ICLP 2020 - *The 36th International Conference on Logic Programming*

Enhancing Linear Algebraic Computation of Logic Programs Using Sparse Representation

Tuan Nguyen Quoc, Katsumi Inoue, Chiaki Sakama

September 16, 2020

Outline

Outline

Representation of Logic Programs in Vector Space

Recently, there are many attempts[1,2,3,4] to tensorize logic programs into matrix representation

- Tensorizing logic programs is the method of encoding the logic programs as tensors (multidimensional array) then using algebraic computation to do logic operators.
- Approaches may different but they share the same idea: determine the vector space of standardized logic programs and then convert to matrices with some elimination strategies.

A *standardized program* is a finite set of rules that are either AND-rule or OR-rule.

$$h \leftarrow b_1 \wedge \cdots \wedge b_m \quad (m \geq 0)$$

$$h \leftarrow b_1 \vee \cdots \vee b_m \quad (m \geq 0)$$

[1] Sakama, Inoue, and Sato, "Linear Algebraic Characterization of Logic Programs", 2017.
[2] Nguyen et al., "Computing Logic Programming Semantics in Linear Algebra", 2018.
[3] Sakama et al., *Partial Evaluation of Logic Programs in Vector Spaces*, EasyChair, 2018.
[4] Kojima and Sato, "A tensorized logic programming language for large-scale data", 2019.

Representation of Logic Programs in Vector Space

Definition 1 (Matrix representation of standardized programs)

Let P be a standardized program and $B_P = \{p_1, \ldots, p_n\}$. Then P is represented by a matrix $M_P \in \mathbb{R}^{n \times n}$ such that for each element a_{ij} $(1 \le i, j \le n)$ in M_P,

1. $a_{ij_k} = \frac{1}{m}$ $(1 \le k \le m;\ 1 \le i, j_k \le n)$ if $p_i \leftarrow p_{j_1} \wedge \cdots \wedge p_{j_m}$ is in P;
2. $a_{ij_k} = 1$ $(1 \le k \le l;\ 1 \le i, j_k \le n)$ if $p_i \leftarrow p_{j_1} \vee \cdots \vee p_{j_l}$ is in P;
3. $a_{ii} = 1$ if $p_i \leftarrow$ is in P;
4. $a_{ij} = 0$, otherwise.

Example 1:
$P = \{p \leftarrow q \wedge r,\ p \leftarrow s \wedge t,\ r \leftarrow s,\ q \leftarrow t,\ s \leftarrow,\ t \leftarrow\}$

Transform P to P'

$P' = \{u \leftarrow q \wedge r,\ v \leftarrow s \wedge t,\ p \leftarrow u \vee v,\ r \leftarrow s,\ q \leftarrow t,\ s \leftarrow,\ t \leftarrow\}$

$$
\begin{array}{c|ccccccc}
 & p & q & r & s & t & u & v \\
\hline
p & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
q & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
r & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
s & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
t & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
u & 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\
v & 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0
\end{array}
$$

Representation of Logic Programs in Vector Space

Definition 2 (**Matrix representation of normal programs**)

Let P be a normal program and $B_P = \{p_1, \ldots, p_n\}$ and its *positive form* P^+ with
$B_{P^+} = \{p_1, \ldots, p_n, \overline{q}_{n+1}, \ldots, \overline{q}_m\}$.
Then P^+ is represented by a matrix $M_P \in \mathbb{R}^{m \times m}$ such that for each element a_{ij} $(1 \leq i, j \leq m)$:

1. $a_{ii} = 1$ for $n+1 \leq i \leq m$;
2. $a_{ij} = 0$ for $n+1 \leq i \leq m$ and $1 \leq j \leq m$ such that $i \neq j$;
3. Otherwise, a_{ij} $(1 \leq i \leq n;\ 1 \leq j \leq m)$ is encoded as in Definition 1.

