Solving CNF-XOR Formulas: A Practical Perspective

Mate Soos

Simons Institute SAT Program Seminar

22nd of February, 2021

About Me

- PhD at INRIA Grenoble 2009
- Maintainer of CryptoMiniSat, ApproxMC, UniGen, STP
- Working as a Senior Research Fellow at National University of Singapore (3mo a year)
- Working as a Senior IT Security Architect at Zalando (9mo a year)
- Interests: SAT solving, Counting, Sampling, Machine Learning, Higher level abstractions

CNF-XOR

CNF with XORs occur in a number of problem domains, for example:

- Cryptography. ex find the key given partially observed circuit data [MohamedBulyginZHW2012]
- Adder and multiplier circuits: ex. see Daniela Kaufmann's presentation "Combining SAT and Computer Algebra for Circuit Verification" (February 16th) [KaufmannBiereKauers2019]
- Polynomial systems over GF(2): ex.

$$\begin{array}{llllllll} ab & \oplus & b & \oplus & efd & \oplus & e & =0 \\ a & \oplus & e & \oplus & fd & & & =0 \\ abcd & \oplus & e & \oplus & f & & & =0 \end{array}$$

  See e.g. Bosphorus [ChooSoosChaiMeel19]

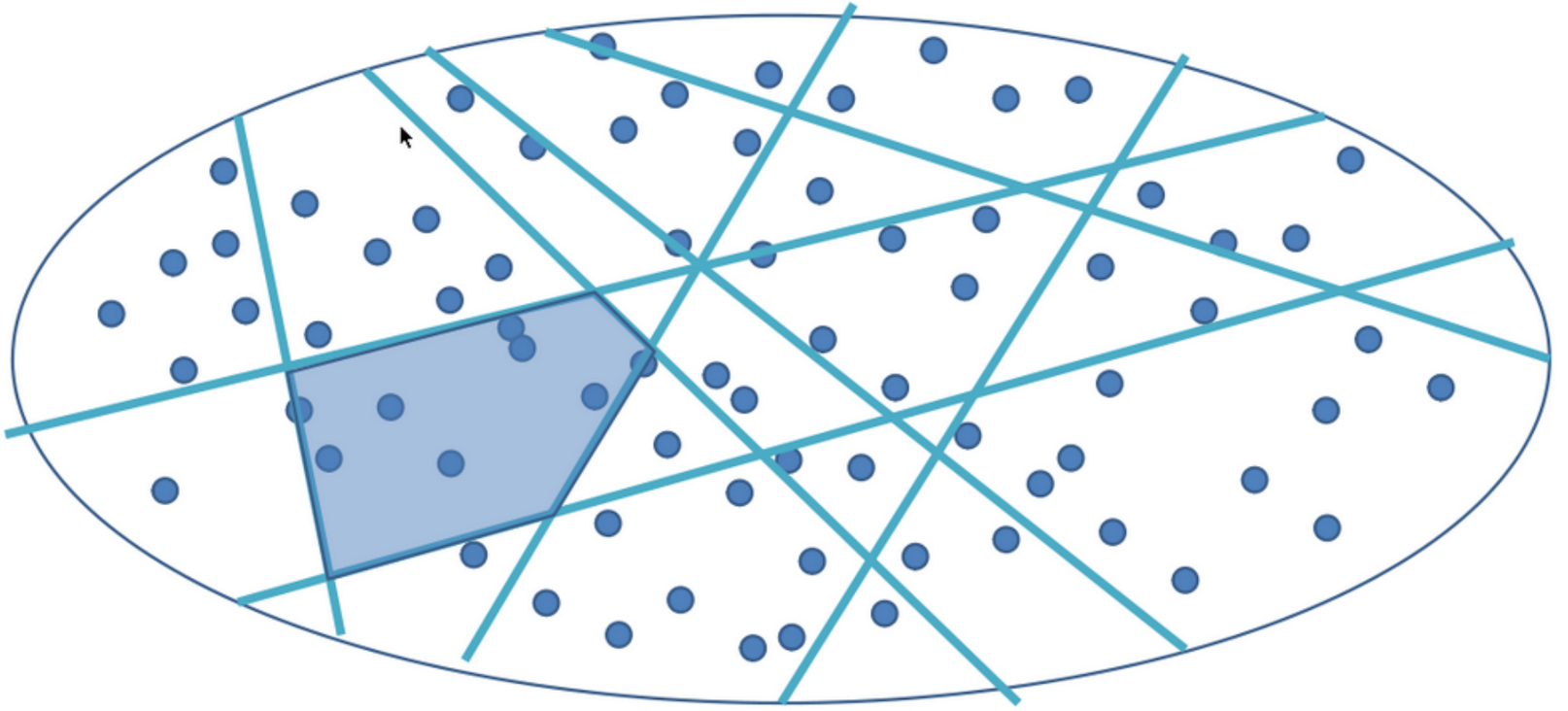
- Hashed-based counting and sampling [ChakrabortyMeelVardi2003]

