GLUCOSE 2.1
Aggressive – but Reactive – Clause Database Management,
Dynamic Restarts

Gilles Audemard – Laurent Simon

POS – 16 June 2012
A short history of GLUCOSE

2009 – Version 1.0

- Built on top of MINISAT 2.0
- Learnt clause measure usefulness: LBD
- Agressive cleaning strategy
- Dynamic restarts
A short history of GLUCOSE

2009 – Version 1.0
- Built on top of MINISAT 2.0
- Learnt clause measure usefulness: LBD
- Aggressive cleaning strategy
- Dynamic restarts

2011 – Version 2.0
- Built on top of MINISAT 2.2 (∼30% faster)
- Focus on cleaning strategy
  - More aggressive cleaning strategy
  - Dynamic
  - Protect promising clauses
- Reducing learnt clauses
A short history of GLUCOSE

■ 2009 – Version 1.0
  ▶ Built on top of MINISAT 2.0
  ▶ Learnt clause measure usefulness: LBD
  ▶ Aggressive cleaning strategy
  ▶ Dynamic restarts

■ 2011 – Version 2.0
  ▶ Built on top of MINISAT 2.2 (≈ 30% faster)
  ▶ Focus on cleaning strategy
    ■ More aggressive cleaning strategy
    ■ Dynamic
    ■ Protect promising clauses
  ▶ Reducing learnt clauses

■ 2012 – Version 2.1
  ▶ Focus on restarts
Litteral Block Distance
An observation

- Before CDCL solvers: solvers implement ideas (lookahead, Mom’s heuristics...) explaining performances was simple

- With CDCL: lookback solvers (VSIDS heuristics, learning,...) explaining performances is hard

We need strong empirical studies in order to understand and improve performances
For each conflict, we store the decision level where it occurs
We also compute the linear regression on these points
Gives an idea of the global behavior of the computation
Some plots ...

- For each conflict, we store the decision level where it occurs
- We also compute the linear regression on these points
- Gives an idea of the global behavior of the computation
For each conflict, we store the decision level where it occurs.

- We also compute the linear regression on these points.

- Gives an idea of the global behavior of the computation.
Remarks

- Of course, we do not expect to feet curves

- We try to make observations of the behavior of a CDCL solver
Decreasing appear in a lot of problems

<table>
<thead>
<tr>
<th>Series</th>
<th>#Benchs</th>
<th>% Decr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>een</td>
<td>8</td>
<td>62%</td>
</tr>
<tr>
<td>goldb</td>
<td>11</td>
<td>100%</td>
</tr>
<tr>
<td>grieu</td>
<td>7</td>
<td>71%</td>
</tr>
<tr>
<td>hoons</td>
<td>5</td>
<td>100%</td>
</tr>
<tr>
<td>ibm-2002</td>
<td>7</td>
<td>71%</td>
</tr>
<tr>
<td>ibm-2004</td>
<td>13</td>
<td>92%</td>
</tr>
<tr>
<td>manol-pipe</td>
<td>55</td>
<td>91%</td>
</tr>
<tr>
<td>miz</td>
<td>13</td>
<td>0%</td>
</tr>
<tr>
<td>schup</td>
<td>5</td>
<td>80%</td>
</tr>
<tr>
<td>simon</td>
<td>10</td>
<td>90%</td>
</tr>
<tr>
<td>vange</td>
<td>3</td>
<td>66%</td>
</tr>
<tr>
<td>velev</td>
<td>54</td>
<td>92%</td>
</tr>
<tr>
<td>all</td>
<td>199</td>
<td>83%</td>
</tr>
</tbody>
</table>
The goal

![Graph](image.png)

grieu-vmpc-s05-25 – SAT – 625 vars and 76,000 clauses
The goal

grieu-vmpc-s05-25 – SAT – 625 vars and 76,000 clauses
Intuitions

- A lot of dependencies between variables
  During search those variables will probably be propagated together inside blocks of propagations
- One needs to collapse independent blocks of propagated literals in order to reduce the decision level

The LBD score of a nogood is the number of different blocks of propagated literals
Intuitions

- A lot of dependencies between variables
  During search those variables will probably be propagated together inside blocks of propagations
- One needs to collapse independent blocks of propagated literals in order to reduce the decision level

The LBD score of a nogood is the number of different blocks of propagated literals

- LBD=2
  - Only one literal from the last decision level (the assertive one)
  - This literal will be glued to the other block
  - binary clauses have LBD equal to 2

- VSIDS + progress saving: this should occurs a lot!!!

