From-Below Approximations in Boolean Matrix Factorization: Geometry and New Algorithm II

Radim Belohlavek, Martin Trnecka

DAMOL
DATA ANALYSIS AND MODELING LAB
Palacky University, Olomouc, Czech Republic

european social fund in the czech republic
EUROPEAN UNION
MINISTRY OF EDUCATION, YOUTH AND SPORTS
OP Education for Competitiveness
INVESTMENTS IN EDUCATION DEVELOPMENT

List of Algorithms for Boolean Matrix Factorization (BMF)

- Algorithm 1
- GreConD
- ASSO
- Tiling (LTM algorithm)
- PaNDa
- Hyper (Hyper+)

ASSO Algorithm in Short

- Author: Pauli Miettinen from Max-Planck-Institut für Informatik, originally from Finland.
- First introduced in: The Discrete Basis Problem (2006), MSC thesis as Association algorithm.
- Probably currently the most discussed BMF algorithm in the data mining literature.

ASSO Algorithm Pseudocode

Input: A matrix $\mathcal{C} \in \{0,1\}^{n \times m}$ for data, a positive integer k, a threshold value $\tau \in [0,1]$, and real-valued weights w^+ and w^-.

for $i = 1, \ldots, m$ **do**

  `Construct matrix` $\mathcal{A}$ `row by row`

  $a_i \leftarrow (1(c(i \Rightarrow j, \mathcal{C}) \geq \tau))_{j=1}^{m}$

end

$\mathcal{B}$ `and` $\mathcal{S}$ `are empty matrices`

$\mathcal{B} \leftarrow []$, $\mathcal{S} \leftarrow []$

`Select the` k `basis vectors from` $\mathcal{A}$

for $l = 1, \ldots, k$ **do**

  $(a_i, s) \leftarrow \max_{a_i, s \in \{0,1\}^{n \times 1}} \mathit{cover}(\begin{bmatrix} \mathcal{B} \\ a_i \end{bmatrix}, [\mathcal{S}\ s], \mathcal{C}, w^+, w^-)$

  $\mathcal{B} \leftarrow \begin{bmatrix} \mathcal{B} \\ a_i \end{bmatrix}$, $\mathcal{S} \leftarrow [\mathcal{S}\ s]$

end

return $\mathcal{B}$ a $\mathcal{S}$

ASSO Continuation

- $cover(\mathcal{B}, \mathcal{S}, \mathcal{C}, w^+, w^-)$ is equal to:

$$w^+ \left|\{(i,j) : c_{ij} = 1, (\mathcal{S} \circ \mathcal{B})_{ij} = 1\}\right| - w^- \left|\{(i,j) : c_{ij} = 0, (\mathcal{S} \circ \mathcal{B})_{ij} = 1\}\right|.$$

- Unfortunately it is more complicated.
- Vector s is compute from a_i in greedy manner: $cover$ function is compute separately for each row of input matrix C, $s_j = 1$ if value of $cover > 0$ for row C_j.
- Take to account covered part (not mentioned in pseudocode).
- $cover$ function does not count covered part.

Example (ASSO Algorithm, Miettinen's implementation - First Factor)

$k = 3, \tau = 0.9, w^+ = 1, w^- = 1$

$$C = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}, covered = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$cover$ for the first row: -2 + 4 + 2 + 0 = 6
$cover$ for the second row: 0 + 2 + 2 + 2 = 6
$cover$ for the third row: 1 + 1 + 1 + 1 = 4
$cover$ for the fourth row: -1 + 3 + 3 + 1 = **7**

$$\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \circ \begin{pmatrix} 0 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

Example (ASSO Algorithm, Miettinen's implementation - Second Factor)

$k = 3, \tau = 0.9, w^+ = 1, w^- = 1$

$$C = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}, covered = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

$cover$ for the first row: -2 + 1 - 1 - 1 = 1
$cover$ for the second row: 0 + 0 + 0 + 0 = 0
$cover$ for the third row: 1 + 0 + 0 + 0 = **1**
$cover$ for the fourth row: -1 + 0 + 0 + 0 = 0

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} \circ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

