Undercover Boolean Matrix Factorization with MaxSAT

Florent Avellaneda and Roger Villemaire

Université du Québec à Montréal (UQAM)

AAAI 2022

Summary

1. Introduction
2. (Max)SAT Encoding
3. Undercover Factorization
4. Block-Optimal Undercover

Plan

1. Introduction
2. (Max)SAT Encoding
3. Undercover Factorization
4. Block-Optimal Undercover

Matrix Factorization Problem

$$M = \begin{vmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

Goal:

Find $A_{m \times k}$ and $B_{k \times n}$ such that $A \times B \approx M$

$$(A \times B)_{i,j} = \sum_{\ell=1}^{k} A_{i,\ell} \times B_{\ell,j}$$

Matrix Factorization Problem

$$M = \begin{vmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

Goal:
Find $A_{m \times k}$ and $B_{k \times n}$ such that $A \times B \approx M$

$$(A \times B)_{i,j} = \sum_{\ell=1}^{k} A_{i,\ell} \times B_{\ell,j}$$

Example of a rank 2 factorization ($k = 2$):

$$\begin{array}{cc} & \begin{vmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{vmatrix} \\ \begin{vmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{vmatrix} & \begin{vmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix} \end{array}$$

Constraints:

$$\forall i, j : \sum_{\ell=0}^{k} a_{i,\ell} \times b_{\ell,j} \approx M_{i,j}$$

Solution with SVD

$$\begin{vmatrix} 0.5 & 0.7 & 0.5 \\ -0.7 & 0 & 0.7 \end{vmatrix}$$

$$\begin{vmatrix} 1.1 & 0.7 \\ 1.7 & 0 \\ 1.2 & -0.7 \end{vmatrix} \quad \begin{vmatrix} 0.11 & 0.84 & 1.09 \\ 0.89 & 1.19 & 0.85 \\ 1.09 & 0.84 & 0.11 \end{vmatrix}$$

Solution with SVD

$$\begin{vmatrix} 0.5 & 0.7 & 0.5 \\ -0.7 & 0 & 0.7 \end{vmatrix}$$

$$\begin{vmatrix} 1.1 & 0.7 \\ 1.7 & 0 \\ 1.2 & -0.7 \end{vmatrix} \quad \begin{vmatrix} 0.11 & 0.84 & 1.09 \\ 0.89 & 1.19 & 0.85 \\ 1.09 & 0.84 & 0.11 \end{vmatrix}$$

Problems:

- No exact solution of rank 2

Solution with SVD

$$\begin{vmatrix} 0.5 & 0.7 & 0.5 \\ -0.7 & 0 & 0.7 \end{vmatrix}$$

$$\begin{vmatrix} 1.1 & 0.7 \\ 1.7 & 0 \\ 1.2 & -0.7 \end{vmatrix} \quad \begin{vmatrix} 0.11 & 0.84 & 1.09 \\ 0.89 & 1.19 & 0.85 \\ 1.09 & 0.84 & 0.11 \end{vmatrix}$$

	Paw Patrol	Shrek	Matrix
Alice	0.11	0.84	1.09
Bob	0.89	1.19	0.85
Charle	1.09	0.84	0.11

Problems:

- No exact solution of rank 2
- Poor interpretability of the factorization

Boolean Matrix Factorization Problem

$$M = \begin{vmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

Goal:

Find $A_{m\times k}$ and $B_{k\times n}$ such that $A \circ B \approx M$

$$(A \circ B)_{i,j} = \bigvee_{\ell=1}^{k} A_{i,\ell} \wedge B_{\ell,j}$$

Boolean Matrix Factorization Problem

$$M = \begin{vmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

Goal:

Find $A_{m \times k}$ and $B_{k \times n}$ such that $A \circ B \approx M$

$$(A \circ B)_{i,j} = \bigvee_{\ell=1}^{k} A_{i,\ell} \wedge B_{\ell,j}$$

Example of a rank 2 factorization ($k = 2$):

$$\begin{vmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{vmatrix}$$

$$\begin{vmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{vmatrix} \begin{vmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

Constraints :

$$\forall i, j : \bigvee_{\ell=0}^{k} a_{i,\ell} \wedge b_{\ell,j} = M_{i,j}$$

Solution with BMF

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

	Paw Patrol	Shrek	Matrix
Alice	0	1	1
Bob	1	1	1
Charle	1	1	0

Solution with BMF

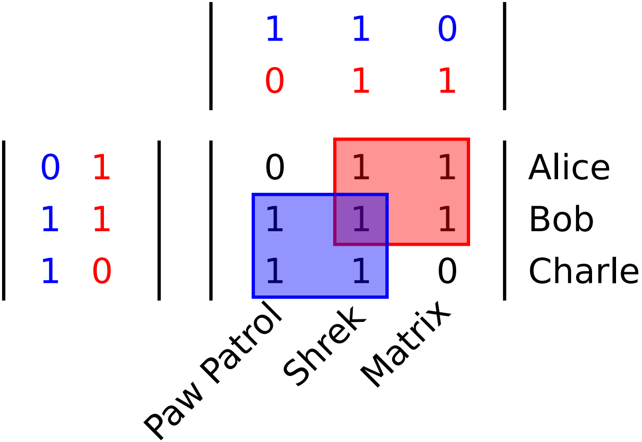
$$
\begin{array}{ccc}
 & \begin{vmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{vmatrix} & \\
\begin{vmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{vmatrix} & \begin{vmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{vmatrix} & \begin{array}{l} \text{Alice} \\ \text{Bob} \\ \text{Charle} \end{array}
\end{array}
$$

