Solving Propositional Satisfiability by Identification of Hard Subformulæ

R. Bruni and A. Sassano
University of Roma “La Sapienza”

ISMP 2000
Atlanta, GA, August 2000
Summary

1. Introduction and notation

2. Hard clauses and Minimally unsatisfiable subformulae (MUS)

3. Branching tree structure

4. The algorithm: Adaptive core search (ACS)

5. Computational results and conclusions
Every propositional logic formula can be expressed in Conjunctive Normal Form

\((\alpha_{i1} \lor \ldots \lor \alpha_{j1} \lor \neg \alpha_{j1+1} \lor \ldots \lor \neg \alpha_{n1}) \land \ldots \land (\alpha_{im} \lor \ldots \lor \alpha_{jm} \lor \neg \alpha_{jm+1} \lor \ldots \lor \neg \alpha_{nm})\)

Satisfiability Problem (NP-complete)

Is there a truth assignment for the logic variables (and if yes which is) such as the whole formula is satisfied (= is True)?
Notation

Ground set of the literals (posited or negated proposition)

\[ A = \{ a_i : a_i = \alpha_i \text{ for } i = 1, \ldots, n; \neg\alpha_{i-n} \text{ for } i = n+1, \ldots, 2n \} \]

Define \( \neg a_i = a_{i+n} \), and \( \neg a_{i+n} = a_i \)

Clause (set of literals)

\[ C_j = \{ a_i : i \in I_j \subseteq I \equiv \{1, \ldots, 2n\} \} \]

Instance (collection of sets of literals)

\[ \mathcal{F} = \{ C_j : j = 1, \ldots, m \} \]
**Truth Assignment**

Truth assignment

S = \{ a_i : a_i \in S \Rightarrow \neg a_i \notin S \}  
(set of literals)

Partial if |S| < n, complete if |S| = n

Completion

C(S) = \{ a_i : a_i \notin S \land \neg a_i \notin S \}  
(set of literals)

S satisfies \mathcal{F}  
\forall C_j \in \mathcal{F}, \ S \cap C_j \neq \phi

\mathcal{F} is unsatisfiable  
\forall S, \ \exists C_j \in \mathcal{F} : S \cap C_j = \phi
**Approaches**

- Complete methods
  
  Given enough time, are guaranteed to find the solution.
  
  Based on branching (DLL) and/or resolution (DP)

- Heuristics
  
  Are faster, but not guaranteed to find the solution.
  
  Not very useful for unsatisfiable instances.
**Generic branching procedure**

Time depends on:

- number of nodes of the search tree
- time needed at every single node

At every node,

1. Choose variable to branch on (branching rule)
   *(needs time)*

2. Fix the variable *(fast)*

3. Simplify the formula *(needs time)*
   (unit propagation: unit resolution and unit subsumption)
Possible solutions

- Reduce size of the branching tree,
  e.g. by improving effectiveness of the branching rule or by cutting some subtrees.

- Reduce time needed for the selection of the branching variable by simplifying its calculation.

- Reduce time needed for unit propagation by delaying some operations.
HARD CLAUSES

Given an instance $\mathcal{F}$, some clauses are more difficult to satisfy, that is are more constraining in the context of that particular instance.

Example of short clauses containing the same variables (hard)

\[
C_1 = \{ a_1, a_2 \} \quad C_2 = \{ a_1, \neg a_2 \} \quad C_3 = \{ \neg a_1, a_2 \}
\]

Example of long clauses containing different variables (easy)

\[
C_1 = \{ a_1, a_2, \neg a_3 \} \quad C_2 = \{ a_4, \neg a_5, a_6 \} \quad C_3 = \{ \neg a_7, a_8, a_9 \}
\]
INDIVIDUATION OF HARD CLAUSES

A priori :
observations made before, length l_j, ....

In itinere :
(solving the problem with a branching procedure)
# of visits v_j of a clause, # of failure f_j due to a clause

We evaluate clause hardness using
\[ \phi(C_j) = (v_j + pf_j)/ l_j \]

Calculation of \( \phi \) requires extremely small overhead and keeps improving throughout the computation.
Branching rule part 1: clause selection

- If we front hard clauses deep in the branching tree (current partial assignment S almost complete, C(S) small) usually we need to backtrack far.