In this talk, we'll use approximate counting as our reference problem

Motivating Problem: Approximate Counting

- ApproxMC by Chakraborty, Meel, and Vardi (2003): A Scalable Approximate Model Counter
- Random XORs over the projection set: a hashing-based strategy to probably approximately count solutions
- 99% of its time is spent in the SAT solver solving CNF-XOR formulas

(1) Pick a random cell, (2) count solutions, (3) multiply by no. cells



XORs in CNF

Translating $a \oplus b \oplus c = 1$ into CNF using Tseitin transformation [Tseitin'66]:

$$
\begin{array}{rcrcr}
a & \vee & b & \vee & c \\
a & \vee & \neg b & \vee & \neg c \\
\neg a & \vee & \neg b & \vee & c \\
\neg a & \vee & b & \vee & \neg c
\end{array}
$$

An XOR for size N needs 2^{n-1} clauses to translate without helper variables.

To translate an XOR $a \oplus b \oplus c \oplus d \oplus e \oplus f = 0$, we cut it into two XORs, using a helper variable x:

$$
\begin{array}{ccccccc}
a & \oplus & b & \oplus & c & \oplus & x \\
x & \oplus & d & \oplus & e & \oplus & f
\end{array}
$$

Using as many helper variables as needed cuts down the blowup to linear size. It increases the number of variables, but that's not really an issue. Modern SAT solvers can deal with millions of variables. However....

XOR manipulation once in CNF format

- XORs can be recovered syntactically and semantically
- Syntactic recovery seems easy but:

$$\begin{array}{rcrcr} a & \vee & b & & \\ a & \vee & \neg b & \vee & \neg c \\ \neg a & \vee & \neg b & \vee & c \\ \neg a & \vee & b & \vee & \neg c \end{array}$$

$\rightarrow$ still implies the XOR, so complication arise

- Once recovered, XORs can be re-assembled:

$$\begin{array}{ccccccc} a & \oplus & b & \oplus & c & \oplus & x \\ x & \oplus & d & \oplus & e & \oplus & f \end{array}$$

becomes

$$a \oplus b \oplus c \oplus d \oplus e \oplus f = 0$$

- XORs in different matrices can be extracted by grouping XORs together that coincide on at least one variable.

  $\rightarrow$ Since naive GJE is O(n^3), if we have two matrices of n/2 size, we'll be ≈ 4x faster working on them separately

- Running GJE on the reassembled XORs is easy, but does not suffice – it will only perform GJE at "toplevel", i.e. without any assignments by the solver.

