Evaluating CDCL Restart Schemes

Armin Biere     Andreas Fröhlich
Johannes Kepler University, Linz, Austria

Pragmatics of SAT 2015
POS’15

The University of Texas at Austin
Austin, TX, USA

Wednesday, 23rd September, 2015
Number of solved instances within a given amount of CPU time

http://satcompetition.org/edacc/sc14
Adding Glucose Restarts

SAT Competition 2014 Application SAT + UNSAT

CPU time (sec)

- Lingeling ayv SC2014
- SWDiA5BY A26
- Lingeling ba2 ema−14
Lingeling actually barely won
  - only for long time limit of 5000 seconds
  - for 900 seconds: no chance

two main reasons
  - selected benchmark biased towards descendents of Glucose / MiniSAT
  - but Glucose restarts are important for many (selected) benchmarks

this paper is about lessons learned while
  - porting the Glucose restart scheme to Lingeling
  - and simplifying by

using exponential moving averages (EMA)
application track instances clustered in buckets (by the organizers):

2d-strip-packing (4), argumentation (20), bio (11),
crypto-aes (8), crypto-des (7), crypto-gos (9),
crypto-md5 (21), crypto-sha (29), crypto-vpmc (4),
diagnosis (28), fpga-routing (1),
hardware-bmc (4), hardware-bmc-ibm (18), hardware-cec (30),
hardware-manolios (6), hardware-velev (27),
planning (19), scheduling (30), scheduling-pesp (3),
software-bit-verif (9), software-bmc (6), symbolic-simulation (1), termination (5)

in total 300 instances clustered in 23 buckets
Status run_CDCL_loop_with_restarts () {
    for (;;) {
        if (bcp ()) {
            if (restarting ()) restart ();
            else if (!decide ()) return SATISFIABLE;
        } else {
            conflicts++;
            if (!analyze ()) return UNSATISFIABLE;
        }
    }
}

- run BCP and conflict analysis (including learning) until completion
- restart if restart policy implemented in restarting says so
- usually based on a global conflicts counter
- otherwise pick next decision (unless all are assigned)
bool restarting () {
    return conflicts >= limit;
}

void static_uniform_restart () {
    restarts++;
    limit = conflicts + interval;
    backtrack (0);
}

void static_geometric_restart () {
    limit = conflicts + interval * pow (1.5, ++restarts);
    backtrack (0);
}

void luby_restart () {
    limit = conflicts + interval * luby (++restarts);
    backtrack (0);
}
Restart Scheme Classification

- **static schemes**
  - fixed schedule of restarts only based on conflicts counter
    - **uniform intervals**: wait a fixed number of conflicts after each restart
    - **non-uniform restart intervals**
      - number of performed restarts determines next interval (in terms of conflicts)
      - arithmetically or geometrically increasing actual interval
      - Luby scheme (also known as reluctant doubling)
      - inner-outer scheme

- **dynamic schemes**
  - agility based restart blocking
  - local restarts (not discussed in the paper nor the talk)
  - reusing the trail implicitly also blocks restarts (even partially)
  - Glucose restart scheme (focus here)
Comparing Static Uniform Restart Schemes

