A New Algorithm for Computing Least Generalization

Hien D. Nguyen
VNU-HCM, Vietnam
Chiaki Sakama
Wakayama University, Japan

ILP 2019, Plovdiv, Bulgaria, September 2019

BACKGROUND: UNIFICATION VS. ANTI-UNIFICATION

- **Robinson's Unification Algorithm** (1965): compute the **greatest common instance** of any finite set of unifiable atomic formulas.

- **Plotkin/Reynolds's Anti-unification Algorithm** (1970): compute the **least common generalization** of any finite set of compatible atomic formulas.

GREATEST COMMON INSTANCE

P(x, f(y)) P(z, f(b)) P(c,w)

mgu: $\sigma_1 = \{ b/y, x/z \}$

P(x, f(b))

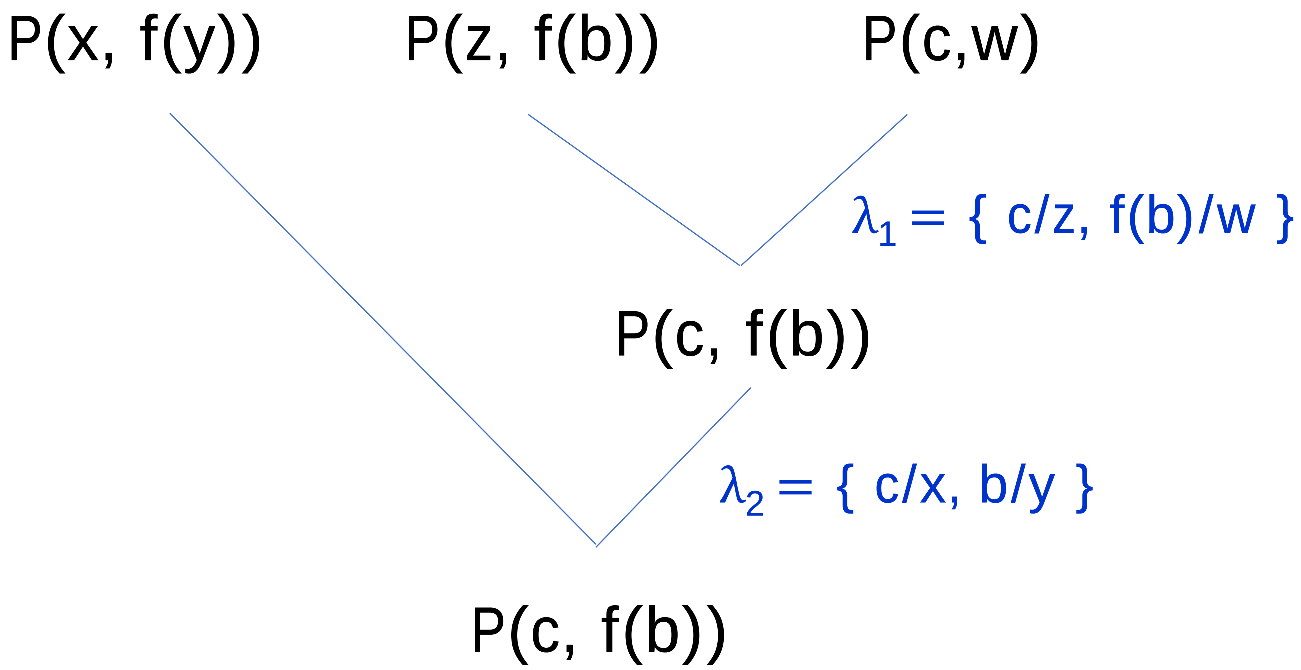
mgu: $\sigma_2 = \{ c/x, f(b)/w \}$

P(c, f(b))

Composition of mgu: $\sigma_1\sigma_2 = \{ b\sigma_2 /y, x\sigma_2 /z, c/x, f(b)/w \}$

$= \{ b/y, c/z, c/x, f(b)/w \}$

GREATEST COMMON INSTANCE



Composition of mgu: $\lambda_1\lambda_2 = \{\ c\lambda_2\,/z,\ f(b)\lambda_2\,/w,\ c/x,\ b/y\ \}$

$= \{\ c/z,\ f(b)/w,\ c/x,\ b/y\ \} = \sigma_1\sigma_2$

GREATEST COMMON INSTANCE

P(x, f(y)) P(z, f(b)) P(c,w)

mgu: $\sigma_1 = \{ b/y, x/z \}$

mgu: $\lambda_1 = \{ c/z, f(b)/w \}$

P(x, f(b)) P(c, f(b))

