2021-06-26

XOR Local Search for Boolean Brent Equations

XOR Local Search for Boolean Brent Equations

Wojciech Nawrocki Zhenjun Liu Andreas Fröhlich Mariin Heule Armin Biere



XOR Local Search for Boolean Brent Equations

Wojciech Nawrocki Zhenjun Liu Andreas Fröhlich Marijn Heule Armin Biere





SAT 2021

When SLS outperforms CDCL

Random k-SAT and satisfiable, hard-combinatorial problems.

XOR Local Search for Boolean Brent Equations

When SLS outperforms CDCL

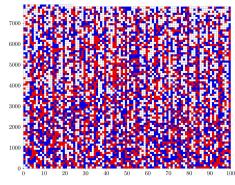
- While CDCL is the dominating SAT solving paradigm, there are problems on which Stochastic Local Search performs significantly better.
 - The largest satisfiable instance of the Boolean Pythagorean Triples problem can be solved using DDFW [Divide and Distribute Fixed Weights] local search in ~one CPU minute. Other algorithms time out.
 - SLS solvers perform well in the search for new matrix multiplication schemes expressed as a SAT problem via the Boolean Brent equations.
- Can we further improve the performance of LS on a class of problems where it already performs best? We look at problems involving XOR constraints, of which matrix multiplication is one instance.

2021-06-26

Random k-SAT and satisfiable, hard-combinatorial problems.

When SLS outperforms CDCL

Random k-SAT and satisfiable, hard-combinatorial problems.



(a) DDFW on Boolean Pythagorean Triples

XOR Local Search for Boolean Brent Equations

└─When SLS outperforms CDCL



When SLS outperforms CDCL

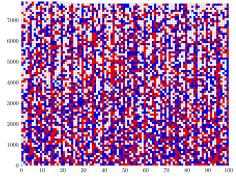
• While CDCL is the dominating SAT solving paradigm, there are problems on which Stochastic Local Search performs significantly better.

- The largest satisfiable instance of the Boolean Pythagorean Triples problem can be solved using DDFW [Divide and Distribute Fixed Weights] local search in ~one CPU minute. Other algorithms time out.
- SLS solvers perform well in the search for new matrix multiplication schemes expressed as a SAT problem via the Boolean Brent equations.
- Can we further improve the performance of LS on a class of problems where it already performs best? We look at problems involving XOR constraints, of which matrix multiplication is one instance.

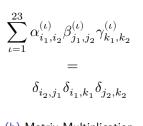
2021-

When SLS outperforms CDCL

Random k-SAT and satisfiable, hard-combinatorial problems.



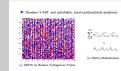




(b) Matrix Multiplication

XOR Local Search for Boolean Brent Equations

└─When SLS outperforms CDCL



When SLS outperforms CDCL

- While CDCL is the dominating SAT solving paradigm, there are problems on which Stochastic Local Search performs significantly better.
 - The largest satisfiable instance of the Boolean Pythagorean Triples problem can be solved using DDFW [Divide and Distribute Fixed Weights] local search in ~one CPU minute. Other algorithms time out.
 - SLS solvers perform well in the search for new matrix multiplication schemes expressed as a SAT problem via the Boolean Brent equations.
- Can we further improve the performance of LS on a class of problems where it already performs best? We look at problems involving XOR constraints, of which matrix multiplication is one instance.

2021-

 $(x_1 \oplus \ldots \oplus x_k) \mapsto ?$

2021-06-26

XOR Local Search for Boolean Brent Equations

Solving XOR in CNF form

 $(x_1\oplus\ldots\oplus x_k)\mapsto ?$

└─Solving XOR in CNF form

- To solve problems involving XOR constraints, we have to pick an encoding into CNF.
- Most straightforwardly, we can use a direct encoding XOR_d. But this produces exponentially many clauses.
- The usual linear approach is Tseitin encoding. It recursively breaks off fixed-size chunks and encodes them directly.
- But we pay for linearity. Tseitin encoding introduces auxiliary variables (*y*, underlined). These interact poorly with the SLS algorithm.

$$(x_1 \oplus \ldots \oplus x_k) \mapsto 1$$

$$\mathrm{XOR}_{\mathrm{d}}(x_1, ..., x_k) = \bigwedge_{\mathrm{even } \# -} (\pm x_1 \vee \ldots \vee \pm x_k)$$

 $(r \oplus \sigma r) \mapsto ?$

2021-06-26

XOR Local Search for Boolean Brent Equations

Solving XOR in CNF form

 $\begin{array}{l} (x_1 \oplus \ldots \oplus x_k) \mapsto ?\\ \mathrm{XOR}_* \mathrm{d}(x_1,...,x_k) = \bigwedge_{- \cdots + s} \ (\pm x_1 \vee \ldots \vee \pm x_k) \end{array}$

└─Solving XOR in CNF form

- To solve problems involving XOR constraints, we have to pick an encoding into CNF.
- Most straightforwardly, we can use a direct encoding XOR_d. But this produces exponentially many clauses.
- The usual linear approach is Tseitin encoding. It recursively breaks off fixed-size chunks and encodes them directly.
- But we pay for linearity. Tseitin encoding introduces auxiliary variables (*y*, underlined). These interact poorly with the SLS algorithm.

$$\operatorname{XOR_d}(x_1,...,x_k) = \bigwedge_{\operatorname{even} \# -} (\pm x_1 \lor \ldots \lor \pm x_k)$$

 $(r, \oplus \oplus r,) \mapsto ?$

 $\texttt{XOR_T_n}(x_1,...,x_k) = \texttt{XOR_d}(x_1,...,x_{n-1},-y) \land \texttt{XOR_T_n}(y,x_n,...,x_k)$

XOR Local Search for Boolean Brent Equations

└─Solving XOR in CNF form

Solving XOR in CNF form

 $(x_1 \oplus ... \oplus x_k) \mapsto ?$

 $\operatorname{XOR}_*\mathrm{d}(x_1,...,x_k) = \bigwedge_{\mathrm{even}\; \#^-} (\pm x_1 \vee \ldots \vee \pm x_k)$

 $\texttt{XOR}_\texttt{T}_\texttt{n}(x_1,...,x_k) = \texttt{XOR}_\texttt{d}(x_1,...,x_{n-1},-y) \land \texttt{XOR}_\texttt{T}_\texttt{n}(y,x_n,...,x_k) = \texttt{XOR}_\texttt{d}(x_1,...,x_{n-1},-y) \land \texttt{XOR}_\texttt{d}(x_1,...,x_k) = \texttt{XOR}_\texttt{d}(x_1,...,x_{n-1},-y) \land \texttt{XOR}_\texttt{d}(x_1,...,x_k) = \texttt{XOR}_\texttt{d}$

