Partial Evaluation of Logic Programs in Vector Spaces

Chiaki Sakama, Wakayama Univ., Japan
Hien D. Nguyen, VNU-HCM, Vietnam
Taisuke Sato, AIST, Japan
Katsumi Inoue, NII, Japan

ASPOCP 2018@Oxford

Background

- Integration of linear algebraic computation and symbolic computation is a challenging topic in AI.
  e.g. Tensor-based predicate calculus (Grefenstette 2013)
  Mining Horn clauses in vector spaces (Yang, et al. 2015)
  Logic Tensor Networks (Serafini&Garcez 2016)
  Computing Datalog in vector spaces (Sato 2017), etc
- Such integration has potential to make symbolic reasoning scalable to real-life datasets.
- Computing logic programming using linear algebra contributes to a step for realizing logical inference in huge scale knowledge bases.

Purpose

- We provide a new method of computing the least model of a propositional definite program using linear algebra.
- We develop a technique of partial evaluation of programs in vector spaces for optimization.
- We implement algorithms on GPGPU and evaluate performance.

Vector Representation of Interpretations

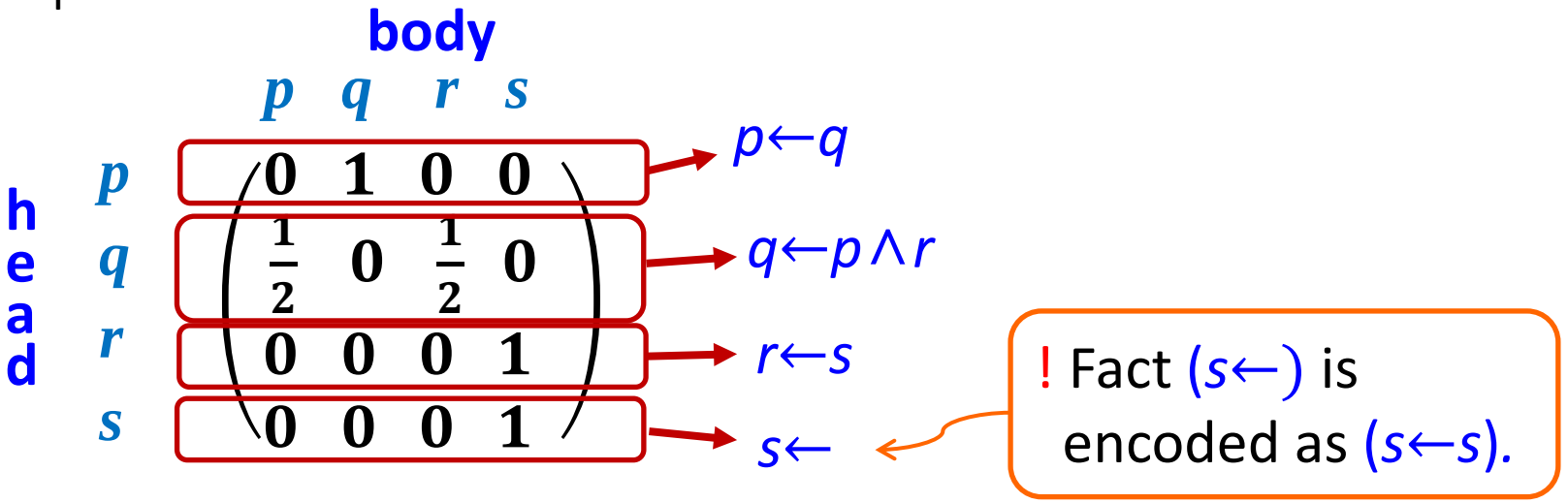
- Given the Herbrand base $B_P = \{ p, q, r, s \}$, an interpretation $I = \{ p, r \}$ is represented by the vector:

$$\boldsymbol{v} = \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{0} \end{pmatrix} \begin{matrix} p \\ q \\ r \\ s \end{matrix}$$

- The i-th element of $\boldsymbol{v}$ represents the truth value of p_i (written $row_1(\boldsymbol{v}) = p$, $row_2(\boldsymbol{v}) = q$, $row_3(\boldsymbol{v}) = r$, etc).
- Given $\boldsymbol{v}=(a_1,\ldots,a_n)^T \in \mathbf{R}^n$, $\boldsymbol{v}[a_1 \ldots a_k]$ represents a (sub)vector $(a_1,\ldots,a_k)^T \in \mathbf{R}^k$ $(k \leq n)$.