Paw Patrol | Shrek | Matrix

Advantages:

- Exact solution of rank 2

Solution with BMF



Advantages:

- Exact solution of rank 2
- Good interpretability of the factorization

Plan

1. Introduction
2. (Max)SAT Encoding
3. Undercover Factorization
4. Block-Optimal Undercover

SAT Encoding

If $m_{i,j} = 0$:

$$
\begin{array}{c|c}
 & \begin{vmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{vmatrix} \\
\begin{vmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{vmatrix} & \begin{vmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}
\end{array}
$$

SAT Encoding

If $m_{i,j} = 0$:

$$\begin{vmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{vmatrix}$$

$$\begin{vmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{vmatrix} \begin{vmatrix} 1 & 1 & \boxed{0} \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

SAT Encoding

$$\text{If } m_{i,j} = 0\text{: } (\neg a_{i,0} \vee \neg b_{0,j})$$

	$b_{0,0}$	$b_{0,1}$	$b_{0,2}$
	$b_{1,0}$	$b_{1,1}$	$b_{1,2}$
$a_{0,0}$ $a_{0,1}$	1	1	0
$a_{1,0}$ $a_{1,1}$	1	1	1
$a_{2,0}$ $a_{2,1}$	0	1	1

SAT Encoding

$$\text{If } m_{i,j} = 0\text{: } (\neg a_{i,0} \vee \neg b_{0,j})$$

$$\begin{array}{c|c} & \begin{vmatrix} b_{0,0} & b_{0,1} & \boxed{b_{0,2}} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{vmatrix} \\ \begin{vmatrix} \boxed{a_{0,0}} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{vmatrix} & \begin{vmatrix} 1 & 1 & \boxed{0} \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix} \end{array}$$

SAT Encoding

If $m_{i,j} = 0$: $(\neg a_{i,0} \vee \neg b_{0,j}) \wedge (\neg a_{i,1} \vee \neg b_{1,j})$

		$b_{0,0}$	$b_{0,1}$	$b_{0,2}$
		$b_{1,0}$	$b_{1,1}$	$b_{1,2}$
$a_{0,0}$	$a_{0,1}$	1	1	0
$a_{1,0}$	$a_{1,1}$	1	1	1
$a_{2,0}$	$a_{2,1}$	0	1	1

SAT Encoding

If $m_{i,j} = 0$: $(\neg a_{i,0} \vee \neg b_{0,j}) \wedge (\neg a_{i,1} \vee \neg b_{1,j})$

$$\begin{array}{cc|ccc} & & b_{0,0} & b_{0,1} & b_{0,2} \\ & & b_{1,0} & b_{1,1} & \boxed{b_{1,2}} \\ \hline a_{0,0} & \boxed{a_{0,1}} & 1 & 1 & \boxed{0} \\ a_{1,0} & a_{1,1} & 1 & 1 & 1 \\ a_{2,0} & a_{2,1} & 0 & 1 & 1 \end{array}$$

SAT Encoding

If $m_{i,j} = 0$: $(\neg a_{i,0} \vee \neg b_{0,j}) \wedge (\neg a_{i,1} \vee \neg b_{1,j})$

If $m_{i,j} = 1$:

$$\begin{array}{cc} & \begin{vmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{vmatrix} \\ \begin{vmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{vmatrix} & \begin{vmatrix} \boxed{1} & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix} \end{array}$$

SAT Encoding

If $m_{i,j} = 0$: $(\neg a_{i,0} \vee \neg b_{0,j}) \wedge (\neg a_{i,1} \vee \neg b_{1,j})$

If $m_{i,j} = 1$: $(a_{i,0} \wedge b_{0,j})$

$$\begin{vmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{vmatrix}$$

$$\begin{vmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{vmatrix} \quad \begin{vmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

SAT Encoding

If $m_{i,j} = 0$: $(\neg a_{i,0} \vee \neg b_{0,j}) \wedge (\neg a_{i,1} \vee \neg b_{1,j})$

If $m_{i,j} = 1$: $(a_{i,0} \wedge b_{0,j}) \vee (a_{i,1} \wedge b_{1,j})$

$$\begin{vmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ \boxed{b_{1,0}} & b_{1,1} & b_{1,2} \end{vmatrix}$$

$$\begin{vmatrix} a_{0,0} & \boxed{a_{0,1}} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{vmatrix} \begin{vmatrix} \boxed{1} & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

SAT Encoding

If $m_{i,j} = 0$: $(\neg a_{i,0} \vee \neg b_{0,j}) \wedge (\neg a_{i,1} \vee \neg b_{1,j})$

If $m_{i,j} = 1$: $(a_{i,0} \wedge b_{0,j}) \vee (a_{i,1} \wedge b_{1,j})$

$$
\begin{array}{cc}
 & \begin{vmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ \boxed{b_{1,0}} & b_{1,1} & b_{1,2} \end{vmatrix} \\
\begin{vmatrix} a_{0,0} & \boxed{a_{0,1}} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{vmatrix} & \begin{vmatrix} \boxed{1} & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}
\end{array}
$$

