香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Rapids @ HKUST

Efficient Parallel Subgraph Enumeration on a Single Machine

Shixuan Sun, *Yulin Che, Lipeng Wang, and Qiong Luo**

The Hong Kong University of Science and Technology

Outline

- Background
- Basic Subgraph Enumeration Algorithm
- Lazy Materialization Subgraph Enumeration
- Evaluation
- Conclusions

Subgraph Isomorphism

Given unlabeled graphs $g = (V, E)$ and $g' = (V', E')$, a subgraph isomorphism from g to g' is an injective function $\varphi: V \to V'$ such that $\forall e(u, u') \in E, e(\varphi(u), \varphi(u')) \in E'$.

Problem Definition

Given a data graph G and a pattern graph P, subgraph enumeration finds all subgraphs in G that are isomorphic to P.

Existing Algorithms on a Single Machine

- DUALSIM partitions data graphs that cannot fit in memory.
- EmptyHeaded utilizes the worst-case optimal join to enumerate subgraphs.

Algorithms	Environment	Year Published
DUALSIM [7]	Single Machine (parallel)	SIGMOD 2016
EmptyHeaded [8]	Single Machine (parallel)	TODS 2017

Existing Distributed Algorithms

Distributed algorithms adopt the parallel join method.

1. Decompose P into a collection of small components.
2. Join the matches of the components in parallel.

Algorithms	Distributed Environment	Year Published
Afrati [1]	MapReduce	ICDE 2013
PSgL [2]	Giraph	SIGMOD 2014
TwinTwig [3]	MapReduce	VLDB 2015
SEED [4]	MapReduce	VLDB 2016
CRYSTAL [5]	MapReduce	VLDB 2017
BiGJoin [6]	Timely Dataflow	VLDB 2018

Outline

- Background
- Basic Subgraph Enumeration Algorithm
- Lazy Materialization Subgraph Enumeration
- Evaluation
- Conclusions

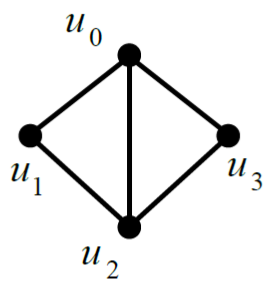
Basic Subgraph Enumeration Algorithm

Input: a data graph G and a pattern graph P.

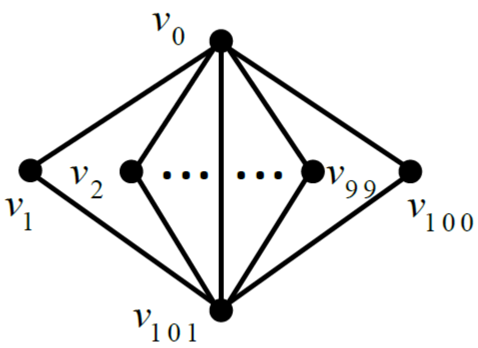
Output: all subgraphs in G that are isomorphic to P.

1. Generate an enumeration order π, which is a permutation of pattern vertices.
2. Enumerate all solutions by recursively extending partial results along π.