- If we front hard clauses at the beginning of the branching tree (S small, C(S) wide) we solve them, or we discover unsatisfiability earlier.

Hard clauses must be fronted first:

We choose \( C_j = \text{argmax } \varphi(C_j) = (v_j + pf_j)/ l_j \)
Like most of complete methods, we use a branching procedure.

No binary branching, but clause-based branching.
Branching rule part 2: variable selection

- Within the clause, we need to choose the variable:

  Two sided Jeroslow Wang (Hooker)

  Choose \( a_i = \text{argmax} \sum_{C_j \subset a_i} 2^{-|C_j|} + \sum_{C_j \subset \neg a_i} 2^{-|C_j|} \)

  approximated as follows:

  let \( J(a_i) = 1 + \# \text{ binary clauses containing } a_i \)

  choose \( a_i = \text{argmax} J(a_i) J(\neg a_i) \)
Minimally Unsatisfiable Subformulae

Given an unsatisfiable instance $\mathcal{F}$, we can have collections of clauses $\mathcal{G} \subseteq \mathcal{F}$ which are still unsatisfiable

\[ \mathcal{G} \subseteq \mathcal{F} \text{ is minimally unsatisfiable (MUS)} \iff \exists S, \forall C_j \in \mathcal{G} : S \cap C_j = \emptyset \] (is unsatisfiable)

\[ \forall \mathcal{H} \subseteq \mathcal{G}, \exists S, \forall C_j \in \mathcal{H} : S \cap C_j \neq \emptyset \] (every subset is satisfiable)

$\mathcal{F}$ is unsatisfiable $\iff$ $\mathcal{F}$ contains a MUS
By collecting enough hard clauses (using $\varphi$) from an unsatisfiable instance, we can identify an unsatisfiable set of clauses.

If we stop collecting as soon as the set is unsatisfiable, we cannot say that it’s minimal.

We have a quick approximation of a minimally unsatisfiable subformula.
THE ALGORITHM - DEFINITIONS

Given an instance $\mathcal{F}$ and values for $\varphi$, define:

$$\varphi(\mathcal{F}) = \mathcal{C} = \{ C_j : C_j \in \mathcal{F}, \varphi(C_j) \geq \varphi(C_k) \ \forall \ C_k \in \mathcal{F}, \ |\mathcal{C}\ | < |\mathcal{F}| \}$$

(selection of the hardest clauses in the clause-set)

and $\mathcal{O} = \mathcal{F} \setminus \mathcal{C}$

Given a partial solution $S_k$ and a set of clauses $\mathcal{O}_k$, define:

$$\mathcal{N}_k = \{ C_j \in \mathcal{O}_k : C(S_k) \cap C_j = \phi \}$$  \hspace{1cm} \text{(falsified clauses)}

$$\mathcal{S}_k = \{ C_j \in \mathcal{O}_k : S_k \cap C_j \neq \phi \}$$  \hspace{1cm} \text{(satisfied clauses)}
Adaptive core search - I

**Preprocessing:** perform $p$ branching iterations on $\mathcal{F}$ to give initial values to $\varphi$

**Base step:** select an initial collection of the hardest clauses $\mathcal{C}_1 = \varphi(\mathcal{F})$. $\mathcal{C}_1$ is the first core, i.e. candidate to be a MUS. Remaining clauses form $\mathcal{O}_1$
Adaptive core search - II

**Iteration**: apply $h$ branching iterations to $\mathcal{C}_k$ ignoring $\mathcal{O}_k$

One of the following:

- $\mathcal{C}_k$ is unsatisfiable $\Rightarrow$ $\mathcal{F}$ is unsatisfiable, then STOP.

- No result after $h$ iteration $\Rightarrow$ contraction (allowed only $t$ times to ensure termination): put $\mathcal{C}_{k+1} = \varphi(\mathcal{C}_k)$, $k = k+1$, goto **Iteration**.