Or in CNF:

$$\bigvee_{\ell}^{k} T_{i,j}^{\ell} \wedge$$

$$\bigwedge_{i,j,\ell} T_{i,j}^{\ell} \Rightarrow a_{i,\ell} \wedge b_{\ell,j}$$

SAT Encoding

If $m_{i,j} = 0$: $(\neg a_{i,0} \vee \neg b_{0,j}) \wedge (\neg a_{i,1} \vee \neg b_{1,j})$

If $m_{i,j} = 1$: $(a_{i,0} \wedge b_{0,j}) \vee (a_{i,1} \wedge b_{1,j})$

			$b_{0,0}$	$b_{0,1}$	$b_{0,2}$
			$b_{1,0}$	$b_{1,1}$	$b_{1,2}$
$a_{0,0}$	**$a_{0,1}$**		**1**	1	0
$a_{1,0}$	$a_{1,1}$		1	1	1
$a_{2,0}$	$a_{2,1}$		0	1	1

Or in CNF:

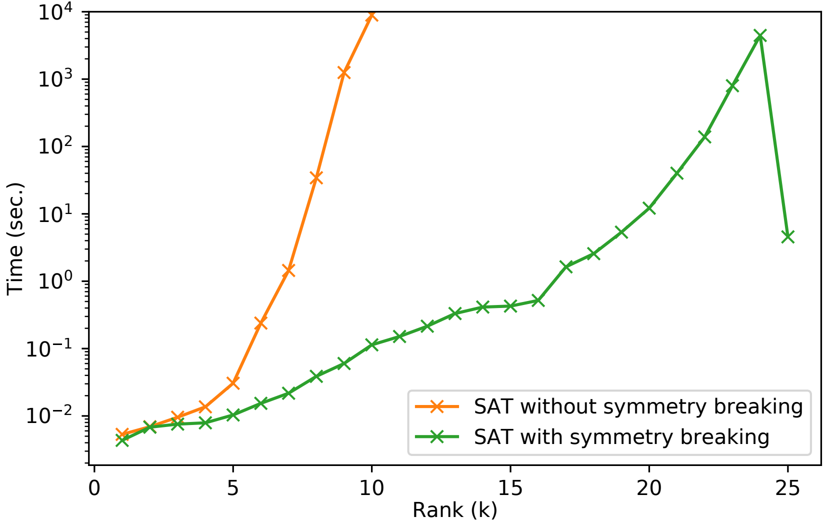
$$\bigvee_{\ell}^{k} T_{i,j}^{\ell} \wedge$$

$$\bigwedge_{i,j,\ell} T_{i,j}^{\ell} \Rightarrow a_{i,\ell} \wedge b_{\ell,j}$$

Symmetry breaking : $(b_{0,0} b_{0,1} \ldots b_{0,j})_{binary} \leq \ldots \leq (b_{k,0} b_{k,1} \ldots b_{k,j})_{binary}$

Benchmark

Execution time to factor the Zoo dataset (101×28) :



MaxSAT Encoding

Si $m_{i,j} = 0 : \neg C_{i,j} \vee ((\neg a_{i,0} \vee \neg b_{0,0}) \wedge (\neg a_{i,1} \vee \neg b_{1,i}))$

Si $m_{i,j} = 1 :$

			$b_{0,0}$	$b_{0,1}$	$b_{0,2}$
			$b_{1,0}$	$b_{1,1}$	$b_{1,2}$
$a_{0,0}$	$a_{0,1}$		1	1	0
$a_{1,0}$	$a_{1,1}$		1	1	1
$a_{2,0}$	$a_{2,1}$		0	1	1

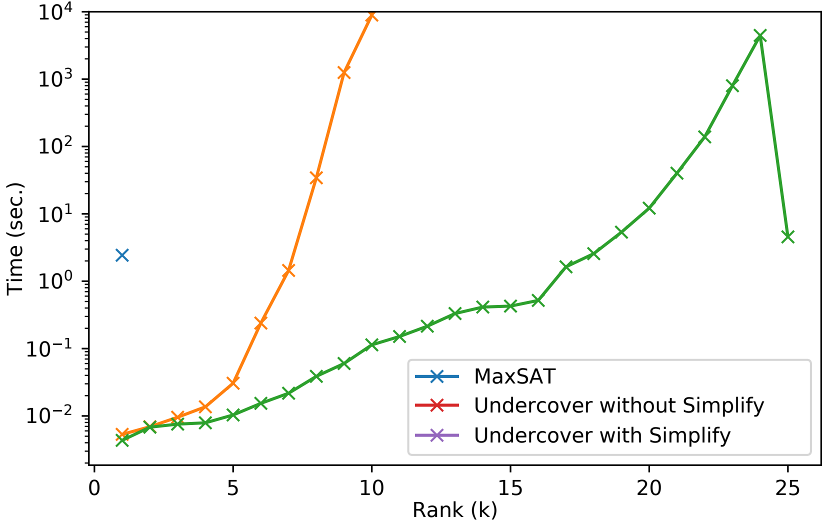
$$(\neg C_{i,j} \vee \bigvee_{\ell}^{k} T^{\ell}_{i,j}) \wedge$$

$$\bigwedge_{i,j,\ell} T^{0}_{i,j} \Rightarrow a_{i,\ell} \wedge b_{\ell,j}$$

$$Max(\sum_{i,j} C_{i,j})$$

Benchmark

Execution time to factor the Zoo dataset (101×28) :



