Revisiting Decision Diagrams for SAT

Tom van Dijk\textsuperscript{1} \quad Rüdiger Ehlers\textsuperscript{2} \quad Armin Biere\textsuperscript{1}

\textsuperscript{1} Johannes Kepler University Linz, Austria
\textsuperscript{2} University of Bremen, Germany

PoCR’17
Programatics for Constraint Reasoning
affiliated to CP’17, ILCP’17, and SAT’17

Melbourne, Australia

August 28, 2017
Motivation

- SAT solvers are used almost everywhere, but also...
- ... increasing use of SAT solvers for hard combinatorial problems
  - Pythagorean Triples Problem (PTN) (CACM August 2017: The Science of Brute Force)
  - verifying arithmetic circuits
  - cryptanalysis

- features of these hard problems
  - “few variables” in the thousands (PTN has 7825)
  - no short resolution proofs (200 TB)
  - plain CDCL SAT solvers do not work

- binary decision diagrams (BDDs)
  - one fixed variable order / bad on large industrial instances
  - symbolic representation might give exponential speed-ups
  - much more memory and many more cores today
  - new paradigms such as cube-and-conquer
Pigeon Hole Problem (PHP$_n$)

- fit $n + 1$ pigeons into $n$ holes

\[
\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} p_{i,j}
\]

each pigeon $i$ in at least one hole $j$

\[
\bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n+1} \bigwedge_{k=1}^{n} (\overline{p_{i,k}} \vee \overline{p_{j,k}})
\]

pigeon $i$ and pigeon $j$ not in the same hole $k$

- [Haken’85] showed that all resolution refutations of PHP$_n$ are exponential
  - thus also hard for plain CDCL SAT solving …
  - … which in principle is as good as general resolution
  - so is a prototypical benchmark to test new ideas

- can be solved faster than with CDCL SAT solving by
  - directly building BDDs, or performing variable elimination over ZDDs
  - empirical results [ChatalicSimon’03] are old ⇒ revisit

- actually trivial to solve by cardinality reasoning
Binary Decision Diagram (BDD)  [Bryant’86]

\[(a \lor c) \land (\bar{a} \lor \bar{c}) = a \oplus c\] XOR

\[
\begin{align*}
\text{a ? 1 : (c ? 1 : 0)} &= \text{a ? (c ? 0 : 1) : 1} \\
\Diamond (a, 1, \Diamond (c, 1, 0)) &= \Diamond (a, \Diamond (c, 0, 1), 1) \\
\Diamond (a, \Diamond (c, 0, 1), \Diamond (c, 1, 0)) &= \Diamond (a, 1, \Diamond (c, 1, 0))
\end{align*}
\]
BDD Apply Algorithm

\[
\begin{align*}
0 \land 0 &= 0 \\
0 \land 1 &= 0 \\
1 \land 0 &= 0 \\
1 \land 1 &= 1 \\
\diamondsuit(x, f_1, f_2) \land \diamondsuit(x, g_1, g_2) &= \diamondsuit(x, (f_1 \land g_1), (f_2 \land g_2)) \\
\text{modulo} \\
\diamondsuit(x, f, f) &= f
\end{align*}
\]

works the same for other boolean operators \( \lor, \oplus, \ldots \)
Zero Suppressed Decision Diagram (ZDD)  [Minato’93, ChatalicSimon’03]

\[
\{\{a, c\}\} \cup \{\\overline{a}, \overline{c}\}\} = \{\{a, c\}, \{\overline{a}, \overline{c}\}\}
\]

\[
\begin{align*}
\text{a.}(c.1 \cup 0) \cup 0 & = \text{\(a\).\(c\).1 \cup 0\) \cup 0} \\
\text{with} \ 0 = \{\}, \ 1 = \{\{\}\}, \ x.P = \{\{x\}\} \cup S \ | \ S \in P
\end{align*}
\]

\[
\begin{align*}
\Delta(a, \Delta(c, 1, 0), 0) & = \Delta(a, \Delta(c, 1, 0), 0) \\
\Delta(\overline{a}, \Delta(\overline{c}, 1, 0), 0) & = \Delta(a, \Delta(b, 1, 0), \Delta(\overline{a}, \Delta(\overline{b}, 1, 0), 0))
\end{align*}
\]
Example clauses:

- $a \lor b \lor \neg c$
- $a \lor b \lor \neg d$
- $\neg b \lor \neg c$
- $\neg b \lor \neg d$
- $c \lor d$
CNF → ZDD (2/4)
CNF $\rightarrow$ ZDD (3/4)
CNF $\rightarrow$ ZDD (4/4)
ZDD Apply Algorithm

\[
\begin{align*}
0 \cup 0 &= 0 \\
0 \cup 1 &= 1 \\
1 \cup 0 &= 1 \\
1 \cup 1 &= 1
\end{align*}
\]

\[
\triangle(x, f_1, f_2) \cup \triangle(x, g_1, g_2) = \triangle(x, (f_1 \cup g_1), (f_2 \cup g_2))
\]

modulo

\[
\triangle(x, 0, f) = 0
\]

works the same for other set operations \(\cap, \setminus, \ldots\)

again with \(0 = \{\} \) and \(1 = \{\{\}\}\)
**CNF → BDD**

- parse CNF and build individual BDD for each clause
- keep a BDD representing conjunction of all previously read clauses
- add BDD for new clause with (parallelized) BDD apply algorithm

**CNF → ZDD**

- parse whole CNF into integer array
- divide-and-conquer recursive union of clauses as ZDD (parallelized)
- base case is to build a ZDD for individual clauses

**ZDD → BDD**

- build BDDs recursively over whole ZDD
  \[
  \text{zdd2bdd}(\triangle(x, f_1, f_2)) = \text{zdd2bdd}(f_2) \lor \Diamond(x, \text{zdd2bdd}(f_1), 0)
  \]
- again using work-stealing and task parallelism in “∨” and “zdd2bdd(⋯)”
<table>
<thead>
<tr>
<th>cores</th>
<th>ph10</th>
<th>ph11</th>
<th>ph12</th>
<th>ph13</th>
<th>ph14</th>
<th>ph15</th>
<th>ph16</th>
<th>ph17</th>
<th>ph18</th>
<th>ph19</th>
<th>ph20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.43</td>
<td>0.46</td>
<td>0.50</td>
<td>0.61</td>
<td>0.90</td>
<td>1.54</td>
<td>3.02</td>
<td>6.31</td>
<td>18.18</td>
<td>37.15</td>
<td>58.95</td>
</tr>
<tr>
<td>2</td>
<td>0.62</td>
<td>0.63</td>
<td>0.73</td>
<td>0.82</td>
<td>1.04</td>
<td>1.19</td>
<td>2.20</td>
<td>4.29</td>
<td>9.61</td>
<td>22.27</td>
<td>33.38</td>
</tr>
<tr>
<td>4</td>
<td>0.61</td>
<td>0.63</td>
<td>0.67</td>
<td>0.72</td>
<td>0.87</td>
<td>1.17</td>
<td>1.77</td>
<td>3.15</td>
<td>8.45</td>
<td>18.55</td>
<td>143.05</td>
</tr>
<tr>
<td>6</td>
<td>0.37</td>
<td>0.41</td>
<td>0.39</td>
<td>0.47</td>
<td>0.59</td>
<td>0.83</td>
<td>1.31</td>
<td>2.44</td>
<td>5.87</td>
<td>12.50</td>
<td>98.28</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>0.71</td>
<td>0.76</td>
<td>0.81</td>
<td>0.87</td>
<td>1.06</td>
<td>1.45</td>
<td>2.45</td>
<td>5.04</td>
<td>10.98</td>
<td>132.32</td>
</tr>
<tr>
<td>10</td>
<td>0.79</td>
<td>0.84</td>
<td>0.87</td>
<td>0.94</td>
<td>1.06</td>
<td>1.30</td>
<td>1.98</td>
<td>2.88</td>
<td>5.63</td>
<td>10.52</td>
<td>93.82</td>
</tr>
<tr>
<td>12</td>
<td>0.26</td>
<td>0.29</td>
<td>0.32</td>
<td>0.42</td>
<td>0.54</td>
<td>0.83</td>
<td>1.28</td>
<td>2.35</td>
<td>5.27</td>
<td>10.06</td>
<td>108.49</td>
</tr>
<tr>
<td>14</td>
<td>0.34</td>
<td>0.37</td>
<td>0.42</td>
<td>0.48</td>
<td>0.63</td>
<td>0.86</td>
<td>1.34</td>
<td>2.31</td>
<td>4.83</td>
<td>9.22</td>
<td>102.03</td>
</tr>
<tr>
<td>16</td>
<td>0.81</td>
<td>0.81</td>
<td>0.86</td>
<td>0.91</td>
<td>1.05</td>
<td>1.31</td>
<td>1.79</td>
<td>2.72</td>
<td>4.92</td>
<td>9.01</td>
<td>57.23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cores</th>
<th>ph10</th>
<th>ph11</th>
<th>ph12</th>
<th>ph13</th>
<th>ph14</th>
<th>ph15</th>
<th>ph16</th>
<th>ph17</th>
<th>ph18</th>
<th>ph19</th>
<th>ph20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.93</td>
<td>0.96</td>
<td>1.00</td>
<td>1.11</td>
<td>1.39</td>
<td>2.04</td>
<td>3.52</td>
<td>6.80</td>
<td>18.80</td>
<td>38.46</td>
<td>54.50</td>
</tr>
<tr>
<td>2</td>
<td>0.94</td>
<td>0.94</td>
<td>1.00</td>
<td>1.08</td>
<td>1.27</td>
<td>1.67</td>
<td>2.53</td>
<td>4.62</td>
<td>11.27</td>
<td>23.04</td>
<td>37.13</td>
</tr>
<tr>
<td>4</td>
<td>0.94</td>
<td>0.96</td>
<td>0.98</td>
<td>1.04</td>
<td>1.17</td>
<td>1.47</td>
<td>2.02</td>
<td>3.61</td>
<td>8.99</td>
<td>16.36</td>
<td>29.42</td>
</tr>
<tr>
<td>6</td>
<td>0.47</td>
<td>0.47</td>
<td>0.48</td>
<td>0.53</td>
<td>0.63</td>
<td>0.95</td>
<td>1.37</td>
<td>2.44</td>
<td>6.96</td>
<td>14.07</td>
<td>105.88</td>
</tr>
<tr>
<td>8</td>
<td>0.18</td>
<td>0.17</td>
<td>0.20</td>
<td>0.23</td>
<td>0.35</td>
<td>0.59</td>
<td>1.00</td>
<td>1.83</td>
<td>5.39</td>
<td>11.08</td>
<td>138.65</td>
</tr>
<tr>
<td>10</td>
<td>0.54</td>
<td>0.50</td>
<td>0.55</td>
<td>0.60</td>
<td>0.68</td>
<td>0.91</td>
<td>1.47</td>
<td>2.76</td>
<td>5.63</td>
<td>11.57</td>
<td>174.56</td>
</tr>
<tr>
<td>12</td>
<td>0.52</td>
<td>0.54</td>
<td>0.57</td>
<td>0.63</td>
<td>0.73</td>
<td>1.00</td>
<td>1.48</td>
<td>2.79</td>
<td>5.84</td>
<td>11.90</td>
<td>91.31</td>
</tr>
<tr>
<td>14</td>
<td>0.49</td>
<td>0.49</td>
<td>0.53</td>
<td>0.56</td>
<td>0.69</td>
<td>0.89</td>
<td>1.32</td>
<td>2.62</td>
<td>5.68</td>
<td>10.90</td>
<td>94.71</td>
</tr>
<tr>
<td>16</td>
<td>0.41</td>
<td>0.46</td>
<td>0.47</td>
<td>0.56</td>
<td>0.65</td>
<td>0.89</td>
<td>1.29</td>
<td>2.34</td>
<td>5.15</td>
<td>10.74</td>
<td>19.01</td>
</tr>
</tbody>
</table>
New Compact Notation for ZDD encoding CNF