- To solve problems involving XOR constraints, we have to pick an encoding into CNF.
- Most straightforwardly, we can use a direct encoding XOR_d. But this produces exponentially many clauses.
- The usual linear approach is Tseitin encoding. It recursively breaks off fixed-size chunks and encodes them directly.
- But we pay for linearity. Tseitin encoding introduces auxiliary variables (*y*, underlined). These interact poorly with the SLS algorithm.

$$(x_1 \oplus \ldots \oplus x_k) + \gamma$$

$$\texttt{XOR_d}(x_1, \ldots, x_k) = \bigwedge_{\text{even } \#-} (\pm x_1 \lor \ldots \lor \pm x_k)$$

 $(r \oplus \phi r) \mapsto ?$

 $\texttt{XOR_T_n}(x_1,...,x_k) = \texttt{XOR_d}(x_1,...,x_{n-1},\underline{-y}) \land \texttt{XOR_T_n}(\underline{y},x_n,...,x_k)$

XOR Local Search for Boolean Brent Equations

└─Solving XOR in CNF form

Solving XOR in CNF form

 $(x_1\oplus\ldots\oplus x_k)\mapsto ?$

 $\mathtt{XOR}_a\mathtt{d}(x_1,...,x_k) = \bigwedge_{\mathtt{even}\; \phi-} (\pm x_1 \lor \ldots \lor \pm x_k)$

 $\texttt{XOR}_\texttt{T}_\texttt{n}(x_1,...,x_k) = \texttt{XOR}_\texttt{d}(x_1,...,x_{n-1},\underline{-y}) \land \texttt{XOR}_\texttt{T}_\texttt{n}(\underline{y},x_n,...,x_k) \land \texttt{XOR}_\texttt{T}_\texttt{n}(\underline{y},x_n,...,x_k) \land \texttt{XOR}_\texttt{T}_\texttt{n}(\underline{y},x_n,...,x_k) \land \texttt{XOR}_\texttt{T}_\texttt{n}(\underline{y},x_n,...,x_k) \land \texttt{XOR}_\texttt{n}(\underline{y},x_n,...,x_k) \land \texttt{X$

- To solve problems involving XOR constraints, we have to pick an encoding into CNF.
- Most straightforwardly, we can use a direct encoding XOR_d. But this produces exponentially many clauses.
- The usual linear approach is Tseitin encoding. It recursively breaks off fixed-size chunks and encodes them directly.
- But we pay for linearity. Tseitin encoding introduces auxiliary variables (*y*, underlined). These interact poorly with the SLS algorithm.

${\tt XOR_T_2}(x_1, x_2, x_3, x_4, x_5) =$

$$\begin{split} & \operatorname{XOR_d}(x_1, x_2, \underline{y_1}) \wedge \operatorname{XOR_d}(-\underline{y_1}, x_3, \underline{y_2}) \wedge \operatorname{XOR_d}(-\underline{y_2}, x_4, x_5) = \\ & (x_1, x_2, \underline{y_1}) \wedge (-x_1, -x_2, \underline{y_1}) \wedge (-x_1, x_2, -\underline{y_1}) \wedge (x_1, -x_2, -\underline{y_1}) \wedge \\ & (-\underline{y_1}, x_3, \underline{y_2}) \wedge (\underline{y_1}, -x_3, \underline{y_2}) \wedge (\underline{y_1}, x_3, -\underline{y_2}) \wedge (-\underline{y_1}, -x_3, -\underline{y_2}) \wedge \\ & (-\underline{y_2}, x_4, x_5) \wedge (\underline{y_2}, -x_4, x_5) \wedge (\underline{y_2}, x_4, -x_5) \wedge (-\underline{y_2}, -x_4, -x_5) \end{split}$$

XOR Local Search for Boolean Brent Equations

-Flipping Tseitin

2021-06-26

 ${\rm XOR}_{-}{\rm T}_{-}{\rm 2}(x_1,x_2,x_3,x_4,x_5) =$

Flipping Tseitin

 $\begin{array}{l} \mathrm{XOR}_{\mathbf{d}}(x_1,x_2,\underline{y_1}) \wedge \mathrm{XOR}_{\mathbf{d}}(\underline{-y_1},x_3,\underline{y_2}) \wedge \mathrm{XOR}_{\mathbf{d}}(\underline{-y_2},x_4,x_5) = \\ (x_1,x_2,\underline{y_1}) \wedge (-x_1,-x_2,\underline{y_1}) \wedge (-x_1,x_2,-\underline{y_1}) \wedge (x_1,-x_2,-\underline{y_1}) \wedge \\ (-\underline{y_1},x_3,\underline{y_2}) \wedge (\underline{y_1},-x_3,\underline{y_2}) \wedge (\underline{y_1},x_3,-\underline{y_2}) \wedge (-\underline{y_1},-x_3,-\underline{y_2}) \wedge \\ (-\underline{y_2},x_4,x_5) \wedge (\underline{y_2},-x_4,x_5) \wedge (\underline{y_2},x_4,-x_5) \wedge (-\underline{y_2},-x_4,-x_5) \wedge \\ \end{array}$

- At its core, SLS proceeds by flipping variables in falsified clauses according to a probabilistic heuristic.
- Consider solving the following XOR with 5 original and 2 auxiliary variables.
 - Suppose we start with the all-true assignment. The high-level constraint is *already* satisified but the solver does not "see" this because it operates on a low-level CNF representation.
 - The solver takes steps flipping variables, even going backwards in a sense by falsifying the high-level constraint in order to solve its CNF encoding.
 - Generally, for a given assignment to original variables there is precisely one assignment satisfying the auxiliary variables.
 Searching for it is unnecessary work.

${\tt XOR_T_2}(x_1, x_2, x_3, x_4, x_5) =$

$$\begin{split} & \operatorname{XOR_d}(x_1, x_2, \underline{y_1}) \wedge \operatorname{XOR_d}(-\underline{y_1}, x_3, \underline{y_2}) \wedge \operatorname{XOR_d}(-\underline{y_2}, x_4, x_5) = \\ & (x_1, x_2, \underline{y_1}) \wedge (-x_1, -x_2, \underline{y_1}) \wedge (-x_1, x_2, -\underline{y_1}) \wedge (x_1, -x_2, -\underline{y_1}) \wedge \\ & (-\underline{y_1}, x_3, \underline{y_2}) \wedge (\underline{y_1}, -x_3, \underline{y_2}) \wedge (\underline{y_1}, x_3, -\underline{y_2}) \wedge (-\underline{y_1}, -x_3, -\underline{y_2}) \wedge \\ & (-\underline{y_2}, x_4, x_5) \wedge (\underline{y_2}, -x_4, x_5) \wedge (\underline{y_2}, x_4, -x_5) \wedge (-\underline{y_2}, -x_4, -x_5) \end{split}$$