$\sigma_1 + \lambda_1 = \{ c/x, b/y, c/z, f(b)/w \}$

P(c, f(b))

Combination $\sigma_1 + \lambda_1$ is computed as the mgu of
Q(b, x, c, f(b)) and Q(y, z, z, w)
where $\sigma_1 = \{ b/y, x/z \}$ and $\lambda_1 = \{ c/z, f(b)/w \}$

COMPOSITION VS. COMBINATION

	Composition	Combination
associative	$(\sigma_1\sigma_2)\sigma_3 = \sigma_1(\sigma_2\sigma_3)$	$(\sigma_1+\sigma_2)+\sigma_3 = \sigma_1+(\sigma_2+\sigma_3)$
commutative	$\sigma_1\sigma_2 \neq \sigma_2\sigma_1$	$\sigma_1+\sigma_2 = \sigma_2+\sigma_1$
idempotent	$\sigma\sigma \neq \sigma$	$\sigma+\sigma = \sigma$
identity	$\sigma\varepsilon = \varepsilon\sigma = \sigma$	$\sigma+\varepsilon = \varepsilon+\sigma = \sigma$

ε : empty substitution

! Composition is neither commutative nor idempotent.

BACKGROUND:
COMBINATION OF SUBSTITUTIONS

- Originally introduced in (Plawitz, 1969) and redefined in (Chang & Lee, 1973).
- Combination is obtained as the greatest lower bound of mgus (Eder, 1985).
- Semantics of Horn logic programs is reformulated using combination of mgus (Yamasaki et al., 1986; Palamidessi, 1990).

MOTIVATION & GOAL

- Combination of substitutions enables to compute greatest common instance **in parallel**.
- Unification and (most general) unifier are used for computing greatest common instance by combination.
- Greatest common instance and least common generalization are **dual** notions.
- We use **anti-unification** and (**most specific**) **anti-unifier** for computing least common generalization by the **inverse operation of combination**.

CONTRIBUTIONS

- We compute least (common) generalization of a set of atoms by an **inverse substitution** of combination, which we call **anti-combination**.
- We develop a **parallel algorithm** for computing least generalization based on anti-combination.
- We perform experimental evaluation and show that anti-combination outperforms sequential computation of anti-unification.

LEAST GENERALIZATION

- Given two atoms A and B, define $A \leq B$ if $A = B\theta$ for some substation θ. B is a **generalization** of A.
- Given a set of atoms $\Sigma = \{ A_1, \ldots, A_k \}$, an atom B is a **(common) generalization** of Σ if $A_i \leq B$ (i=1,…,k).
- An atom B is a **least (common) generalization** of Σ (written $lg(\Sigma)$) if B is a generalization of Σ and $B \leq C$ for any generalization C of Σ.

ANTI-UNIFIER

- Given a set of atoms $\Sigma = \{ A_1, \ldots, A_k \}$, a tuple of substitutions $\tau = (\sigma_1, \ldots, \sigma_k)$ is an **anti-unifier** of Σ if $A_i = lg(\Sigma)\sigma_i$ for i=1,…,k.

- An anti-unifier τ of Σ is a **most specific anti-unifier (msau)** if for each anti-unifier $(\theta_1, \ldots, \theta_k)$ there is a substitution λ_i s.t. $\sigma_i = \lambda_i \theta_i$ $(1 \leq i \leq k)$.

ANTI-UNIFICATION

- Given two atoms A and B, an **anti-unification algorithm** outputs $lg(\{A,B\})$ and an msau $\tau = (\sigma_1, \sigma_2)$.
  (Plotkin 1970; Reynolds 1970)
- For a set of atoms $\Sigma = \{A_1, \ldots, A_k\}$, $lg(\Sigma)$ is sequentially computed as $lg(A_1, lg(A_2, \ldots, lg(A_{k-1}, A_k) \ldots))$.
- The method is inefficient when the number of atoms increases.