<table>
<thead>
<tr>
<th>r</th>
<th>002</th>
<th>004</th>
<th>008</th>
<th>016</th>
<th>032</th>
<th>064</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>tot</td>
<td>122</td>
<td>127</td>
<td>139</td>
<td>144</td>
<td>144</td>
<td>161</td>
<td>163</td>
<td><strong>168</strong></td>
<td>158</td>
</tr>
<tr>
<td>sat</td>
<td>40</td>
<td>41</td>
<td>49</td>
<td>56</td>
<td>56</td>
<td>73</td>
<td>76</td>
<td><strong>83</strong></td>
<td>79</td>
</tr>
<tr>
<td>uns</td>
<td>82</td>
<td>86</td>
<td><strong>90</strong></td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>87</td>
<td>85</td>
<td>79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r</th>
<th>002</th>
<th>004</th>
<th>008</th>
<th>016</th>
<th>032</th>
<th>064</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>2d-strip-packing</td>
<td>0/2</td>
<td>0/2</td>
<td>0/2</td>
<td>0/2</td>
<td>0/2</td>
<td>1/2</td>
<td>1/2</td>
<td><strong>1/2</strong></td>
<td>2/2</td>
</tr>
<tr>
<td>crypto-sha</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>1/0</td>
<td>7/0</td>
<td>11/0</td>
<td><strong>13/0</strong></td>
<td>10/0</td>
</tr>
<tr>
<td>hardware-cec</td>
<td>0/22</td>
<td>0/23</td>
<td><strong>0/24</strong></td>
<td>0/22</td>
<td>0/22</td>
<td>0/23</td>
<td>0/22</td>
<td>0/21</td>
<td>0/21</td>
</tr>
<tr>
<td>hardware-manolios</td>
<td>0/4</td>
<td>0/5</td>
<td>0/5</td>
<td>0/5</td>
<td><strong>0/6</strong></td>
<td>0/6</td>
<td>0/6</td>
<td>0/6</td>
<td>0/6</td>
</tr>
<tr>
<td>hardware-velev</td>
<td>5/9</td>
<td>6/10</td>
<td>8/11</td>
<td>8/12</td>
<td>8/12</td>
<td><strong>8/13</strong></td>
<td>8/12</td>
<td>8/11</td>
<td>8/6</td>
</tr>
<tr>
<td>planning</td>
<td>6/3</td>
<td>6/5</td>
<td>7/4</td>
<td>7/4</td>
<td>8/4</td>
<td>9/3</td>
<td>9/4</td>
<td><strong>11/4</strong></td>
<td>10/4</td>
</tr>
<tr>
<td>scheduling</td>
<td>1/7</td>
<td>0/7</td>
<td>1/7</td>
<td>4/7</td>
<td>6/7</td>
<td>9/7</td>
<td>9/7</td>
<td><strong>11/7</strong></td>
<td>12/7</td>
</tr>
</tbody>
</table>

SAT / UNSAT

underlined best
### Comparing Static but Non-Uniform Restart Schemes

<table>
<thead>
<tr>
<th></th>
<th>luby-(b)</th>
<th>inner-outer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01 02 04 08 16 32</td>
<td>001 002 004 008 032 128</td>
</tr>
<tr>
<td>tot</td>
<td>156 168 161 163 160 159</td>
<td>161 161 158 153 154 150</td>
</tr>
<tr>
<td>sat</td>
<td>72 80 74 77 74 74</td>
<td>80 81 77 74 76 76</td>
</tr>
<tr>
<td>uns</td>
<td>84 88 87 86 86 85</td>
<td>81 80 81 79 78 74</td>
</tr>
<tr>
<td>avgc</td>
<td>9 17 31 58 108 203</td>
<td>443 509 601 732 1084 1740</td>
</tr>
</tbody>
</table>

\( \text{avgc} = \text{average restart interval (over all instances)} \) in conflicts
bool restarting () {
    return conflicts >= limit &&
    average_RECENT_lbdss () > 1.25 * average_ALL_lbdss ();
}

void glucose_restart () { // same as static_uniform_restart
    restarts++;
    limit = conflicts + 50;
    backtrack (0);
}

- glucose level (LBD) of learned clause:
  - number of different decision levels in a learned clauses
  - calculated at the point the clause is learned during conflict analysis
- last 50 LBDs are stored and considered recent (explicit LBD queue)
- total average of all LBDs is simply \( \frac{\text{sum}_lbd}{\text{conflicts}} \)
- for discussion of blocking restarts since Glucose 2.1 see the paper
Glucose uses *simple moving average* (SMA) for the average of recent LBDs and *cumulative moving average* (CMA) for the average of all LBDs and

\[
SMA(n, w) = \frac{1}{w} \cdot (t_n + t_{n-1} + \ldots + t_{n-w+1}) \quad \text{with } n \geq w \geq 1
\]

\[
CMA(n) = SMA(n, n)
\]

\[
CMA(n) = CMA(n-1) + \frac{t_n - CMA(n-1)}{n}
\]

\[
SMA(n, w) = SMA(n-1, w) + \frac{t_n}{w} - \frac{t_{n-w}}{w}
\]