XOR Local Search for Boolean Brent Equations

-Flipping Tseitin

2021-06-26

 ${\rm NDR_{a}T_{a}2}(x_{1},x_{2},x_{3},x_{4},x_{5}) =$

Flipping Tseitin

$$\begin{split} & \mathsf{XGR}_{*} d(x_1, x_2, \underline{u}_1) \wedge \mathsf{XGR}_{*} d(-\underline{w}_1, x_3, \underline{w}_2) \wedge \mathsf{XGR}_{*} d(-\underline{w}_2, x_4, x_3) = \\ & (x_1, x_2, \underline{w}_1) \wedge (-x_1, -x_2, \underline{w}_1) \wedge (-x_1, x_2, -\underline{w}_1) \wedge (x_1, -x_2, -\underline{w}_1) \wedge \\ & (-\underline{w}_1, x_3, \underline{w}_2) \wedge (\underline{w}_1, -x_3, \underline{w}_2) \wedge (\underline{w}_1, x_3, -\underline{w}_2) \wedge (-\underline{w}_1, -x_3, -\underline{y}_2) \wedge \\ & (-\underline{w}_1, x_2, x_3) \wedge (\underline{w}_2, -x_4, x_3) \wedge (\underline{w}_2, x_4, -x_5) \wedge (-\underline{w}_2, -x_4, -x_5) \end{split}$$

- At its core, SLS proceeds by flipping variables in falsified clauses according to a probabilistic heuristic.
- Consider solving the following XOR with 5 original and 2 auxiliary variables.
 - Suppose we start with the all-true assignment. The high-level constraint is *already* satisified but the solver does not "see" this because it operates on a low-level CNF representation.
 - The solver takes steps flipping variables, even going backwards in a sense by falsifying the high-level constraint in order to solve its CNF encoding.
 - Generally, for a given assignment to original variables there is precisely one assignment satisfying the auxiliary variables.
 Searching for it is unnecessary work.

 \downarrow XOR_T_2(x_1, x_2, x_3, x_4, x_5) =

$$\begin{split} & \operatorname{XOR_d}(x_1, x_2, \underline{y_1}) \wedge \operatorname{XOR_d}(-\underline{y_1}, x_3, \underline{y_2}) \wedge \operatorname{XOR_d}(-\underline{y_2}, x_4, x_5) = \\ & (x_1, x_2, \underline{y_1}) \wedge (-x_1, -x_2, \underline{y_1}) \wedge (-x_1, x_2, -\underline{y_1}) \wedge (x_1, -x_2, -\underline{y_1}) \wedge \\ & (-\underline{y_1}, x_3, \underline{y_2}) \wedge (\underline{y_1}, -x_3, \underline{y_2}) \wedge (\underline{y_1}, x_3, -\underline{y_2}) \wedge (-\underline{y_1}, -x_3, -\underline{y_2}) \wedge \\ & (-\underline{y_2}, x_4, x_5) \wedge (\underline{y_2}, -x_4, x_5) \wedge (\underline{y_2}, x_4, -x_5) \wedge (-\underline{y_2}, -x_4, -x_5) \end{split}$$

XOR Local Search for Boolean Brent Equations

-Flipping Tseitin

2021-06-26

 $\downarrow \\ \texttt{NOR}_{\mathtt{a}}\mathtt{T}_{\mathtt{a}}\mathtt{2}(x_1, x_2, x_3, x_4, x_5) =$

Flipping Tseitin

 $\begin{array}{l} \mathrm{XOR}_{\mathbf{z}} \mathrm{d}(x_1, x_2, \underline{y_1}) \wedge \mathrm{XOR}_{\mathbf{z}} \mathrm{d}(-\underline{y_1}, x_3, \underline{y_2}) \wedge \mathrm{XOR}_{\mathbf{z}} \mathrm{d}(-\underline{y_2}, x_4, z_5) = \\ (x_1, x_2, \underline{y_1}) \wedge (-x_1, -x_2, \underline{y_1}) \wedge (-x_1, x_2, -\underline{y_1}) \wedge (x_1, -x_2, -\underline{y_1}) \wedge \\ (-\underline{y_1}, x_3, \underline{y_2}) \wedge (\underline{y_1}, -x_3, \underline{y_2}) \wedge (\underline{y_1}, x_3, -\underline{y_2}) \wedge (-\underline{y_1}, -x_3, -\underline{y_2}) \wedge \\ (-\underline{y_2}, x_4, x_5) \wedge (\underline{y_2}, -x_4, z_5) \wedge (\underline{y_2}, x_4, -z_5) \wedge (-\underline{y_2}, -x_4, -z_5) \end{array}$

- At its core, SLS proceeds by flipping variables in falsified clauses according to a probabilistic heuristic.
- Consider solving the following XOR with 5 original and 2 auxiliary variables.
 - Suppose we start with the all-true assignment. The high-level constraint is *already* satisified but the solver does not "see" this because it operates on a low-level CNF representation.
 - The solver takes steps flipping variables, even going backwards in a sense by falsifying the high-level constraint in order to solve its CNF encoding.
 - Generally, for a given assignment to original variables there is precisely one assignment satisfying the auxiliary variables.
 Searching for it is unnecessary work.

$\texttt{XOR_T_2}(x_1, x_2, \!\!\! x_3, x_4, x_5) =$

$$\begin{split} & \operatorname{XOR_d}(x_1, x_2, \underline{y_1}) \wedge \operatorname{XOR_d}(-\underline{y_1}, x_3, \underline{y_2}) \wedge \operatorname{XOR_d}(-\underline{y_2}, x_4, x_5) = \\ & (x_1, x_2, \underline{y_1}) \wedge (-x_1, -x_2, \underline{y_1}) \wedge (-x_1, x_2, -\underline{y_1}) \wedge (x_1, -x_2, -\underline{y_1}) \wedge \\ & (-\underline{y_1}, x_3, \underline{y_2}) \wedge (\underline{y_1}, -x_3, \underline{y_2}) \wedge (\underline{y_1}, x_3, -\underline{y_2}) \wedge (-\underline{y_1}, -x_3, -\underline{y_2}) \wedge \\ & (-\underline{y_2}, x_4, x_5) \wedge (\underline{y_2}, -x_4, x_5) \wedge (\underline{y_2}, x_4, -x_5) \wedge (-\underline{y_2}, -x_4, -x_5) \end{split}$$

XOR Local Search for Boolean Brent Equations

Flipping Tseitin

2021-06-26

 $\mathtt{XOR}_{\mathtt{a}}\mathtt{T}_{\mathtt{a}}\mathtt{2}(x_1, x_2, \mathbf{x}_3, x_4, x_5) =$

