Compressing UNSAT Search Trees with Caching: an update

Anthony BLOMME, Daniel LE BERRE, Anne PARRAIN, Olivier ROUSSEL

CRIL, Université d’Artois & CNRS

July, 2023
Machine Learning is everywhere

The rationale of the outcome of those “black boxes” is hard to explain making XAI a very trendy topic

Many people no longer trust computer programs

Even if it is a deterministic constraint programming solver
What is an explanation for a SAT solver?

- several target users (solver expert, modeling expert or user)
- several levels of explanation (clauses, high level constraints, …)
- foundation of the explanation (logical reasoning, statistical reasoning, …)
- …
What is an explanation for a SAT solver?

- several target users (solver expert, modeling expert or user)
- several levels of explanation (clauses, high level constraints, ...)
- foundation of the explanation (logical reasoning, statistical reasoning, ...)
- ...

This work: focus on the information provided by the solver, i.e. it's search tree
Case of satisfiable formulas

Why is it a solution?

- All the clauses are satisfied
- Compact representation (prime implicant)

Why this particular solution?

- Logical justification: logical implication (which reduces to UNSAT proof), backbone
- Statistical justification: probability(\(x = \text{true}\))
- Solver’s decisions have no logical explanation!
$ java -Dcolor -jar org.sat4j.core.jar file.cnf

s SATISFIABLE
v 1 2 -3 4 5 -6 -7 -8 -9 -10 11 -12 -13 14 15 16 -17 -18 19 -20 -21 -22 23 24 25 26 -
27 28 -29 -30 -31 32 -33 34 35 36 37 38 39 -40 -41 42 -43 -44 -45 -46 -47 -48 49 50 -
-75 -76 -77 -78 79 -80 -81 -82 83 -84 -85 86 -87 88 89 -90 -91 -92 -93 94 95 -96 97 -
98 99 -100 0
c UNASSIGNED: 0 DECIDED: 0 PROPAGATED_ORIGINAL: 69 PROPAGATED_LEARNED: 29 DECIDED_PROPAGATED: 1 DECIDED_PROPAGATED_LEARNED: 0 DECIDED_CYCLE: 1
c Total wall clock time (in seconds) : 0.01
Why is there no solution?

Prove the impossibility of a solution

- UNSAT certificate or MUS (too large in general)
- Reduce a posteriori the size of the search tree:
  - Delete useless decisions and propagations
  - Reorder the nodes
  - Recognize recurring patterns
Case of unsatisfiable formulas

Why is there no solution?

*Prove* the impossibility of a solution

- UNSAT certificate or MUS (too large in general)
- Reduce a posteriori the size of the search tree:
  - Delete useless decisions and propagations
  - Reorder the nodes
  - Recognize recurring patterns
Recognize recurring patterns

Recognize equivalent subformulas (renamings, inclusions).

Do not explain the unsatisfiability of a formula twice.

Link similar proofs together (single explanation).
Motivating example: the pigeon hole problem

\[
\begin{align*}
\text{PHP}_n & \quad x_{1,1} \lor x_{1,2} \lor \ldots \lor x_{1,n} \\
\text{PHP}_{n-1} & \quad x_{2,1} \lor x_{2,2} \lor \ldots \lor x_{2,n} \\
& \quad \vdots \\
\text{PHP}_2 & \quad x_{n-1,1} \lor x_{n-1,2} \lor \ldots \lor x_{n-1,n} \\
& \quad x_{n,1} \lor x_{n,2} \lor \ldots \lor x_{n,n} \\
& \quad x_{n+1,1} \lor x_{n+1,2} \lor \ldots \lor x_{n+1,n}
\end{align*}
\]
Idea: Build a cache with proven unsatisfiable subformulas and try to recognize them later

Inspired by model counters and compilers, here specialized to the UNSAT case.

- Light minimization: use only the clauses involved in the conflict (sources)
- Use a normalized representation to register subformulas
- If a subformula is equal to an entry of the cache, we can prune the branch
Idea: Build a cache with proven unsatisfiable subformulas and try to recognize them later

Inspired by model counters and compilers, here specialized to the UNSAT case.

- Light minimization: use only the clauses involved in the conflict (sources)
- Use a normalized representation to register subformulas
- If a subformula is equal to an entry of the cache, we can prune the branch

Does not work on PHP example: sub-PHP instances are built on different variables and clauses

If the subformula contain the cache entry it is also UNSAT
Generalize equality:

- detect if an entry of the cache is a subset of the current subformula
- allow variable renaming

**Subgraph isomorphism** allows to test if, after renaming the variables, an entry of the cache is included in the current subformula. If it is the case, we can prune the branch.

Glasgow Subgraph Solver is used to detect subgraph isomorphisms (⇒ classic encoding of subformulas).
Colored graph representation of the formula

\((\neg x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2)\)
Colored graph representation of the formula
\((\neg x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2)\)
When to cache?