Example 2:
$P = \{p \leftarrow q \wedge s,\ q \leftarrow p \wedge t,$
$s \leftarrow \neg t,\ t \leftarrow,\ u \leftarrow v\}$

Transform P to P^+

$P^+ = \{p \leftarrow q \wedge s,\ q \leftarrow p \wedge t,$
$s \leftarrow \overline{t},\ t \leftarrow,\ u \leftarrow v\}.$

	p	q	s	t	u	v	$\overline{t}$
p	0	1/2	1/2	0	0	0	0
q	1/2	0	0	1/2	0	0	0
s	0	0	0	0	0	0	1
t	0	0	0	1	0	0	0
u	0	0	0	0	0	1	0
v	0	0	0	0	0	0	0
$\overline{t}$	0	0	0	0	0	0	1

Algorithm 1: Matrix computation of least model

input : a *definite program* P and its Herbrand base $B_P = \{p_1, p_2, \dots, p_n\}$

output: a vector v representing the least model

```
transform P to a standardized program P^δ
create matrix M_{P^δ} ∈ ℝ^{m×m} representing P^δ
create initial vector v_0 = (v_1, v_2, ..., v_m)^T of P^δ
v = v_0
u = θ(M_{P^δ} v)                    ▷ θ thresholding method
while u ≠ v do
    v = u
    u = θ(M_{P^δ} v)                ▷ θ thresholding method
end
return v
```

- *Interpretation vector* v is a vector which represents the truth value of the proposition in P.
- *Initial vector* v_0 is the starting point of v in which only propositions are fact have the truth value is 1.

Algorithm 2: Matrix computation of stable models

input : a *normal program* P and its Herbrand base $B_P = \{p_1, p_2, ..., p_n\}$

output: a set of vectors V representing the stable models or P

transform P to a *standardized program* P^+ with $B_{P^+} = \{p_1, \dots, p_n, \overline{q}_{n+1}, \dots, \overline{q}_m\}$.

create the matrix $M_P \in \mathbb{R}^{m \times m}$ representing P^+

create the initial matrix $M_0 \in \mathbb{R}^{m \times h}$

$M = M_0$, $U = \theta(M_{P^+} M)$ ▷ θ thresholding method

while $U \neq M$ **do**

    $M = U$, $U = \theta(M_{P^+} M)$ ▷ θ thresholding method

end

V = find stable models of P ▷ refer to Algorithm 3 in the paper

return V

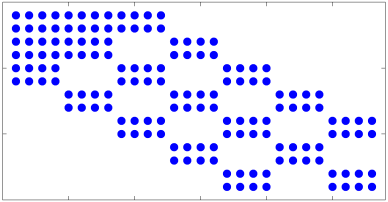
- V is a set of *interpretation vector* v.
- M_0 is the initial point of V.
- M_0 is created by enumerating all the combinations of the truth value of negations appear in P.

Outline

Sparse Representation of Logic Programs

- Matrix multiplication is the most time-consuming task.
- Noticeably, matrices representing logic programs are sparse.
- This is because $|body(r)| \ll |B_P|$ for each rule r of program p.

$$sparsity(P) = 1 - \frac{\sum_{r \in P} |body(r)|}{n^2} \quad (1)$$



The complexity of these aformentioned algebraic methods could be enhanced remarkably from $O(m^3)$ or $O(m^2 n)$ to approximate *the number of non-zero elements*[5].

[5] Gustavson, "Two fast algorithms for sparse matrices: Multiplication and permuted transposition", 1978.

Sparse Representation of Logic Programs

$$P = \{p \leftarrow q \wedge r,\ p \leftarrow s \wedge t,\ r \leftarrow s,\ q \leftarrow t,\ s \leftarrow,\ t \leftarrow\}$$
$$P' = \{u \leftarrow q \wedge r,\ v \leftarrow s \wedge t,\ p \leftarrow u \vee v,\ r \leftarrow s,\ q \leftarrow t,\ s \leftarrow,\ t \leftarrow\}$$

$$\begin{array}{c c} & \begin{array}{ccccccc} p & q & r & s & t & u & v \end{array} \\ \begin{array}{c} p \\ q \\ r \\ s \\ t \\ u \\ v \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 \end{pmatrix} \end{array}$$

$$sparsity = 1 - \frac{10}{7^2} = 0.796$$

Example 3:

Coordinate (COO) representation for P in *Example 1*

Row index	0	0	1	2	3	4	5	5	6	6
Col index	5	6	4	3	3	4	1	2	3	4
Value	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.5	0.5	0.5

Example 4:

Compressed Sparse Row (CSR) representation for P in *Example 1*

Row index	0	2	3	4	5	6	8	10		
Col index	5	6	4	3	3	4	1	2	3	4
Value	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.5	0.5	0.5

Outline

Experimental results

- We use the same method of Logic Programming (LP) generation conducted in[6] that the size of logic program defined by the size $n = |B_P|$ of the Herband base B_P and the number of rules $m = |P|$ in P.

Table: Proportion of rules in P based on the number of propositional variables in their bodies.

Body length	0	1	2	3	4	5	6	7	8
Allocated proportion	$< n/3$	4%	4%	10%	40%	35%	4%	2%	1%

- Further experiment using non-random problems with definite programs using transitive closure problem. The graph is selected from the Koblenz network collection[7]. This dataset contains binary tuples and we compute transitive closure of them using the following rules: $path(X, Y) \leftarrow edge(X, Y)$ and $path(X, Y) \leftarrow edge(X, Z) \wedge path(Z, Y)$

[6] Nguyen et al., "Computing Logic Programming Semantics in Linear Algebra", 2018.
[7] Kunegis, "Konect: the koblenz network collection", 2013.

Experimental results

- IWe compare our methods with *T_P-operator* and *Clasp* (Clingo v5.4.1 running with flag −−method=clasp)[8] .
- Our implementations are done with *dense matrix method* and *sparse matrix method* using C++ with CPU x64 as a targeted device (we do not use GPU accelerated code).
- In terms of matrix representations and operators, we use *Eigen 3 library*[9].
- Environment configurations: CPU: Intel Cote i7-4770 (4 cores, 8 threads) @3.4GHz; RAM: 16GB DDR3 @1333MHz; Operating system: Ubuntu 18.04 LTS 64bit.

[8] Gebser et al., "Theory solving made easy with clingo 5", 2016.

[9] Guennebaud and Jacob, *Eigen v3*, 2010.

Experimental results - *definite programs, artificial data*

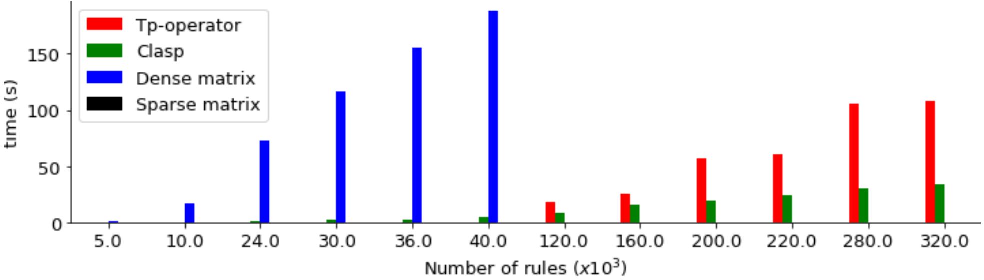


Figure: Execution time comparison of T_P-operator, Clasp and linear algebraic methods (with dense and sparse representation) on definite programs.

Experimental results - *definite programs, artificial data*

Table: Details of experimental results on definite programs of T_P-operator, Clasp and linear algebraic methods (with dense and sparse representation). n' indicates the actual matrix size after transformation.