Flipping Tseitin

 $\begin{array}{l} \mathfrak{XOR}_{\mathbf{d}}(x_1,x_2,\underline{u}_1) \wedge \mathfrak{XOR}_{\mathbf{d}}(-\underline{w}_1,x_3,\underline{w}_2) \wedge \mathfrak{XOR}_{\mathbf{d}}(-\underline{w}_2,x_4,x_3) = \\ (x_1,x_2,\underline{u}_1) \wedge (-x_1,-x_2,\underline{u}_1) \wedge (-x_1,x_2,-\underline{u}_1) \wedge (x_1,-x_2,-\underline{u}_1) \wedge \\ (-\underline{u}_1,x_3,\underline{u}_2) \wedge (\underline{u}_1,-x_3,\underline{u}_2) \wedge (\underline{u}_1,x_3,-\underline{u}_2) \wedge (-\underline{u}_1,-x_3,-\underline{u}_2) \wedge \\ (-\underline{u}_2,x_4,x_5) \wedge (\underline{u}_2,-x_4,x_5) \wedge (\underline{u}_2,x_4,-x_5) \wedge (-\underline{u}_2,-x_4,-x_5) \end{pmatrix}$

- At its core, SLS proceeds by flipping variables in falsified clauses according to a probabilistic heuristic.
- Consider solving the following XOR with 5 original and 2 auxiliary variables.
 - Suppose we start with the all-true assignment. The high-level constraint is *already* satisified but the solver does not "see" this because it operates on a low-level CNF representation.
 - The solver takes steps flipping variables, even going backwards in a sense by falsifying the high-level constraint in order to solve its CNF encoding.
 - Generally, for a given assignment to original variables there is precisely one assignment satisfying the auxiliary variables.
 Searching for it is unnecessary work.

 \downarrow XOR_T_2(x_1, x_2, x_3, x_4, x_5) =

$$\begin{split} & \operatorname{XOR_d}(x_1, x_2, \underline{y_1}) \wedge \operatorname{XOR_d}(-\underline{y_1}, x_3, \underline{y_2}) \wedge \operatorname{XOR_d}(-\underline{y_2}, x_4, x_5) = \\ & (x_1, x_2, \underline{y_1}) \wedge (-x_1, -x_2, \underline{y_1}) \wedge (-x_1, x_2, -\underline{y_1}) \wedge (x_1, -x_2, -\underline{y_1}) \wedge \\ & (-\underline{y_1}, x_3, \underline{y_2}) \wedge (\underline{y_1}, -x_3, \underline{y_2}) \wedge (\underline{y_1}, x_3, -\underline{y_2}) \wedge (-\underline{y_1}, -x_3, -\underline{y_2}) \wedge \\ & (-\underline{y_2}, x_4, x_5) \wedge (\underline{y_2}, -x_4, x_5) \wedge (\underline{y_2}, x_4, -x_5) \wedge (-\underline{y_2}, -x_4, -x_5) \end{split}$$

XOR Local Search for Boolean Brent Equations

-Flipping Tseitin

2021-06-26

Ļ

 $\mathtt{NDR}_{a}\mathtt{T}_{a}\mathtt{2}(x_{1},x_{2},\!x_{3},x_{4},x_{5}) =$

Flipping Tseitin

 $\begin{array}{l} \mathrm{XOR}_{\mathbf{z}} \mathrm{d}(x_1, x_2, \underline{y}_1) \wedge \mathrm{XOR}_{\mathbf{z}} \mathrm{d}(-\underline{w}_1, x_3, \underline{y}_2) \wedge \mathrm{XOR}_{\mathbf{z}} \mathrm{d}(-\underline{w}_2, x_4, z_5) = \\ (x_1, x_2, \underline{y}_1) \wedge (-x_1, -x_2, \underline{y}_1) \wedge (-x_1, x_2, -\underline{y}_1) \wedge (x_1, -x_2, -\underline{y}_1) \wedge \\ (-\underline{y}_1, x_3, \underline{y}_2) \wedge (\underline{y}_1, -x_3, \underline{y}_2) \wedge (\underline{y}_1, x_3, -\underline{y}_2) \wedge (-\underline{y}_1, -x_3, -\underline{y}_2) \wedge \\ (-\underline{y}_1, x_4, x_5) \wedge (\underline{y}_2, -x_4, z_5) \wedge (\underline{y}_2, x_4, -z_5) \wedge (-\underline{y}_2, -x_4, -z_5) \end{array}$

- At its core, SLS proceeds by flipping variables in falsified clauses according to a probabilistic heuristic.
- Consider solving the following XOR with 5 original and 2 auxiliary variables.
 - Suppose we start with the all-true assignment. The high-level constraint is *already* satisified but the solver does not "see" this because it operates on a low-level CNF representation.
 - The solver takes steps flipping variables, even going backwards in a sense by falsifying the high-level constraint in order to solve its CNF encoding.
 - Generally, for a given assignment to original variables there is precisely one assignment satisfying the auxiliary variables.
 Searching for it is unnecessary work.

XNF

2021-06-26

XNF XOR Local Search for Boolean Brent Equations p xnf 3 2 1 2 3 0 x 1 2 0

 $(x_1 \lor x_2 \lor x_3) \land$ $(x_1 \oplus x_2)$

- We propose to experiment with native XOR representations more widely. In the spirit of DIMACS CNF, an XNF format could be used.
- Worth noting that we later found out XNF is already implemented in CryptoMiniSAT.

xnfSAT: Stochastic Local Search with native XOR

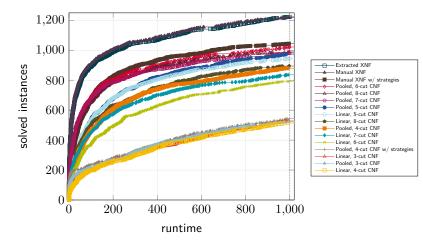
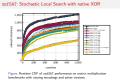


Figure: Runtime CDF of xnfSAT performance on matrix multiplication benchmarks with varying encodings and solver versions.

XOR Local Search for Boolean Brent Equations

L_xnfSAT: Stochastic Local Search with native XOR



- To solve XNF, we present xnfSAT, an SLS solver supporting native XOR.
- Its performance on matrix multiplication benchmarks significantly improves upon the best CNF-based solver, YalSAT.
- To go with xnfSAT, we implemented a tool to extract XOR gates from CNF files.

