

CS137: Electronic Design Automation

Day 13: May 17, 2004
Modern SAT Solvers
(Chaff)



CALTECH CS137 Spring2004 -- DeHon

Today

- SAT
- Davis-Putnam
- Data Structures
- Optimizations
 - Watch2
 - VSIDS
 - ?restarts

CALTECH CS137 Spring2004 -- DeHon

Problem

- SAT: Boolean Satisfiability
- **Given:** logical formula f in CNF
- Find a set of variable assignments that makes f **true**
- Or conclude no such assignment exists

CNF

- Conjunctive Normal Form
- Logical AND of a set of **clauses**
- **Clauses:** logical OR of a set of literals
- **Literal:** a variable or its complement
- *E.g.*

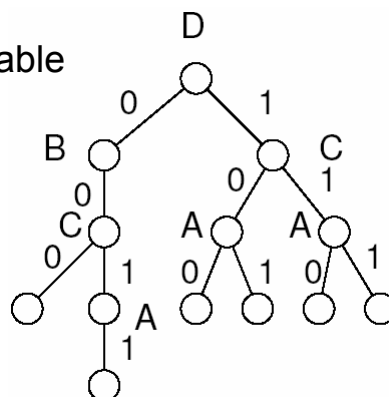
$$(A+B+/C)*(/B+D)*(C+/A+/E)$$

CNF

- Conjunctive Normal Form
- Logical AND of a set of **clauses**
- To be satisfied:
 - Every clause must be made **true**
- $(A+B+/C)*(/B+D)*(C+/A+/E)$
 - If know $D=\mathbf{false}$
 - B must be **false**

Previously

- Argued could be solved with pruning search
 - Pick an unassigned variable
 - Branch on true/false
 - Compute implications



Previously

- Also looked at PODEM
 - Backtracking search on variable assignment

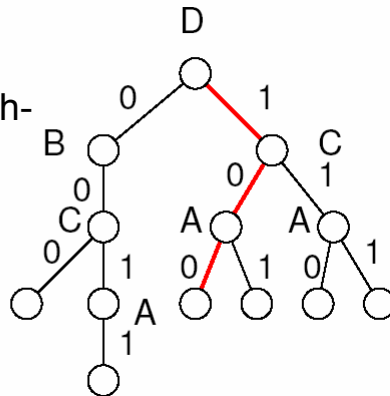
Davis-Putnam

```
while (true) {  
  if (!decide()) // no unassigned vars  
    return(satisfiable);  
  while ( !bcp()) { // constraint propagation  
    if (!resolveConflict()) // backtrack  
      return(not satisfiable);  
    }  
}
```

decide()

- Picks an unassigned variable
- Gives it a value
- Push on *decision stack*
 - Efficient structure for depth-first search tree

| |
|-----|
| A=0 |
| C=0 |
| D=1 |



CALTECH CS137 Spring2004 -- DeHon

Data Structures

- Variable “array”
- Clause “DB”
- Each clause is a set of variables
- Decision “stack”

| |
|-----|
| A=0 |
| C=0 |
| D=1 |

| |
|---|
| A |
| B |
| C |
| D |
| E |

| | | |
|----|---|----|
| A | B | /C |
| /B | D | |
| /A | C | /E |

CALTECH CS137 Spring2004 -- DeHon

bcp

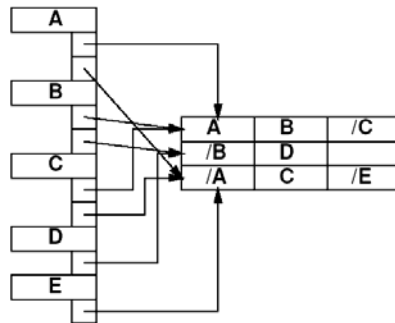
- What do we need to do on each variable assignment?
 - Find implications
 - Implication when all other literals in a clause are **false**
 - Look through all clauses this assignment effects
 - See if any now have all **false** and one unassigned
 - Assign implied values
 - Propagate that assignment
 - Conflict if get implications for **true** and **false**

bcp()

- Q=new queue();
- Q.insert(top of decision stack);
- while (!Q.empty())
 - V=Q.pop();
 - For each clause C in DB with V
 - If C has one unassigned literal, rest **false**
 - Vnew=unassigned literal in C
 - val=value Vnew must take
 - If (Vnew assigned to value other than val)
 - » return (**false**); // conflict
 - Q.add(Vnew=val);
- return(**true**)

Variable array

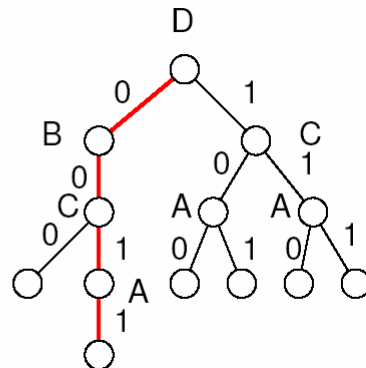
- Each variable has a list pointing to all clauses in which it appears?
 - Avoid need to look at every clause



Tracking Implications

- Each implication made at some tree level
 - Associated with some entry on decision stack
 - Has associated decision stack height
- On backtrack
 - Unassign implications above changed decision level

| | |
|-------------|-----|
| A=1 at DL=2 | C=1 |
| B=0 at DL=1 | D=0 |



Track Variable Assignment

- Each clause has counter
 - Count number of unassigned literals
 - Decrement when assign **false** literal
 - Mark clause as satisfied when assign **true** literal (remove from clause database?)