Example (ASSO Algorithm, Miettinen's implementation - Third Factor)

$k = 3, \tau = 0.9, w^{+} = 1, w^{-} = 1$

$$C = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}, covered = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

$cover$ for the first row: -3 + 1 + 0 - 1 = **1**
$cover$ for the second row: -1 + 0 + 0 + 0 = 0
$cover$ for the third row: 0 + 0 + 0 + 0 = 0
$cover$ for the fourth row: -2 + 0 + 0 + 0 = 0

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \circ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

ASSO Algorithm Remark

There exist several variants of ASSO:

- ASSO + transpose
- ASSO + clustering
- ASSO + exhaustive search
- ASSO + iterative search

- One specific feature: **There is no way how to remove E_o!**
- Empirical time complexity: 4-5x slower than GreConD.

Tiling Algorithm in Short

- First introduced in: Tiling Databases (2004).
- Originally called LTM algorithm.
- Base on idea from CHARM, LPMiner and BAMBOO algorithm.
- Introduced notion of tile $\tau(I,D) = \{(tid,i) \in cover(I,D), i \in I\}$.
- Tiles can be seen as rectangles in binary data.

Tiling Algorithm Pseudocode

Input: Binary database D, minimal size of tile δ, itemset I initially called with $I = \emptyset$.

$T \leftarrow \emptyset$
$D \leftarrow \text{PRUNE}(D, \delta, I)$
forall the i *occurring in* D **do**
  if $|cover(\{i\})| \cdot (|I| + 1) \geq \delta$ **then**
    $T \leftarrow T \cup \tau(I \cup \{i\})$
  end
  $D^i \leftarrow \emptyset$
  forall the j *occurring in* D *such that* $j > i$ **do**
    $C \leftarrow cover(\{i\}) \cap cover(\{j\})$
    Add (j, C) to D^i
    Recursively compute for $I \cup \{i\}$
  end
end

PRUNE Procedure Pseudocode

Input: D, δ, I.

repeat
  forall the i *occurring in* D **do**
    if $UB_{I \cup \{i\}} < \delta$ **then**
      Remove i from D
    end
    forall the $tid \in cover(\{i\})$ **do**
      if $size(tid) < ML_{I \cup \{i\}}$ **then**
        Remove tid from $cover(\{i\})$
      end
    end
  end
until *nothing change*

Explain Prune Procedure

$UB_{I\cup\{i\}} = max\{(|I| + l) \cdot supp_{\geq l}(i, D^I) \mid l \in \{1, 2, \dots\}\}$

- If $UB_{I\cup\{i\}} < \delta$ then remove item (column) i from D.
- Upper bound of the largest possible tile containing $I \cup \{i\}$.

$ML_{I\cup\{i\}} = min\{l \mid l \cdot supp_{\geq l}(i, D^I) \geq \delta\}$

- If $size(tid) < ML_{I\cup\{i\}}$ then remove transaction (row) tid from D.
- Minimal size of transaction containing i, that can still generate a tile with area at least δ.

Tiling Algorithm

- Produces all tiles with support $> \delta$.
- Easily can be change to find top-k largest tiles.
- Can be modified to solve (approximately) Minimum tiling problem (completely cover input database with the smallest number of tiles).
- We must modify algorithm cost function - it must take into account already covered area.
- In each step is extracted tile with the highest still uncovered part.
- Step $=$ run whole Tiling algorithm!

Tiling Algorithm Remarks

- Modification of Tiling algorithm for BMF, is very slow!
- Produce the same results as Algorithm 1 (in special settings, fix order of tiles).
- In normal case there is more tiles with the same uncovered part - leads to the clash of decision function for selecting tiles.

PaNDa Algorithm in Short

- First introduced in: Mining Top-K Patterns from Binary Datasets in Presence of Noise.
- Basic idea: Select noise-free pattern and extend this pattern with some noise.
- Notion of pattern $P = \langle P_T, P_I \rangle$ in binary database, where $P_T \in \{0,1\}^n, P_I \in \{0,1\}^m$.
- Pattern can be seen as rectangle in binary database.
- Compute the set of patterns Π with the smallest cost.
- Cost function of PaNDa algorithm for set of patterns Π and database $\mathcal{D}$:

$$\gamma(\Pi, \mathcal{D}) = \sum_{P \in \Pi} \gamma_P(P_T, P_I) + \gamma_N(N)$$

where $\gamma_P(P_T, P_I) = |P_T| + |P_I|$ and $\gamma_N(N) = |N|$.