Idea for scaling up: Undercover Factorization

Definition: A matrix M' "undercover" a matrix M ($M' \leq M$) if:

$$\forall i, j : \neg M_{i,j} \Rightarrow \neg M'_{i,j}$$

Idea for scaling up: Undercover Factorization

Definition: A matrix M' "undercover" a matrix M ($M' \leq M$) if:

$$\forall i, j : \neg M_{i,j} \Rightarrow \neg M'_{i,j}$$

Definition: $(A_{m \times k}, B_{k \times n})$ is an optimal k-undercover for M if:

- $A \circ B \leq M$
- For every $(A'_{m \times k}, B'_{k \times n})$ such that $A' \circ B' \leq M$ we have $|A' \circ B'|_1 \leq |A \circ B|_1$

Idea for scaling up: Undercover Factorization

Definition: A matrix M' "undercover" a matrix M ($M' \leq M$) if:

$$\forall i, j : \neg M_{i,j} \Rightarrow \neg M'_{i,j}$$

Definition: $(A_{m \times k}, B_{k \times n})$ is an optimal k-undercover for M if:

- $A \circ B \leq M$
- For every $(A'_{m \times k}, B'_{k \times n})$ such that $A' \circ B' \leq M$ we have $|A' \circ B'|_1 \leq |A \circ B|_1$

Advantages:

- Simpler formulas
- Opportunity to use an iterative approach

Plan

1 Introduction

2 (Max)SAT Encoding

3 Undercover Factorization

4 Block-Optimal Undercover

Undercover Encoding

Si $m_{i,j} = 0$: ~~$\neg C_{i,j} \vee$~~ $((\neg a_{i,0} \vee \neg b_{0,0}) \wedge (\neg a_{i,1} \vee \neg b_{1,i}))$

Si $m_{i,j} = 1$:

		$b_{0,0}$	$b_{0,1}$	$b_{0,2}$
		$b_{1,0}$	$b_{1,1}$	$b_{1,2}$
$a_{0,0}$	$a_{0,1}$	1	1	0
$a_{1,0}$	$a_{1,1}$	1	1	1
$a_{2,0}$	$a_{2,1}$	0	1	1

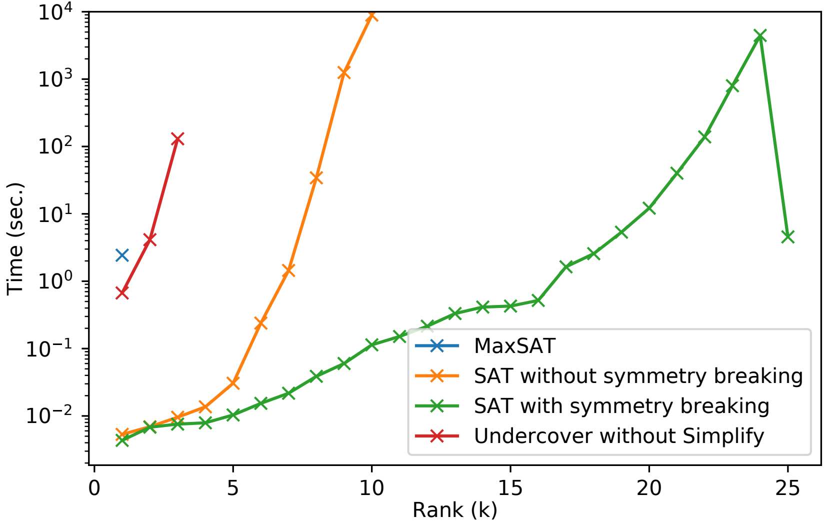
$$(\neg C_{i,j} \vee \bigvee_{\ell}^{k} T_{i,j}^{\ell}) \wedge$$

$$\bigwedge_{i,j,\ell} T_{i,j}^{0} \Rightarrow a_{i,\ell} \wedge b_{\ell,j}$$

$$Max(\sum_{i,j} C_{i,j})$$

Benchmark

Execution time to factor the Zoo dataset (101×28) :



How Most MaxSAT Solvers Work

Input: A SAT formula ϕ and a set of soft clauses $\mathcal{S}$

1: $cost \leftarrow 0$
2: **while** $SAT(\phi \cup \mathcal{S}) = false$ **do**
3: $\quad C \leftarrow unsat\_core(\phi \cup \mathcal{S})$
4: $\quad \mathcal{S} \leftarrow \mathcal{S} \setminus C$
5: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{(\sum_{v \in C} v) \geq |C| - 1\}$
6: $\quad cost \leftarrow cost + 1$
7: **end while**
8: **return** $cost$

How Most MaxSAT Solvers Work

Input: A SAT formula ϕ and a set of soft clauses $\mathcal{S}$

1: $cost \leftarrow 0$
2: **while** $SAT(\phi \cup \mathcal{S}) = false$ **do**
3: $\quad C \leftarrow unsat\_core(\phi \cup \mathcal{S})$
4: $\quad \mathcal{S} \leftarrow \mathcal{S} \setminus C$
5: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{(\sum_{v \in C} v) \geq |C| - 1\}$
6: $\quad cost \leftarrow cost + 1$
7: **end while**
8: **return** $cost$