Resolution on XOR formulas

- Standard (non-extended) resolution is bad at proving UNSAT of XORs blasted into CNF
- Actually, the minimum size resolution proof can be made to be exponential in the number of variables
- But naive Gauss-Jordan Elimination (GJE) is only O(n^3)
- We need to combine GJE with CDCL or we won't have a hope of solving some of these formulas

N.B.: We don't always need GJE just because there are XORs. E.g. if there is an UNSAT core that doesn't contain a single clause from the XORs, it can be OK (though there may be a smaller UNSAT proof with the XORs)

Gaussian Elimination

Gaussian elimination [UnknownChineseAuthor(s)179, Newton1670] is an algorithm in linear algebra for solving a system of linear equations. Let's restrict ourselves to linear equations over GF(2), i.e. to:

$$\begin{array}{lllll} x_1 & & \oplus & x_3 & = 0 \\ x_1 & \oplus\ x_2 & \oplus & x_3 & = 1 \\ x_1 & \oplus\ x_2 & & & = 0 \end{array}$$

$\rightarrow$ In matrix notation:

$$\left[\begin{array}{ccc|c} x_1 & x_2 & x_3 & \text{RHS} \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{array}\right]$$

In Gaussian Elimination, we aim to modify the matrix until the lower left-hand corner of the matrix is filled with zeros, also called the **upper triangular form**:

$$\left[\begin{array}{ccc|c} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{array}\right]$$

$\rightarrow$ XOR row of leftmost "1" into all rows where '1" is present in that column

$$\left[\begin{array}{ccc|c} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right]$$

$\rightarrow$ XOR row of 2nd leftmost "1" into all rows where '1" is present in that column

$$\left[\begin{array}{ccc|c} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right]$$

Gaussian Elimination cont.

$$\left[\begin{array}{ccc|c} \color{red}{1} & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{array}\right]$$

$\rightarrow$ XOR row of leftmost "1" into all rows where "1" is present in that column

$$\left[\begin{array}{ccc|c} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 1 & 0 & 1 & 0 \\ 0 & \color{red}{1} & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right]$$

$\rightarrow$ XOR row of 2nd leftmost "1" into all rows where "1" is present in that column

$$\left[\begin{array}{ccc|c} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right]$$

What if the matrix doesn't look so pretty?

- We can **swap rows** to make sure the 1st row has a "1" in the leftmost column
- If a row contains $[000\ldots0|1]$ then the set of linear equations cannot be satisfied, i.e. **UNSAT**
- If the matrix is **underdetermined**, if there is a solution, it's never a single solution. E.g. 20 variables but only 5 equations.

Gauss-Jordan Elimination

Gauss-Jordan Elimination lets Gaussian Elimination finish, then goes from right to left, bottom to top:

$$\left[\begin{array}{ccc|c} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right]$$

$\rightarrow$ XOR the row of rightmost "1" into all rows above where a "1" is present

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right]$$

This is known as the **row echelon form**. We can now trivially read out the solution:

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 1 \\ x_3 &= 1 \end{aligned}$$

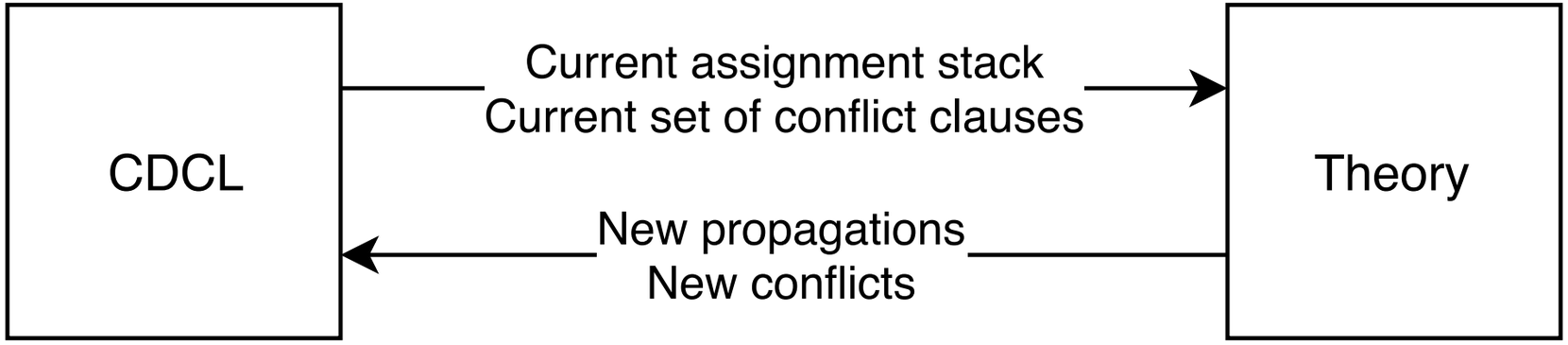
This indeed satisfies:

$$\begin{aligned} x_1 \qquad\quad \oplus\ x_3 &= 0 \\ x_1 \oplus x_2 \oplus x_3 &= 1 \\ x_1 \oplus x_2 &= 0 \end{aligned}$$

CDCL(T)

For theories that are not efficiently simulated by CDCL

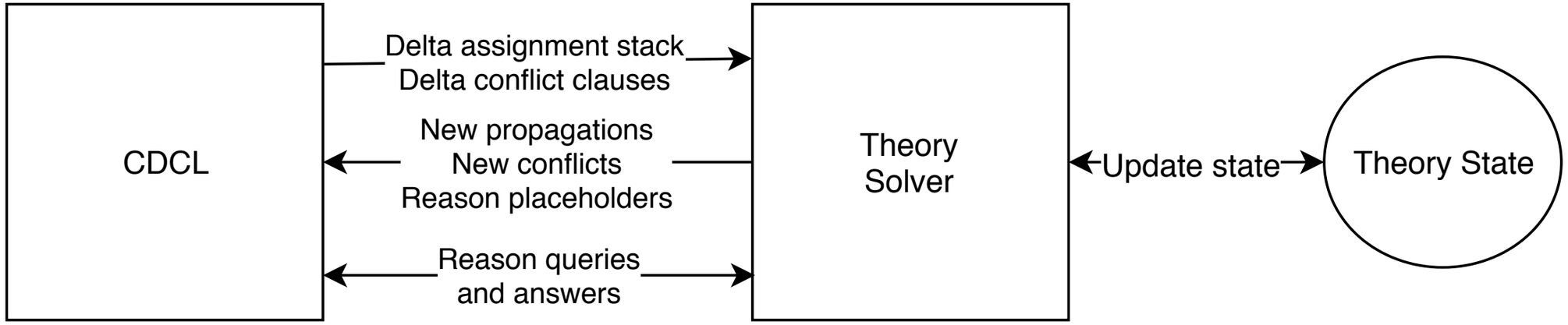
- T is the theory, e.g.:
  - Gauss-Jordan Elimination [SoosNohlCastelluccia'2010]
  - Pseudo-Boolean Reasoning [ChaiKuehlmann'2006]
  - Symmetric Explanation Learning [DevriendtBogaertsBruynooghe'2017]
- Theory is run side-by-side to the CDCL algorithm
- **Propagate** values implied by Theory given current assignment stack of CDCL
- **Conflict** if Theory implies 1=0 given current assignment stack of CDCL
- Theory must give reason for propagations&conflicts



CDCL(T) Cont.

Optimizations:

- Should only send delta of assignment stack + conflict clauses
  - Variables assigned (decisions + propagations)
  - Variables unassigned (backtracking, restarting)
  - New conflict clauses
- Theory only needs to compute delta relative to old state
- Theory can give placeholders for reasons
  - If reason is needed during conflict generation, Theory is queried
  - Called “lazy” (vs “greedy”) interpolant generation



CDCL(T) Gauss-Jordan Elimination: Ingredients

What components do we need?