\[ \nabla(v, f_1, f_2, f_3) = \triangle(v, f_1, \triangle(\bar{v}, f_2, f_3)) \]

assuming no node \( \triangle(v, \triangle(\bar{v}, \ldots, \ldots), \ldots) \) exists

corresponds to CNF containing trivial clauses with both \( v \) and \( \bar{v} \)

\[ \triangle(a, \triangle(b, 1, 0), \triangle(\bar{a}, \triangle(\bar{b}, 1, 0), 0)) \]

\[ \nabla(a, \nabla(b, 1, 0, 0), \nabla(\bar{b}, 1, 0, 0)) \]
Union with New Notation

\[
\begin{align*}
0 \cup 0 &= 0 \\
0 \cup 1 &= 1 \\
1 \cup 0 &= 1 \\
1 \cup 1 &= 1 \\
\triangle(x, f_1, f_2, f_3) \cup \triangle(x, g_1, g_2, g_3) &= \nabla(x, (f_1 \cup g_1), (f_2 \cup g_2), (f_3 \cup g_3)) \\
\text{modulo} \\
\nabla(u, f_1, f_2, f_3) \cup \nabla(v, g_1, g_2, g_3) &= \nabla(u, f_1, f_2, f_3) \cup \nabla(u, 0, 0, \nabla(v, g_1, g_2, g_3)) \\
\text{if } u < v
\end{align*}
\]
Subsumption with New Notation

\[
\begin{align*}
0 \triangleright f &= 0 \\
1 \triangleright 1 &= 1 \\
1 \triangleright \nabla(x, f_1, f_2, f_3) &= 1 \\
\nabla(x, f_1, f_2, f_3) \triangleright 1 &= 1 \\
\nabla(x, f_1, f_2, f_3) \triangleright 0 &= \nabla(x, f_1, f_2, f_3)
\end{align*}
\]

\[
\nabla(x, f_1, f_2, f_3) \triangleright \nabla(x, g_1, g_2, g_3) = \nabla(x, (f_1 \triangleright g_1) \triangleright g_3, (f_2 \triangleright g_2) \triangleright g_3, f_3 \triangleright g_3)
\]