Matrix Representation of Definite Programs

- $P = \{\ p \leftarrow q,\quad q \leftarrow p \wedge r,\quad r \leftarrow s,\quad s \leftarrow\ \}$ is represented by $\boldsymbol{M}_P \in \mathbf{R}^{4\times 4}$:



- The i-th row represents the atom p_i in the head, and the j-th column represents the atom p_j in the body of a rule (written: $row_1(\boldsymbol{M}_P) = p$, $col_2(\boldsymbol{M}_P) = q$, … etc)

Matrix Representation of Rules with the Same Head

- $P = \{\ p \leftarrow q,\quad q \leftarrow p \wedge r,\quad q \leftarrow s,\quad s \leftarrow\ \}$ is transformed to the program $P^{\delta} = Q \cup D$ where
  $Q = \{\ p \leftarrow q,\quad t \leftarrow p \wedge r,\quad u \leftarrow s,\quad s \leftarrow\ \}$ and $D = \{\ q \leftarrow t \vee u\ \}$.
- P^{δ} is represented by $\boldsymbol{M}_{P^{\delta}} \in \mathbf{R}^{6 \times 6}$:

$$\begin{array}{c} \\ p \\ q \\ r \\ s \\ t \\ u \end{array} \begin{array}{c} \begin{array}{cccccc} p & q & r & s & t & u \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

- Rules in D are called **d-rules**.
- Note: $q \leftarrow t \vee u$ is a shorthand of $q \leftarrow t$ and $q \leftarrow u$, so P^{δ} is considered a definite program.

Thresholding

- Given a vector $\boldsymbol{v} = (a_1, \ldots, a_n)^T$, define a function θ:

$$\theta(\boldsymbol{v}) = (a'_1, \ldots, a'_n)^T$$

where $a'_i = 1$ $(1 \leq i \leq n)$ if $a_i \geq 1$; otherwise, $a'_i = 0$.

Computing Least Models

- Given P ={ $p \leftarrow q$, $q \leftarrow p \wedge r$, $r \leftarrow s$, $s \leftarrow$ }, the **initial vector** $\boldsymbol{v}_0$ =(0,0,0,1)$^{\mathsf{T}}$ represents facts in P. Then,

$$\boldsymbol{M}_{\mathsf{P}}\boldsymbol{v}_0 = \begin{array}{c} \\ p \\ q \\ r \\ s \end{array} \begin{array}{c} \begin{array}{cccc} p & q & r & s \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \qquad \boldsymbol{v}_1 = \boldsymbol{\theta}(\boldsymbol{M}_{\mathsf{P}}\boldsymbol{v}_0)$$

$$\boldsymbol{M}_{\mathsf{P}}\boldsymbol{v}_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{2} \\ 1 \\ 1 \end{pmatrix} \qquad \boldsymbol{v}_2 = \boldsymbol{\theta}(\boldsymbol{M}_{\mathsf{P}}\boldsymbol{v}_1) = \boldsymbol{v}_1$$

- $\boldsymbol{v}_1$ is a fixpoint of $\boldsymbol{v}_k = \boldsymbol{\theta}(\boldsymbol{M}_{\mathsf{P}}\boldsymbol{v}_{k-1})$ ($k \geq 1$).
- $\boldsymbol{v}_1$ =(0,0,1,1)$^{\mathsf{T}}$ represents the least model { r, s } of P.

Constraints

- Given P ={ $p \leftarrow q$, $q \leftarrow p \wedge r$, $r \leftarrow s$, $s \leftarrow$}, and C ={ $\leftarrow q \wedge s$, $\leftarrow r$ }, first compute the least model of P.
- C is represented by

$$\boldsymbol{M}_{\mathrm{C}} = \begin{array}{c} \begin{matrix} p & q & r & s \end{matrix} \\ \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

- Applying $\boldsymbol{v}$ =$(0,0,1,1)^{\mathrm{T}}$ to $\boldsymbol{M}_{\mathrm{C}}$:

$$\boldsymbol{M}_{\mathrm{C}}\boldsymbol{v} = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 1 \end{pmatrix}$$

Existence of 1 means $P \cup C$ is inconsistent.

Column Reduction

- Consider $P^{\delta} = Q \cup D$ where

  $Q = \{ p \leftarrow q,\ \ t \leftarrow p \wedge r,\ \ u \leftarrow s,\ \ s \leftarrow \}$ and $D = \{ q \leftarrow t \vee u \}$.

- Reduce columns for newly introduced atoms and produce $\boldsymbol{N}_{P^{\delta}} \in \mathbf{R}^{6 \times 4}$:

$$\boldsymbol{M}_{P^{\delta}} = \begin{array}{c} \\ p \\ q \\ r \\ s \\ t \\ u \end{array} \begin{array}{c} \begin{array}{cccccc} p & q & r & s & t & u \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \quad \Rightarrow \quad \boldsymbol{N}_{P^{\delta}} = \begin{array}{c} \\ p \\ q \\ r \\ s \\ t \\ u \end{array} \begin{array}{c} \begin{array}{cccccc} p & q & r & s & t & u \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Example (cont.)

