Input file "einstein.cnf" is open.
c Reading clauses in DIMACS's format.
c Max_atom = 125, Max_clause = 1000
There are 885 input clauses (3 unit, 251 subsumed, 637 retained).

Model #1: (indices of true atoms)
   3  9  15  17  21  28  35  37  41  49  52  59  61  70  73  77  81  90  93  99
   103  106  112  120  124
The number of found models is 1.
There are 13 branches (1 succeeded, 9 failed, 0 jumped).

---------- Stats ----------
run time (seconds)          0.00
  build time                0.00
  search time               0.00
mallocated (K bytes        96.49
-----------------------------------
```

Recursive Learning (RL) in CNF

- Recursive evaluation of clause satisfiability requirements for identifying common assignments

$\varphi = (a \vee b)(\neg a \vee d)\,(\neg b \vee d)$

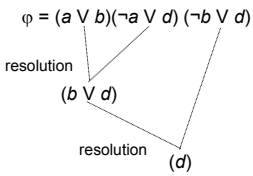
Try $a = 1$: $(a = 1) \Rightarrow (d = 1)$ $\quad C(a = 1) = \{a = 1, d = 1\}$

Try $b = 1$: $(b = 1) \Rightarrow (d = 1)$ $\quad C(b = 1) = \{b = 1, d = 1\}$

$C(a = 1) \cap C(b = 1) = \{d = 1\}$ Every way of satisfying $(a \vee b)$ implies $d = 1$. Hence, $d = 1$ is a necessary assignment !

Recursion can be of arbitrary depth

An Alternative Explanation for RL



Sequence of resolution operations for finding necessary assignments

Comment: RL provides yet another mechanism for identifying suitable resolution operations

Comparison

- Local search is incomplete
  - If instances are known to be SAT, local search can be competitive
- Resolution is in general impractical
- Recursive Learning (RL) are in general slow, though robust
  - RL can derive too much unnecessary information

Techniques for Backtrack Search

- Conflict analysis
  - Clause/implicate recording
  - Non-chronological backtracking
- Incorporate and extend ideas from:
  - Resolution
  - Recursive learning
- Formula simplification & Clause inference
- Randomization & Restarts

Clause Recording

- During backtrack search, for each conflict create clause that explains and prevents recurrence of same conflict

$\varphi = (a \vee b)(\neg b \vee c \vee d)\,(\neg b \vee e)(\neg d \vee \neg e \vee f)\ldots$

Assume (decisions) $c = 0$ and $f = 0$

Assign $a = 0$ and imply assignments

A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfiable

$(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$

$(\varphi = 1) \Rightarrow (a = 1) \vee (c = 1) \vee (f = 1)$

Create new clause: $(a \vee c \vee f)$

Clause Recording

- Clauses derived from conflicts can also be viewed as the result of applying selective resolution

$\varphi = (a \vee b)(\neg b \vee c \vee d)\,(\neg b \vee e)(\neg d \vee \neg e \vee f)\ldots$

resolution

$(a \vee c \vee d)$ $\qquad$ $(a \vee e)$

$(a \vee c \vee \neg e \vee f)$

$(a \vee c \vee f)$

Unit clause $(a \vee c \vee f)$ implies assignment !

Clauses $(a \vee c \vee f)$ and $(\neg a)$ conflict and implies assignment !

Non-Chronological Backtracking

- During backtrack search, in the presence of conflicts, backtrack to one of the causes of the conflict

$\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$
$(a \vee c \vee f)(\neg a \vee g)(\neg g \vee b)(\neg h \vee j)(\neg i \vee k)\ldots$

Assume (decisions) $c = 0$, $f = 0$, $h = 0$ and $i = 0$

Assignment $a = 0$ caused conflict $\Rightarrow$ clause $(a \vee c \vee f)$ created
$(a \vee c \vee f)$ implies $a = 1$

A conflict is again reached: $(\neg d \vee \neg e \vee f)$ is unsat

$(a = 1) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$

$(\varphi = 1) \Rightarrow (a = 0) \vee (c = 1) \vee (f = 1)$

Creates new clause: $(a \vee c \vee f)$

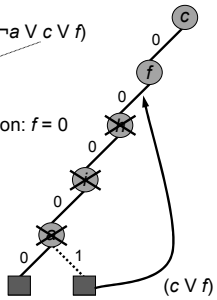
Non-Chronological Backtracking

Created clauses: $(a \vee c \vee f)$ and $(\neg a \vee c \vee f)$

Apply resolution:
new unsat clause $(c \vee f)$

$\therefore$ backtrack to most recent decision: $f = 0$

$\therefore$ created clauses/implicates:
$(a \vee c \vee f)$,
$(\neg a \vee c \vee f)$, and
$(c \vee f)$



Ideas from other Approaches

- Resolution and recursive learning can be incorporated into backtrack search (DP)
  - create additional clauses/implicates
  - anticipate and prevent conflicting conditions
  - identify necessary assignments
  - allow for non-chronological backtracking

Resolution within DP:

$(a \vee b \vee c)$ $(\neg a \vee b \vee d)$

resolution

$(b \vee c \vee d)$

$(b \vee c \vee d)$: Unit clause !