- **Extractor for XOR constraints**: XORs may be encoded as CNF
- **Disjoint matrix detection**: disjoint matrices should be handled separately
- **Delta update mechanism** for row-echelon form matrix:
  - how to handle when variable is set
  - how to handle when variable is unset
- **Efficient data structures** to allow for quick updates
- **Reason generation**

CDCL(T) Gauss-Jordan Elimination: None of that row swapping please!

Observations:

- We are using binary matrices (1/0), so bit-packed format is best
- Packed format: row-swapping becomes expensive – it's a copy
- Row-echelon form is nice for the eyes [HanJiang2012]:
  - But we only need a row to be responsible for a column's "1"
  - What we loose: have to check all rows, not only ones below
- So, any row can be responsible for being a column's "1"

$$\left[\begin{array}{cccccccc|c} x_1 & x_2 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & \text{RHS} \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}\right]$$

The top line reads: $x_3 \oplus x_4 \oplus x_5 \oplus x_7 = 1$

CDCL(T) Gauss-Jordan Elimination: 2-variable watchlist scheme

Let's use a 2-variable watch scheme [HanJiang2012]:

- If 2 or more variables are unset in XOR constraint, it cannot propagate or conflict
- If 1 variable is unset, it must propagate
- If 0 variable is unset, it is either satisfied or is in conflict

We'll use the Simplex Method's terminology [Danzig'82]:

- Let's call the column that the row is responsible for "basic"
- Let's call the column that the row is NOT responsible for "nonbasic"

What data structures do we need for this? Let's see:

- Watchlist for variables (not literals!)
- column-has-responsible-row[column] = 1/0
- row-to-nonbasic-column[row] = column

CDCL(T) Gauss-Jordan Elimination: Propagation

A rough outline:

- Observe that the matrix is usually underdetermined: more columns than rows
- Many unset columns will have no responsible rows
- If we set a variable, its column doesn't need a responsible row
- The more variables we decide on, the more the matrix will be determined

$$\left[\begin{array}{cccccccc|c} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}\right]$$

Let's set the first column to "1" $\rightarrow$

$$\left[\begin{array}{cccccccc|c} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

we get a propagation! $\rightarrow$

$$\left[\begin{array}{cccccccc|c} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Notice: we were were watching both of this row's variables where it has a "1". It's a 2-variable watch scheme!

CDCL(T) Gauss-Jordan Elimination: Propagation

We got a propagation from last slide:

$$\left[\begin{array}{cccccccc|c} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Variable is now set by Gauss-Jordan $\rightarrow$

$$\left[\begin{array}{cccccccc|c} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Variable is decided on $\rightarrow$

$$\left[\begin{array}{cccccccc|c} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Need new basic variable $\rightarrow$

$$\left[\begin{array}{cccccccc|c} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Must adjust matrix $\rightarrow$

$$\left[\begin{array}{cccccccc|c} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

New propagation $\rightarrow$

$$\left[\begin{array}{cccccccc|c} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

And the story goes on...

CDCL(T) Gauss-Jordan Elimination: Reason Clauses

What combination of XOR constraints gave us the propagation?

- The above set of matrices cannot give us the reason clause
- Easy solution: the “green” columns are actually not zeroed out
- When looking for propagations/conflicts, we check if columns’ variable is set. If yes, we pretend it’s a 0
- When looking for reasons, we use the actual values
- All the row-XOR operations happen as before

Hence:

- Each row is a combination of input XOR constraints
- It is guaranteed to propagate/conflict under current variable assignment

When a variable is set, we are just wearing **“green glasses”, a bitmask**

CDCL(T) Gauss-Jordan Elimination: Backtracking

If we don’t zero out the columns, we get a free bonus! If we need to unset an assignment due to backtracking, we pretend we never set it (remove “green glasses”):