- $P^{\delta} = \{\ p \leftarrow q,\quad t \leftarrow p \wedge r,\quad u \leftarrow s,\quad s \leftarrow,\quad q \leftarrow t \vee u\ \}$.
- Given $\boldsymbol{v} = (0,0,0,1)^{\mathsf{T}}$, it becomes $\boldsymbol{w} = \boldsymbol{N}_{P^{\delta}}\boldsymbol{v} = (0,0,0,1,0,1)^{\mathsf{T}}$.
- Introduce the rule: *if an element in the body of a d-rule is 1, then the element in the head of the d-rule is set to 1.* Add this rule to the θ-theresholding (written θ_D).

$$\boldsymbol{N}_{P^{\delta}} = \begin{array}{c} \begin{matrix} p & q & r & s \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} p \\ q \\ r \\ s \\ t \\ u \end{matrix}$$

- Put $d = (q \leftarrow t \vee u)$. Since $row_6(\boldsymbol{w}) = u \in body(d)$ and $head(d) = q$, applying θ_D to $\boldsymbol{N}_{P^{\delta}}\boldsymbol{v}$ produces $\theta_D(\boldsymbol{N}_{P^{\delta}}\boldsymbol{v}) = (0,1,0,1,0,1)^{\mathsf{T}}$.

Computing Least Models

- $P^{\delta} = \{\ p \leftarrow q,\quad t \leftarrow p \wedge r,\quad u \leftarrow s,\quad s \leftarrow,\quad q \leftarrow t \vee u\ \}$.
- Given $\boldsymbol{v}_0 = (0,0,0,1,0,0)^{\mathsf{T}}$, $\boldsymbol{v}_0[1\ldots4] = (0,0,0,1)^{\mathsf{T}}$:

$\boldsymbol{v}_1 = \theta_D(\boldsymbol{N}_{P^{\delta}} \boldsymbol{v}_0[1\ldots4]) = (0,1,0,1,0,1)^{\mathsf{T}}$
$\boldsymbol{v}_2 = \theta_D(\boldsymbol{N}_{P^{\delta}} \boldsymbol{v}_1[1\ldots4]) = (1,1,0,1,0,1)^{\mathsf{T}}$
$\boldsymbol{v}_3 = \theta_D(\boldsymbol{N}_{P^{\delta}} \boldsymbol{v}_2[1\ldots4]) = (1,1,0,1,0,1)^{\mathsf{T}} = \boldsymbol{v}_2$

$$\boldsymbol{N}_{P^{\delta}} = \begin{array}{c} \begin{matrix} p & q & r & s \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} p \\ q \\ r \\ s \\ t \\ u \end{matrix}$$

- Then $\boldsymbol{v}_2$ represents the least model of P^{δ} and $\boldsymbol{v}_2[1\ldots4] = (1,1,0,1)$ is a vector representing the least model $\{\ p, q, s\ \}$ of P.

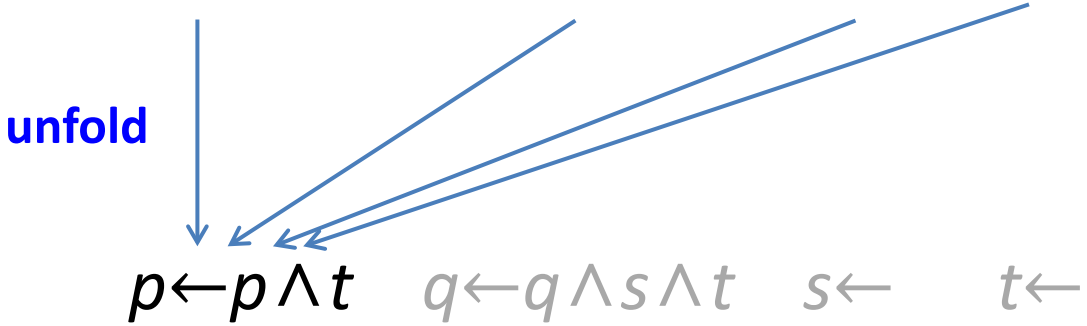
Complexities

- In matrix computation, complexity of computing $\boldsymbol{M}_{\mathrm{P}^\delta}\,\boldsymbol{v}$ is $O(m^2)$ and computing $\theta()$ is $O(m)$. The number of times for iterating $\boldsymbol{M}_{\mathrm{P}^\delta}\,\boldsymbol{v}$ is at most $(m+1)$ times. So the complexity of fixpoint computation is $O((m+1)\times(m+m^2))= O(m^3)$.
- In column reduction, the complexity of computing $\boldsymbol{N}_{\mathrm{P}^\delta}\,\boldsymbol{v}$ is $O(m\times n)$ and computing $\theta_{\mathrm{D}}()$ is $O(m\times n)$. The number of times for iterating $\boldsymbol{N}_{\mathrm{P}^\delta}\,\boldsymbol{v}$ is at most $(m+1)$ times. So the complexity of fixpoint computation is $O((m+1)\times(m\times n+m\times n))= O(m^2\times n)$.
- Column reduction reduces complexity as $m \gg n$ in general.

Partial Evaluation

- Unfolding every rule in a program simultaneously.