INVERSE SUBSTITUTION

- ***Var*** : set of variables, ***Term***: set of terms
- A **substitution** is a mapping σ: ***Var*** → ***Term***.
  When $\sigma(x_i)=t_i$ (i=1,…,n), written $\sigma = \{ t_1/x_1,\ldots, t_n/x_n \}$.
  The set $D(\sigma)=\{ x_1,\ldots, x_n \}$ is the **domain** of σ.
- Given an injective substitution σ, an **inverse substitution** σ^{-1} : ***Term***→***Var*** is defined as

  - $t\sigma^{-1} = x$ if $(t/x)\in \sigma$
  - $f(t_1,\ldots,t_n)\sigma^{-1} = f(t_1\sigma^{-1},\ldots, t_n\sigma^{-1})$ if $(f(t_1,\ldots,t_n)/x) \notin \sigma$ for any $x\in$ ***Var***
  - $y\sigma^{-1} = y$ if $(y/x)\notin \sigma$ for any $x\in$ ***Var***

  where t and $D(\sigma)$ have no common variable.

REMARK

- If t and $D(\sigma)$ have common variables, variables in t are renamed to make them different from those in $D(\sigma)$.

- If σ is not injective, a technique of (N-Cheng&deWolf, 1997) is applied to compute σ^{-1}. For instance, given $\sigma = \{ a/x, a/y \}$, it becomes $\sigma^{-1} = \{ (x/a, \langle 1 \rangle), (y/a, \langle 2 \rangle) \}$ meaning that a at position $\langle 1 \rangle$ is mapped to x and a at position $\langle 2 \rangle$ is mapped to y.

REMARK

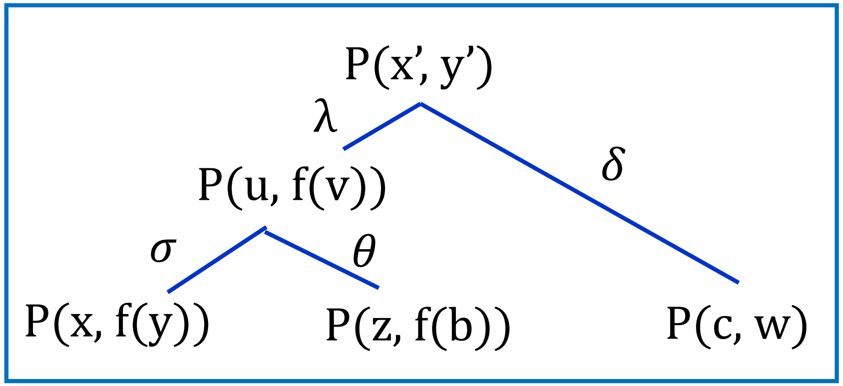
- Combining injective substitutions may produce a non-injective substitution. To compute its inverse substitution, incorporate information of substitutions from which each binding comes from.
- For instance, $\sigma_1 = \{\, a/x \,\}$ and $\sigma_2 = \{\, a/y \,\}$ produce $\sigma_1 + \sigma_2 = \{\, a/x,\ a/y \,\}$. Then, define $(\sigma_1 + \sigma_2)^{-1} = \{\, (x/a, \langle\sigma_1\rangle),\ (y/a, \langle\sigma_2\rangle) \,\}$ which means a from σ_1 is mapped to x and a from σ_2 is mapped to y.

ANTI-COMBINATION

- Let $\sigma = \theta_1 + \cdots + \theta_n$ be a combination of $\theta_1,\ldots,\theta_n$. Then the inverse substitution σ^{-1} is called an **anti-combination** of $\theta_1,\ldots,\theta_n$.

- Let $\Sigma=\{ A_1,\ldots, A_n \}$ be a set of atoms, $\tau_{1k}=(\sigma_{1k}, \lambda_{1k})$ $(2\leq k \leq n)$ an msau of $\{ A_1, A_k \}$ s.t. $D(\tau_{1i}) \cap D(\tau_{1j})=\emptyset$ $(1 \leq i, j \leq n; i\neq j)$. Then $lg(\Sigma)=A_1\theta^{-1}$ where $\theta = \sigma_{12} + \cdots + \sigma_{1n}$ (modulo variable renaming).