- $\mathcal{C}_k$ is satisfied by $S_k$ $\Rightarrow$ If $\mathcal{C}_k = \mathcal{F}$, $\mathcal{F}$ is satisfiable, STOP. Otherwise test $S_k$ on $\mathcal{O}_k$. One of the following: [next]
Adaptive core search - III

Test $S_k$ on $O_k$. One of the following:

- $S_k = O_k$, $N_k = \emptyset \Rightarrow \mathcal{F}$ is satisfied, then STOP

- $N_k \neq \emptyset \Rightarrow$ expansion: add $N_k$ to the core, obtain

  $\mathcal{C}_{k+1} = \mathcal{C}_k \cup N_k$ (note $\mathcal{C}_{k+1} \subseteq \mathcal{F}$), $k = k+1$, goto Iteration.

- $N_k = \emptyset$, $S_k \subset O_k \Rightarrow$ extension: keep $S_k$,

  put $\mathcal{C}_{k+1} = \mathcal{C}_k \cup \varphi(O_k)$, $k = k+1$, goto Iteration.
Features

- Complete method: In the worst case performs a complete branching.

- Factors of size reduction for the branching tree:
  1) To front hard clauses first.
  2) To prove unsatisfiability exploring only the core subtree.

- Factors of time reduction for branching variable selection:
  1) Easy to compute branching rule.
  2) To choose only within the current core.

- Reduces time needed for unit propagation by delaying it for clauses out of the core.
Many sat instances encode real world problems.

If a formula is unsatisfiable, it is useful to know which part of the formula cause this unsolvability.

That is the part respectively to remove or to keep when we respectively want such formula to be satisfiable or unsatisfiable.
<table>
<thead>
<tr>
<th>Problem</th>
<th>$n$</th>
<th>$m$</th>
<th>ACS 1.0</th>
<th>SATO 3.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>par16-1</td>
<td>1015</td>
<td>3310</td>
<td>10.10</td>
<td>24.16</td>
</tr>
<tr>
<td>par16-1-c</td>
<td>317</td>
<td>1264</td>
<td>11.36</td>
<td>2.62</td>
</tr>
<tr>
<td>par16-2</td>
<td>1015</td>
<td>3374</td>
<td>52.36</td>
<td>49.22</td>
</tr>
<tr>
<td>par16-2-c</td>
<td>349</td>
<td>1392</td>
<td>100.73</td>
<td>128.15</td>
</tr>
<tr>
<td>par16-3</td>
<td>1015</td>
<td>3344</td>
<td>103.92</td>
<td>40.81</td>
</tr>
<tr>
<td>par16-3-c</td>
<td>334</td>
<td>1332</td>
<td>8.19</td>
<td>78.91</td>
</tr>
<tr>
<td>par16-4</td>
<td>1015</td>
<td>3324</td>
<td>70.82</td>
<td>1.51</td>
</tr>
<tr>
<td>par16-4-c</td>
<td>324</td>
<td>1292</td>
<td>5.10</td>
<td>133.07</td>
</tr>
<tr>
<td>par16-5</td>
<td>1015</td>
<td>3358</td>
<td>224.84</td>
<td>4.92</td>
</tr>
<tr>
<td>par16-5-c</td>
<td>341</td>
<td>1360</td>
<td>72.29</td>
<td>196.33</td>
</tr>
</tbody>
</table>
# Series AIM-100 unsat

