GRASP: A Search Algorithm for Propositional Satisfiability

Authors: João P. Marques-Silva, and Karem A. Sakallah

Presentors: Jing Ji, Qilu Guo
Introduction

• Boolean Formula

• Boolean Satisfiability Problem (SAT)

• Conjunctive normal form (CNF)

• DPLL (David-Putnam-Longemann-Loveland)
  • Decision Tree
  • Backtrack

• Boolean constraint propagation (BCP)
Introduction

• **Boolean Formula**

Boolean Functions can be represented by formulae defined as well-formed sequence of:

- Literals: $a, \bar{a}, b, \bar{b}$
- Boolean operators: $OR (\lor), AND (\land), NOT (\neg)$
- Parentheses: ()

Example:

$$f = \bar{a}b + a\bar{b}$$

- Literals: $a, \bar{a}, b, \bar{b}$
- Sum of Products (SoP): can intuitively think of it as disjunction of conjunctions of literals
- Product of Sum (PoS): can intuitively think of it as conjunction of disjunctions of literals

$$f = (a + b) \cdot (\bar{a} + \bar{b})$$
Introduction

• **Boolean Satisfiability Problem (SAT)**

  The problem of determining if there exists an interpretation that satisfies a given Boolean formula

**Definition:**
- Given a Boolean formula \( f(a, b, \ldots) \), is there an assignment \((a_1, b_1, \ldots)\) such that \( f(a, b, \ldots) = 1 \)?
- If the answer is yes, then we say the formula is *satisfiable*
- Otherwise we say the formula is *unsatisfiable*

**Examples:**
- Is \( a \cdot \overline{a} \) satisfiable?
- Is \((a + c) \cdot (b + c) \cdot (\neg a + \neg b + \neg c)\) satisfiable?
- Is \((a + b) \cdot (\neg a + \neg b) \cdot (\neg a + b)\) satisfiable?
Introduction

• Conjunctive normal form (CNF)

  A product-of-sums (PoS) representation of a Boolean function
  • A sum term in a CNF is also called as a clause
  • Clausal normal form: a conjunction of clauses

  Unit Clause Rule:
  A clause is a unit clause if it has exactly one unassigned literal

Example:

$$\varphi = (a + c)(b + c)(\neg a + \neg b + \neg c)$$

Suppose $a$ and $b$ are assigned to 1. Then

$$\varphi = (1)(1)(\neg c)$$

The third clause is now a unit clause, and it implies that $c$ must be set to 0 to have the formula satisfied
Introduction

- **DPLL (David-Putnam-Longemann-Loveland)**
  Complete, backtracking-based depth-first search algorithm

**Example:**

\[ f = \neg(a + \neg b) \]

**Decision Tree:**
- Nodes represent variables
- Edges represent decisions
- Assignments are associated with decision level
- Ends either satisfiable (green) or unsatisfiable (red)
Introduction

• **DPLL (David-Putnam-Longemann-Loveland)**

  Example:

  \[ f = \neg(\neg a + \neg b) \]

  • Is actually the CNF form of AND logic

  **Backtracking:** If reaches an unsatisfiable conclusion

  • Return back one decision level
  • Redo the decision at that decision level
Introduction

• **Boolean constraint propagation (BCP)**
  - The basic mechanism for deriving implications from a given clause database
  - Unit propagation: The procedure is based on *unit clause*
  - The sequence of implications generated by BCP is captured by a *directed implication graph*

Example:

\[ \varphi = (a + c)(b + c)(\neg a + \neg b + \neg c) \]

If \(a\) and \(b\) are both assigned to 1,
\[ \varphi = (1)(1)(\neg c) \]
Then \(c\) is implied to be 0.
Outline

• Search Algorithm Template
• Conflict Analysis Procedure
• Experimental Results
• Conclusion
GRASP — Search Algorithm Template

- Search Algorithm Template
- Conflict Analysis Procedure
- Experimental Results
- Conclusion
GRASP — Search Algorithm Template

• **Decision Engine**
  • Choose a decision assignment for one literal at each stage
  • Maximize the number of clauses that are directly satisfied by this assignment
GRASP – Search Algorithm Template