Remark: A high cost implies a large number of calls to the solver

How Most MaxSAT Solvers Work

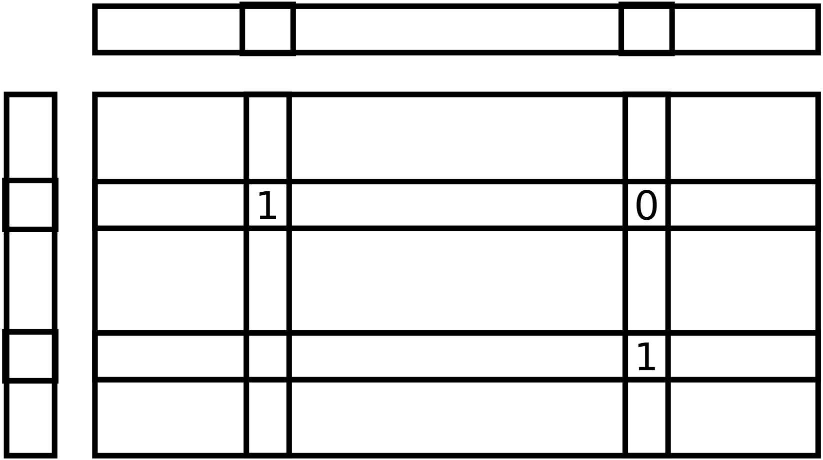
Input: A SAT formula ϕ and a set of soft clauses $\mathcal{S}$

$cost \leftarrow 0$
while $SAT(\phi \cup \mathcal{S}) = false$ **do**
  $C \leftarrow unsat\_core(\phi \cup \mathcal{S})$
  $\mathcal{S} \leftarrow \mathcal{S} \setminus C$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\sum_{v \in C} v) \geq |C| - 1\}$
  $cost \leftarrow cost + 1$
end while
return $cost$

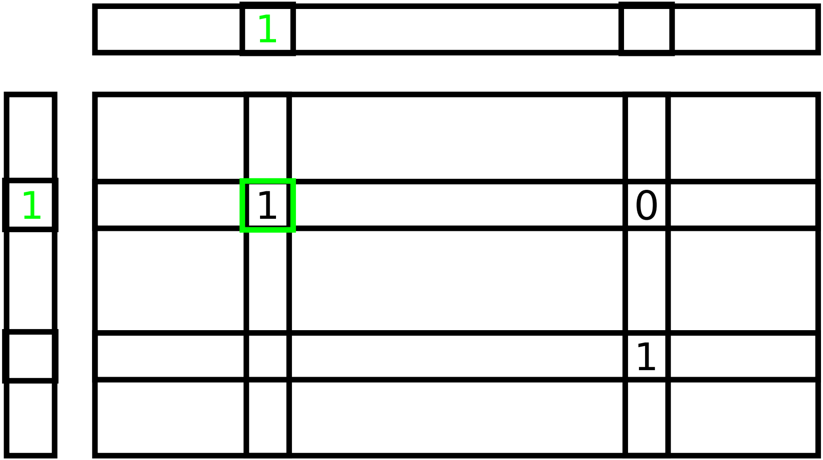
Remark: A high cost implies a large number of calls to the solver

Idea: Do not start with a cost equal to zero (use domain knowledge to find unsat core quickly)