Self-Subsumption Makes ZDD / CNF Subsumption-Free

\[
\begin{align*}
SF(0) &= 0 \\
SF(1) &= 1
\end{align*}
\]

\[
SF(\nabla(x, f_1, f_2, f_3)) = \nabla(x, SF(f_1) \triangleright SF(f_3), SF(f_2) \triangleright SF(f_3), SF(f_3))
\]
Subsumption-Free Union (Logical Conjunction)

\[ 0 \cup_S f = f \]
\[ 1 \cup_S f = 1 \]
\[ f \cup_S 0 = f \]
\[ f \cup_S 1 = 1 \]

\[ \nabla(x, f_1, f_2, f_3) \cup_S \nabla(x, g_1, g_2, g_3) = \]
\[ \nabla(x, (f_1 \cup_S g_1) \setminus (f_3 \cup_S g_3), (f_2 \cup_S g_2) \setminus (f_3 \cup_S g_3), (f_3 \cup_S g_3)) \]

Subsumption-Free Clause Distribution (Logical Disjunction)

\[ 0 \times_S f = 0 \]
\[ 1 \times_S f = f \]
\[ f \times_S 0 = 0 \]
\[ f \times_S 1 = f \]

\[ \nabla(x, f_1, f_2, f_3) \times_S \nabla(x, g_1, g_2, g_3) = \]
\[ \nabla(x, ((f_1 \times_S g_1) \cup_S (f_1 \times_S g_3) \cup_S (f_3 \times_S g_1)) \setminus (f_3 \times_S g_3),
   ((f_2 \times_S g_2) \cup_S (f_2 \times_S g_3) \cup_S (f_3 \times_S g_2)) \setminus (f_3 \times_S g_3), (f_3 \times_S g_3)) \]
Clause Distribution – Davis Putnam Procedure (DP)

- eliminate variables from CNF one-by-one  [DavisPutnam’58]
- resolve all clauses with variable \( x \) with all clauses with \( \overline{x} \)
- add resolvents after removing clauses with \( x \) and \( \overline{x} \)

Symbolic Variable Elimination

- DP but on ZDD encoded CNF  [ChatalicSimon’03]
- was combined with subsumption removal and solves PHP problems
- high compression ratio  \#clauses / \#nodes

Bounded Variable Elimination (BVE)

- only eliminate variables if CNF size does not increase  [EenBiere’05]
- combined with subsumption removal
- most effective preprocessing
Bounded Symbolic Variable Elimination

- symbolic variable elimination / clause distribution
- eagerly eliminate variables which do not increase size
- if all variable increase size eliminate one with smallest increase

Experiments

<table>
<thead>
<tr>
<th>strategy</th>
<th>cores</th>
<th>ph10</th>
<th>ph11</th>
<th>ph12</th>
<th>ph13</th>
<th>ph14</th>
<th>ph15</th>
<th>ph16</th>
<th>ph17</th>
<th>ph18</th>
<th>ph19</th>
<th>ph20</th>
</tr>
</thead>
<tbody>
<tr>
<td>original</td>
<td>1</td>
<td>40</td>
<td>218</td>
<td>1231</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5</td>
<td>21</td>
<td>117</td>
<td>673</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>3</td>
<td>13</td>
<td>64</td>
<td>357</td>
<td>2002</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>node</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>10</td>
<td>15</td>
<td>26</td>
<td>43</td>
<td>64</td>
<td>99</td>
<td>147</td>
<td>209</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>11</td>
<td>17</td>
<td>26</td>
<td>44</td>
<td>66</td>
<td>100</td>
<td>146</td>
<td>206</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>22</td>
<td>35</td>
<td>55</td>
<td>83</td>
<td>124</td>
<td>182</td>
<td>260</td>
</tr>
<tr>
<td>clause</td>
<td>1</td>
<td>53</td>
<td>351</td>
<td>2108</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>7</td>
<td>39</td>
<td>215</td>
<td>1305</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>5</td>
<td>24</td>
<td>120</td>
<td>713</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

substantial amount of time spent in “trial elimination attempts” for “node” and “clause” size bounding
Conclusion

- started to revisit both BDD and ZDD based SAT solving
- contribution: simpler notation, parallel, bounded symbolic variable elimination
- but did not cover parallelization in BDD library Sylvan by Tom van Dijk

Things we tried without Success

- played with different variable orderings for PHP
- stronger (more expensive) simplifiers (not just subsumption)
- other similar hard combinatorial examples

Future Work

- symbolic cube-and-conquer
- low-level parallelization of CDCL SAT solvers
- unit propagation on BDDs / ZDDs