Algorithm YalSAT, a WalkSAT-based solver

- 1: for clause in input file do
- 2: parse and store clause to data structure
- 3: end for
- 4: preprocess formula
- 5: $\alpha \leftarrow \text{complete initial assignment of truth values}$
- 6: while there exists a clause falsified by $\alpha~{\rm do}$
- 7: $C \leftarrow \texttt{pickUnsatClause}()$
- 8: $x \leftarrow \texttt{pickVarIn}(C)$
- 9: $\alpha \leftarrow \alpha$ with x flipped
- 10: update solver state
- 11: end while

XOR Local Search for Boolean Brent Equations

We build on YalSAT

Algorithm ValleT, a ValleT hand solver 1 for classe in legate for do 2 graphic distance classe to do dist structures 3 graphics formal 5 $a \sim complex initial algorithm of the values$ $5 <math>a \sim complex initial algorithm of the values$ $5 <math>a \sim complex initial algorithm of the values$ $5 <math>a \sim complex initial algorithm of the values$ $5 <math>a \sim complex initial (0)$ 1 $a \sim complex initial (0)$ 2 $a \sim complex initial (0)$ 3 $a \sim complex initial (0)$ 3

We build on Ya1SAT

- xnfSAT is based on YalSAT. Instructive to understand its outline.
- Unsurprisingly, supporting XOR needs no modifications to the high-level structure.
- We adapt parsing (XNF), preprocessing and variable selection (pickVarIn).

Algorithm YalSAT, a WalkSAT-based solver

- 1: for clause in input file do
- 2: parse and store clause to data structure
- 3: end for
- 4: preprocess formula
- 5: $\alpha \leftarrow \text{complete initial assignment of truth values}$
- 6: while there exists a clause falsified by $\alpha~{\rm do}$
- $\textbf{7:} \quad C \leftarrow \texttt{pickUnsatClause}()$
- 8: $x \leftarrow \texttt{pickVarIn}(C)$
- 9: $\alpha \leftarrow \alpha$ with x flipped
- 10: update solver state
- 11: end while

XOR Local Search for Boolean Brent Equations

└─We build on YalSAT



We build on Ya1SAT

- xnfSAT is based on YalSAT. Instructive to understand its outline.
- Unsurprisingly, supporting XOR needs no modifications to the high-level structure.
- We adapt parsing (XNF), preprocessing and variable selection (pickVarIn).

Algorithm YalSAT, a WalkSAT-based solver

- 1: for clause in input file do
- 2: parse and store clause to data structure
- 3: end for
- 4: preprocess formula
- 5: $\alpha \leftarrow \text{complete initial assignment of truth values}$
- 6: while there exists a clause falsified by $\alpha~{\rm do}$
- $\textbf{7:} \quad C \leftarrow \texttt{pickUnsatClause}()$
- 8: $x \leftarrow \texttt{pickVarIn}(C)$
- 9: $\alpha \leftarrow \alpha$ with x flipped
- 10: update solver state
- 11: end while

XOR Local Search for Boolean Brent Equations

We build on YalSAT



- xnfSAT is based on YalSAT. Instructive to understand its outline.
- Unsurprisingly, supporting XOR needs no modifications to the high-level structure.
- We adapt parsing (XNF), preprocessing and variable selection (pickVarIn).

Algorithm YalSAT, a WalkSAT-based solver

- 1: for clause in input file do
- 2: parse and store clause to data structure
- 3: end for
- 4: preprocess formula
- 5: $\alpha \leftarrow \text{complete initial assignment of truth values}$
- 6: while there exists a clause falsified by $\alpha~{\rm do}$
- $\textbf{7:} \quad C \leftarrow \texttt{pickUnsatClause}()$
- 8: $x \leftarrow \texttt{pickVarIn}(C)$
- 9: $\alpha \leftarrow \alpha$ with x flipped
- 10: update solver state
- 11: end while

XOR Local Search for Boolean Brent Equations

-We build on YalSAT

Algorithm Y1227, a U12374 based solver 1 for classe in spec file do x_{press} provide the dot in the structure x_{press} provide the dot in the structure x_{press} provide the solution of the structure x_{press} provide the solution x_{press} solution x_{press} and x_{press} x_{press} and x_{press} and x_{press} and x_{press} x_{press} and x_{press} x_{press} and x_{pres

We build on Ya1SAT

- xnfSAT is based on YalSAT. Instructive to understand its outline.
- Unsurprisingly, supporting XOR needs no modifications to the high-level structure.
- We adapt parsing (XNF), preprocessing and variable selection (pickVarIn).

Parsing and clause storage

XOR Local Search for Boolean Brent Equations

Parsing and clause storage

Parsing and clause storage

 XOR constraints are stored as buffers of variable indices, forgetting negations.
 Track the parity of each so that ⊕_i x_i is satisfied iff ∑x_i + oarity = 1 mod 2.

- XOR constraints are stored as buffers of variable indices, forgetting negations.
- Track the *parity* of each so that $\bigoplus_i x_i$ is satisfied iff $\Sigma x_i + \text{parity} \equiv 1 \mod 2$.

Parsing and clause storage

XOR Local Search for Boolean Brent Equations

Parsing and clause storage

- XOR constraints are stored as buffers of variable indices, forgetting negations.
- ► Track the parity of each so that ⊕_i x_i is satisfied iff Σx_i + parity ≡ 1 mod 2.
- For example, $(x_1 \oplus -x_2)$ is $(x_1 \oplus x_2, parity = 1)$.

- XOR constraints are stored as buffers of variable indices, forgetting negations.
- Track the *parity* of each so that $\bigoplus_i x_i$ is satisfied iff $\Sigma x_i + \text{parity} \equiv 1 \mod 2$.
- $\blacktriangleright \ \, {\rm For \ example,} \ \, (x_1\oplus -x_2) \ \, {\rm is} \ \, (x_1\oplus x_2, {\tt parity}=1).$

XOR Local Search for Boolean Brent Equations

└─Preprocessing

2021-06-26

Preprocessing

 During preprocessing, to remove a propagating unit from an XOR constraint we simply flip the parity.

XOR Local Search for Boolean Brent Equations

└─Preprocessing

2021-06-26

During preprocessing, to remove a propagating unit from an XOR constraint we simply flip the parity.
 Have (x₁ ⊕ x₂ ⊕ x₃, parity = 0)

Preprocessing

```
\blacktriangleright \text{ Have } (x_1 \oplus x_2 \oplus x_3, \texttt{parity} = 0)
```

XOR Local Search for Boolean Brent Equations

Preprocessing

2021-06-26

During preprocessing, to remove a propagating unit from an XOR constraint we simply flip the parity.
 Have (x₁ ⊕ x₂ ⊕ x₃, parity = 0)
 Then x₁ = 1 propagates

Preprocessing

▶ Have
$$(x_1 \oplus x_2 \oplus x_3, parity = 0)$$
 ▶ Then $x_1 = 1$ propagates

XOR Local Search for Boolean Brent Equations

└─Preprocessing

2021-06-26

During preprocessing, to remove a propagating unit from an XOR constraint we simply lip the parky.
 How (x_i = 0, x_i = x_i = x_i) = x_i = x_i = 1 propagates
 Then x_i = 1 propagates
 Then how (x_i = 0, x_i = x_i = x_i = x_i)