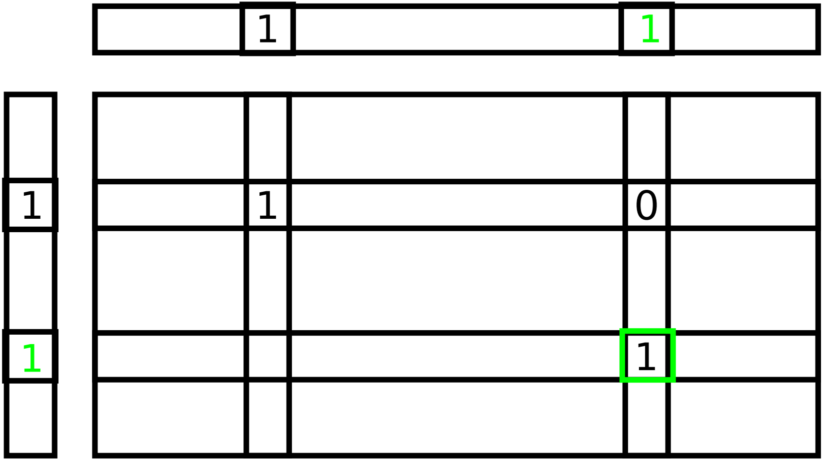
Optimisation



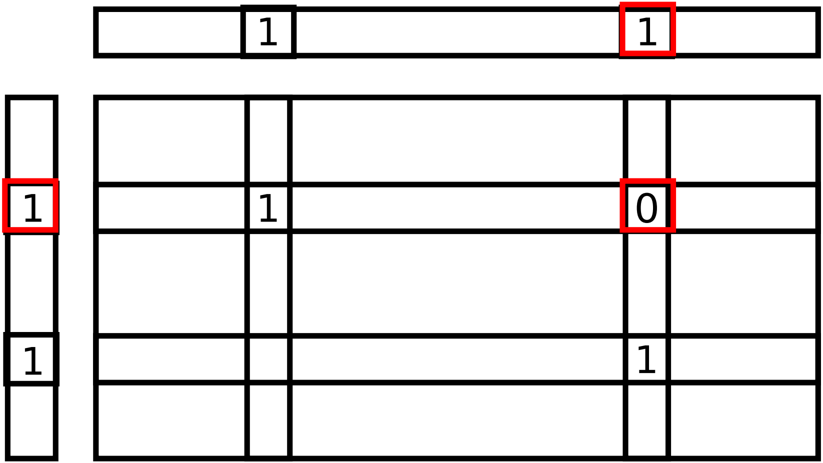
Optimisation



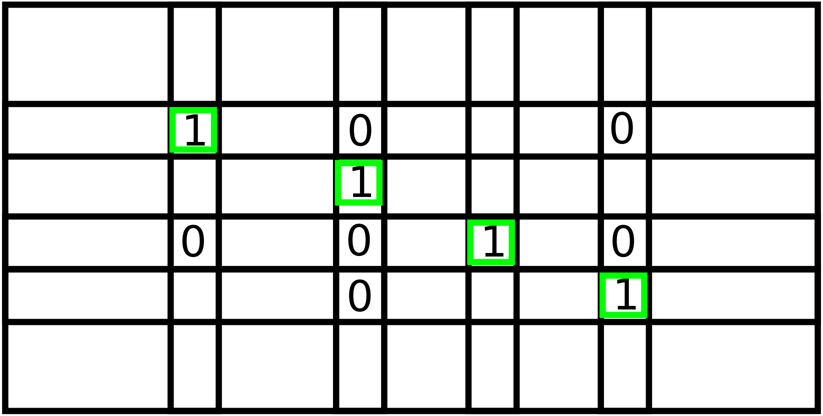
Optimisation



Optimisation

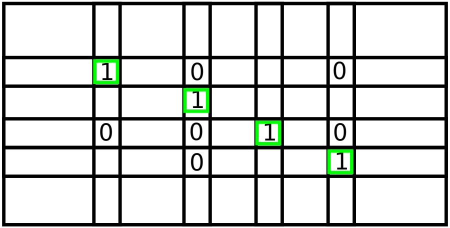


Optimisation



At most k 1s can be covered in a set of $|I|$ 1s two by two incompatible

Optimisation

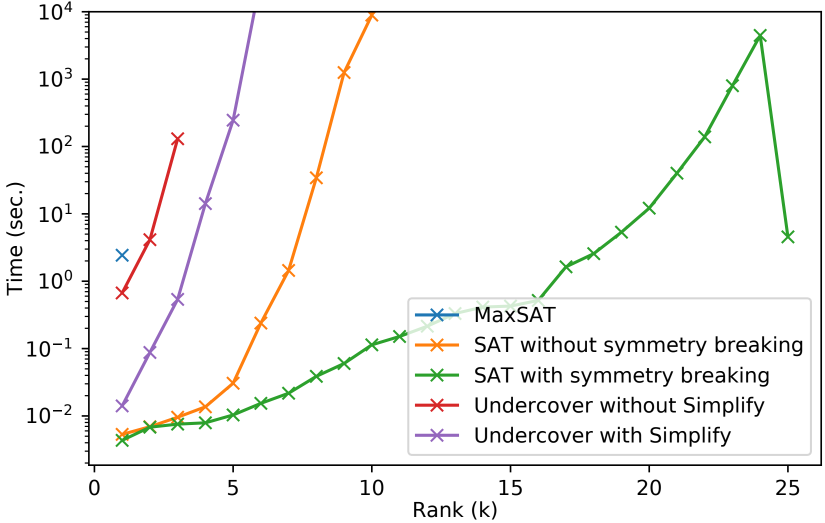


For each set I of 1s, two by two incompatible:

1. $\forall M_{i,j}$, remove $c_{i,j}$ from the soft clauses set.
2. add $(\sum_{M_{i,j} \in I} c_{i,j}) \geq k$ in the soft clauses set.
3. Increment the cost of the MaxSAT formula by $|I| - k$

Benchmark

Execution time to factor the Zoo dataset (101×28):



Plan

1. Introduction
2. (Max)SAT Encoding
3. Undercover Factorization
4. Block-Optimal Undercover

Scaling Up: Incremental Approach

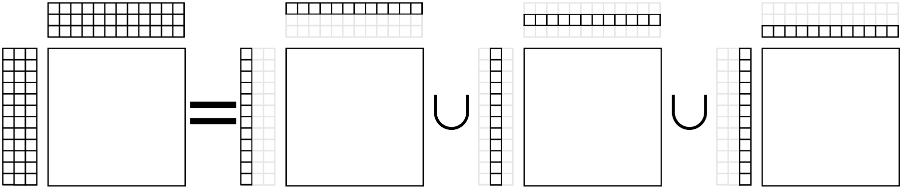
Definition : Two matrices $A_{m \times k}, B_{k \times n}$ are *block-optimal* k-undercover for a matrix M if :

$$\forall p \in [1, k] : (A_{:,p} \circ B_{p,:}) \text{ is an optimal 1-undercovers for } X - \bigcup_{\ell \neq p} (A_{:,\ell} \circ B_{\ell,:})$$