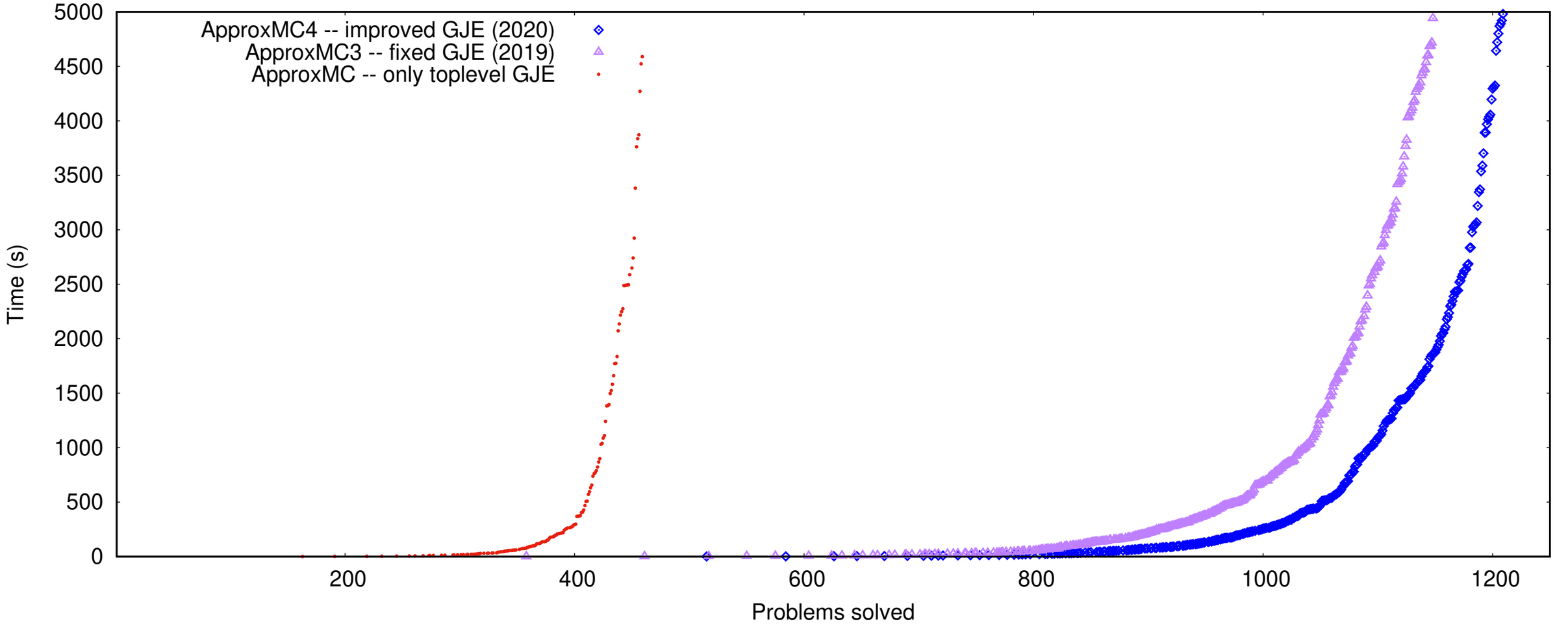
- All previous invariants still hold
- If the column had a responsible row, it still has it
- Both watches of the row are still good and in the watchlists
- Matrix looks differently than when we last had this assignment... is that a problem?
- No! Observe: new matrix could have been reached from the starting position, pivoting differently(!)

CDCL(T) Gauss-Jordan Elimination: Recap

Let’s recap! What was hard:

- Extracting XOR constraints
- Keeping CDCL and GJ in sync:
  - Fast update for variable setting (propagation)
  - Fast update for backtracking (conflict)
- Fast checking for propagation/conflict
- Lazy reason clause generation

Thank you!


ApproxMC4 -- improved GJE (2020)
ApproxMC3 -- fixed GJE (2019)
ApproxMC -- only toplevel GJE
Time (s)
Problems solved
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
200
400
600
800
1000
1200