Preprocessing

Have
$$(x_1 \oplus x_2 \oplus x_3, \text{parity} = 0)$$
Then $x_1 = 1$ propagates
Then have $(x_2 \oplus x_3, \text{parity} = 1)$

▶ $x_i \leftarrow \texttt{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_w(x_i)}$

XOR Local Search for Boolean Brent Equations

probSAT-like variable selection

▶ $x_i \leftarrow \text{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability ~ $\frac{1}{break_w(x_i)}$

└─probSAT-like variable selection

- Clause and variable selection heuristics are the most important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.
- While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

▶ $x_i \leftarrow \operatorname{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_w(x_i)}$ ▶ Let $B(x_i)$ be the clauses falsified on flipping x_i XOR Local Search for Boolean Brent Equations

probSAT-like variable selection

x_i ← PickVarIn(x₁ ∨ ... ∨ x_k) with probability ~ ¹/_{break_u(x_i)}
 Let B(x_i) be the clauses falsified on flipping x_i

└─probSAT-like variable selection

- Clause and variable selection heuristics are *the most* important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.
- While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

▶ $x_i \leftarrow \texttt{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_w(x_i)}$

Let B(x_i) be the clauses falsified on flipping x_i
 and w be a clause weighing function;

XOR Local Search for Boolean Brent Equations

probSAT-like variable selection

 $\begin{array}{l} \bullet \quad x_i \leftarrow \operatorname{PickVarIn}(x_1 \lor \dots \lor x_k) \text{ with probability} \sim \frac{1}{break_w(x_i)} \\ \bullet \quad \operatorname{Let} B(x_i) \text{ be the clauses failsified on flipping } x_i \\ \bullet \quad \operatorname{and} w \text{ be a clause weighing function;} \end{array}$

probSAT-like variable selection

- Clause and variable selection heuristics are *the most* important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.
- While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

▶ $x_i \leftarrow \texttt{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_m(x_i)}$

Let $B(x_i)$ be the clauses falsified on flipping x_i and w be a clause weighing function;

 $\blacktriangleright \text{ Then } break_w(x_i) = \Sigma_{C \in B(x_i)} w(C)$

XOR Local Search for Boolean Brent Equations

probSAT-like variable selection

- Clause and variable selection heuristics are the most important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.
- While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

x_i ← PickVarIn(x₁ ∨ ... ∨ x_k) with probability ~ ¹/_{Mreak_u(x_i)}
 Let B(x_i) be the clauses faithful on flipping x_i
 and w be a clause swighing function;
 Then break_u(x_i) = 2_{CO(Re)_i}w(C)

▶ $x_i \leftarrow \operatorname{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_{\cdots}(x_i)}$

- Let $B(x_i)$ be the clauses falsified on flipping x_i
- \blacktriangleright and w be a clause weighing function;
- $\blacktriangleright \text{ Then } break_w(x_i) = \Sigma_{C \in B(x_i)} w(C)$
- \blacktriangleright While $w(x_1 \lor \ldots \lor x_n) \sim n$, we set $w(x_1 \oplus \ldots \oplus x_m)$ constant. Why?

XOR Local Search for Boolean Brent Equations

-probSAT-like variable selection

- Clause and variable selection heuristics are the most important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.
- While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

2021-06-26

probSAT-like variable selection

$$\begin{split} & \textbf{k}_i \leftarrow \mathsf{PickWarIn}(x_1 \vee \ldots \vee x_k) \text{ with probability} \sim \frac{1}{4 \max_{x \in X_i}(x_i)} \\ & \textbf{k} tat B(x_i) \text{ be the classes failsfield on flipping } x_i \\ & \textbf{k} and w be a classe weighing function; \\ & \mathsf{Tash herod}_w(x_i) = 2 \sum_{x \in X_i}(x_i) (w_i) \\ & \mathsf{TWeile} w(x_i \vee \ldots \vee x_k) \sim n, \text{ we sat } w(x_i \oplus \ldots \oplus x_m) \text{ constant.} \\ & \mathsf{Marcel} \end{aligned}$$

▶ $x_i \leftarrow \operatorname{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_{\cdots}(x_i)}$

- Let $B(x_i)$ be the clauses falsified on flipping x_i
- \blacktriangleright and w be a clause weighing function;
- $\blacktriangleright \text{ Then } break_w(x_i) = \Sigma_{C \in B(x_i)} w(C)$
- \blacktriangleright While $w(x_1 \lor \ldots \lor x_n) \sim n,$ we set $w(x_1 \oplus \ldots \oplus x_m)$ constant. Why?
 - 1. Simplicity.

XOR Local Search for Boolean Brent Equations

 $\begin{array}{l} \textbf{x}_1 \leftarrow \mathsf{PickVarIn}(x_1 \lor \dots \lor x_k) \text{ with probability} \sim \frac{1}{\operatorname{Resch}_{\mathbb{R}^2}(x_1)}\\ \textbf{b} \ \operatorname{Lst} B(x_1) \ b \ \operatorname{the classes fabriced on flipping} x_1\\ \textbf{b} \ \operatorname{and} w \ ba \ s \ classes weighting function;\\ \textbf{Then} \ for a \ ba_{\mathbb{R}^2}(x_2) = \sum_{C \in [d_{\mathbb{R}^2}(w)]}(C)\\ \textbf{Write} \ w(x_1 \lor \dots \lor x_m) < w, \ we \ \operatorname{stet} w(x_1 \oplus \dots \oplus x_m) \ \operatorname{constant}. \end{array}$

probSAT-like variable selection

1. Simplicity

└─probSAT-like variable selection

- Clause and variable selection heuristics are *the most* important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.
- While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

▶ $x_i \leftarrow \operatorname{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_w(x_i)}$

- Let $B(x_i)$ be the clauses falsified on flipping x_i
- \blacktriangleright and w be a clause weighing function;
- $\blacktriangleright \text{ Then } break_w(x_i) = \Sigma_{C \in B(x_i)} w(C)$
- \blacktriangleright While $w(x_1 \lor \ldots \lor x_n) \sim n,$ we set $w(x_1 \oplus \ldots \oplus x_m)$ constant. Why?
 - 1. Simplicity.
 - 2. A literal x is *critical* in C if flipping x breaks C.

