Enhanced Gaussian Elimination in DPLL-based SAT Solvers

Mate Soos
INRIA SALSA Team
10th of July 2010
1. Context

2. Gaussian elimination in SAT Solvers

3. Results

4. Conclusions
1. **Context**
   - Cryptographic problems
   - Gaussian elimination

2. **Gaussian elimination in SAT Solvers**
   - Datastructures, algorithms
   - Row and Column Elimination by XOR
   - Independent sub-matrixes
   - Skipping parts of matrix to treat

3. **Results**

4. **Conclusions**
DPLL-based SAT solvers

Solves a problem in CNF

CNF is an “and of or-s”

\[ \neg x_1 \lor \neg x_3 \quad \neg x_2 \lor x_3 \quad x_1 \lor x_2 \]

Uses DPLL(\(\varphi\)) algorithm

1. If formula \(\varphi\) is trivial, return SAT/UNSAT
2. Picks a variable \(v\) to branch on
3. \(v := \text{true}\)
4. Simplifies formula to \(\varphi'\) and calls DPLL(\(\varphi'\))
5. if SAT, output SAT
6. if UNSAT, \(v := \text{false}\)
7. Simplifies formula to \(\varphi''\) and calls DPLL(\(\varphi''\))
8. if SAT, output SAT
9. if UNSAT, output UNSAT
Cryptographic problems

Crypto problems are given in ANF

\[ 0 = ab \oplus b \oplus bc \]
\[ 0 = a \oplus d \oplus c \oplus bd \]
\[ 0 = bc \oplus cd \oplus bd \]
\[ 0 = d \oplus ab \oplus 1 \]

Methods to solve ANF

1. Put into matrix, Gauss eliminate:

\[
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

2. Convert to CNF. Notice: it’s same as above, but \( ab = a \times b \) is included, and less info (rows) needed.

3. Other methods (e.g. F4/F5)
Gaussian elimination

Theory

- Solving a Gaussian elimination problem with DPLL-based SAT solvers is exponentially difficult
- Even though Gaussian elimination is poly-time
  - Theoretically, Gaussian elimination in SAT solvers is useful

Practise

- Designers of SAT solvers have grown accustomed to solving worst-case exponential problems really fast
- But Gaussian elimination is different:

| Matrix size: $n \times n$, MiniSat time (s) |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 20              | 22              | 24              | 26              | 28              | 30              | 32              | 34              | 36              | 38              |
| 0.02            | 0.09            | 0.22            | 0.8             | 1.84            | 8.2             | 30.9            | 90.0            | 331.3           | 1539.9           |

- Practical usefulness is still elusive
**Gauss and Crypto**

**The two approaches**

- Only-Gauss approach problem: too many rows needed, too large matrix
- Only-SAT approach problem: Can’t “see” the matrix, can’t find truths from it

**A hybrid approach**

Executing Gauss. elim. at every decision step in the SAT solver, we can mix the two approaches.
1 Context
- Cryptographic problems
- Gaussian elimination

2 Gaussian elimination in SAT Solvers
- Datastructures, algorithms
- Row and Column Elimination by XOR
- Independent sub-matrixes
- Skipping parts of matrix to treat

3 Results

4 Conclusions
### Implementation

<table>
<thead>
<tr>
<th></th>
<th>v10</th>
<th>v8</th>
<th>v9</th>
<th>v12</th>
<th>aug</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-matrix</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>v10</th>
<th>v8</th>
<th>v9</th>
<th>v12</th>
<th>aug</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-matrix</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### Implementation

<table>
<thead>
<tr>
<th>A-matrix with $v_8$ assigned to true</th>
<th>N-matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_{10}$ $v_8$ $v_9$ $v_{12}$ $\text{aug}$</td>
<td>$v_{10}$ $v_8$ $v_9$ $v_{12}$ $\text{aug}$</td>
</tr>
</tbody>
</table>
| \[
\begin{bmatrix}
1 & - & 1 & 1 & 1 \\
0 & - & 1 & 1 & 1 \\
0 & - & 0 & 1 & 0 \\
0 & - & 0 & 0 & 0 \\
\end{bmatrix}
\] | \[
\begin{bmatrix}
1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 \\
\end{bmatrix}
\] |
### Implementation