| | | | |
|---|-----------|----------|-----------|
| 3 | A | B | /C |
| 2 | /B | D | |
| 3 | /A | C | /E |

CALTECH CS137 Spring2004 -- DeHon

Track Variable Assignment

- Each clause has counter
 - Count number of unassigned literals
 - Decrement when assign **false** literal
 - Mark clause as satisfied when assign **true** literal (remove from clause database?)

| | | | |
|---|-----------|----------|-----------|
| 3 | A | B | /C |
| 2 | /B | D | |
| 3 | /A | C | /E |

E=1

| | | | |
|---|--------------|-----------|-----------|
| 3 | A | B | /C |
| 2 | /B | D | |
| 2 | 3 | /A | C |
| | | /E | |

CALTECH CS137 Spring2004 -- DeHon

Track Variable Assignment

- Each clause has counter
 - Count number of unassigned literals
 - Decrement when assign **false** literal
 - Mark clause as satisfied when assign **true** literal
 - Watch for counter decrement 2→1
 - That's when a literal is implied.

| | | | |
|---|-----------|----------|-----------|
| 3 | A | B | /C |
| 2 | /B | D | |
| 2 | /A | C | /E |

resolveConflict()

- What does resolveConflict need to do?
 - Look at most recent decision
 - If can go other way, switch value
 - (clear implications to this depth)
 - Else pop and recurse on previous decision
 - If pop top decision,
 - unsatisfiable

Chaff Optimizations

How will this perform?

- 10,000's of variables
- 100,000's of clauses (millions)
- Every assignment walks to the clause database
- Cache performance?

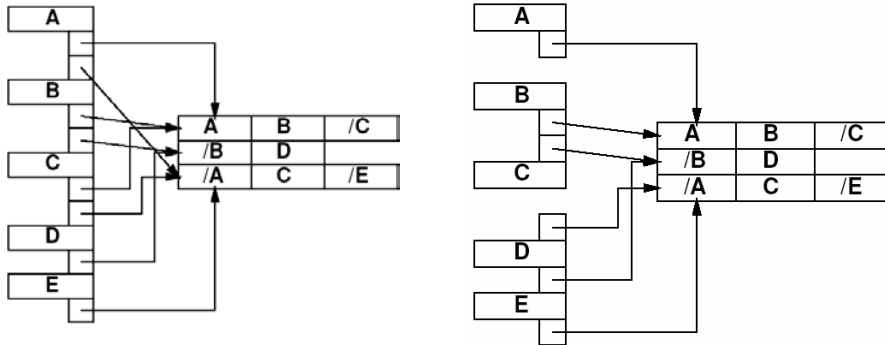
Challenge 1

- Currently, visit every clause on each assignment
 - Clause with K variables
 - Visited $K-1$ times
 - $K-2$ of which just to discover it's not the last
- Can we avoid visiting every clause on every assignment?
 - Every clause in which a variable appears?

Avoiding Clause Visits

- **Idea:** watch only 2 variables in each clause
- Only care about final set of next to last variable
- If set other $k-2$, won't force an implication
- When set one of these (and everything else set)
 - Then we have an implication

Watch 2 Data Structure



Avoiding Clause Visits

- **Idea:** watch only 2 variables in each clause
- Only care about final set of next to last variable
- What if we set of these two “watched” variables?
 - If not last, change the watch to one of the unset variables

Watch 2

- If watched literal becomes false
 - Check if all non-watched are set
 - if so, set implication on other watched
 - else, update watch literal

Note

- Watch pair is arbitrary
- Unassigning a variable (during backtrack)
 - Does not require reset of watch set
 - Constant time to “unset” a variable

Challenge 2: Variable Ordering

- How do we decide() which variable to use next?
 - Want to pick one that facilitates lots of pruning

Variable Ordering

- Old Ideas:
 - Random
 - (DLIS) Dynamic largest individual sum
 - Used most frequently in unresolved clauses
 - BAD?
 - Must re-sort with every variable assignment?
 - ...none clearly superior
 - DLIS competitive
 - Rand good on CAD benchmarks?

New: VSIDS

- Variable State Independent Decaying Sum
 - Each literal has a counter
 - When clause added to DB, increment counter for each literal
 - Select unassigned literal with highest count
 - Periodically, all counters are divided by a constant

New: VSIDS

- Variable State Independent Decaying Sum
 - Each literal has a counter
 - When clause added to DB, increment counter for each literal
 - Remove clauses when satisfied?
 - Reinsert on backtrack
 - Select unassigned literal with highest count
 - Periodically, all counters are divided by a constant

New: VSIDS

- Variable State Independent Decaying Sum
 - Each literal has a counter
 - When clause added to DB, increment counter for each literal
 - **Select unassigned literal with highest count**
 - Don't need to re-sort each selection
 - Only re-sort on backtrack
 - Maybe priority queue insert?
 - Periodically, all counters are divided by a constant

31

VSIDS

- **Goal:** satisfy *recent* conflict clauses
- Decaying sum weights things being added
 - Clauses not conflicting for a while, have values reduced
 - (? Avoid walking through them by increasing weight on new stuff rather than decreasing all old?)
- **Impact:** order of magnitude speedup

32

Restarts

- Periodically restart
 - Clearing the state of all variables
 - i.e. clear decision stack
 - Leave clauses in clause database
 - ? Keep ordering based on recent costs
 - ? Re-insert clauses must reinsert on restart?
 - State of clause database drives variable ordering
 - Benefit: new variable ordering based on lessons of previous search

Overall

- Two orders of magnitude benefit on unsatisfiable instances
- One order of magnitude on satisfiable instances

Next Time

- SAT for Physical Design
 - Routing
 - Placement

Big Ideas

- Exploit Structure
 - Constraint propagation
 - Pruning search technique
- Constants matter
 - Exploit hierarchy in modern memory systems