Clause provides explanation
for necessary assignment $b = 1$

Recursive Learning within DP

$\varphi = (\ldots)(\neg a \vee d \vee e)(\neg b \vee d \vee c)$

Implications:

$(a = 1) \wedge (e = 0) \Rightarrow (d = 1)$

$(b = 1) \wedge (c = 0) \Rightarrow (d = 1)$

$(c = 0) \wedge ((e = 0) \wedge (c = 0)) \Rightarrow (d = 1)$

resolution

$(b \vee c \vee e \vee d)$

resolution

$(c \vee e \vee d)$

Clausal form:

$(c \vee e \vee d)$: Unit clause !

Clause provides explanation
for necessary assignment $d = 1$

Formula Simplification

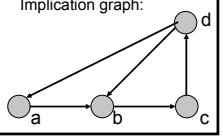
- Eliminate clauses and variables
  - If $(x \vee \neg y)$ and $(\neg x \vee y)$ exist, then x and y are equivalent, $(x \leftrightarrow y)$
    - eliminate y, and replace by x
    - remove satisfied clauses
  - Utilize 2CNF sub-formula for identifying equivalent variables

$(\neg a \vee b)(\neg b \vee c)(\neg c \vee d)(\neg d \vee b)(\neg d \vee a)$

$\equiv (a \rightarrow b)(b \rightarrow c)(c \rightarrow d)(d \rightarrow b)(d \rightarrow a)$

a, b, c and d are pairwise equivalent
$\therefore$replace all variables by a

Implication graph:



2-Variable Equivalence

$(a \vee \neg x)(b \vee \neg x)(\neg a \vee \neg b \vee x)$ $(a \vee \neg y)(b \vee \neg y)(\neg a \vee \neg b \vee y)$

resolution

$(\neg x \vee \neg b \vee y)$ $(\neg y \vee \neg b \vee x)$

resolution

$(\neg x \vee y)$ $(\neg y \vee x)$

$x \leftrightarrow y$

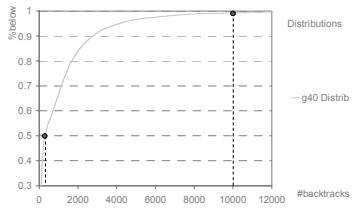
The Power of Resolution

- Most search pruning techniques can be explained as particular ways of applying selective resolution
  - Conflict-based clause recording
  - Non-chronological backtracking
  - Extending recursive learning to backtrack search
  - Clause inference conditions
- General resolution is computationally too expensive **!**
- Most techniques indirectly identify which resolution operations to apply **!**
  - To create new clauses/implicates
  - To identify necessary assignments

Randomization & Restarts

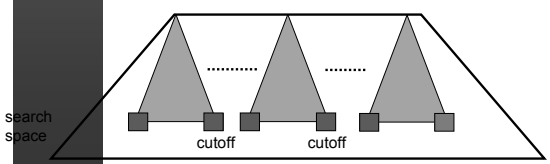
- Run times of backtrack search SAT solvers characterized by heavy-tail distributions
  - For a fixed problem instance, run times can exhibit large variations with different branching heuristics and/or branching randomization

Processor verification instance w/ branching randomization and 10000 runs



Randomization & Restarts

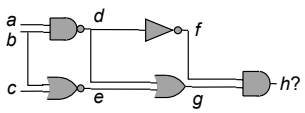
- Search strategy: Rapid Randomized Restarts
  - Randomize variable selection heuristic
  - Utilize a "small" backtrack cutoff value