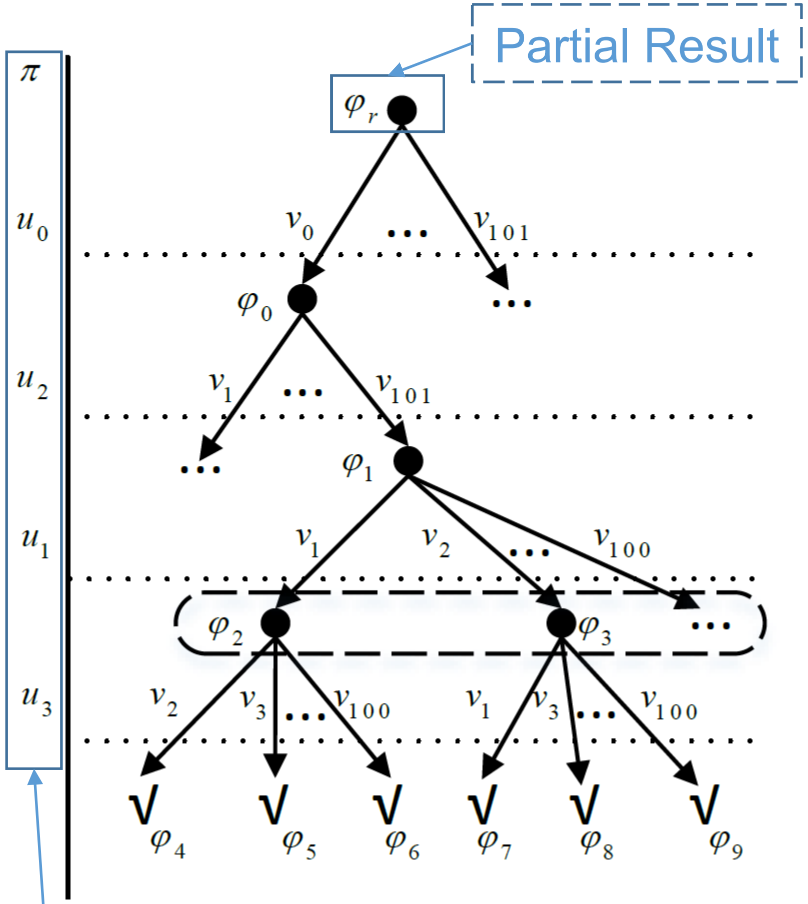
Example of SE



Pattern Graph P.



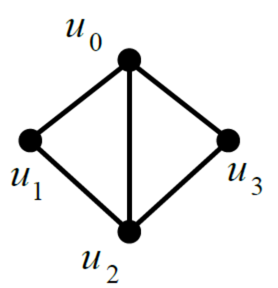
Data Graph G.



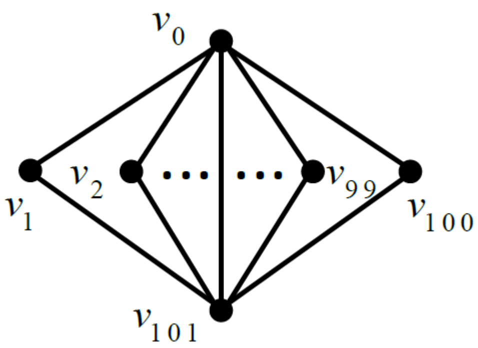
Search Tree of SE.

