Enhanced Gaussian Elimination in DPLL-based SAT Solvers

Mate Soos

UPMC LIP6, PLANETE team INRIA, SALSA team INRIA

10th of July 2010

Table of Contents

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

DPLL-based SAT solvers

Solves a problem in CNF

CNF is an "`and` of `or`-s"

$$\neg x_1 \vee \neg x_3 \qquad \neg x_2 \vee x_3 \qquad x_1 \vee x_2$$

Uses DPLL(φ) algorithm

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

$$\begin{array}{c} \begin{array}{ccccccccc} ab & bc & cd & bd & a & b & c & d & \text{aug} \end{array} \\ \left[\begin{array}{cccccccc|c} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] \end{array}$$

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 elim. problem with DPLL-based SAT solvers is exponentially difficult
- Even though Gaussian elimination is poly-time

$\longrightarrow$ Theoretically, Gauss. elim in SAT solvers is useful

Practise

- Designers of SAT solvers have grown accustomed to solving worst-case exponential problems *really* fast
- But Gauss 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

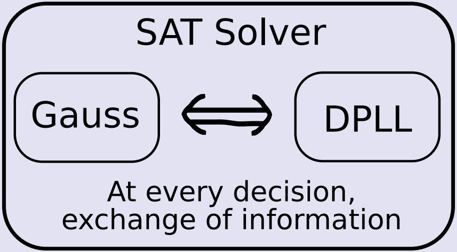
Gauss and Crypto

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



Outline

Datastructures, algorithms

Implementation

A-matrix

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \\ \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

N-matrix

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \\ \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Datastructures, algorithms

Implementation

A-matrix with $v8$ assigned to true	N-matrix
$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$ $$\left[\begin{array}{cccc\|c} 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}\right]$$	$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$ $$\left[\begin{array}{cccc\|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Datastructures, algorithms

Implementation

A-matrix
with $v8$ assigned to `true`

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}\right]$$

N-matrix

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Resulting xor-clause:

$$v8 \oplus v12$$

Datastructures, algorithms

Implementation

A-matrix with $v8$ assigned to `true`	N-matrix
$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$ $\left[\begin{array}{cccc\|c} 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}\right]$	$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$ $\left[\begin{array}{cccc\|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$

Resulting xor-clause:

$$v12 = \texttt{false} \quad \leftarrow \quad v8 \oplus v12$$

Row and Column Elimination by XOR — RCX

Example

- If variable a is not present anywhere but in 2 XOR-s:

$$a \oplus b \oplus c \oplus d = \texttt{false}$$
$$a \oplus f \oplus g \oplus h = \texttt{false}$$

- Then we can remove a, the two XOR-s, and add the XOR:

$$f \oplus g \oplus h \oplus b \oplus c \oplus d = \texttt{false}$$

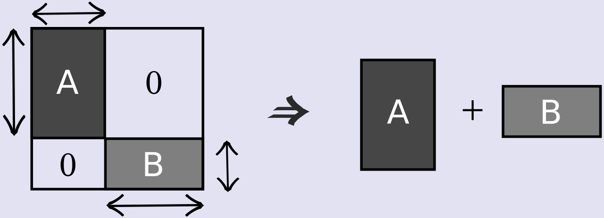
Theory

- 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:
  $\rightarrow$ 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

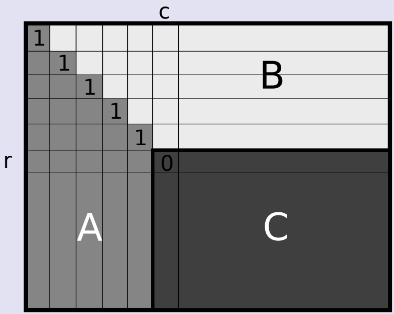
- In case of 2 roughly equal independent sub-matrixes: $cnm^2 \rightarrow 2c'(n/2)(m/2)^2 = c'nm^2/4$
- Better understanding of problem structure:
  - E.g. number of shift registers in a cipher
  - Number of S-boxes in cipher
  - Problem similarities

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



Auto turn-off heuristics

Reasoning

- Gauss. doesn't work well for all restarts
- If it doesn't bring enough benefits, switch it off
- Performance is measured by percentage of times confl/prop is generated

Qantitatively

- If $2\textsc{numGaussConfl} + \textsc{numGaussProp} < 0.05\textsc{numGaussCalled}$

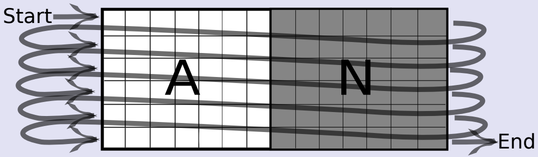
$\rightarrow$ Then turn it off

- 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
- $A[0][0] \ldots A[0][n], N[0][0] \ldots N[0][n], \ldots A[m][0] \ldots N[m][n]$



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

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

	Bivium					
no. help bits	55	54	53	52	51	50
no RCX + no Gauss	0.69	1.26	1.38	2.19	6.25	10.40
RCX + no Gauss	0.65	0.89	1.30	2.36	5.76	8.87
no RCX + Gauss	0.55	0.91	1.06	1.89	3.87	7.76
RCX + Gauss	0.52	0.69	0.90	1.85	3.81	6.20
Vars removed on avg	36.27	36.42	37.30	37.07	38.32	37.94

Results — Gauss

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

			Bivium		
no. help bits	54	53	52	51	50
RCX	0.89	1.30	2.36	5.76	8.87
Gauss+RCX	0.69	0.90	1.85	3.81	6.20
			Trivium		
no. help bits	157	156	155	154	153
RCX	66.57	86.42	146.17	261.75	472.27
Gauss+RCX	40.57	68.16	84.13	146.35	259.07

Results — Gauss cont.

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

HiTag2							
no. help bits	15	14	13	12	11	10	9
RCX	4.78	11.73	30.70	76.44	233.61	719.86	1666.99
Gauss+RCX	4.76	11.64	29.03	77.19	220.64	701.46	1636.77

Grain				
no. help bits	109	108	107	106
RCX	168.51	291.29	540.14	1123.08
Gauss+RCX	193.09	359.58	608.47	1133.75

Outline

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