  - Repeatedly restart the search each time backtrack cutoff reached
    - Use randomization to explore different paths in search tree



Randomization & Restarts

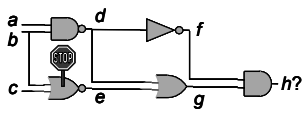
- Can make the search strategy complete
  - Increase backtrack cutoff value after each restart
- Can utilize learning
  - Useful for proving unsatisfiability
- Can utilize portfolios of algorithms and/or algorithm configurations
  - Either, run *K* algorithms (or algorithm configurations)
    - concurrently, in different processors, or
    - sequentially, in a single processor
  - Or, after each restart, pick an algorithm from a portfolio
  - Also useful for proving unsatisfiability

Circuit Satisfiability



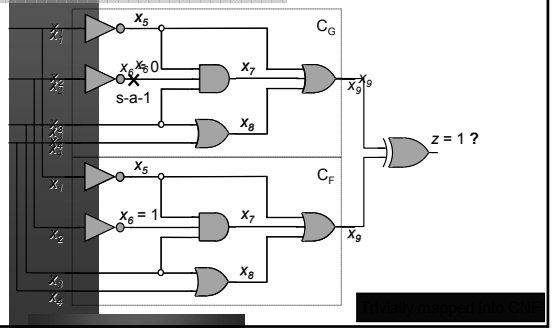
$\varphi = [d \equiv \neg(ab)]\ [e \equiv \neg(b \vee c)]\ [f \equiv \neg d]\ [g \equiv d \vee e]\ [h \equiv fg]\ h$

Circuit Satisfiability



$\varphi = h\ [d \equiv \neg(ab)]\ [e \equiv \neg(b \vee c)]\ [f \equiv \neg d]\ [g \equiv d \vee e]\ [h \equiv fg]$
$= \mathbf{h}$
$\mathbf{(a \vee d)(b \vee d)(\neg a \vee \neg b \vee \neg d)}$
$\mathbf{(\neg b \vee \neg e)(\neg c \vee \neg e)(b \vee c \vee e)}$
$\mathbf{(\neg d \vee \neg f)(d \vee f)}$
$\mathbf{(\neg d \vee g)(\neg e \vee g)(d \vee e \vee \neg g)}$
$\mathbf{(f \vee \neg h)(g \vee \neg h)(\neg f \vee \neg g \vee h)}$

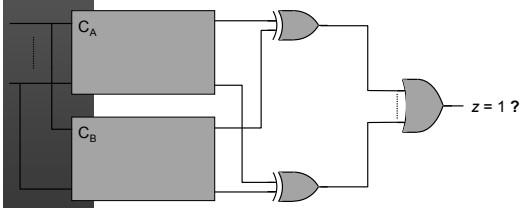
ATPG (Automatic Test Pattern Generation)



Typically mapped into CNF

Equivalence Checking

- Combinational Circuits:



$z = 1$ is unsatisfiable if two circuits are equivalent

Bounded Model Checking

- Sequential Circuits:
- Problem formulation,
  - System property P does not hold in one of the first k states following initial state I_0
  - Transition function ρ

$$I_0 \wedge \rho(0,1) \wedge \rho(1,2) \wedge \ldots \wedge \rho(k\text{-}1,k) \wedge (\neg P_0 \vee \neg P_1 \ldots \vee \neg P_k)$$

  - Represent formula as an instance of SAT in CNF format

Comparison

- CEC is harder than ATPG
  - In ATPG the two circuits are known to be similar
  - In CEC the two circuits can be fairly different
    - Hard instances of CEC are unsatisfiable
- BMC is hard
  - Large number of (apparently) unrelated variables
  - Identification of conflicts may require large number of decisions

SAT Problem Hardness in EDA

- Bounded Model Checking (BMC)
- Superscalar processor verification
- FPGA routing
- Equivalence Checking (CEC)
- Circuit Delay Computation
- Test Pattern Generation (ATPG):
  - Stuck-at, Delay faults, etc.
  - Redundancy Removal
- Noise analysis
- ...