Enumeration Order

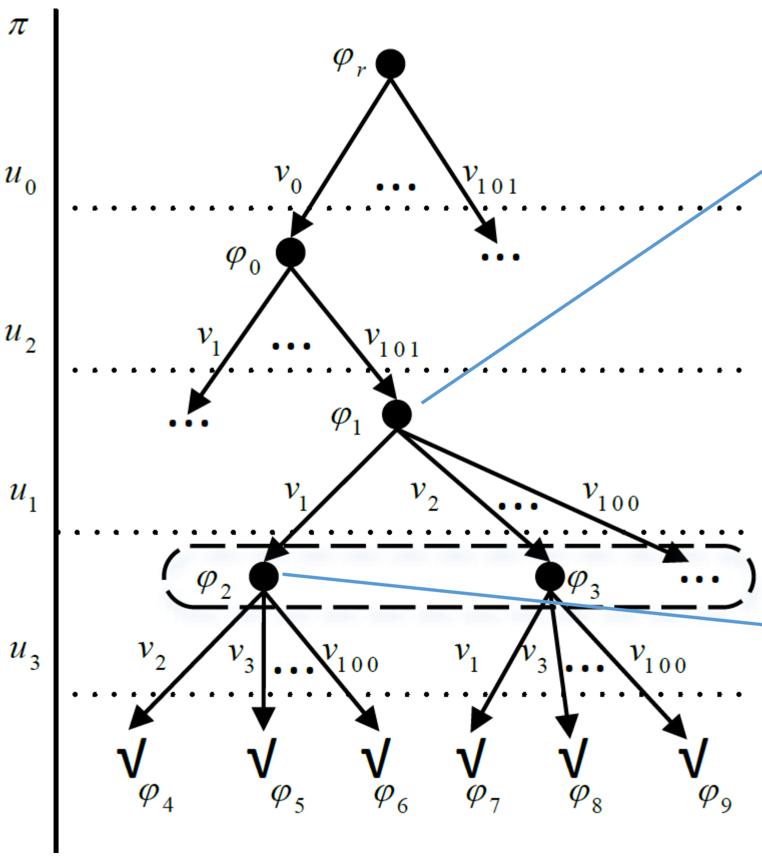
Example of SE



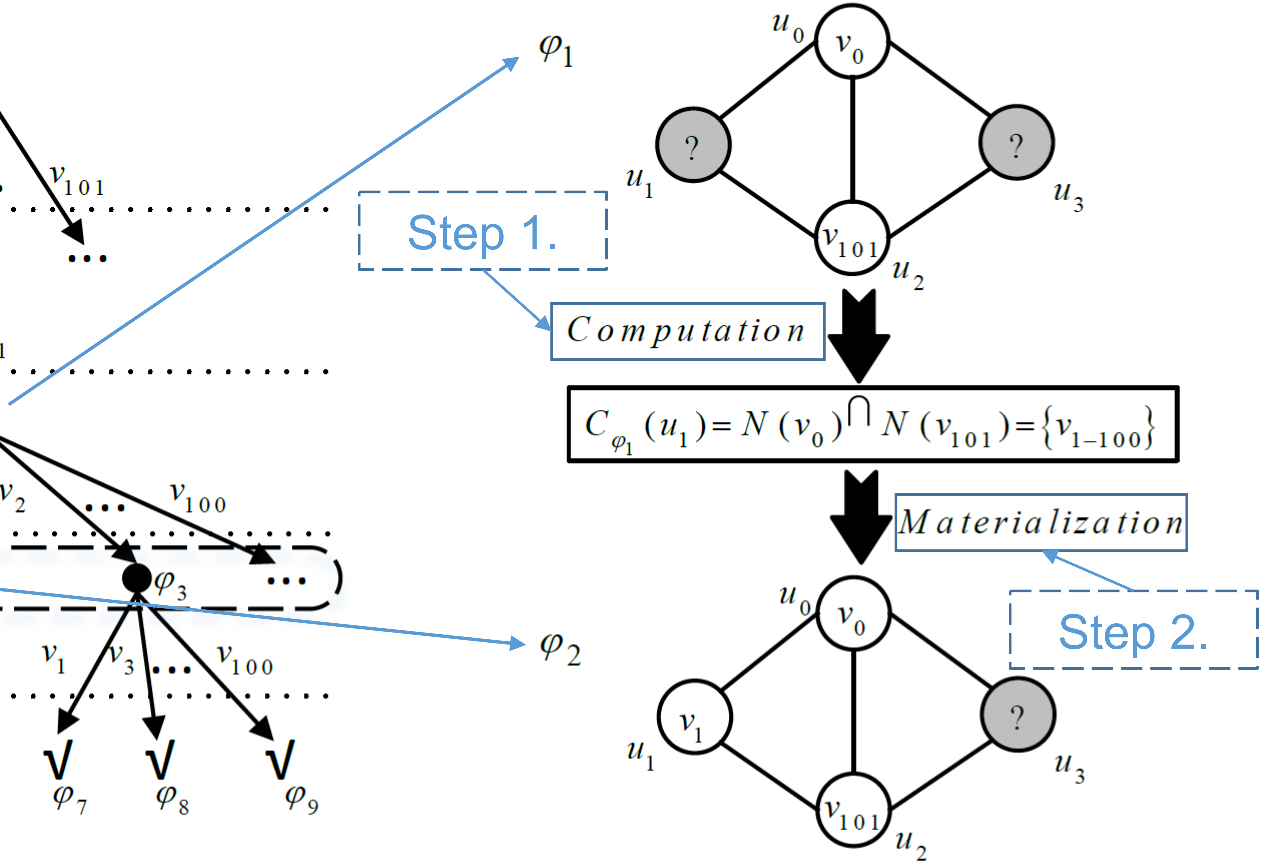
Pattern Graph P.



Data Graph G.