• **Deduction Engine (BCP)**
  - Implements BCP and (implicitly) maintains the resulting implication graph
  - Repeatedly applies the unit clause rule and check for unsatisfiable clauses
GRASP – Search Algorithm Template

• Deduction Engine (BCP)

\[ \omega_1 = (\neg x_1 + x_2) \quad \rightarrow \quad \omega_1 = (x_2) \]
\[ \omega_2 = (\neg x_1 + x_3 + x_9) \quad \rightarrow \quad \omega_2 = (x_3) \]
\[ \omega_3 = (\neg x_2 + \neg x_3 + x_4) \]
\[ \omega_4 = (\neg x_4 + x_5 + x_{10}) \]
\[ \omega_5 = (\neg x_4 + x_6 + x_{11}) \]
\[ \omega_6 = (\neg x_5 + \neg x_6) \]
GRASP — Search Algorithm Template

- **Diagnosis Engine**
  - Identify the cause of conflict
    - Conflict learning
  - Determine the backtrack level
    - Nonchronological backtracking
Outline

• Conflict Analysis Procedure
• Experimental Results
• Conclusion
Conflict Analysis

- Conflict Analysis Procedure
  - Identify the causes of conflict
    - $x_1 = 1$, $x_9 = 0$, $x_{10} = 0$, $x_{11} = 0$
  - Create conflict-induced clause
    - $\omega_C(\kappa) = (\neg x_1 + x_9 + x_{10} + x_{11})$
  - Add $\omega_C(\kappa)$ to the clause database
  - Determine a backtrack level
Conflict Analysis

• **Backtracking**

  • Backtrack to the highest decision level

![Diagram of decision levels and variables](image)
Drawbacks of Conflict Diagnosis Engine

• Overhead due to conflict analysis:
  • Outweighed by the performance gain

• Exponentially growth in the size of clause database:
  • Selectively add the conflict-induced clause to the clause database
    • $\omega_{C1}(\kappa) = (\neg x_1 + x_9 + x_4)$ ✓
    • $\omega_{C2}(\kappa) = (\neg x_1 + x_9 + x_{10} + x_{11})$ ✗
  • Reduce the size of the implicates
    • $\omega_{C}(\kappa) = (\neg x_1 + x_9 + x_{10} + x_{11})$ ✗
    • $\omega_{C1}(\kappa) = (\neg x_1 + x_9 + x_4)$ & $\omega_{C2}(\kappa) = (\neg x_4 + x_{10} + x_{11})$ ✓
Outline

• Experimental Results

• Conclusion
Experimental Results

- **CPU Time (s)**
  - Performs better at some cases
  - Performs similar to those cases
  - POSIT performs better
  - Other solvers only perform better on certain cases