PaNDa Algorithm Pseudocode

Input: $\mathcal{D}$ binary database, k number of extracted patterns

$\Pi \leftarrow \emptyset$
$\mathcal{D}_{\mathcal{R}} \leftarrow \mathcal{D}$
for $i = 1 \ldots k$ **do**
  $C, E \leftarrow$ FIND-CORE($\mathcal{D}_{\mathcal{R}}, \Pi, \mathcal{D}$)
  $C^+ \leftarrow$ EXTEND-CORE($C, E, \Pi, \mathcal{D}$)
  if $\gamma(\Pi, \mathcal{D}) \leq \gamma(\Pi \cup C^+, \mathcal{D})$ **then**
    break
  end
  $\Pi \leftarrow \Pi \cup C^+$
  $\mathcal{D}_{\mathcal{R}}(i, j) \leftarrow 0 \; \forall i, j$ s.t. $C^+_T(i) = 1 \wedge C^+_I(i) = 1$
end
return Π

Core Pattern Discovery, FIND-CORE Procedure

Input: $\mathcal{D}_{\mathcal{R}}, \Pi, \mathcal{D}$

$E \leftarrow \emptyset$
$S = \{s_1, \ldots, s_m\} \leftarrow$ SORT-ITEMS-IN-DB($\mathcal{D}_{\mathcal{R}}$)
$C \leftarrow \langle C_T = 0^n, C_I = 0^m \rangle$
$C_I(s_1) \leftarrow 1$
$C_T(i) \leftarrow 1 \ \forall i$ s.t. $\mathcal{D}_{\mathcal{R}}(i, s_1) = 1$
for $h = 2, \ldots, m$ **do**
    $C^* \leftarrow C$
    $C^*_I(s_h) \leftarrow 1$
    $C^*_T(i) \leftarrow 0 \ \forall i$ s.t. $\mathcal{D}_{\mathcal{R}}(i, s_h) = 0$
    if $\gamma(\Pi \cup C^*, \mathcal{D}) \leq \gamma(\Pi \cup C, \mathcal{D})$ **then**
        $C \leftarrow C^*$
    else
        E.append(s_h)
    end
end
return C, E

Extension of Core Pattern, EXTEND-CORE Procedure

Input: $C, E, \Pi, \mathcal{D}$

```
while E ≠ ∅ do
    e ← E.pop()
    C* ← C
    C*_I(e) ← 1
    if γ(Π ∪ C*, D) ≤ γ(Π ∪ C, D) then
        C ← C*
    end
    for i = 1, ..., n s.t. C*_T(i) = 0 do
        C* ← C
        C*_T(i) ← 1
        if γ(Π ∪ C*, D) ≤ γ(Π ∪ C, D) then
            C ← C*
        end
    end
end
return C
```

PaNDa Remarks

- Sorting items in database:
  - Frequency (best results)
  - Correlation
  - Prefix-tree
  - Couples frequency
- Empirical time complexity: 5x slower than GreConD.

Hyper Algorithm

- First introduced in: Summarizing Transactional Databases with Overlapped Hyperrectangles (2011).
- Notion of hyperrectangle $H = T \times I = \{(i,j) : i \in T, j \in I\}$.
- T is subset of all transactions and I is subset of all items.
- Hyperrectangle can be seen as rectangle in binary database.
- Price of hyperrectangle H_i:

$$\gamma(H_i) = \frac{|T_i| + |I_i|}{|T_i \times I_i \setminus R|}$$

where R is already covered part of input database.