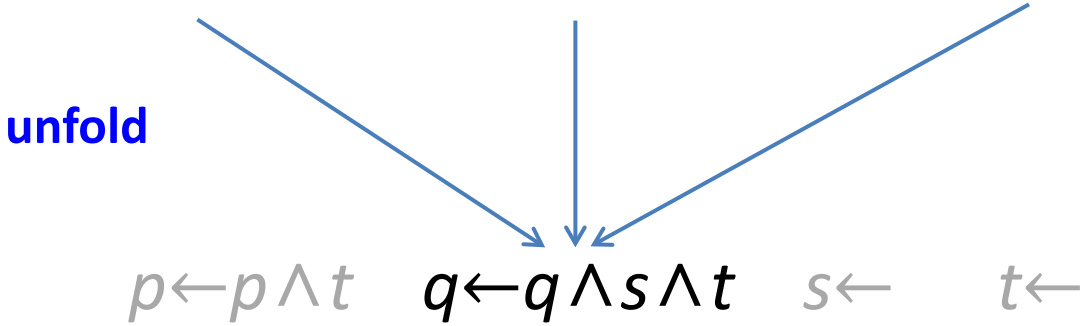
ex) $P = \{\ p \leftarrow q \wedge s \wedge t,\quad q \leftarrow p \wedge t,\quad s \leftarrow t,\quad t \leftarrow\ \}$



Partial Evaluation

- Unfolding every rule in a program simultaneously.

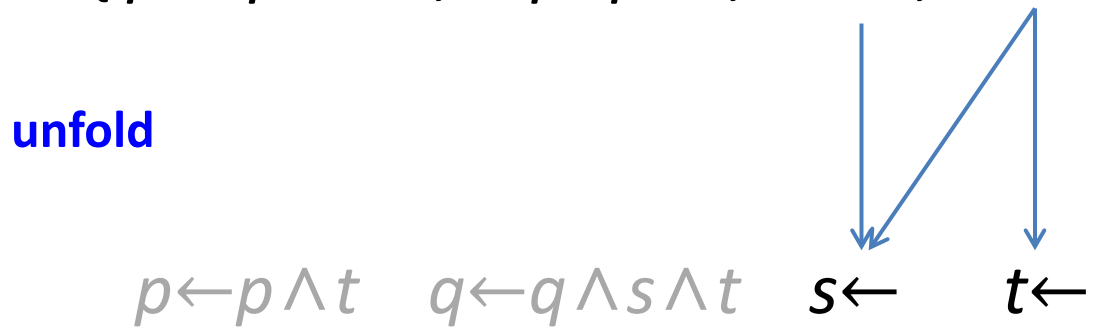
ex) $P = \{\ p \leftarrow q \wedge s \wedge t,\quad q \leftarrow p \wedge t,\quad s \leftarrow t,\quad t \leftarrow\ \}$



Partial Evaluation

- Unfolding every rule in a program simultaneously.

ex) $P = \{\ p \leftarrow q \wedge s \wedge t,\quad q \leftarrow p \wedge t,\quad s \leftarrow t,\quad t \leftarrow\ \}$



peval$(P) = \{\ p \leftarrow p \wedge t,\quad q \leftarrow q \wedge s \wedge t,\quad s \leftarrow,\quad t \leftarrow\ \}$

Computing peval

- $P = \{\ p \leftarrow q \wedge s \wedge t,\quad q \leftarrow p \wedge t,\quad s \leftarrow t,\quad t \leftarrow\ \}$ is represented by

$$\boldsymbol{M}_{\mathrm{P}} = \begin{array}{c} \\ p \\ q \\ s \\ t \end{array} \begin{array}{c} \begin{array}{cccc} p & q & s & t \end{array} \\ \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

- Then $\boldsymbol{M}_{\mathrm{P}}{}^{2}$ becomes

$$\boldsymbol{M}_{\mathrm{P}}{}^{2} = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{array}{c} \begin{array}{cccc} p & q & s & t \end{array} \\ \begin{pmatrix} \frac{1}{6} & 0 & 0 & \frac{5}{6} \\ 0 & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{array}{c} p \\ q \\ s \\ t \end{array}$$

Computing peval

- Viewing non-zero elements as conjuncts:
  - 1st row represents $p \leftarrow p \wedge t$
  - 2nd row represents $q \leftarrow q \wedge s \wedge t$

$$\boldsymbol{M}_{\mathrm{P}}{}^{2} = \begin{array}{c} \\ p \\ q \\ s \\ t \end{array} \begin{array}{c} \begin{array}{cccc} p & q & s & t \end{array} \\ \begin{pmatrix} \frac{1}{6} & 0 & 0 & \frac{5}{6} \\ 0 & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

- $\boldsymbol{M}_{\mathrm{P}}{}^{2}$ then represents
  $P' = \{\ p \leftarrow p \wedge t,\quad q \leftarrow q \wedge s \wedge t,\quad s \leftarrow t,\quad t \leftarrow\ \}$.
- By contrast, **peval**(P)=$\{\ p \leftarrow p \wedge t,\ q \leftarrow q \wedge s \wedge t,\ s \leftarrow,\ t \leftarrow\ \}$.
- $\boldsymbol{M}_{\mathrm{P}}{}^{2}$ represents the result of unfolding rules by rules (not by facts) (called **r-r partial evaluation**).
- Since P' and **peval**(P) have the same least model, the difference does not affect the result of computing the least model of P.