Hardest
Easiest
Unknown

Empirical Evidence (in EDA)

- Illustrate scalability of modern SAT solvers
  - Ability to solve large problem instances
- Illustrate practical application of the techniques described for backtrack search
  - Clause recording and non-chronological backtracking
  - Recursive Learning / Stalmarck's Method
  - Formula simplification
  - Randomization and restarts
  - Portfolio of algorithm configurations
- Utilize modern backtrack search SAT algorithm,
  - GRASP, REL_SAT, SATO, ...

Empirical Evidence (in EDA)

Can solve large problem instances

domain	instance	variables	clauses	CPU time
FPGA routing	bigkeyv1w6	25896	91400	11.9

Empirical Evidence (in EDA)

Non-chronological backtracking (NCB)
and clause recording (CR)
can be observed often and can be crucial

		w/o NCB and CR		w/ NCB and w/o CR				w/ NCB and CR			
domain	instance	B	T	B	NCB	LJ	T	B	NCB	LJ	T
testing	ssa2670-130	> 11250000	> 10000	7142	3538	20	60.3	100	42	15	0.65

Empirical Evidence (in EDA)

Formula simplification can be significant

		before		after	
domain	instance	variables	clauses	variables	clauses
BMC	barrel7	3523	13765	805	3467

Empirical Evidence (in EDA)

Randomization & Restarts can be effective

		w/o restarts				w/ restarts				
domain	instance	B	NCB	LJ	T	B	NCB	LJ	T	R
verification	2dlx_cc_bug54	42645	15156	35	2299.3	222	147	36	53.5	3

Restart search every 100 backtracks
Keep recorded clauses of size ≤ 10

Empirical Evidence (in EDA)

Portfolio of algorithm configurations can be essential

		w/o portfolio				w/ portfolio			
domain	instance	B	NCB	LJ	T	B	NCB	LJ	T
verification	2dlx2_cc	2273327	688891	39	> 100000	100498	32066	70	2032

Configurations:
Clauses recorded: sizes 1 to 30
Clauses kept: sizes 10 to 20
Each configuration w/ equal probability
4 different configurations used
Initial cutoff = 250; increments = 50

Conclusions

- Many recent SAT algorithms and (EDA) applications
- Hard Applications
  - Bounded Model Checking
  - Combinational Equivalence Checking
  - Superscalar processor verification
  - FPGA routing
- "Easy" Applications
  - Test Pattern Generation: Stuck-at, Delay faults, etc.
  - Redundancy Removal
  - Circuit Delay Computation
- Other Applications
  - Noise analysis, etc.

Conclusions

- Complete vs. Incomplete algorithms
  - Backtrack search (DP)
  - Resolution (original DP)
  - Recursive learning
  - Local search
- Techniques for backtrack search (infer implicates)
  - conflict-induced clause recording
  - non-chronological backtracking
  - resolution, SM and RL within backtrack search
  - formula simplification & clause inference conditions
  - randomization & restarts

Research Directions

- Algorithms:
  - Explore relation between different techniques
    - backtrack search; conflict analysis; recursive learning; branch-merge rule; randomization & restarts; clause inference; local search (?); BDDs (?)
  - Address specific solvers (circuits, incremental, etc.)
  - Develop visualization aids for helping to better understand problem hardness
- Applications:
  - Industry has applied SAT solvers to different applications
  - SAT research requires challenging and representative publicly available benchmark instances **!**

A Few References

- EDA Applications
  - Marques-Silva&Sakallah, DAC'00
- Resolution
  - Davis&Putnam, JACM'60
- Backtrack Search
  - Davis et. al, CACM'62
  - Non-chronological backtracking and clause recording
    - Marques-Silva&Sakallah, ICCAD'96; Bayardo&Schrag, AAAI'97; Zhang, CADE'97
  - Relevance-based learning
    - Bayardo&Schrag, AAAI'97
  - Conflict-induced necessary assignments
    - Marques-Silva&Sakallah, ICCAD'96

A Few References (Cont'd)

- Backtrack Search (Cont'd)
  - Randomization and restarts
    - Gomes&Selman, AAAI'98; Baptista&Marques-Silva, CP'2000
  - Formula simplification
    - Li, AAAI'2000; Marques-Silva, CP'2000
- Stalmarck's Method
  - Stalmarck, Patent'89; Groote&Warners, CWI TechRep'1999
- Recursive Learning
  - Kunz&Pradhan, ITC'92; Marques-Silva&Glass, DATE'99
- Local Search
  - Selman&Kautz, IJCAI'93