HYPER Algorithm Pseudocode

Input: DB binary database, C_α set of frequent itemsets with minimal support α union with singleton sets of itemsets (sets with only one item)

```
R ← ∅
CBD ← ∅
while R ≠ DB do
    forall the H_i ∈ C_α do
        X_i ← FIND-HYPER(H_i, R)
    end
    H' ← min_{X_i} γ(X_i)
    R ← R ∪ H'
end
return CDB
```

Procedure for Finding Sub-hyperrectangle with Cheapest Price

Input: Hyperrectangle H, already covered part of input database R

forall the $S = \{t_j\} \times I_i \subseteq H$ **do**
    calculate the number of uncovered DB elements in S, $|S \setminus R|$
end
sort S according to $|S \setminus R|$ and put in U
$H' \leftarrow$ first hyperrectangle S popped from U
while $U \neq \emptyset$ **do**
    pop single transaction hyperrectangle S prom U
    if $\gamma(H' \cup S) > \gamma(H')$ **then**
        break
    else
        $H' \leftarrow H' \cup S$
    end
end
return H'

Example (HYPER Finding Sub-hyperrectangle with Cheapest Price)

$$H = \begin{array}{c|cccc} & i_2 & i_4 & i_5 & i_7 \\ t_1 & 1 & 1 & 1 & 1 \\ t_3 & 1 & 1 & 1 & 1 \\ t_4 & 1 & 1 & 1 & 1 \\ t_6 & 1 & 1 & 1 & 1 \\ t_8 & 1 & 1 & 1 & 1 \\ t_9 & 1 & 1 & 1 & 1 \end{array}, \quad R = \begin{array}{c|cccc} & i_2 & i_4 & i_5 & i_7 \\ t_1 & 0 & 1 & 0 & 1 \\ t_3 & 1 & 0 & 1 & 1 \\ t_4 & 0 & 0 & 0 & 0 \\ t_6 & 0 & 1 & 0 & 1 \\ t_8 & 0 & 0 & 0 & 0 \\ t_9 & 1 & 0 & 1 & 1 \end{array}$$

Sort transaction according to size of their uncovered part: $t_4, t_8, t_1, t_6, t_3, t_9$.

$H' = t_4 \times I, \gamma(H') = \frac{4+1}{4} = 1.25$
add $t_8 \times I$ to $H', \gamma(H') = \frac{4+2}{4+4} = 0.75$ decrease
add $t_1 \times I$ to $H', \gamma(H') = \frac{4+3}{4+4+4-2} = 0.70$ decrease
add $t_6 \times I$ to $H', \gamma(H') = \frac{4+4}{4+4+4+4-2-2} = 0.67$ decrease
add $t_3 \times I$ to $H', \gamma(H') = \frac{4+5}{4+4+4+4+4-2-2-3} = 0.69$ increase - stop

Final hyperrectangle $H' = t_4, t_8, t_1, t_6 \times I$.

Hyper Algorithm Remark

- In basic setting C_α is from Apriori algorithm.
- Variants of Hyper algorithm:
  - Maximal frequent itemset.
  - Frequent closed itemset (best results).
- Prune technique can accelerate HYPER algorithm.
- Empirical time complexity: it depends on size C_α for “reasonable” size is 4-5x slower than GreConD.
- Exact factorization (there is no E_o error BMF).
- Hyper+ approximate factorization (merges hyperrectangles, the merged ones contain 0s).
- Hyper+ is post-processing algorithm.

HYPER+ Algorithm Pseudocode

Input: Binary database DB, set of hyperrectangles CBD, final number of hyperrectangles β

$SCDB \leftarrow CBD$

while $\frac{|SCDB^C| \setminus |DB|}{|DB|} \leq \beta$ **do**

find $H_i, H_j \in SCDB$ whose merge is within the false positive budget

$$\frac{|(SCDB \setminus \{H_i, H_j\} \cup \{H_i \oplus H_j\})^C \setminus DB|}{|DB|} \leq \beta$$

and produce the maximum saving-false positive ratio

$$\frac{|T_i| + |T_j| + |I_i| + |I_j| - |T_i \cup T_j| - |I_i \cup I_j|}{|(H_i \oplus H_j) \setminus SCDB^C|}$$

$SCBD \leftarrow SCBD \setminus \{H_i, H_j\}$

$SCBD \leftarrow SCBD \cup \{H_i \oplus H_j\}$

end

return $SCDB$

Noise in Binary Data

- Big topic in Data Minig in general.
- Current consideration regarding noise in binary data seem not mature (ad hoc).
- For example: in work about ASSO algorithm - 40% of noise (means practically new data).
- It is reasonable to take it as some kind of error.