Computing peval

- Let P be a program s.t. $head(r_1) \neq head(r_2)$ for any two rules r_1 and r_2 in P (called **singly defined (SD) program**).
- Given the initial vector $\boldsymbol{v}_0$ representing facts of P,

$$\theta(\boldsymbol{M}_P^2 \boldsymbol{v}_0) = \theta(\boldsymbol{M}_P(\theta(\boldsymbol{M}_P \boldsymbol{v}_0))) \qquad (\dagger)$$

- Partial evaluation has the effect of reducing deduction steps by unfolding rules in advance. (†) realizes this effect by computing matrix products in advance.

Iterative Partial Evaluation

- Partial evaluation is performed iteratively as

  $\mathbf{peval}^{k}(P)=\mathbf{peval}(\mathbf{peval}^{k-1}(P))\ (k\geqq 1)$ and $\mathbf{peval}^{0}(P)=P$

- Let P be an SD program and $\boldsymbol{M}_{\mathrm{P}}$ its program matrix. Define $\mathbf{\Gamma}_{\mathrm{P}}{}^{1}=\boldsymbol{M}_{\mathrm{P}}{}^{2}$ and $\mathbf{\Gamma}_{\mathrm{P}}{}^{k+1}=(\mathbf{\Gamma}_{\mathrm{P}}{}^{k})^{2}\ (k\geqq 1)$.

- Then $\mathbf{\Gamma}_{\mathrm{P}}{}^{k}$ is a matrix representing a program that is obtained by k-th iteration of (r-r) partial evaluation.

Peval of non-SD Program

- When P is a non-SD program, first transform P to $P^{\delta} = Q \cup D$ where Q is an SD-program and D is a set of d-rules.
- Define $(\boldsymbol{\Gamma}_P{}^{\delta})^k = \boldsymbol{\Gamma}_Q{}^k + \boldsymbol{M}_D$ where $\boldsymbol{\Gamma}_Q{}^k$ represents the k-th iteration of (r-r) partial evaluation of Q and $\boldsymbol{M}_D$ is the matrix representing D.
- Compute (r-r) partial evaluation of P^{δ} as (r-r) partial evaluation of an SD program Q plus d-rules D.

Example

- P = { $p \leftarrow q$, $p \leftarrow r$, $q \leftarrow r$, $r \leftarrow s$, $s \leftarrow$ } is transformed to $P^{\delta} = Q \cup D$ where Q = { $u \leftarrow q$, $v \leftarrow r$, $q \leftarrow r$, $r \leftarrow s$, $s \leftarrow$ } and D= { $p \leftarrow u \vee v$ }.
- $(\mathbf{\Gamma}_{P^{\delta}})^2 = \mathbf{\Gamma}_Q{}^2 + \boldsymbol{M}_D =$

$$\begin{array}{c} \\ p \\ q \\ r \\ s \\ u \\ v \end{array} \begin{array}{c} \begin{array}{cccccc} p & q & r & s & u & v \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} + \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

which represents the result of the 2nd iteration of (r-r) peval of Q plus D:

{ $u \leftarrow s$, $v \leftarrow s$, $q \leftarrow s$, $r \leftarrow s$, $s \leftarrow$, $p \leftarrow u \vee v$ }.

Experiments

- Compare 4 algorithms for computing the least model of a definite program:
  - fixpoint by the T_P-operator
  - matrix computation
  - column reduction
  - partial evaluation
- Testing is done on a machine with the configuration:
  - OS: Linux Ubuntu 16.04 LTS 64bit
  - CPU: Intel Core i7-6800K (3.4GHz/14nm/Cores=6/ Threads=12 /Cache15MB), Memory 32GB, DDR-2400
  - GPU: GeForce GTX1070TI GDDR5 8GB
  - Implementation Language: Maple 2017, 64bit

Parameters

- Runtime is measured by changing the parameters:
  - n: size of the Herbrand base B_P
  - m: number of rules in P
  - k: number of iteration in peval (k=1, 5, n/2, n)
- Based on (n,m), randomly generate a definite program having rules as follows:

N	0	1	2	3	4	5	6	7	8
rate	< n/3	4%	4%	10%	40%	35%	4%	2%	∼1%

+ N is the number of atoms in the body of a rule
+ Every program has > 95% rules with |body(r)| > 1