- We have to be sure that the entry corresponds to an UNSAT formula.
- With a DPLL approach, it can be done for any node in the search tree
  - on the leaves, corresponding to conflicts
  - on internal nodes, once both children are found UNSAT
- With a CDCL approach, things are more complicated ...
Problem 1: When backjumping, the search is not complete and we do not know if the unexplored subformulas are unsatisfiable.

The caching is performed at the leaves, when encountering a conflict.
Problem 2 (technical): When we hit an entry in the cache, we need a conflict clause to backtrack. How to build it?

When recognizing an entry, we create a conflict composed of the falsified literals in the matching clauses. The conflict analysis can be performed with this clause.

If those literals are not from the current decision level, backtrack to the lowest decision level before performing conflict analysis.
marg2x3.cnf instance with CDCL

Without cache

With cache

New entry
Cache hit
Generalized isomorphisms

Expecting $x_1 \lor x_2$ and got $x_a \lor x_b \lor x_c \lor x_d$: matches?
Expecting $x_1 \lor x_2$ and got $x_a \lor x_b \lor x_c \lor x_d$: matches?

Expecting $x_1 \lor x_2 \lor x_3$ and got $x_a \lor x_b \lor x_c \lor x_d$: matches?
Generalized isomorphisms

Expecting $x_1 \lor x_2$ and got $x_a \lor x_b \lor x_c \lor x_d$: matches?

Expecting $x_1 \lor x_2 \lor x_3$ and got $x_a \lor x_b \lor x_c \lor x_d$: matches?

Detect entries of the cache even if some literals are falsified.

- Do not delete satisfied clauses and satisfied literals are considered unassigned
- Create variants of clauses with falsified literals. Create all the possibilities from the complete original clause to the clause with all falsified literals removed
- Selector nodes are used to avoid using several variants of a same clause

Encoding of exponential size but the number of added clauses is, in general, reasonable compared to the original number of clauses.
Generalized isomorphisms colored graph

\[ C_{5,1} \]

\[ C_{5,2} \]

\[ C_{5,3} \]

\[ C_{5,4} \]
Implemented on top of Minisat.

The cache lookup is performed before taking a decision.

Cache lookup is translated into a subgraph isomorphism problem and then given to Glasgow Subgraph Solver.

Time limit of 2 seconds (regular isomorphisms) or 4 seconds (generalized isomorphisms) for each call to Glasgow Subgraph Solver.
Environment for the experiments

- We consider UNSAT instances from the SAT’02 and SAT’03 competitions
  - “Easy” for DPLL and CDCL
  - Small enough for expensive algorithms
- A total of 579 UNSAT instances
  - SAT’02: 381 instances
  - SAT’03: 198 instances
- Time limit:
  - Regular isomorphisms: 15 minutes
  - Generalized isomorphisms: 30 minutes
Some results (pruning after search, regular)

<table>
<thead>
<tr>
<th>Instance</th>
<th>Size</th>
<th>Conflicts (no cache)</th>
<th>Conflicts (cache)</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>(PHP_7)</td>
<td>448</td>
<td>(5.6 \times 10^3)</td>
<td>853</td>
<td>(1.5 \times 10^{-1})</td>
</tr>
<tr>
<td>(PHP_{12})</td>
<td>2,028</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>marg2x6.sat03-1444</td>
<td>528</td>
<td>(3.0 \times 10^4)</td>
<td>20</td>
<td>(6.6 \times 10^{-4})</td>
</tr>
<tr>
<td>marg3x3add8.sat03-1449</td>
<td>1,056</td>
<td>(1.8 \times 10^5)</td>
<td>32</td>
<td>(1.8 \times 10^{-4})</td>
</tr>
<tr>
<td>Urquhart-s3-b9</td>
<td>1,240</td>
<td>(1.9 \times 10^4)</td>
<td>21</td>
<td>(1.1 \times 10^{-3})</td>
</tr>
<tr>
<td>Urquhart-s3-b3</td>
<td>2,152</td>
<td>(1.6 \times 10^6)</td>
<td>29</td>
<td>(1.8 \times 10^{-5})</td>
</tr>
<tr>
<td>x1.16</td>
<td>364</td>
<td>(2.2 \times 10^3)</td>
<td>20</td>
<td>(9.1 \times 10^{-3})</td>
</tr>
<tr>
<td>x1.24</td>
<td>556</td>
<td>(2.0 \times 10^5)</td>
<td>78</td>
<td>(3.9 \times 10^{-4})</td>
</tr>
<tr>
<td>3col20.5.6</td>
<td>646</td>
<td>27</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>3col40.5.4</td>
<td>1,286</td>
<td>92</td>
<td>64</td>
<td>(7.0 \times 10^{-1})</td>
</tr>
<tr>
<td>homer06</td>
<td>1,800</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- 117/579 instances solved
- Some traces were too large to be postprocessed
Some results (pruning during search, regular)