Experiments Scenario

- We use various random and real datasets.
- We compare GreEss (pessimistic estimate), GreConD, ASSO (basic), Tiling, PaNDa, and Hyper (frequent closed itemsets) algorithm.
- Error:

$$
\begin{aligned}
E(I, A \circ B) &= E_u(I, A \circ B) + E_o(I, A \circ B), \text{where} \\
E_u(I, A \circ B) &= |\{\langle i, j \rangle \,;\, I_{ij} = 1, (A \circ B)_{ij} = 0\}|, \\
E_o(I, A \circ B) &= |\{\langle i, j \rangle \,;\, I_{ij} = 0, (A \circ B)_{ij} = 1\}|.
\end{aligned}
$$

- We are interested in from-below approximation of binary data.
- For each algorithm we compute coverage quality for each obtained factor.
- Coverage quality:

$$c = 1 - \frac{E_u(I, A \circ B)}{||I||}$$

Synthetic Data - Randomly Generated Matrices

- It is senseless compute Boolean matrix factorization on fully random data (there are no rational factors).
- We compute random binary matrices $n \times k$ and $k \times m$.
- Boolean product of these matrices represent our synthetic data with size $n \times m$.
- Such matrix is decomposable with (at most) k factors.

Synthetic Data

dataset	size	basis	dens A	dens B
Set 1	300×100	20	0.10	0.10
Set 2	500×250	20	0.05	0.05
Set 3	500×250	20	0.10	0.05
Set 4	500×250	30	0.12	0.12
Set 5	1000×500	50	0.10	0.10
Set 6	10000×1000	50	0.10	0.10

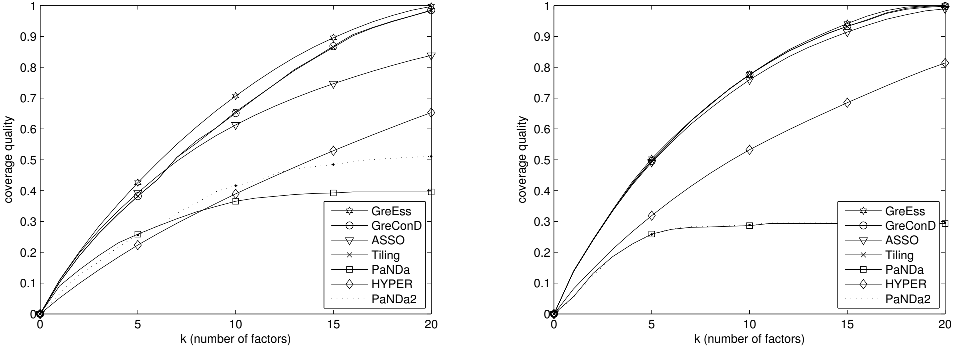
Table: Synthetic datasets $I = A \circ B$

Statistics on Synthetic Data

dataset	avg $\|\|I\|\|$	avg $\|\|\mathcal{E}(I)\|\|$	avg $\|\|\mathcal{E}(I)\|\|/\|\|I\|\|$
Set 1	5039	2764	0.549
Set 2	11966	6412	0.536
Set 3	22841	5207	0.228
Set 4	44027	8894	0.202
Set 5	195990	34005	0.174
Set 6	3907433	47359	0.012

Table: Essential part of I (synthetic data)

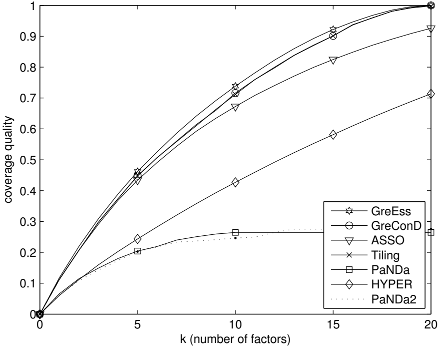
Results for Synthetic Data I.