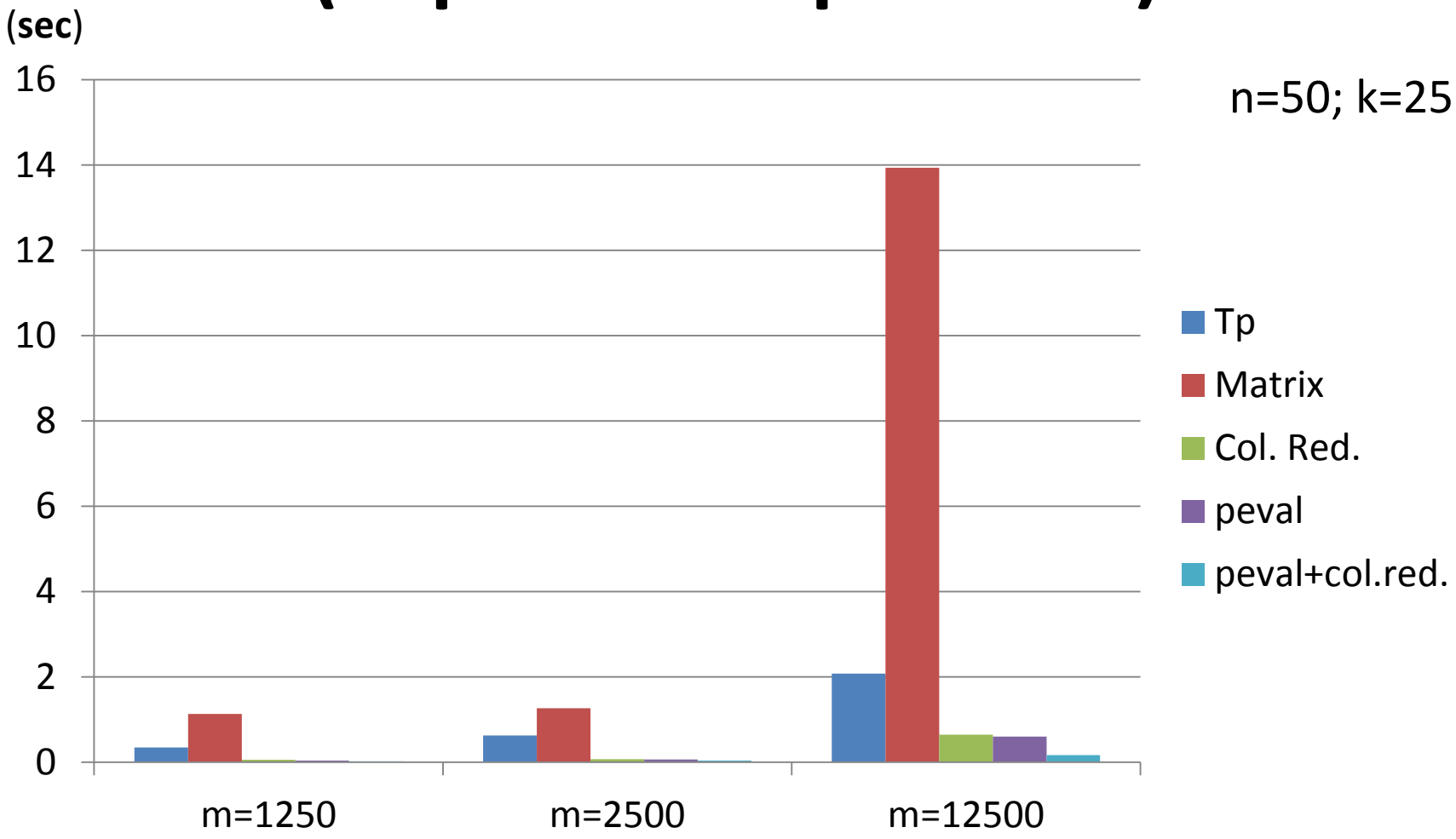
Results

(sec)