Scaling Up: Incremental Approach

Definition : Two matrices $A_{m\times k}, B_{k\times n}$ are *block-optimal* k-undercover for a matrix M if :

$$\forall p \in [1,k] : (A_{:,p} \circ B_{p,:}) \text{ is an optimal 1-undercovers for } X - \bigcup_{\ell \neq p} (A_{:,\ell} \circ B_{\ell,:})$$



Scaling Up: Incremental Approach

Definition : Two matrices $A_{m\times k}, B_{k\times n}$ are *block-optimal* k-undercover for a matrix M if :

$$\forall p \in [1,k] : (A_{:,p} \circ B_{p,:}) \text{ is an optimal 1-undercovers for } X - \bigcup_{\ell \neq p} (A_{:,\ell} \circ B_{\ell,:})$$

Theorem : A block-optimal k-undercover is a $\frac{1}{2}$-approximation of the problem k-undercover

Scaling Up: Incremental Approach

Definition : Two matrices $A_{m\times k}, B_{k\times n}$ are *block-optimal* k-undercover for a matrix M if :

$$\forall p \in [1,k] : (A_{:,p} \circ B_{p,:}) \text{ is an optimal 1-undercovers for } X - \bigcup_{\ell \neq p} (A_{:,\ell} \circ B_{\ell,:})$$

Theorem : A block-optimal k-undercover is a $\frac{1}{2}$-approximation of the problem k-undercover

Algorithm OptiBlock : Compute an optimal 1-undercover of M and remove the 1s covered by the solution. Iterate k times saving the solutions to build A and B.

Theorem: The algorithm OptiBlock gives a 0.632-approximation

Method	# TOP	Time (min)
OptiBlock	46	544
CG (with timeout)	20	469
MP	18	372
GreConD+	9	107
Tile	9	220
IterEss	9	0.4
k-greedy	5	59
MEBF	0	4.0
Asso	0	96

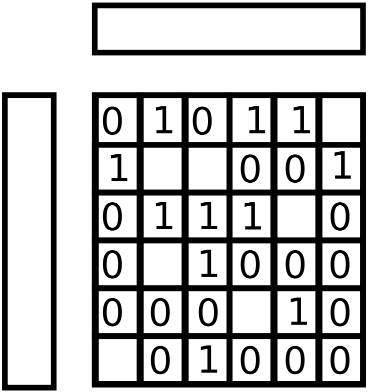
Table 1: Benchmark on 25[1] real data sets with three values of k (75 cases)

[1]Datasets from UCI, namely: Audiology, Autism Screening Adult, Balance Scale, Breast Cancer, Car Evaluation, Chess (King-Rook vs. King), Congressional Voting Records, Contraceptive Method Choice, Dermatology, Hepatitis, Iris, Lung Cancer, Lymphography, Mushroom, Nursery, Primary Tumor, Solar Flare, Soybean (Large), Statlog (Heart), Student Performance, Thoracic Surgery, Tic-Tac-Toe Endgame, Website Phishing, Wine and Zoo.

Initialisation (FastUndercover)

Algorithm:

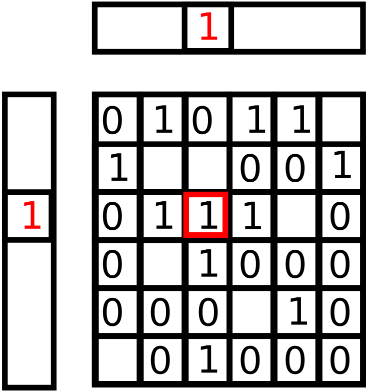
1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

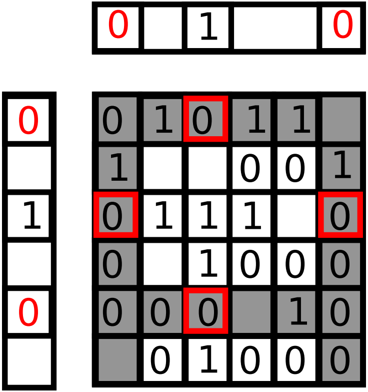
1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

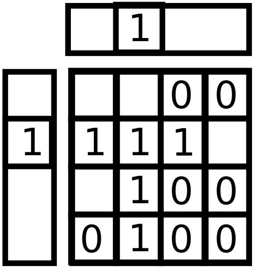
1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

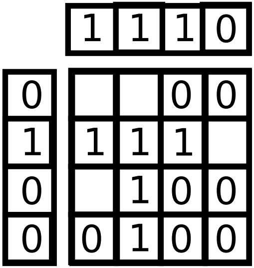
1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

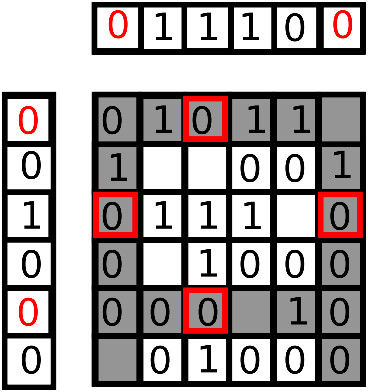
1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

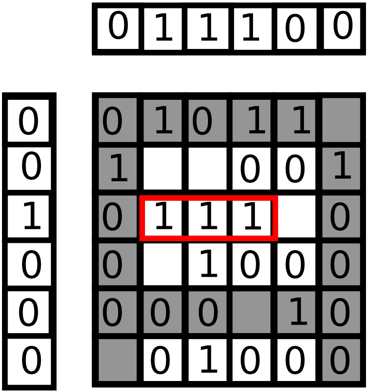
1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