<table>
<thead>
<tr>
<th>A-matrix with ( v_8 ) assigned to true</th>
<th>N-matrix</th>
</tr>
</thead>
</table>
| \[
\begin{bmatrix}
1 & -1 & 1 & 1 & 1 \\
0 & -1 & 1 & 1 & 1 \\
0 & -0 & 1 & 0 & 0 \\
0 & -0 & 0 & 0 & 0 \\
\end{bmatrix}
\] | \[
\begin{bmatrix}
1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 \\
\end{bmatrix}
\] |

Resulting xor-clause:

\[ v_8 \oplus v_{12} \]
### Implementation

#### A-matrix

with \( v_8 \) assigned to true

\[
\begin{bmatrix}
v_{10} & v_8 & v_9 & v_{12} & \text{aug}
\end{bmatrix}
\begin{bmatrix}
1 & -1 & 1 & 1 & 1 \\
0 & - & 1 & 1 & 1 \\
0 & - & 0 & 1 & 0 \\
0 & - & 0 & 0 & 0 \\
\end{bmatrix}
\]

#### N-matrix

\[
\begin{bmatrix}
v_{10} & v_8 & v_9 & v_{12} & \text{aug}
\end{bmatrix}
\begin{bmatrix}
1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Resulting xor-clause:

\[ v_{12} = \text{false} \iff v_8 \oplus v_{12} \]
Example

- If variable $a$ is not present anywhere but in 2 XOR-s:
  
  
  \[ a \oplus b \oplus c \oplus d = \text{false} \]
  
  \[ a \oplus f \oplus g \oplus h = \text{false} \]

- Then we can remove $a$, the two XOR-s, and add the XOR:
  
  \[ f \oplus g \oplus h \oplus b \oplus c \oplus d = \text{false} \]

Theory

- This is variable elimination at the XOR-level
- It is equivalent to VE at CNF level
- But it doesn’t make sense to do this at CNF level:
  \[ \rightarrow \text{results in far more (and larger) clauses} \]
- For us it helps: removes 1 column ($a$) and one row from the matrix
Independent sub-matrixes

Reasoning

- Gaussian elimination is approx. $O(nm^2)$ algorithm
- Making two smaller matrixes from one bigger one leads to speedup
- If matrix has non-connected components, cutting up is orthogonal to algorithm output
Independent sub-matrixes

Algorithm
Let us build a graph from the XOR-s:
- Vertexes are the variables
- Edge runs between two vertexes if they appear in an XOR
- Independent graph components are extracted

Advantages
- In case of 2 roughly equal independent sub-matrixes:
  \[ cnm^2 \rightarrow 2c'(n/2)(m/2)^2 = c'n m^2 / 4 \]
- Better understanding of problem structure:
  - E.g. number of shift registers in a cipher
  - Number of S-boxes in cipher
  - Problem similarities
Not treating parts of the matrix

Reasoning
- Let’s assume the leftmost column updated is the \( c^{th} \)
- Let’s assume the topmost “1” in this column was in row \( r \)
- Then, the rows above \( r \) cannot have changed their leading 1

Example

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0
\end{array}
\]

A

B

C

Mate Soos (INRIA SALSA Team)
Auto turn-off

- If Gauss doesn’t bring enough benefits, it is switched off.
- Performance is measured by percentage of times confl/prop is generated.
- Conflict is preferred — we can return immediately.
More efficient data structure

**Data structure**

- Bits are packed — faster row xor/swap
- Augmented column is non-packed — faster checking
- Two matrixes are stored as an interlaced continuous array
- $A[0][0] \ldots A[0][n], N[0][0] \ldots N[0][n], \ldots A[m][0] \ldots N[m][n]$

![Diagram of matrix interlacing]

**Advantages**

- When doing row-xor both matrixes’ rows are xor-ed
- When doing row-swap both matrixes’ rows are swapped
- We can now operate on *one* continuous data in both operations
Outline

1. Context
   - Cryptographic problems
   - Gaussian elimination

2. Gaussian elimination in SAT Solvers
   - Datastructures, algorithms
   - Row and Column Elimination by XOR
   - Independent sub-matrixes
   - Skipping parts of matrix to treat

