ESE535:
Electronic Design Automation

Day 6: February 4, 2009
Modern SAT Solvers
({z}Chaff, GRASP, miniSAT)

Today

• SAT
• Davis-Putnam
• Data Structures
• Optimizations
  – Watch2
  – VSIDS
  – ? restarts
• Learning

Problem

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

CNF

• Conjunctive Normal Form
• Logical AND of a set of clauses
  – Product of sums
• Clauses: logical OR of a set of literals
• Literal: a variable or its complement
• E.g.
  \[(A+B+/C)*(/B+D)*(C+/A+/E)\]

3-SAT Universal

• Can express any set of boolean constraints in CNF w/ at most 3 literals per clause
• Canonical NP-complete problem
Convert to 3-SAT
- $A = B + \overline{C} \iff (B + C)$ → universal primitive
  - We know can build any logic expression from nor2
- 3-CNF for $A = B \iff C$
  - $(A + B + C) \iff (A + B) \iff (A + C)$
  - If $(B = 0 \land C = 0)$ then $A = 1$
  - If $(B = 1 \lor C = 1)$ then $A = 0$
- Strategy:
  1. Convert to nor2’s
  2. Then use above to convert nor2 expressions to set of clauses
  3. Combine the clauses resulting from all the nor’s

Brute Force Exhaustive
- How could we find satisfying assignment?
- How long would it take?
  - With N binary variables

Search Formulation
- Think of as search tree on variables
- Each variable can be true or false
  - Branch on values
- All variables determined at leaves of tree

Key Trick
- Avoid searching down to leaf on all subtrees
- “Prune” away branches of tree
- $(A + B + C) \iff (A + B) \iff (A + C)$
  - Consider $A = 1$
  - In this subtree becomes
    - $(B + C) \iff B \iff C$
Key Trick

- \((A+B+C)^*\(A/B)^*\(A/C\)
- Consider \(A=1\)
- In this subtree becomes
  - \((B+C)^*B^*/C\)
- Consider \(B=1\)

- Implication
  - When there is only one literal left in a clause
  - Can conclude it must be true
  - Select it and prune other branch

- Contradiction
  - If implications lead to a conflicting assignments
  - Can conclude this subtree is unsatisfiable
  - Prune branch

Prospect

- Use implications and pruning to avoid visiting full space

Pruning Search

- Solve with pruning search
  - Pick an unassigned variable
  - Branch on true/false
  - Compute implications
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

Data Structures
• Variable “array”
• Clause “DB”
  – Each clause is a set of variables
• Decision “stack”

\[
\text{(A+B+/C)}(\text{+/B+D})(\text{C+/A+/E})
\]

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

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

\[
\text{(A+B+/C)}(\text{+/B+D})(\text{C+/A+/E})
\]
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

\[(A+B+/C)^*(/B+D)^*(C+/A+/E)\]

Track Variable Assignment

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

\[
\begin{array}{c|c|c}
3 & A & B \\ \hline
2 & /B & D \\ \hline
3 & /A & C & /E
\end{array}
\]

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
- Alternates:
  - Treat literals separately
    - Unassign and pick another literal
  - Learning (later in lecture)
    - May allow more direct backtracking

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?
- How big is L1 cache? L2 cache?
- Ratio of main-memory speed to L1 cache speed?

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 one of these "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
  - 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
**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

---

**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

---

**Learning**

- When encounter a conflict
  - Determine variable assignment contributing to conflict
  - Add new clause to database
- New clause allows pruning

---

**Davis-Putnam w/ Learning**

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

• As perform bcp propagation
  – When set variable, insert back link to previous variable set forcing this variable set
  – Graph captures what this implication depends upon
• When encounter a conflict
  – Identify what variable values caused

Conflict Resolution

• x1 & /x9 & /x10 & /x11 lead to conflict
• /x1 & /x9 & /x10 & /x11
• /x1+x9+x10+x11 \rightarrow new clause for DB

More Implications

• x4 & /x10 & /x11 lead to conflict
• /x4+x10+x11 \rightarrow new clause for DB
• Also (/x4+x9+x4) since x1*/x9 \rightarrow x4

New Clause

• New clause does not include x12, x13
• May encounter this case again

Unique Implication Point

• UIP = vetext that dominates vertices leading to conflict
  – x1 is UIP (decision variable causing is always a UIP)
  – x4 is UIP
New Clauses

- \( x_4 + x_{10} + x_{11} \)
- Does not depend on \( x_9 \)
- \( \neg(x_1 + x_9 + x_4) \)
- \( x_4 \) not in decision tree
- Will be useful for later pruning

Clause Tradeoff

- Adding clauses facilitates implications
  - Increases pruning
  - Must make fewer decisions
- Adding clauses increases size of clause database
  - Increases memory
  - Could add exponential clauses
  - Requires more work to push implications

Learned Clauses

- Runtime = Decisions * ImplicationTime
  - Decisions decreasing
  - Implication Time increasing
- Starting from 0 learned clauses,
  - Net decrease in runtime
- Eventually, Implication Time too large and slows down
- Optimum with limited number of learned clauses

Limiting Learned Clauses

- Filter out dominated clauses
- Keep smaller clauses (fewer literals)
  - Have most relevance
- zChaff study suggests inserting only UIP closest to conflict [Zhang et al., ICCAD2001]
- Treat like cache and evict learned clauses
  - Use activity statistics as with variables so keep most useful clauses [minisat 1.2]

(Recall) Restarts

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

Impact of Learning

- zChaff [ICCAD2001] showed 2x improvement based on tuning the learning scheme
- Learning can be orders of magnitude benefit
Impact of Learning

Admin

• Reading
  – No new reading for Monday
  – Reading online for Wednesday
• Assignment 2: more graphs
  – More to come

Big Ideas

• Exploit Structure
  – Constraint propagation
  – Pruning search technique
  – Learning (discover structure)
• Constants matter
  – Exploit hierarchy in modern memory systems