EXAMPLE: ANTI-UNIFICATION



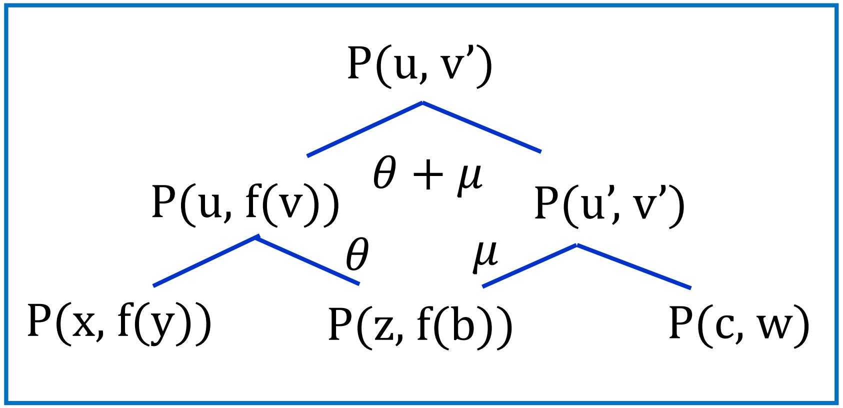
Σ={ P(x,f(y)), P(z,f(b)), P(c,w) }

1. lg({P(x,f(y)), P(z,f(b))}) = P(u,f(v)) and msau (σ , θ)
   where $\sigma = \{ x/u, y/v \}$ and $\theta = \{ z/u, b/v \}$.

2. lg({P(u,f(v)), P(c,w)}) = P(x',y') and msau (λ, δ)
   where $\lambda = \{ u/x', f(v)/y' \}$ and $\delta = \{ c/x', w/y' \}$.

3. lg(Σ)=P(x',y') and msau ($\lambda\sigma$, $\lambda\theta$, δ) where
   $\lambda\sigma = \{ x/x', f(y)/y' \}$ and $\lambda\theta = \{ z/x', f(b)/y' \}$.

EXAMPLE: ANTI-COMBINATION



1. lg({P(x,f(y)), P(z,f(b))}) = P(u,f(v)) and $\theta = \{$ z/u, b/v $\}$.
   lg({P(z,f(b)), P(c,w)}) = P(u',v') and $\mu = \{$ z/u', f(b)/v' $\}$.

2. $\theta + \mu = \{$ z/u, b/v , z/u', f(b)/v' $\}$ and
   $(\theta + \mu)^{-1} = \{$ (u/z, $\langle\theta\rangle$), (v/b, $\langle\theta\rangle$) , (u'/z, $\langle\mu\rangle$), (v'/f(b), $\langle\mu\rangle$) $\}$.

3. Applying $(\theta + \mu)^{-1}$ to P(z,f(b)), lg(Σ)=P(u,v') is obtained.

ALGORITHM: AntiUnif

- **Input**: a set $\Sigma = \{ A_1, \ldots, A_n \}$ ($n \geq 2$) of compatible atoms
- **Output**: least generalization of Σ

1. Put G:= Σ[1] where Σ[i] means the i-th element of Σ.
2. Put i:=2; while $i \leq n$ do;

   Compute G:=lg({G, Σ[i]}) by the anti-unification algorithm.†

   Put i:= i+1.
3. Return G.

† This is the algorithm by Plotkin/Reynolds (1970), which is reformulated by N-Cheng & de Wolf (1997).

ALGORITHM: AntiComb

- **Input**: a set $\Sigma = \{ A_1, \ldots, A_n \}$ (n$\geq$2) of compatible atoms
- **Output**: least generalization of Σ

1. Put $\theta := \varepsilon$ (empty substitution)
2. Put i:=2; while i $\leq$ n do;
   Compute $G_i := lg(\{ A_1, A_i \})$ by the anti-unification algorithm.
   Get a substitution θ_i s.t. $A_1 = G_i\theta_i$ and $D(\theta_i) \cap D(\theta) = \emptyset$.
   Put $\theta := \theta + \theta_i$ and i:= i+1.
3. Compute the inverse substitution θ^{-1} .
4. Compute G= $A_1\theta^{-1}$ and return G.

! When k ($\geq$2) processors are available, Step 2 is split into k procedures, and combination is computed in parallel.

COMPLEXITY

- Complexity of the anti-unification algorithm is $O(N \log N)$ (Kostylev & Zakharov, 2008) where N is the size of the lub of θ_1 and θ_2 .
- Using the result, the complexity of **AntiUnif** is $O(n \times N \log N)$ where n is the number of atoms in Σ.
- Step 2 of **AntiComb** is also done in $O(n \times N \log N)$. If k processors are available, the lower bound of computation is given as $O(\frac{n \times N \log N}{k})$.