No.	n	m	n'	Sparsity	T_P-operator	Clasp	Dense matrix	Sparse matrix
1	1000	5000	5788	0.99	0.0402	0.1680	2.0559	**0.0071**
2	1000	10000	10799	0.99	0.1226	0.2940	17.9986	**0.0127**
3	1600	24000	25198	0.99	0.3952	1.8480	73.3541	**0.0357**
4	1600	30000	31285	0.99	0.4793	2.5360	116.1158	**0.0605**
5	2000	36000	37596	0.99	0.7511	3.1690	155.4312	**0.0692**
6	2000	40000	41936	0.99	0.9763	5.1610	187.6549	**0.0675**
7	10000	120000	127119	0.99	18.5608	9.0720	-	**0.3798**
8	10000	160000	167504	0.99	25.6532	15.7760	-	**0.4832**
9	16000	200000	211039	0.99	57.0223	19.9760	-	**0.8643**
10	16000	220000	231439	0.99	60.4486	24.7860	-	**0.9429**
11	20000	280000	297293	0.99	104.9978	30.5730	-	**0.9048**
12	20000	320000	337056	0.99	108.5883	34.4030	-	**1.0614**

Experimental results - *definite programs, real data*

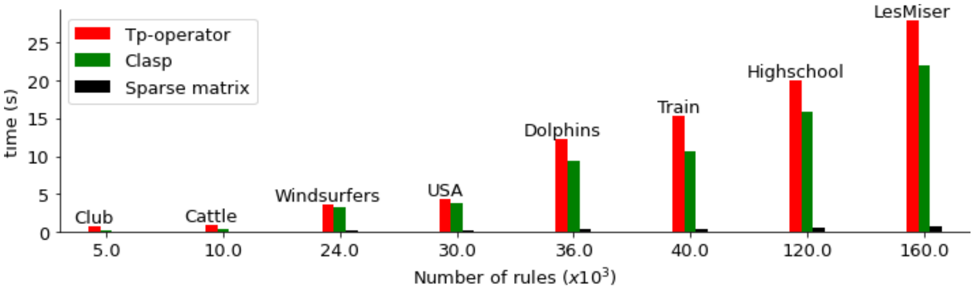


Figure: Execution time comparison of T_P-operator, Clasp and sparse representation method on definite programs with Transitive closure problem using Koblenz network datasets.

Experimental results - *definite programs, real data*

Table: Details of experimental results on transitive closure problem of T_P-operator, Clasp and sparse representation approach. n' indicates the actual matrix size after transformation.

Data name ($\lvert V\rvert, \lvert E\rvert$)	n	m	n'	Sparsity	**T_P-operator**	**Clasp**	**Sparse matrix**
Club membership (65, 95)	1200	14492	15600	0.99	0.8397	0.3370	**0.0255**
Cattle (28, 217)	1512	20629	21924	0.99	0.9541	0.5060	**0.0365**
Windsurfers (43, 336)	4324	99788	103776	0.99	3.6453	3.3690	**0.1824**
Contiguous USA (49, 107)	4704	113003	117600	0.99	4.2975	3.8830	**0.1830**
Dolphins (62, 159)	7564	230861	238266	0.99	12.3067	9.3820	**0.4019**
Train bombing (64, 243)	8064	254259	262080	0.99	15.2257	10.6350	**0.4524**
Highschool (70, 366)	9660	333636	342930	0.99	19.9622	15.8010	**0.6618**
Les Miserables (77, 254)	11704	445006	456456	0.99	27.7931	21.9560	**0.8300**

Experimental results - *normal programs, artificial data*

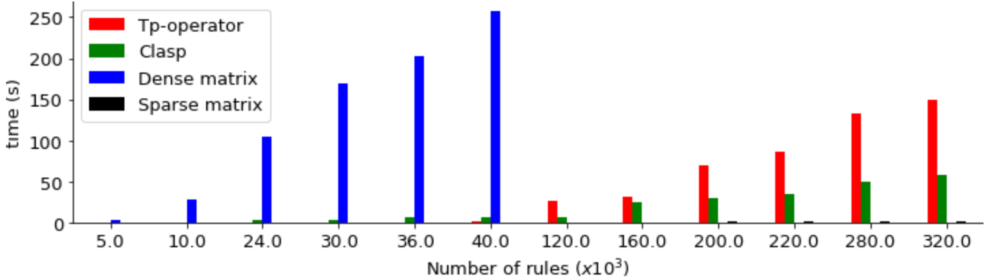


Figure: Execution time comparison of T_P-operator, Clasp and linear algebraic methods (with dense and sparse representation) on normal programs.