Good clauses are GLUE clauses
Managing Learnt Clauses
Previous works

- Before GLUCOSE, managing learnt clauses was not considered as an important component of CDCL solvers.
- Previous measures were not so accurate.
- Clause database size followed a geometric progression.
- Dependent of the size of the input formula: No cleaning are performed for huge formulas.

Use the LBD measure
Agressive strategies

- Small LBD are good ones
- In case of equality, prefer clauses with recent activity (VSIDS like)
- No matter the size of the initial formula
- Remove half of learnt clauses every :
  - GLUCOSE 1.0 (2009): $20000 + 500 \times x$ conflicts
  - GLUCOSE 2.X (2011): $4000 + 300 \times x$ conflicts
A first step towards a dynamic management

- Performances of GLUCOSE heavily depend on the quality of LBD
- A very good indicator on many instances
- However, it may not be discriminating enough

A special case:
- Half of clauses have a LBD less than 3 (we are going to remove potentially good clauses)
- Too much good clauses

- We need to keep more of them
- We postpone the next cleaning by a constant of 1000
A first step towards a dynamic management

- Performances of GLUCOSE heavily depend on the quality of LBD
- A very good indicator on many instances
- However, it may not be discriminating enough

A special case:
- Half of clauses have a LBD less than 3 (we are going to remove potentially good clauses)
- Too much good clauses

- We need to keep more of them
- We postpone the next cleaning by a constant of 1000

When performing cleaning??
Behavior

Graph showing the relationship between the number of conflicts and the clause database size for different strategies:
- All good
- All bad
- Bad, then good
- Good, then Bad
- Bad, then change every 100,000
Protec[t promising clauses

- Reminder: LBD is computed when the clause is learnt

- We computed it again it when a clause is used during BCP

- We change it, if it becomes smaller

- Such clauses seem interesting

- They are protected for one round
Restarts
Initially, restarts were introduced to prevent trashing

Now, restarts must be seen as dynamic rearrangements of variables dependencies

Restarts are more and more frequent

GLUCOSE uses a dynamic restart strategy
Targetting UNSAT

- GLUCOSE aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We use
  - bounded queue (of size X) called $\text{queueLBD}$
  - the sum of all LBD clauses $\text{sumLBD}$
Targetting UNSAT

- GLUCOSE aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We use
  - bounded queue (of size X) called queueLBD
  - the sum of all LBD clauses sumLBD

```cpp
// In case of conflict
compute learnt clause c;
sumLBD+=c.lbd();
queueLBD.push(c.lbd());
if(queueLBD.isFull() && queueLBD.avg()*K>sumLBD/nbConflicts) {
    queueLBD.clear();
    restart();
}
```
Targetting UNSAT

- GLUCOSE aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We use
  - bounded queue (of size $X$) called $queueLBD$
  - the sum of all LBD clauses $sumLBD$

```c
// In case of conflict
compute learnt clause $c$;
$sumLBD$+=c.lbd();
$queueLBD$.push(c.lbd());
if($queueLBD$.isFull() && $queueLBD$.avg()*$K>$sumLBD/$nbConflicts$) {
    $queueLBD$.clear();
    restart();
}
```

- Perform at least $X$ conflicts before restarting
Targetting UNSAT

- GLUCOSE aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We use
  - bounded queue (of size X) called queueLBD
  - the sum of all LBD clauses sumLBD

```c
// In case of conflict
compute learnt clause c;
sumLBD+=c.lbd();
queueLBD.push(c.lbd());
if(queueLBD.isFull() && queueLBD.avg()*K>sumLBD/nbConflicts) {
    queueLBD.clear();
    restart();
}
```

- Perform at least X conflicts before restarting
- Average over last X LBD become too big wrt total average
Targetting UNSAT

- GLUCOSE aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We use
  - bounded queue (of size X) called queueLBD
  - the sum of all LBD clauses sumLBD

```c
// In case of conflict
compute learnt clause c;
sumLBD+=c.lbd();
queueLBD.push(c.lbd());
if(queueLBD.isFull() && queueLBD.avg()*K>sumLBD/nbConflicts) {
    queueLBD.clear();
    restart();
}
```

- GLUCOSE 1.0 and 2.0: \( X=100 \) and \( K=0.7 \)
- GLUEMINISAT and GLUCOSE 2.1: \( X=50 \) and \( K=0.8 \)
Impact of different $K$ and $X$
Impact of different K and X

SAT 2011 Application benchmarks (limit 900 seconds)
Targeting SAT too (NEW in GLUCOSE 2.1)