<table>
<thead>
<tr>
<th>Problem</th>
<th>n</th>
<th>m</th>
<th>C-sat</th>
<th>2cl</th>
<th>TabuS</th>
<th>BRR</th>
<th>ACS sel</th>
<th>ACS sol</th>
<th>ACS tot</th>
<th>n core</th>
<th>m core</th>
</tr>
</thead>
<tbody>
<tr>
<td>aim-100-1_6-no-1</td>
<td>100</td>
<td>160</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>0.17</td>
<td>0.03</td>
<td>0.20</td>
<td>43</td>
<td>48</td>
</tr>
<tr>
<td>aim-100-1_6-no-2</td>
<td>100</td>
<td>160</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>0.54</td>
<td>0.39</td>
<td>0.93</td>
<td>46</td>
<td>54</td>
</tr>
<tr>
<td>aim-100-1_6-no-3</td>
<td>100</td>
<td>160</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>0.62</td>
<td>0.73</td>
<td>1.35</td>
<td>51</td>
<td>57</td>
</tr>
<tr>
<td>aim-100-1_6-no-4</td>
<td>100</td>
<td>160</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>0.61</td>
<td>0.35</td>
<td>0.96</td>
<td>43</td>
<td>48</td>
</tr>
<tr>
<td>aim-100-2_0-no-1</td>
<td>100</td>
<td>200</td>
<td>52.19</td>
<td>19.77</td>
<td>409.50</td>
<td>5.78</td>
<td>0.03</td>
<td>0.01</td>
<td>0.04</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>aim-100-2_0-no-2</td>
<td>100</td>
<td>200</td>
<td>14.63</td>
<td>11.00</td>
<td>258.58</td>
<td>0.57</td>
<td>0.05</td>
<td>0.04</td>
<td>0.09</td>
<td>37</td>
<td>40</td>
</tr>
<tr>
<td>aim-100-2_0-no-3</td>
<td>100</td>
<td>200</td>
<td>56.21</td>
<td>6.53</td>
<td>201.15</td>
<td>2.95</td>
<td>0.04</td>
<td>0.01</td>
<td>0.05</td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td>aim-100-2_0-no-4</td>
<td>100</td>
<td>200</td>
<td>0.05</td>
<td>11.66</td>
<td>392.15</td>
<td>4.80</td>
<td>0.04</td>
<td>0.01</td>
<td>0.05</td>
<td>26</td>
<td>32</td>
</tr>
</tbody>
</table>
# Series II32 (sat)

<table>
<thead>
<tr>
<th>Problem</th>
<th>$n$</th>
<th>$m$</th>
<th>ACS 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ii32a1</td>
<td>459</td>
<td>9212</td>
<td>0.02</td>
</tr>
<tr>
<td>ii32b1</td>
<td>228</td>
<td>1374</td>
<td>0.00</td>
</tr>
<tr>
<td>ii32b2</td>
<td>261</td>
<td>2558</td>
<td>0.03</td>
</tr>
<tr>
<td>ii32b3</td>
<td>348</td>
<td>5734</td>
<td>0.03</td>
</tr>
<tr>
<td>ii32b4</td>
<td>381</td>
<td>6918</td>
<td>1.53</td>
</tr>
<tr>
<td>ii32c1</td>
<td>225</td>
<td>1280</td>
<td>0.00</td>
</tr>
<tr>
<td>ii32c2</td>
<td>249</td>
<td>2182</td>
<td>0.00</td>
</tr>
<tr>
<td>ii32c3</td>
<td>279</td>
<td>3272</td>
<td>2.84</td>
</tr>
<tr>
<td>ii32c4</td>
<td>759</td>
<td>20862</td>
<td>5.07</td>
</tr>
<tr>
<td>ii32d1</td>
<td>332</td>
<td>2730</td>
<td>0.01</td>
</tr>
<tr>
<td>ii32d2</td>
<td>404</td>
<td>5153</td>
<td>0.76</td>
</tr>
<tr>
<td>ii32d3</td>
<td>824</td>
<td>19478</td>
<td>7.49</td>
</tr>
<tr>
<td>ii32e1</td>
<td>222</td>
<td>1186</td>
<td>0.00</td>
</tr>
<tr>
<td>ii32e2</td>
<td>267</td>
<td>2746</td>
<td>0.01</td>
</tr>
<tr>
<td>ii32e3</td>
<td>330</td>
<td>5020</td>
<td>0.08</td>
</tr>
<tr>
<td>ii32e4</td>
<td>387</td>
<td>7106</td>
<td>0.02</td>
</tr>
<tr>
<td>ii32e5</td>
<td>522</td>
<td>11636</td>
<td>1.03</td>
</tr>
</tbody>
</table>
Conclusions

- Techniques to perform a fast complete enumeration are widely proposed in literature (e.g. sophisticated data structure, ...).

- Here a technique is presented to reduce the set that enumeration works on.

- In practical scenarios it is useful to know which part of the instance cause the unsolvability.

  Results are extremely encouraging. We believe better performances can be obtained by using both of the above techniques.