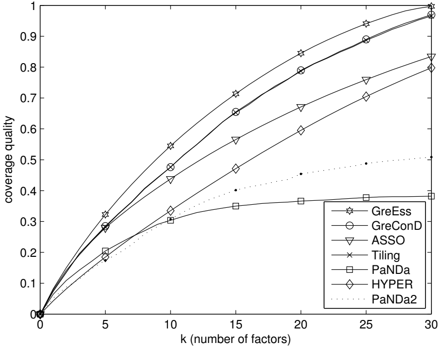
(a) Set 1 (b) Set 2

Figure: Coverage by the first k factors (synthetic data)

Results for Synthetic Data II.



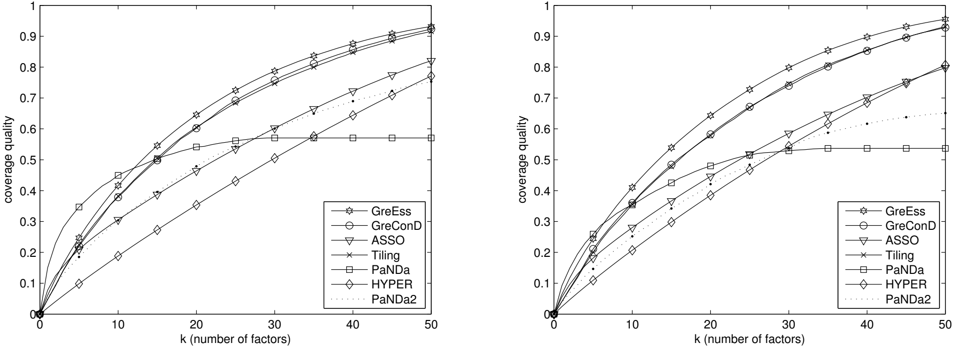
(a) Set 3



(b) Set 4

Figure: Coverage by the first k factors (synthetic data)

Results for Synthetic Data III.



(a) Set 5 (b) Set 6

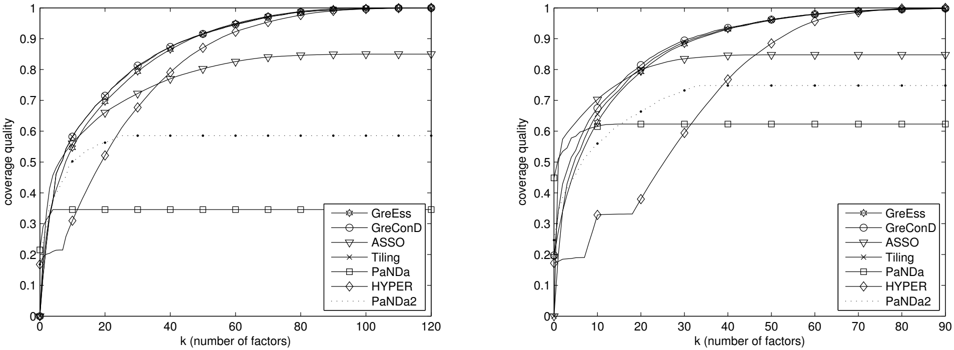
Figure: Coverage by the first k factors (synthetic data)

Real Datasets

dataset	size	$\|\|I\|\|$	$\|\|\mathcal{E}(I)\|\|$	$\|\|\mathcal{E}(I)\|\|/\|\|I\|\|$
Mushroom	8124×119	186852	82965	0.444
DBLP	19×6980	40637	1601	0.039
Paleo	501×139	3537	1906	0.539
Chess	3196×76	118252	71296	0.603
DNA	4590×392	26527	1685	0.064

Table: Essential part of I (real data)

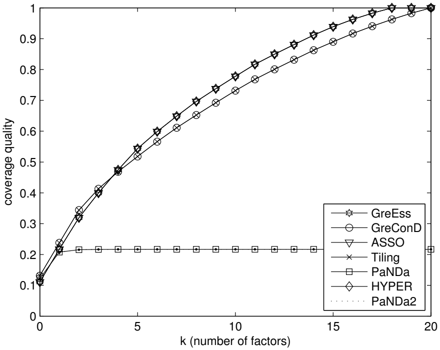
Results for Real Datasets I.