EXPERIMENTAL EVALUATION
GENERATING TEST DATA

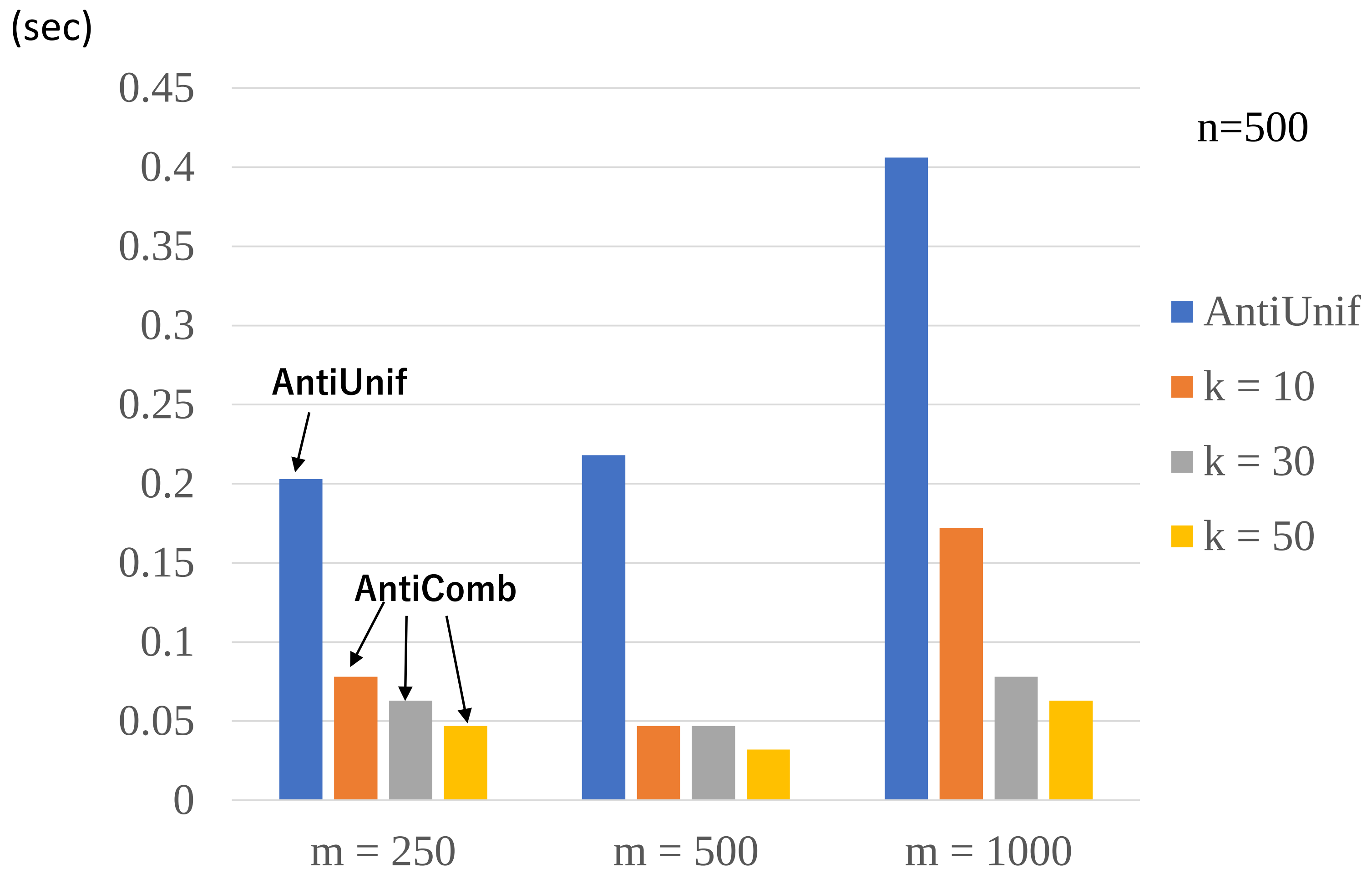
- A set **Prog** of atoms is randomly created.
- Each atom in **Prog** has the same ternary predicate P and is of the form $P(t_1,t_2,t_3)$ where t_i ($i=1,2,3$) are terms.
- The depth of each atom in **Prog** is ≤ 5.
- For any $P(t_1,t_2,t_3)$ in **Prog**, if a function *f* appears in the outermost of the term t_i, then the outermost function appearing in the corresponding term s_i of another atom $P(s_1,s_2,s_3)$ in **Prog** is set to the same function *f*.

EXPERIMENTAL EVALUATION
ENVIRONMENT AND SETTING

- Compare runtime of **AntiUnif** and **AntiComb**.
- Implementation language: Maple 2018, 64bit
- Environment: Intel® CPU 2GHz, RAM 8GB, Win10/64bit
- Parameters: the number of atoms in **Prog** is set to: $n = 500, 1000, 3000, 5000, 10000$; the number of functions in **Prog** is set to $m = n/2$, n, and $2n$.
- In **AntiComb**, the number of processors is set to $k = 10$, 30, and 50.