Experimental results - *normal programs, artificial data*

Table: Details of experimental results on normal programs of T_P-operator, Clasp and linear algebraic methods (with dense and sparse representation). n' indicates the actual matrix size after transformation.

No.	n	m	n'	k [10]	Sparsity	T_P-operator	Clasp	Dense matrix	Sparse matrix
1	1000	5000	6379	8	0.99	0.0472	0.3070	3.9560	**0.0119**
2	1000	10000	12745	8	0.99	0.1838	1.0920	28.1806	**0.0178**
3	1600	24000	30061	8	0.99	0.5525	3.2760	105.4931	**0.0559**
4	1600	30000	36402	7	0.99	0.6801	4.3050	168.8044	**0.0832**
5	2000	36000	42039	5	0.99	1.2378	6.7180	203.2749	**0.0897**
6	2000	40000	48187	8	0.99	1.5437	7.1800	256.9701	**0.0991**
7	10000	120000	171967	6	0.99	27.3162	7.6820	-	**0.7124**
8	10000	160000	207432	7	0.99	32.5547	24.6990	-	**0.8424**
9	16000	200000	250194	5	0.99	70.3114	30.7180	-	**1.5603**
10	16000	220000	278190	6	0.99	86.5192	35.4050	-	**1.8314**
11	20000	280000	357001	4	0.99	133.7881	50.1970	-	**1.9170**
12	20000	320000	396128	4	0.99	150.3377	58.6090	-	**2.1066**

[10]

Outline

Conclusions and Future Works

1. Analyze the sparsity of matrix representation for LP
2. Demonstrate the improvement using sparse matrix representation in terms of computation performance even when compared to Clasp.
3. Apply a sampling method to reduce the number of guesses in the initial matrix for normal programs that also reduce the dimension of the matrix representation.
4. Conduct more experiments on real Answer Set Programming (ASP) problems (usually including many negations).

References I

- Gebser, Martin et al. "Theory solving made easy with clingo 5". In: *OASIcs-OpenAccess Series in Informatics*. Vol. 52. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2016. DOI: 10.4230/OASIcs.ICLP.2016.2.
- Guennebaud, Gaël, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.
- Gustavson, Fred G. "Two fast algorithms for sparse matrices: Multiplication and permuted transposition". In: *ACM Transactions on Mathematical Software (TOMS)* 4.3 (1978), pp. 250–269. DOI: 10.1145/355791.355796.
- Kojima, Ryosuke and Taisuke Sato. "A tensorized logic programming language for large-scale data". In: *arXiv preprint arXiv:1901.08548* (2019).
- Kunegis, Jérôme. "Konect: the koblenz network collection". In: *Proceedings of the 22nd International Conference on World Wide Web*. 2013, pp. 1343–1350. DOI: 10.1145/2487788.2488173.

References II

- Nguyen, Hien D. et al. "Computing Logic Programming Semantics in Linear Algebra". In: *Proceedings of the 12th International Conference on Multi-disciplinary Trends in Artificial Intelligence (MIWAI 2018)*. Lecture Notes in Artificial Intelligence, Vol. 11248: Springer International Publishing, 2018, pp. 32–48. ISBN: 978-3-030-03014-8. DOI: 10.1007/978-3-030-03014-8_3.
- Sakama, Chiaki, Katsumi Inoue, and Taisuke Sato. "Linear Algebraic Characterization of Logic Programs". In: *Proceedings of the 10th International Conference on Knowledge Science, Engineering and Management (KSEM 2017)*. Lecture Notes in Artificial Intelligence, Vol. 10412: Springer International Publishing, 2017, pp. 520–533. ISBN: 978-3-319-63558-3. DOI: 10.1007/978-3-319-63558-3_44.
- Sakama, Chiaki et al. *Partial Evaluation of Logic Programs in Vector Spaces*. EasyChair Preprint no. 172. EasyChair, 2018. DOI: 10.29007/9d61.

Thank you for your attention