<table>
<thead>
<tr>
<th>Benchmark Class</th>
<th>#M</th>
<th>GRASP</th>
<th>POSIT</th>
<th>SATO</th>
<th>TEGUS</th>
<th>DPL</th>
<th>GSAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIM-100</td>
<td>24</td>
<td>1.8</td>
<td>1290</td>
<td>60390</td>
<td>107.9</td>
<td>58510</td>
<td>n/a</td>
</tr>
<tr>
<td>AIM-200</td>
<td>24</td>
<td>10.8</td>
<td>117991</td>
<td>150095</td>
<td>14059</td>
<td>156196</td>
<td>n/a</td>
</tr>
<tr>
<td>BF</td>
<td>4</td>
<td>7.2</td>
<td>20037</td>
<td>35695</td>
<td>26654</td>
<td>40000</td>
<td>n/a</td>
</tr>
<tr>
<td>DUBOIS</td>
<td>13</td>
<td>34.4</td>
<td>77189</td>
<td>71528</td>
<td>90333</td>
<td>96977</td>
<td>n/a</td>
</tr>
<tr>
<td>II32</td>
<td>17</td>
<td>7</td>
<td>650.1</td>
<td>10004</td>
<td>1231</td>
<td>21520</td>
<td>83814</td>
</tr>
<tr>
<td>PRET</td>
<td>8</td>
<td>18.2</td>
<td>40691</td>
<td>40430</td>
<td>42579</td>
<td>41429</td>
<td>n/a</td>
</tr>
<tr>
<td>SSA</td>
<td>8</td>
<td>6.5</td>
<td>85.3</td>
<td>30092</td>
<td>20230</td>
<td>80000</td>
<td>n/a</td>
</tr>
<tr>
<td>AIM-50</td>
<td>24</td>
<td>0.4</td>
<td>0.4</td>
<td>12.7</td>
<td>2.2</td>
<td>10.7</td>
<td>n/a</td>
</tr>
<tr>
<td>II8</td>
<td>14</td>
<td>23.4</td>
<td>2.3</td>
<td>0.4</td>
<td>11.8</td>
<td>84189</td>
<td>27647</td>
</tr>
<tr>
<td>JNH</td>
<td>50</td>
<td>21.3</td>
<td>0.8</td>
<td>11</td>
<td>6055</td>
<td>40</td>
<td>n/a</td>
</tr>
<tr>
<td>PAR8</td>
<td>10</td>
<td>0.4</td>
<td>0.1</td>
<td>0.2</td>
<td>1.5</td>
<td>0.8</td>
<td>50005</td>
</tr>
<tr>
<td>PAR16</td>
<td>10</td>
<td>9844</td>
<td>72.1</td>
<td>10447</td>
<td>9983</td>
<td>11741</td>
<td>100000</td>
</tr>
<tr>
<td>II16</td>
<td>10</td>
<td>10311</td>
<td>10120</td>
<td>85522</td>
<td>269.6</td>
<td>83933</td>
<td>11670</td>
</tr>
<tr>
<td>HANOI</td>
<td>2</td>
<td>14480</td>
<td>10117</td>
<td>20000</td>
<td>11.641</td>
<td>20000</td>
<td>20000</td>
</tr>
<tr>
<td>HOLE</td>
<td>5</td>
<td>12704</td>
<td>937.9</td>
<td>362.2</td>
<td>21301</td>
<td>11404</td>
<td>n/a</td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td>40000</td>
<td>40000</td>
<td>40000</td>
<td>40000</td>
<td>20079</td>
<td></td>
</tr>
</tbody>
</table>
Experimental Results

- **Statistics of Running GRASP**
  - Nonchronological backtracks are common
  - The growth of the clause database is acceptable

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#B</th>
<th>#NCB</th>
<th>%Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>aim-200-2_yes1-2</td>
<td>109</td>
<td>50</td>
<td>152.63</td>
</tr>
<tr>
<td>aim-200-2_no-2</td>
<td>39</td>
<td>20</td>
<td>43.6</td>
</tr>
<tr>
<td>bf0432-007</td>
<td>335</td>
<td>124</td>
<td>47.99</td>
</tr>
<tr>
<td>bf1355-075</td>
<td>40</td>
<td>20</td>
<td>6.5</td>
</tr>
<tr>
<td>dubois50</td>
<td>485</td>
<td>175</td>
<td>631.92</td>
</tr>
<tr>
<td>dubois100</td>
<td>1438</td>
<td>639</td>
<td>1033.54</td>
</tr>
<tr>
<td>pret60_40</td>
<td>147</td>
<td>98</td>
<td>407.08</td>
</tr>
<tr>
<td>pret150_75</td>
<td>388</td>
<td>257</td>
<td>446.75</td>
</tr>
<tr>
<td>ssa0432-003</td>
<td>37</td>
<td>6</td>
<td>30.8</td>
</tr>
<tr>
<td>ssa2670-141</td>
<td>377</td>
<td>97</td>
<td>65.71</td>
</tr>
<tr>
<td>ii16b1</td>
<td>88325</td>
<td>2588</td>
<td>131.94</td>
</tr>
</tbody>
</table>

#B: number of backtracks  
#NCB: number of nonchronological backtracks  
%Growth: the growth in size of the clause database
Conclusion

• GRASP
  • A faster search algorithm for solving SAT
  • Conflict learning to identify equivalent conflicting conditions
  • Nonchronological backtracking

• Future research work
  • Heuristic control of the rate of growth of the clause database
  • Improve the deduction engine
Q & A
Debate

• Will it be beneficial to split one large clause into several smaller ones?

• When doing nonchronological backtracking, is it better to return to the closest decision level, or to the level as far as possible?