<table>
<thead>
<tr>
<th>Instance</th>
<th>Conflicts</th>
<th>Cache size</th>
<th>Subgraph Isomorphisms</th>
<th>Calls</th>
<th>Time (Search)</th>
<th>Time (GSS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PHP_7$</td>
<td>47</td>
<td>41</td>
<td>29 (8)</td>
<td>259</td>
<td>0.025</td>
<td>1.550</td>
</tr>
<tr>
<td>$PHP_{12}$</td>
<td>116</td>
<td>107</td>
<td>96 (20)</td>
<td>589</td>
<td>0.189</td>
<td>18.412</td>
</tr>
<tr>
<td>$PHP_{16}$</td>
<td>187</td>
<td>178</td>
<td>167 (32)</td>
<td>1020</td>
<td>0.731</td>
<td>166.088</td>
</tr>
<tr>
<td>marg2x6.sat03-1444</td>
<td>20</td>
<td>17</td>
<td>18 (17)</td>
<td>44</td>
<td>0.007</td>
<td>0.341</td>
</tr>
<tr>
<td>marg3x3add8.sat03-1449</td>
<td>32</td>
<td>25</td>
<td>20 (20)</td>
<td>55</td>
<td>0.022</td>
<td>0.524</td>
</tr>
<tr>
<td>marg6x6.sat03-1456</td>
<td>86</td>
<td>84</td>
<td>84 (84)</td>
<td>276</td>
<td>0.181</td>
<td>15.190</td>
</tr>
<tr>
<td>Urquhart-s3-b9</td>
<td>21</td>
<td>18</td>
<td>17 (17)</td>
<td>38</td>
<td>0.009</td>
<td>0.329</td>
</tr>
<tr>
<td>Urquhart-s3-b3</td>
<td>29</td>
<td>26</td>
<td>27 (25)</td>
<td>59</td>
<td>0.023</td>
<td>0.861</td>
</tr>
<tr>
<td>Urquhart-s5-b5</td>
<td>95</td>
<td>91</td>
<td>91 (90)</td>
<td>259</td>
<td>0.367</td>
<td>55.682</td>
</tr>
<tr>
<td>x1_16</td>
<td>18</td>
<td>15</td>
<td>14 (14)</td>
<td>60</td>
<td>0.010</td>
<td>0.693</td>
</tr>
<tr>
<td>x1_24</td>
<td>40</td>
<td>35</td>
<td>32 (18)</td>
<td>202</td>
<td>0.022</td>
<td>1.665</td>
</tr>
<tr>
<td>x1_96</td>
<td>2177</td>
<td>471</td>
<td>106 (76)</td>
<td>8513</td>
<td>1.759</td>
<td>423.444</td>
</tr>
<tr>
<td>3col20_5_6</td>
<td>27</td>
<td>5</td>
<td>0 (0)</td>
<td>15</td>
<td>0.005</td>
<td>0.099</td>
</tr>
<tr>
<td>3col40_5_4</td>
<td>110</td>
<td>22</td>
<td>54 (3)</td>
<td>786</td>
<td>0.107</td>
<td>5.535</td>
</tr>
<tr>
<td>homer06</td>
<td>102</td>
<td>95</td>
<td>92 (20)</td>
<td>462</td>
<td>0.485</td>
<td>47.701</td>
</tr>
</tbody>
</table>

- 185/579 instances solved
Some results (pruning during search, generalized)

<table>
<thead>
<tr>
<th>Instance</th>
<th>CDCL (integrated cache, generalized isomorphisms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Conflicts</td>
</tr>
<tr>
<td>PHP₅</td>
<td>23</td>
</tr>
<tr>
<td>PHP₇</td>
<td>42</td>
</tr>
<tr>
<td>marg2x6.sat03-1444</td>
<td>20</td>
</tr>
<tr>
<td>marg3x3add8.sat03-1449</td>
<td>31</td>
</tr>
<tr>
<td>marg4x4.sat03-1454</td>
<td>41</td>
</tr>
<tr>
<td>Urquhart-s3-b9</td>
<td>21</td>
</tr>
<tr>
<td>Urquhart-s3-b3</td>
<td>29</td>
</tr>
<tr>
<td>x₁₁₆</td>
<td>18</td>
</tr>
<tr>
<td>x₁₂₄</td>
<td>29</td>
</tr>
<tr>
<td>x₂₃₂</td>
<td>65</td>
</tr>
<tr>
<td>3col20_5_6</td>
<td>23</td>
</tr>
<tr>
<td>3col40_5_4</td>
<td>57</td>
</tr>
</tbody>
</table>

- 89/579 instances solved
Conclusion

- Our goal is to reduce drastically some UNSAT search trees so that they can be shown to the user
- We propose a syntactic approach based on the detection of subgraph isomorphisms
- Interesting results obtained on some highly structured families of instances
- Future research directions:
  - Better encoding for managing assigned literals in cache entries
  - Delete entries that do not seem useful
  - Incremental use of Glasgow Subgraph Solver
  - Try other heuristics
Compressing UNSAT Search Trees with Caching: an update

Anthony BLOMME, Daniel LE BERRE, Anne PARRAIN, Olivier ROUSSEL

CRIL, Université d’Artois & CNRS

July, 2023