requires \(SMA(n, 50) > 1.25 \cdot CMA(n)\) to restart

and 50 conflicts have passed
Exponential Moving Average

we suggest to use EMAs instead of the “fast” SMA and/or “slow” CMA

exponential  \( EMA(n, \alpha) = \alpha \cdot t_n + (1 - \alpha) \cdot EMA(n - 1, \alpha) \) with \( 0 < \alpha < 1 \)

\[ a \approx \frac{2}{1 + w} \]

alternative  \( EMA(n, \alpha) = EMA(n - 1, \alpha) + \alpha \cdot (t_n - EMA(n - 1, \alpha)) \)

to restart version average requires  \( EMA(n, 2^{-5}) > 1.25 \cdot CMA(n) \)

to restart version ema-14 requires  \( EMA(n, 2^{-5}) > 1.25 \cdot EMA(n, 2^{-14}) \)

and again in both cases that a certain number of conflicts say 50 have passed
LBD — fast EMA of LBD with $\alpha = 2^{-5}$

restart — slow EMA of LBD with $\alpha = 2^{-14}$ (ema-14)

inprocessing — CMA of LBD (average)
Comparing EMAs with SMA and CMA

<table>
<thead>
<tr>
<th>solver</th>
<th>Glucose 4.0</th>
<th></th>
<th></th>
<th></th>
<th>Lingeling ba2</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ss</td>
<td>es</td>
<td>ee</td>
<td>avg</td>
<td>e8</td>
<td>e10</td>
<td>e12</td>
<td>e14</td>
<td>e16</td>
</tr>
<tr>
<td>tot</td>
<td>163</td>
<td>163</td>
<td>165</td>
<td>178</td>
<td>167</td>
<td>170</td>
<td>180</td>
<td>181</td>
<td>180</td>
</tr>
<tr>
<td>sat</td>
<td>72</td>
<td>73</td>
<td>76</td>
<td>83</td>
<td>80</td>
<td>78</td>
<td>86</td>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>uns</td>
<td>91</td>
<td>90</td>
<td>89</td>
<td>95</td>
<td>87</td>
<td>92</td>
<td>94</td>
<td>95</td>
<td>94</td>
</tr>
<tr>
<td>avgc</td>
<td>192</td>
<td>166</td>
<td>167</td>
<td>145</td>
<td>230</td>
<td>204</td>
<td>195</td>
<td>186</td>
<td>172</td>
</tr>
</tbody>
</table>

Glucose 4.0 column ss correspond to the original Glucose version

column es  to adding EMAs for only forcing restarts

column ee includes using EMA for blocking restarts too

column avg is Lingeling version average of Glucose version ee

columns eX correspond to Lingeling versions ema-X

using a slow EMA with $\alpha = 2^{-X}$ instead of CMA
double fast, slow;
...

bool analyze () {
    int lbd;
    ...
    slow += (lbd - slow)/(double)(1<<14);
    fast += (lbd - fast)/(double)(1<<5);
    ...
}

bool restarting () {
    return conflicts > limit && fast > 1.25 * slow;
}
fast 64-bit fixed point implementation avoiding floating point
inspired by Donald Knuth's implementation of our agility metric

```c
long fast, slow;  // assume (sizeof (long) == 8);
...  // initialization code skipped ...

bool analyze () {
    int lbd;  // assume (sizeof (int) == 4);
    ...
    fast -= fast >> 5;
    fast += lbd << (32 - 5);
    slow -= slow >> 14;
    slow += lbd << (32 - 14);
    ...
}

bool restarting () {
    return conflicts > limit && fast / 125 > slow / 100;
}
```

Evaluating CDCL Restart Schemes @ POS’15
data and source: http://fmv.jku.at/evalrestart/evalrestart.7z

optimal restart interval varies with benchmark bucket
- for miters fast restarts essential
- for crypto benchmarks longer intervals necessary
- disabling restarts completely is bad
- Glucose restarts superior to Luby style

presented an EMA variant of the Glucose restart scheme
- simpler model, simpler to implement
- similar performance (slightly faster)

future work
- how to improve blocking of restarts
- restart intervals still not optimal: really need machine learning?
- finally cross-fertilize ideas from SAT and Stock Market Analysis

originally proposed title for this paper
solved SAT competition 2014 application instances (ordered by solving time)