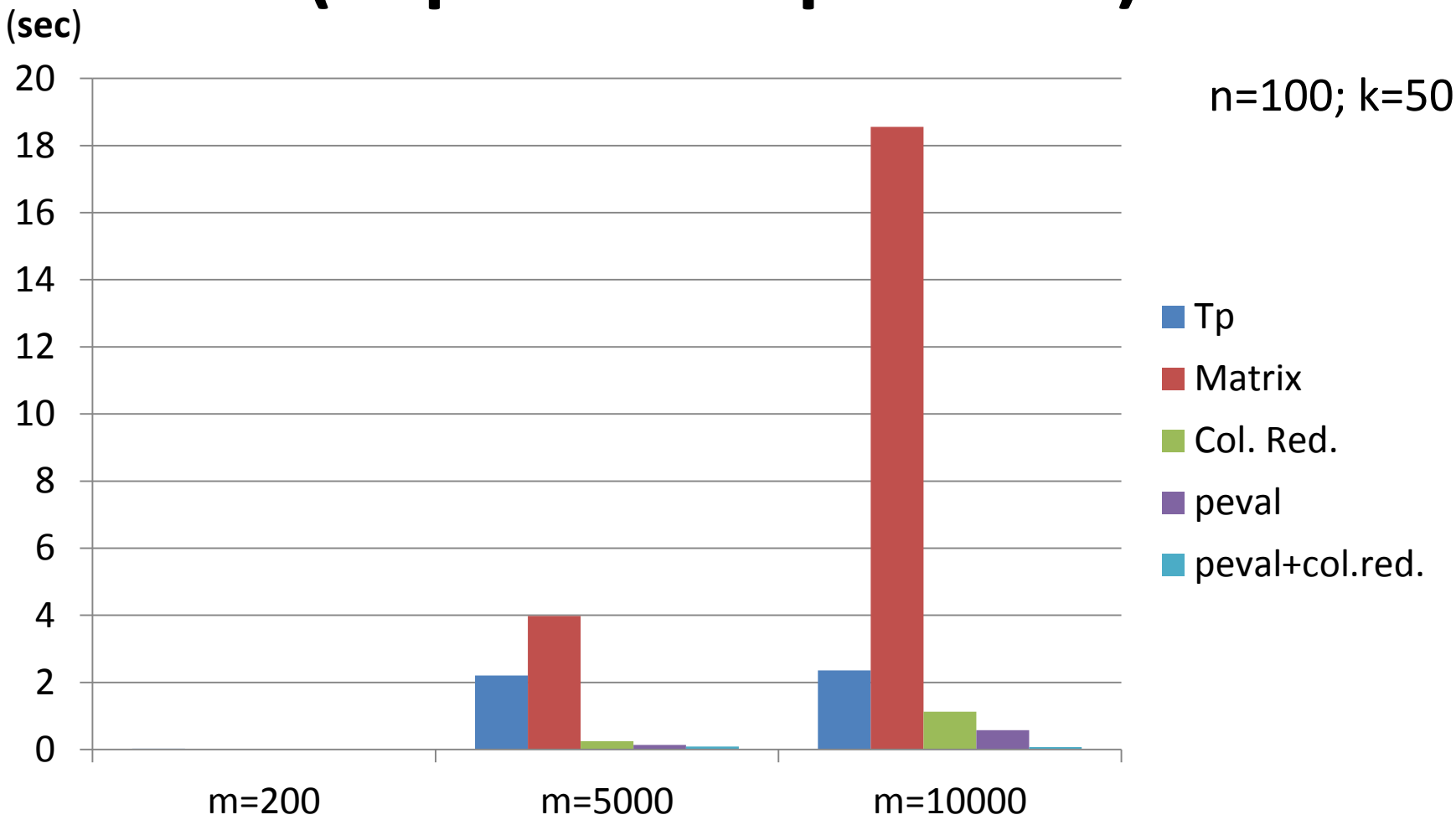
n	m	T_p	Matrix Fixpoint/All	Column Red Fixpoint/All	partial evaluation Γ^k/matrix/col.reduct.
50	100	0.008	0.007/ 0.155	0.005/ 0.13	0.006/ 0.005/ 0.005 (k=25)
50	1,250	0.35	1.135/ 1.158	0.061/ 0.247	0.565/ 0.029/ 0.012 (k=25)
50	2,500	0.627	1.269/ 1.3	0.071/ 0.142	1.81/ 0.063/ 0.043 (k=25)
50	12,500	2.081	13.937/ 14.358	0.649/ 1.024	79.625/ 0.604/ 0.168 (k=25)
100	200	0.029	0.014 / 0.05	0.003 / 0.021	0.017/ 0.007/ 0.01 (k=50)
100	5,000	2.206	3.981 / 4.044	0.249 / 0.485	6.696/ 0.136/ 0.094 (k=50)
100	10,000	2.355	18.553 / 18.836	1.131 / 1.807	78.037/ 0.576/ 0.076 (k=50)
200	400	0.06	0.063 / 0.075	0.013 / 0.06	0.102/ 0.018/ 0.024 (k=100)
200	20,000	6.391	25.161 / 25.833	4.48 / 7.771	289.564/ 1.15/ 0.519 (k=100)

+ "All" means time for creating a program matrix + computing the fixpoint.
+ Γ^k means time for computing partial evaluation
+ After peval, fixpoint is computed in 2 ways: by matrix computation or column reduction