XOR Local Search for Boolean Brent Equations

▶ $x_i \leftarrow \operatorname{PickWarIn}(x_1 \lor \Box \lor x_k)$ with probability $\sim \frac{1}{\operatorname{derat}_{U(x_i)}}$ ▶ Let $B(x_i)$ be the classes fairfind on flipping x_i ▶ and w be a classe weighing function; ▶ Then $\operatorname{bread}_{U(x_i)} = \Sigma_{Cy(k_i), w}(G)$ While $w(x_i) \cup w_i > x_i$ as w set $w(x_i) \oplus \dots \oplus x_m)$ constant

2. A literal x is critical in C if flipping x breaks C

probSAT-like variable selection

└─probSAT-like variable selection

- Clause and variable selection heuristics are *the most* important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.
- While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

▶ $x_i \leftarrow \operatorname{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_w(x_i)}$

- Let $B(x_i)$ be the clauses falsified on flipping x_i
- ▶ and w be a clause weighing function;
- $\blacktriangleright \text{ Then } break_w(x_i) = \Sigma_{C \in B(x_i)} w(C)$
- \blacktriangleright While $w(x_1 \lor \ldots \lor x_n) \sim n,$ we set $w(x_1 \oplus \ldots \oplus x_m)$ constant. Why?
 - 1. Simplicity.
 - 2. A literal x is *critical* in C if flipping x breaks C.
 - $\blacktriangleright \ \ln \ (x_1 \lor x_2 \lor x_3), \ x_1 \ \text{critical}.$

XOR Local Search for Boolean Brent Equations

└─probSAT-like variable selection

 $\label{eq:constant} \begin{array}{l} \mathbf{b} \quad \mathrm{tot} \ B(\mathbf{c}_1) \ \mathrm{is} \ \mathrm{tots} \ \mathrm{dots} \ \mathrm{d$

probSAT-like variable selection

x_i ← PickVarIn(x₁ ∨ ... ∨ x_k) with probability ~ 1/(N=1/4)

- Clause and variable selection heuristics are *the most* impo
- Clause and variable selection heuristics are the most important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.
- While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

-06-26

2021-

▶ $x_i \leftarrow \operatorname{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_w(x_i)}$

- Let $B(x_i)$ be the clauses falsified on flipping x_i
- ▶ and w be a clause weighing function;
- $\blacktriangleright \text{ Then } break_w(x_i) = \Sigma_{C \in B(x_i)} w(C)$
- \blacktriangleright While $w(x_1 \lor \ldots \lor x_n) \sim n,$ we set $w(x_1 \oplus \ldots \oplus x_m)$ constant. Why?
 - 1. Simplicity.
 - 2. A literal x is *critical* in C if flipping x breaks C.
 - In $(x_1 \lor x_2 \lor x_3)$, x_1 critical.
 - $\blacktriangleright In \ (x_1 \lor x_2 \lor x_3), \text{ nothing critical.}$

XOR Local Search for Boolean Brent Equations

probSAT-like variable selection

- Clause and variable selection heuristics are *the most* important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.
- While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

-06-26

2021-

probSAT-like variable selection

x_i ← PickWarIn(x₁ ∨ ... ∨ x_k) with probability ~ ^k_{invok_i(x_i)}
 Let B(x_i) be the classes fabrified on flipping x_i
 and w be a classe weighing function;
 Then break_i(x_i) = Σ<sub>CoRe_i(x_i)(C)
 Write w(x_i, ⊕ x_i) = x_i, w sets w(x_i, ⊕ x_i, ⊕ x_i) constant
</sub>

A literal x is critical in C if flipping x breaks 0
 In (x₁ ∨ x_n ∨ x_n), x_n critical.

In (x1 V x2 V x3), nothing critical

▶ $x_i \leftarrow \texttt{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_w(x_i)}$

- \blacktriangleright Let $B(x_i)$ be the clauses falsified on flipping x_i
- \blacktriangleright and w be a clause weighing function;
- $\blacktriangleright \text{ Then } break_w(x_i) = \Sigma_{C \in B(x_i)} w(C)$
- \blacktriangleright While $w(x_1 \lor \ldots \lor x_n) \sim n,$ we set $w(x_1 \oplus \ldots \oplus x_m)$ constant. Why?
 - 1. Simplicity.
 - 2. A literal x is *critical* in C if flipping x breaks C.
 - $\blacktriangleright \text{ In } (x_1 \lor x_2 \lor x_3), x_1 \text{ critical.}$
 - $\blacktriangleright \ \ \mathsf{In} \ (x_1 \lor x_2 \lor x_3), \text{ nothing critical}.$
 - In a satisfied XOR, all literals are critical.

XOR Local Search for Boolean Brent Equations

└─probSAT-like variable selection

• Clause and variable selection heuristics are *the most* important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.

probSAT-like variable selection

x₁ ← PickVarIn(x₁ ∨ ... ∨ x_k) with probability ~ ¹/_{break_u(x_i)}
 Let B(x_i) be the classes falsified on flipping x_i
 and w be a classe subplied function;
 Then break_u(x_i) = 2_{CYME_i} w(C)

While $w(x, \forall ..., \forall x_{-}) \sim n$, we set $w(x, \oplus ..., \oplus x_{-})$ constant

A literal x is critical in C if flipping x breaks 0
 In (x₁ ∨ x_n ∨ x_n), x_n critical.

In (x1 V x2 V x3), nothing critical. In a satisfied XOR, all literals are critical

 While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

-26

-06

2021-

▶ $x_i \leftarrow \texttt{PickVarIn}(x_1 \lor ... \lor x_k)$ with probability $\sim \frac{1}{break_w(x_i)}$

- \blacktriangleright Let $B(x_i)$ be the clauses falsified on flipping x_i
- \blacktriangleright and w be a clause weighing function;
- $\blacktriangleright \text{ Then } break_w(x_i) = \Sigma_{C \in B(x_i)} w(C)$
- \blacktriangleright While $w(x_1 \lor \ldots \lor x_n) \sim n,$ we set $w(x_1 \oplus \ldots \oplus x_m)$ constant. Why?
 - 1. Simplicity.
 - 2. A literal x is *critical* in C if flipping x breaks C.
 - $\blacktriangleright \text{ In } (x_1 \lor x_2 \lor x_3), x_1 \text{ critical.}$
 - ln $(x_1 \lor x_2 \lor x_3)$, nothing critical.
 - In a satisfied XOR, all literals are critical.
- ▶ For efficiency, *break_w* tables are cached. Tracking critical literals allows for fast updates.

XOR Local Search for Boolean Brent Equations

└─probSAT-like variable selection



probSAT-like variable selection

- Clause and variable selection heuristics are the most important parts of LS solvers. YalSAT and consequently xnfSAT use probSAT-like selection extended with weights.
- While long OR clauses with many satisfied literals are hard to break, XOR constraints can always be broken. Length is not a good measure of the importance of an XOR constraint.