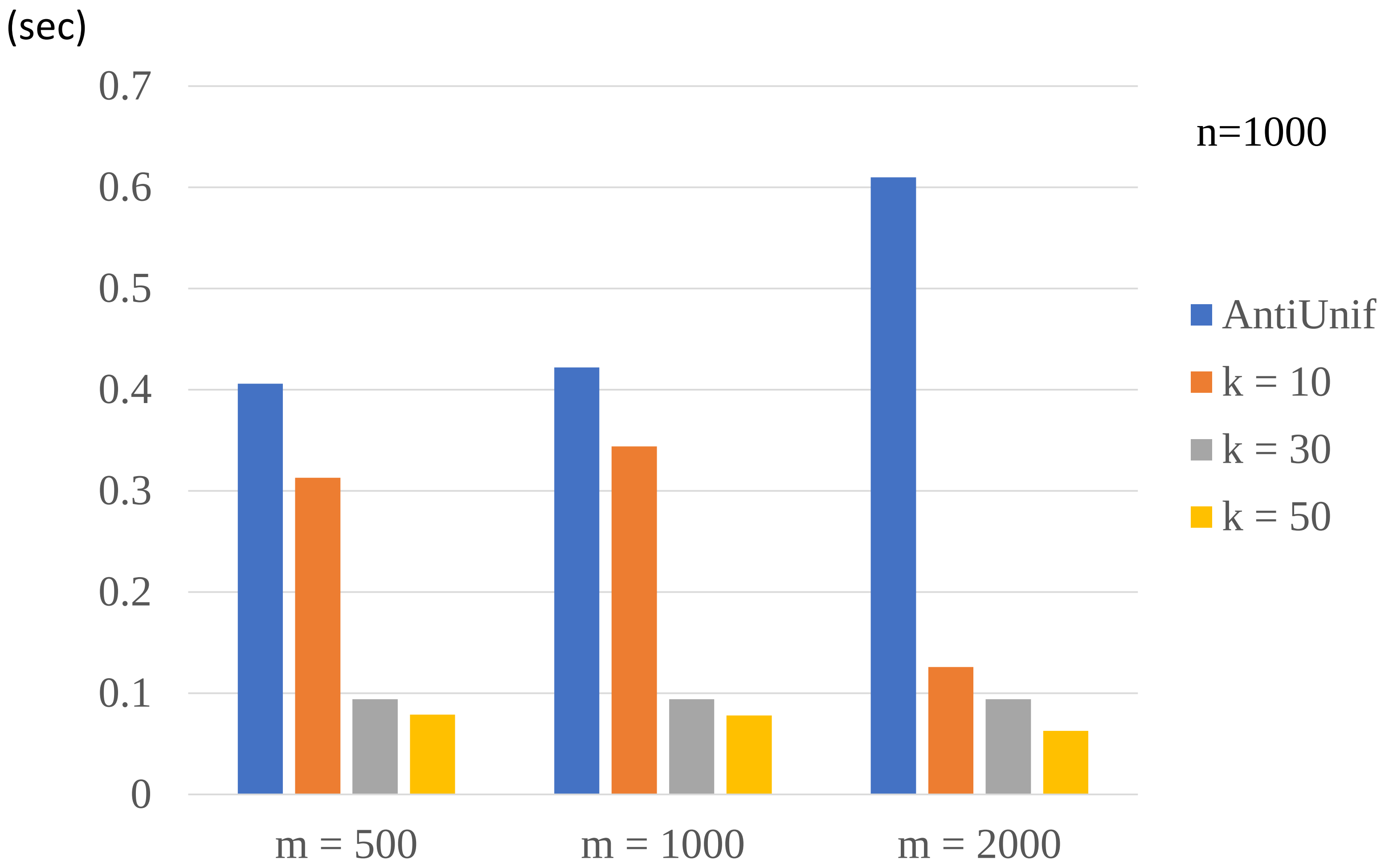
EXPERIMENTAL EVALUATION

RESULTS (500 ATOMS)