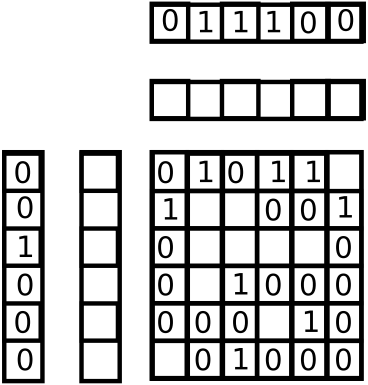
1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

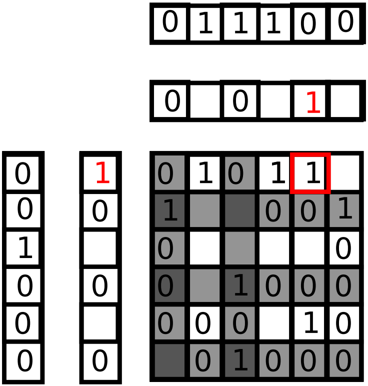
1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

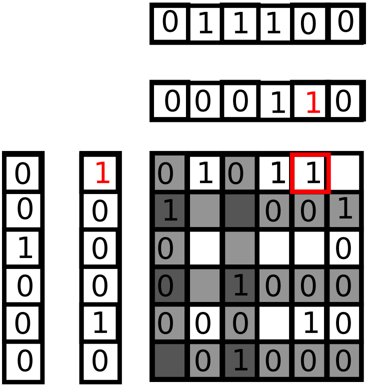
1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

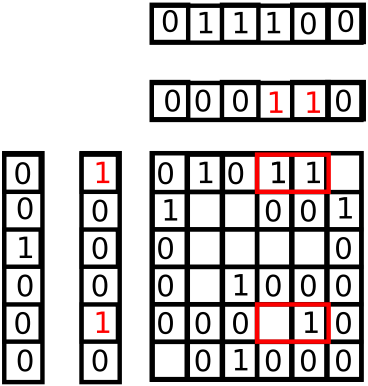
1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times

0	1	1	1	0	0
0	0	0	1	1	0

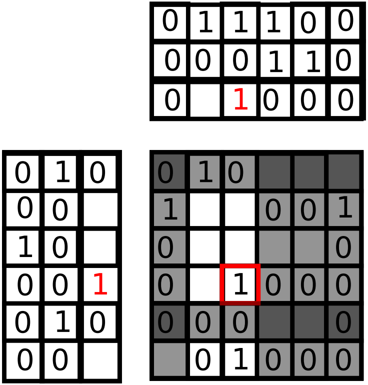
0	1	
0	0	
1	0	
0	0	
0	1	
0	0	

0	1	0			
1			0	0	1
0					0
0		1	0	0	0
0	0	0			0
	0	1	0	0	0

Initialisation (FastUndercover)

Algorithm:

1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times



Initialisation (FastUndercover)

Algorithm:

1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times

0	1	1	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0

0	1	0
0	0	0
1	0	0
0	0	1
0	1	0
0	0	1

0	1	0			
1			0	0	1
0					0
0		1	0	0	0
0	0	0			0
	0	1	0	0	0

Initialisation (FastUndercover)

Algorithm:

1. Choose a "1" to cover
2. Perform a 1-undercover that covers this "1" and remove the covered 1s
3. Repeat k times

0	1	1	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0

0	1	0
0	0	0
1	0	0
0	0	1
0	1	0
0	0	1

0	1	0	1	1	
1			0	0	1
0	1	1	1		0
0		1	0	0	0
0	0	0		1	0
	0	1	0	0	0

Method	# TOP	Time (min)
OptiBlock*	**45**	98
OptiBlock	29	544
CG (with timeout)	19	469
MP	14	372
GreConD+	9	107
FastUndercover	8	3.5
Tile	7	220
IterEss	6	0.4
k-greedy	5	59
MEBF	0	4.0
Asso	0	96

Table 2: Benchmark on 25[2] real data sets with three values of k (75 cases)

[2]Datasets from UCI, namely: Audiology, Autism Screening Adult, Balance Scale, Breast Cancer, Car Evaluation, Chess (King-Rook vs. King), Congressional Voting Records, Contraceptive Method Choice, Dermatology, Hepatitis, Iris, Lung Cancer, Lymphography, Mushroom, Nursery, Primary Tumor, Solar Flare, Soybean (Large),

Conclusion

- Optimal factorization is possible for small matrices, but scaling up is complicated.

Conclusion

- Optimal factorization is possible for small matrices, but scaling up is complicated.
- Propose a performance guarantee algorithm that finds undercover factorization.

Conclusion

- Optimal factorization is possible for small matrices, but scaling up is complicated.
- Propose a performance guarantee algorithm that finds undercover factorization.
- Thanks to a MaxSAT encoding and some optimizations, the algorithm can work on large matrix.

Conclusion

- Optimal factorization is possible for small matrices, but scaling up is complicated.
- Propose a performance guarantee algorithm that finds undercover factorization.
- Thanks to a MaxSAT encoding and some optimizations, the algorithm can work on large matrix.
- The quality of the factorizations generated by our algorithm gave on average better results than the classical algorithms of the literature in our experiments.