26

-00

2021-

How fast can CNF get? Pooled encoding

2021-06-26

XOR Local Search for Boolean Brent Equations

How fast can CNF get? Pooled encoding

 $\texttt{XOR}_\texttt{l}_\texttt{n}(x_1,...,x_k) = \texttt{XOR}_\texttt{d}(x_1,...,x_{n-1},-y) \land \texttt{XOR}_\texttt{l}_\texttt{n}(y,x_n,...,x_k)$

└─How fast can CNF get? Pooled encoding

- To make sure solving XNF strongly outperforms CNF, we performed heavy tuning on the CNF formula. We tried two variants of the Tseitin encoding.
 - First, the linear encoding seen earlier, produced with a stack.
 - Then, a *pooled* encoding produced with a queue. To our knowledge, this encoding is novel.
- We also tried various *cutting numbers* sizes of the directly-encoded chunks.

 $\texttt{XOR_l_n}(x_1,...,x_k) = \texttt{XOR_d}(x_1,...,x_{n-1},-y) \land \texttt{XOR_l_n}(y,x_n,...,x_k)$

How fast can CNF get? Pooled encoding

$$\texttt{XOR_l_n}(x_1,...,x_k) = \texttt{XOR_d}(x_1,...,x_{n-1},-y) \land \texttt{XOR_l_n}(y,x_n,...,x_k)$$

$$\texttt{XOR_p_n}(x_1,...,x_k) = \texttt{XOR_d}(x_1,...,x_{n-1},-y) ~\land~ \texttt{XOR_p_n}(x_n,...,x_k,y)$$

XOR Local Search for Boolean Brent Equations

└─How fast can CNF get? Pooled encoding

How fast can CNF get? Pooled encoding

 $\texttt{XOR}_\texttt{l}_\texttt{n}(x_1,...,x_k) = \texttt{XOR}_\texttt{d}(x_1,...,x_{n-1},-y) \land \texttt{XOR}_\texttt{l}_\texttt{n}(y,x_n,...,x_k)$

 $\texttt{XOR}_\texttt{p}_\texttt{n}(x_1,...,x_k) = \texttt{XOR}_\texttt{d}(x_1,...,x_{n-1},-y) \ \land \ \texttt{XOR}_\texttt{p}_\texttt{n}(x_n,...,x_k,y)$

- To make sure solving XNF strongly outperforms CNF, we performed heavy tuning on the CNF formula. We tried two variants of the Tseitin encoding.
 - First, the linear encoding seen earlier, produced with a stack.
 - Then, a *pooled* encoding produced with a queue. To our knowledge, this encoding is novel.
- We also tried various *cutting numbers* sizes of the directly-encoded chunks.

How fast can CNF get? Pooled encoding

$$\texttt{XOR_l_n}(x_1,...,x_k) = \texttt{XOR_d}(x_1,...,x_{n-1},-y) \land \texttt{XOR_l_n}(y,x_n,...,x_k)$$

$$\begin{split} \text{XOR_p_n}(x_1,...,x_k) = \text{XOR_d}(x_1,...,x_{n-1},-y) ~ \wedge ~ \text{XOR_p_n}(x_n,...,x_k,y) \\ \uparrow \\ \text{cutting number} \end{split}$$

XOR Local Search for Boolean Brent Equations

How fast can CNF get? Pooled encoding

How fast can CNF get? Pooled encoding

 $\texttt{XOR}_\texttt{l}_\texttt{n}(x_1,...,x_k) = \texttt{XOR}_\texttt{d}(x_1,...,x_{n-1},-y) \land \texttt{XOR}_\texttt{l}_\texttt{n}(y,x_n,...,x_k)$

 $\begin{array}{l} \mathrm{XOR}_{\mathbf{z}} \mathrm{P}_{\mathbf{z}} \mathrm{n}(x_1,...,x_k) = \mathrm{XOR}_{\mathbf{z}} \mathrm{d}(x_1,...,x_{n-1},-y) \ \land \ \mathrm{XOR}_{\mathbf{z}} \mathrm{P}_{\mathbf{z}} \mathrm{n}(x_n,...,x_k,y) \\ \uparrow \\ \mathrm{cutting number} \end{array}$

- To make sure solving XNF strongly outperforms CNF, we performed heavy tuning on the CNF formula. We tried two variants of the Tseitin encoding.
 - First, the linear encoding seen earlier, produced with a stack.
 - Then, a *pooled* encoding produced with a queue. To our knowledge, this encoding is novel.
- We also tried various *cutting numbers* sizes of the directly-encoded chunks.

How fast can CNF get?

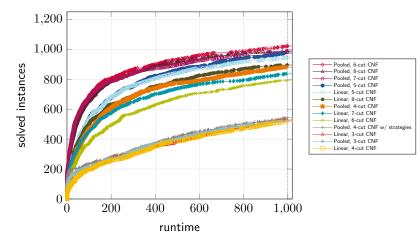
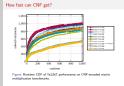


Figure: Runtime CDF of YalSAT performance on CNF-encoded matrix multiplication benchmarks.

XOR Local Search for Boolean Brent Equations

└─How fast can CNF get?



- On these instances, pooled encodings are better across the board.
- Interestingly, performance initially increases with cutting number and plateaus at 6.

Not as fast as XNF!

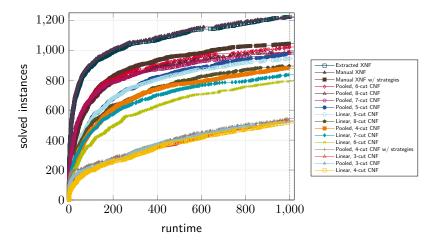
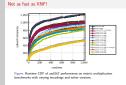


Figure: Runtime CDF of xnfSAT performance on matrix multiplication benchmarks with varying encodings and solver versions.

XOR Local Search for Boolean Brent Equations

└─Not as fast as XNF!



- Here, XNF solving improves over even highly tuned CNF, without having to spend any computational power on optimising the XOR encoding.
- Within a 1000s timeout, our solver operating on XNF can find between 200 and 700 more solutions compared to CNF-based runs in various configurations.

Conclusion

XOR Local Search for Boolean Brent Equations

└─Conclusion

- Implemented SLS with native XOR constraints in xnfSAT.
- Observed strong performance improvements on matrix multiplication benchmarks where SLS already outperformed CDCL.
- Propose to experiment with native XOR more widely and to standardise the XNF format.
- https://github.com/Wtec234/xnfSA1
- https://github.com/arminbiere/cnf2xnf

- Implemented SLS with native XOR constraints in xnfSAT.
- Observed strong performance improvements on matrix multiplication benchmarks where SLS already outperformed CDCL.
- Propose to experiment with native XOR more widely and to standardise the XNF format.
- https://github.com/Vtec234/xnfSAT
- https://github.com/arminbiere/cnf2xnf