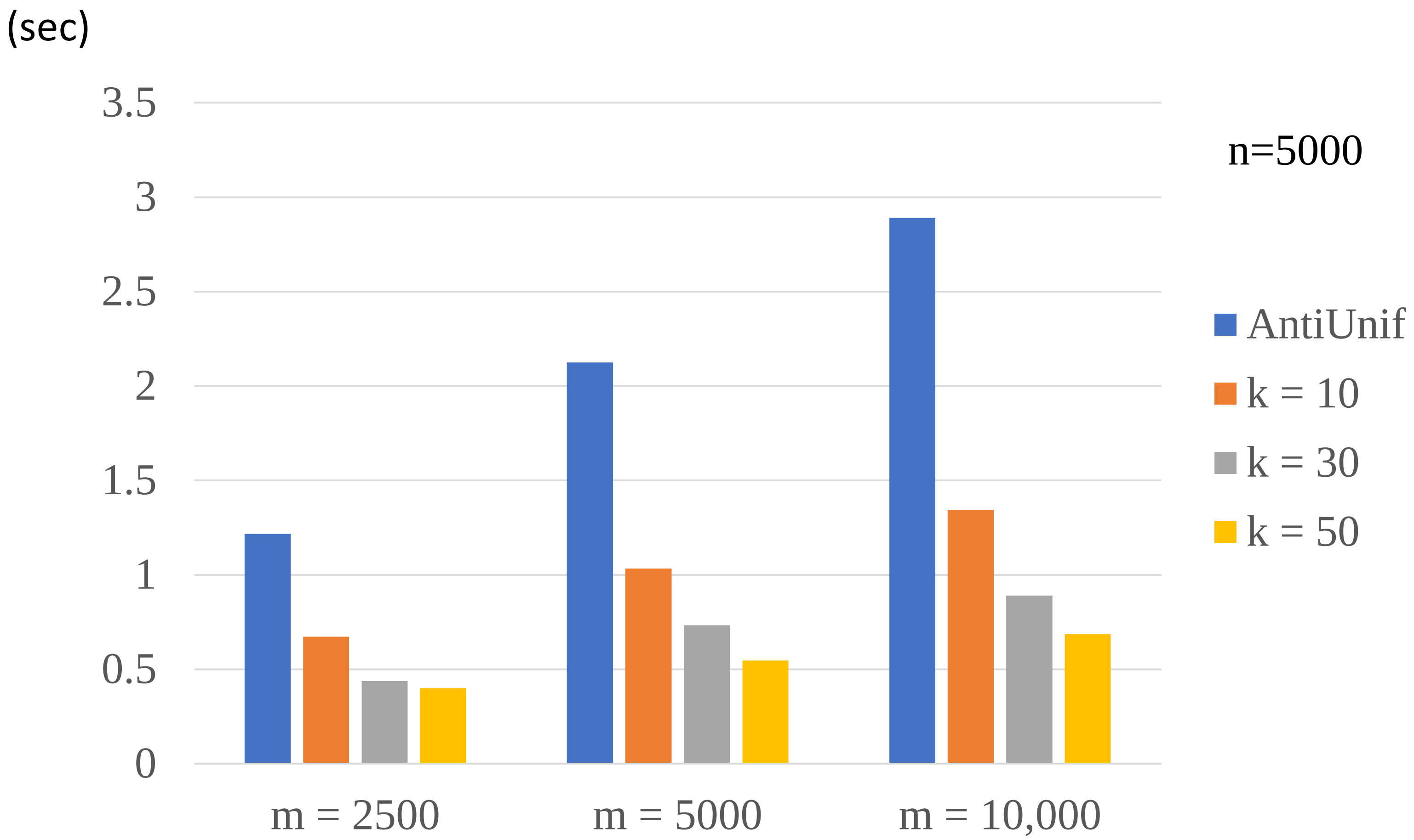


EXPERIMENTAL EVALUATION

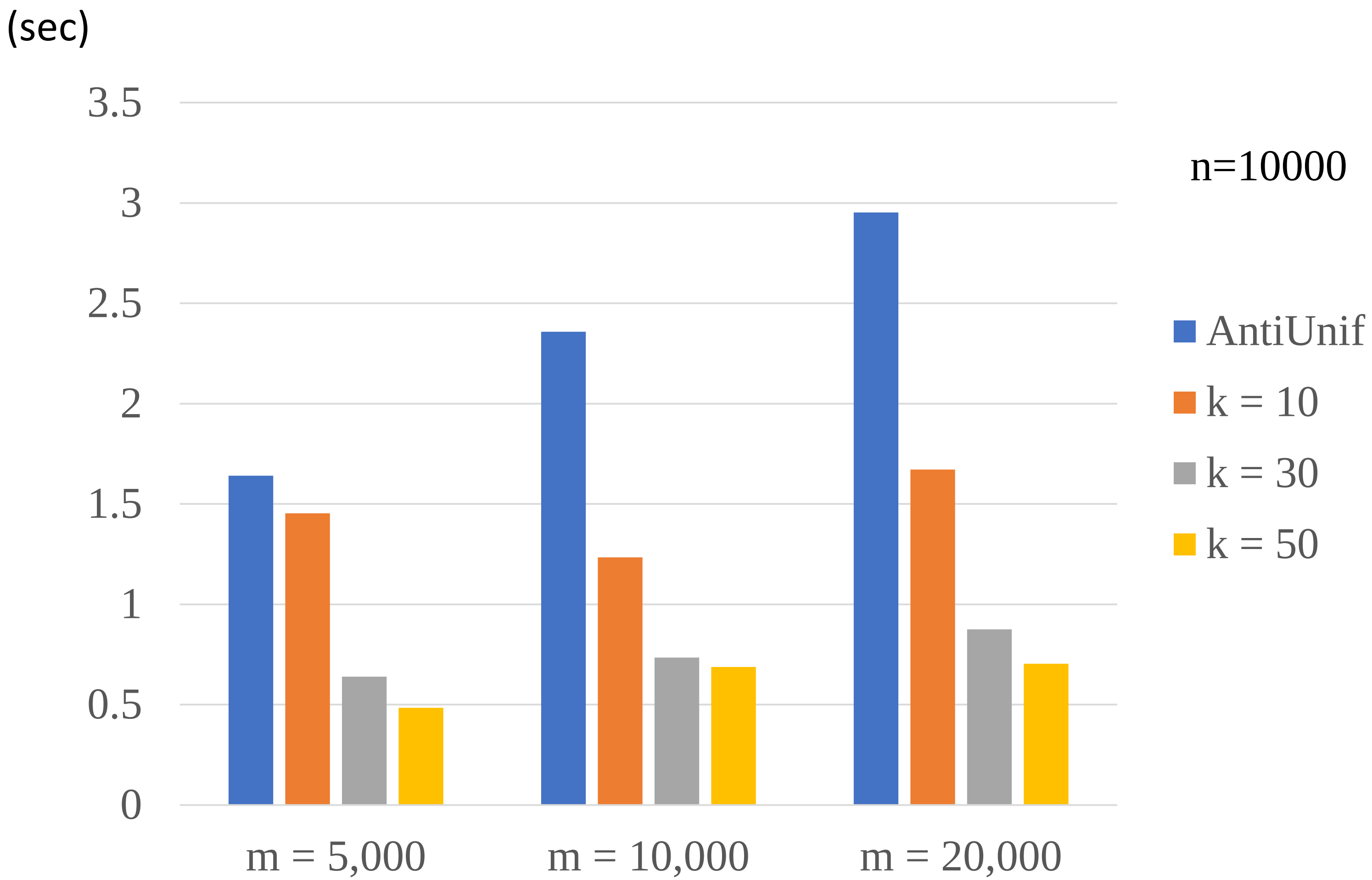
RESULTS (1000 ATOMS)



EXPERIMENTAL EVALUATION
RESULTS (5000 ATOMS)



EXPERIMENTAL EVALUATION
RESULTS (10000 ATOMS)



DISCUSSION

- In **AntiComb**, combination is computed in parallel, while inverse substitution and lg is computed in serial.
- We compute runtime for *k*-parallel processing by $\max\{t_1,\ldots,t_k\}$ where t_i is time for computing combination by each processor.
- These factors make the speedup of **AntiComb** seemingly smaller than the number of processors used.

DISCUSSION

- Kuper et al. (1992) represent terms in trees and show that
  - anti-unification of 2 terms of size n is computed in time $O(\log^2 n)$ using n processors.
  - anti-unification of m terms, each having at most $O(n)$ symbols, is computed in time $O(\log mn \times \log^2 n)$ using mn processors.
- If we use their anti-unification algorithm of two atoms in **AntiComb**, anti-unification of m atoms takes $O(m \times \log^2 n)$ using n processors. Using mn processors, it is done in $O(\log^2 n)$.
- Hence, **AntiComb** will be faster than anti-unification of m atoms by Kuper's method.

CONCLUSION

- We introduced a new algorithm for computing least generalization of a set of atoms based on anti-combination.
- Experimental results show that the proposed algorithm has potential to compute **induction from big data** in the form of **relational facts** in parallel.
- Future study includes exploiting further opportunities for parallelization in practical ILP applications.