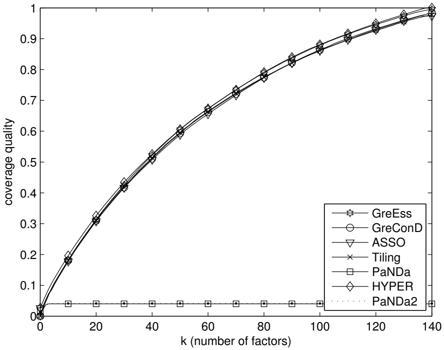
(a) Mushroom (b) Chess

Figure: Coverage by the first k factors (real data)

Results for Real Datasets II.



(a) DBLP



(b) Paleo

Figure: Coverage by the first k factors (real data)

Results for Real Datasets III.

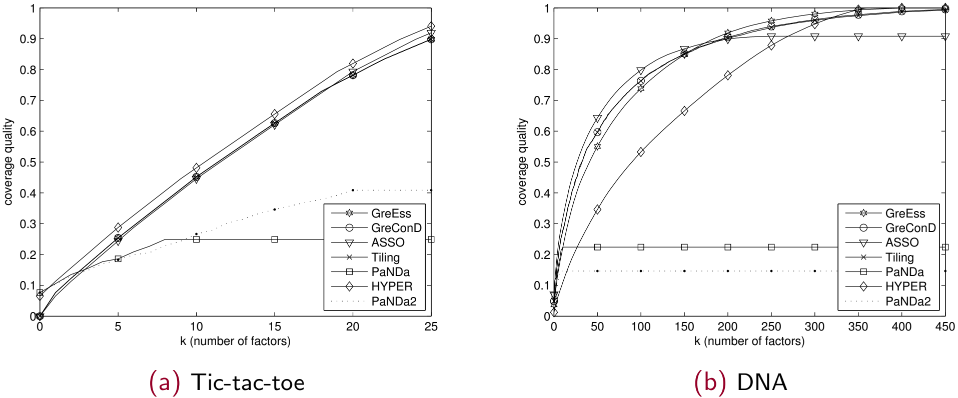


Figure: Coverage by the first k factors (real data)

dataset	coverage ($100c\%$)	Tiling	number of factors required for the prescribed coverage ASSO	GreConD	PaNDa	PaNDa2	Hyper	GreEss
Mushroom	25%	3	2	3	1	2	8	2
	50%	7	6	7	NA	10	19	8
	75%	24	36	24	NA	NA	37	26
	100%	119	NA	120	NA	NA	122	105
DBLP	25%	2	2	2	NA	NA	2	2
	50%	5	5	5	NA	NA	5	5
	75%	10	10	11	NA	NA	10	10
	100%	21	19	20	NA	NA	19	19
Paleo	25%	16	16	16	NA	NA	14	15
	50%	39	40	39	NA	NA	38	38
	75%	75	76	76	NA	NA	73	73
	100%	151	NA	152	NA	NA	139	145
Chess	25%	2	1	1	1	1	9	1
	50%	5	2	4	NA	7	26	6
	75%	16	15	15	NA	NA	39	17
	100%	124	NA	124	NA	NA	90	113
DNA	25%	8	6	8	NA	NA	24	13
	50%	32	27	33	NA	NA	67	41
	75%	94	80	96	NA	NA	155	105
	100%	489	NA	496	NA	NA	392	408
Tic-tac-toe	25%	5	6	5	NA	10	5	5
	50%	12	12	12	NA	NA	11	12
	75%	19	19	19	NA	NA	18	19
	100%	31	29	32	NA	NA	29	32

Conclusion

- GreEss algorithm outperform existing algorithms in both number of factors and coverage quality.
- Empirical time complexity: GreEss is 2x slower than GreConD algorithm.
- Extracted factors seem to be more reasonable than factors from different algorithms (more detailed study is needed for support this claim).