- Frequent restarts seems not very good in case of SAT instances
- Some lessons of SAT 2011 competition – Second Phase, SAT instances
  - CONTRASAT: 1\textsuperscript{st} with 99 instances
  - GLUCOSE: 10\textsuperscript{th} with 94 instances
  - 6 of first ten solvers come from minisat hack (luby restarts)
  - 18 instances separate 1\textsuperscript{st} and 10\textsuperscript{th} in UNSAT
Targeting SAT too (NEW in GLUCOSE 2.1)

- Frequent restarts seems not very good in case of SAT instances
- Some lessons of SAT 2011 competition – Second Phase, SAT instances
  - CONTRASAT: 1\textsuperscript{st} with 99 instances
  - GLUCOSE: 10\textsuperscript{th} with 94 instances
  - 6 of first ten solvers come from minisat hack (luby restarts)
  - 18 instances separate 1\textsuperscript{st} and 10\textsuperscript{th} in UNSAT

- Agressive clauses deletion: some clauses may be bad for UNSAT but good for SAT
- Agressive restarts: some global assignments can be dropped!!

Delay restarts if total of assignments suddenly increase
Now, suppose dots represent trail stack assignment size

grieu-vmpc-s05-25 – SAT – 625 vars and 76,000 clauses
Now, suppose dots represent trail stack assignment size.

GLUCOSE is unlucky, a restart is performed!
Targeting SAT too (NEW in GLUCOSE 2.1)

- We use
  - trail the assignment stack
  - Bounded queue of the last Y trail size when reaching a conflict (queueTrail)
Targeting SAT too (NEW in GLUCOSE 2.1)

We use
- trail the assignment stack
- Bounded queue of the last Y trail size when reaching a conflict (queueTrail)

```java
// In case of conflict
queueTrail.push(trail.size());
if (queueLBD.isFull() && queueTrail.isFull() && trail.size() > T * queueTrail.avg()) {
    queueLBD.clear();
}
```

compute learnt clause c
...
Targeting SAT too (NEW in GLUCOSE 2.1)

- We use
  - trail the assignment stack
  - Bounded queue of the last Y trail size when reaching a conflict (queueTrail)

```java
// In case of conflict
queueTrail.push(trail.size());
if (queueLBD.isFull() && queueTrail.isFull() && trail.size() > T * queueTrail.avg()) {
  queueLBD.clear();
}
```

compute learnt clause c
...

- The total number of assignments suddenly increase
Targeting SAT too (NEW in GLUCOSE 2.1)

- We use
  - trail the assignment stack
  - Bounded queue of the last Y trail size when reaching a conflict (queueTrail)

```java
// In case of conflict
queueTrail.push(trail.size());
if (queueLBD.isFull() && queueTrail.isFull() && trail.size() > T * queueTrail.avg()) {
    queueLBD.clear();
}
```

compute learnt clause c

- The total number of assignments suddenly increase
- Postpone restart
Targeting SAT too (NEW in GLUCOSE 2.1)

- We use
  - trail the assignment stack
  - Bounded queue of the last Y trail size when reaching a conflict (queueTrail)

```java
// In case of conflict
queueTrail.push(trail.size());
if(queueLBD.isFull() && queueTrail.isFull() && 
   trail.size()>T*queueTrail.avg()) {
    queueLBD.clear();
}
```

compute learnt clause c...

- The total number of assignments suddenly increase
- Postpone restart
- Y=5000 and T=1.4 appears to be good
Conclusion
Evolution of GLUCOSE

GLUCOSE 2.1 vs GLUCOSE 2.0

SAT 2011 application benchmarks (limit 900 seconds)
Evolution of GLUCOSE

GLUCOSE 2.1 vs GLUCOSE 1.0

SAT 2011 application benchmarks (limit 900 seconds)
Evolution of GLUCOSE

SAT 2009 + SAT 2011 application benchmarks (536, non redundant)
## Evolution of GLUCOSE

<table>
<thead>
<tr>
<th>Version</th>
<th>SAT</th>
<th>UNSAT</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>113</td>
<td>157</td>
<td>270</td>
</tr>
<tr>
<td>2.0</td>
<td>136</td>
<td>192</td>
<td>328</td>
</tr>
<tr>
<td>2.1</td>
<td>148</td>
<td>204</td>
<td>352</td>
</tr>
</tbody>
</table>

SAT 2009 + SAT 2011 application benchmarks (536, non redundant)
The future of GLUCOSE ... It is a secret :-)  
Current work with Daniel and Laurent
Conclusion

- The future of GLUCOSE . . . It is a secret :-)  
  Current work with Daniel and Laurent

- A possible controversy
  - Are CDCL solvers still complete?
    - Very frequent restarts
    - Many deleted clauses (more than 93% for GLUCOSE (total for SAT 2011 Application benchmarks))
  - Are CDCL solvers closer to DPLL62 or local search??