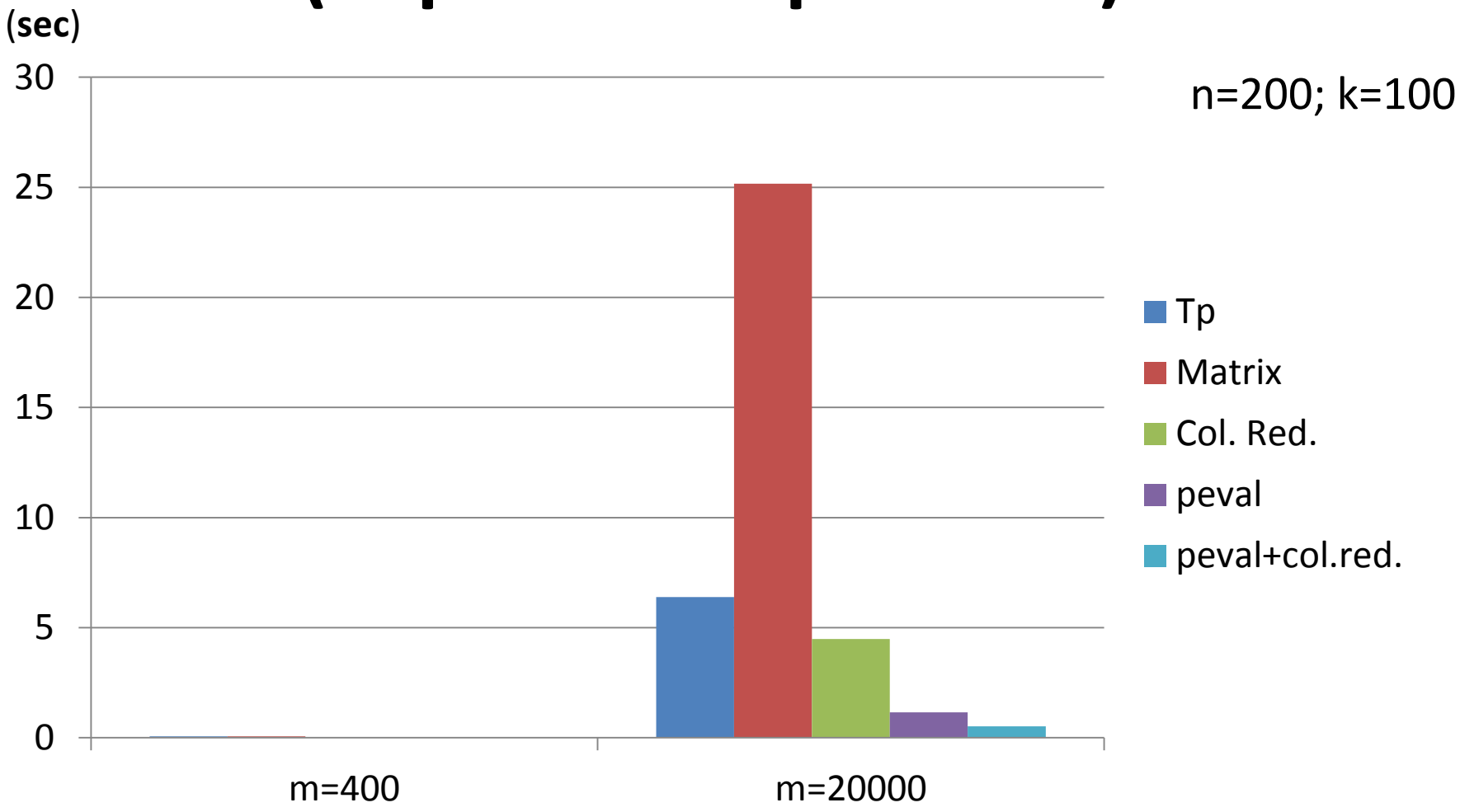
Comparison (fixpoint computation)



Comparison (fixpoint computation)



Comparison (fixpoint computation)



Discussion

- For fixpoint computation, efficiency increases as: **matrix computation < T_P-operator < col. reduction**.
- Column reduction is often more than 10 times faster than naïve computation by program matrices.
- By performing partial evaluation, time for fixpoint computation is significantly reduced.
- Partial evaluation + column reduction is most effective in large scale of programs.

Discussion

- Computing the least model of a definite program costs $O(N)$ where N is the size (the number of literals) of a program (Dowling&Gallier 1984).
- Since column reduction takes $O(m^2 \times n)$ time, it would be effective when $m^2 \times n < N$, i.e., the size of a program is large with a relatively small number of atoms.
- Partial evaluation is performed apart from fixpoint computation, then combination of column reduction and partial evaluation is effective in practice.

Summary

- Linear algebraic computation of definite programs is introduced. Experimental results show that it can realize efficient computation in a large scale of programs and partial evaluation helps to reduce runtime significantly.
- The linear algebraic approach enables us to use efficient algorithms of numerical linear algebra and opens perspective for **parallel** computation of logic programming.
- Performance of our implementation heavily depends on the environment of LA computation, and further acceleration would be gained on more powerful platforms.

Future Work

- An important question is whether linear algebraic computation is applied to **answer set programming**.
- A method for computing stable models was introduced in (Sakama *et al*. 2017) where normal programs are represented by 3rd order tensors.
- We plan to develop a new algorithm for ASP in vector spaces and evaluate it using benchmark testing.

C.Sakama, K.Inoue, T.Sato: **Linear algebraic characterization of logic programs**. *Knowledge Science, Engineering and Management*. LNAI 10412, pp.520-533 (2017)