3. Results

4. Conclusions
Results overview

Before: “Extending SAT Solvers to Cryptographic Problems”
- Worked only on few instances
- Had to be tuned for each instance
- Gave approx. 5-10% speedup

Now: “Enhanced Gaussian Elimination in DPLL-based SAT Solvers”
- Matrix discovery is automatic
- Less tuning necessary – turn-off is automatic
- Works on more types of instances
- Gives up to 30%-45% speedup
### Table: Avg. time (in sec.) to solve 100 random problems

<table>
<thead>
<tr>
<th>no. help bits</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>no RCX + no Gauss</td>
<td>0.69</td>
<td>1.26</td>
<td>1.38</td>
<td>2.19</td>
<td>6.25</td>
<td>10.40</td>
</tr>
<tr>
<td>RCX + no Gauss</td>
<td>0.65</td>
<td>0.89</td>
<td>1.30</td>
<td>2.36</td>
<td>5.76</td>
<td>8.87</td>
</tr>
<tr>
<td>no RCX + Gauss</td>
<td>0.55</td>
<td>0.91</td>
<td>1.06</td>
<td>1.89</td>
<td>3.87</td>
<td>7.76</td>
</tr>
<tr>
<td>RCX + Gauss</td>
<td>0.52</td>
<td>0.69</td>
<td>0.90</td>
<td>1.85</td>
<td>3.81</td>
<td>6.20</td>
</tr>
<tr>
<td>Vars removed on avg</td>
<td>36.27</td>
<td>36.42</td>
<td>37.30</td>
<td>37.07</td>
<td>38.32</td>
<td>37.94</td>
</tr>
</tbody>
</table>
# Results — Gauss

<table>
<thead>
<tr>
<th></th>
<th>Bivium</th>
<th>Trivium</th>
</tr>
</thead>
<tbody>
<tr>
<td>no. help bits</td>
<td>100 random problems</td>
<td></td>
</tr>
<tr>
<td>RCX</td>
<td>0.89 1.30 2.36 5.76 8.87</td>
<td>66.57 86.42 146.17 261.75 472.27</td>
</tr>
<tr>
<td>Gauss+RCX</td>
<td>0.69 0.90 1.85 3.81 6.20</td>
<td>40.57 68.16 84.13 146.35 259.07</td>
</tr>
</tbody>
</table>

Table: Avg. time (in sec.) to solve 100 random problems
### Table: Avg. time (in sec.) to solve 100 random problems

<table>
<thead>
<tr>
<th></th>
<th>HiTag2</th>
<th>Grain</th>
</tr>
</thead>
<tbody>
<tr>
<td>no. help bits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>4.78</td>
<td>168.51</td>
</tr>
<tr>
<td>14</td>
<td>11.73</td>
<td>291.29</td>
</tr>
<tr>
<td>13</td>
<td>30.70</td>
<td>540.14</td>
</tr>
<tr>
<td>12</td>
<td>76.44</td>
<td>1123.08</td>
</tr>
<tr>
<td>11</td>
<td>233.61</td>
<td>540.14</td>
</tr>
<tr>
<td>10</td>
<td>719.86</td>
<td>608.47</td>
</tr>
<tr>
<td>9</td>
<td>1666.99</td>
<td>1133.75</td>
</tr>
<tr>
<td>RCX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gauss+RCX</td>
<td>4.76</td>
<td>193.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>359.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td>608.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1133.75</td>
</tr>
</tbody>
</table>
Outline

1. Context
   - Cryptographic problems
   - Gaussian elimination

2. Gaussian elimination in SAT Solvers
   - Datastructures, algorithms
   - Row and Column Elimination by XOR
   - Independent sub-matrixes
   - Skipping parts of matrix to treat

3. Results

4. Conclusions
Conclusions

- Gaussian elimination can bring benefits for specific applications
- Better understanding of the problem could be gained

Possible future work

- Automatic cut-off value finding
- Better heuristics to decide when to execute Gaussian elim.
- Add support for sparse matrix representation
Thank you for your time