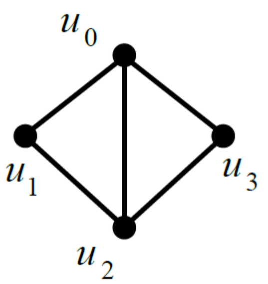
Search Tree of SE.



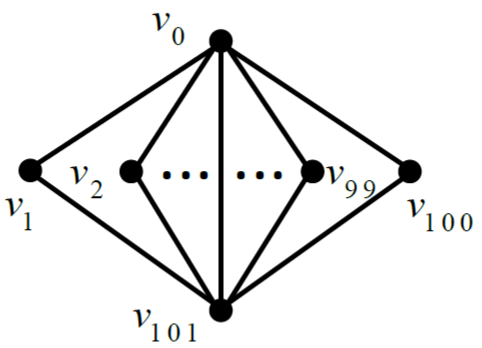
Expand a Partial Result.

We find that there is a large amount of redundant computation in the enumeration.

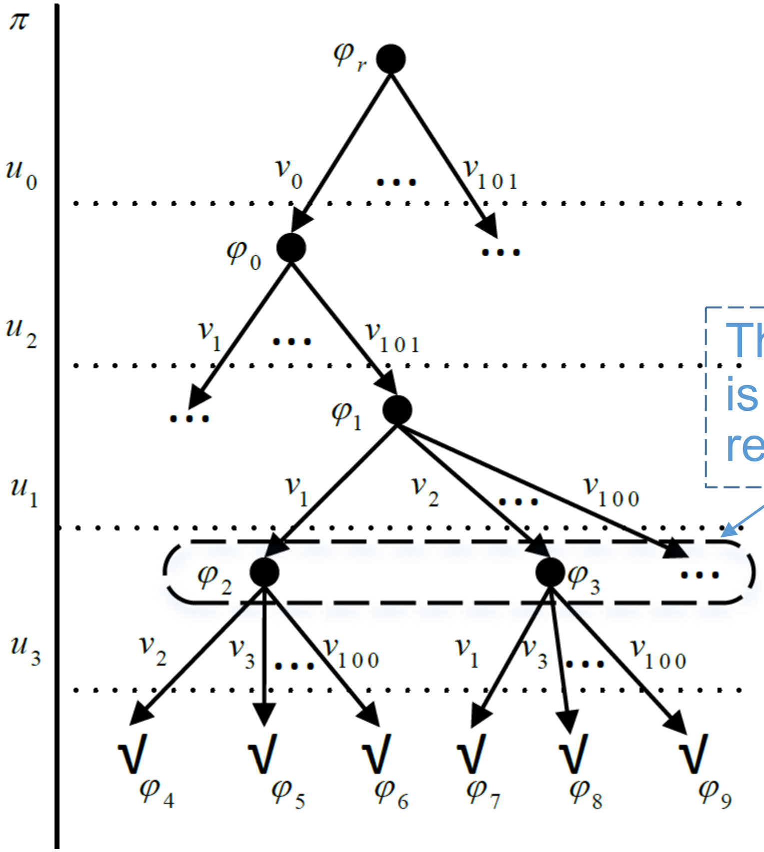
Observation One



Pattern Graph P.



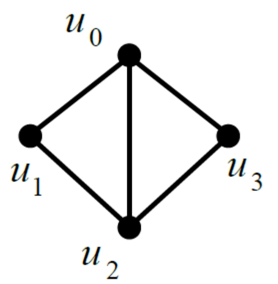
Data Graph G.



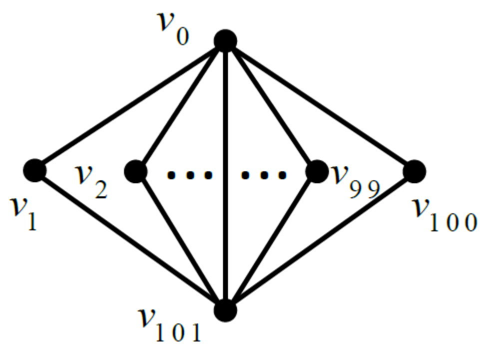
Search Tree of SE.

The same set intersection $N(v_0) \cap N(v_{101})$ is repeated in the computation of partial results in the dashed rectangle for u_3.

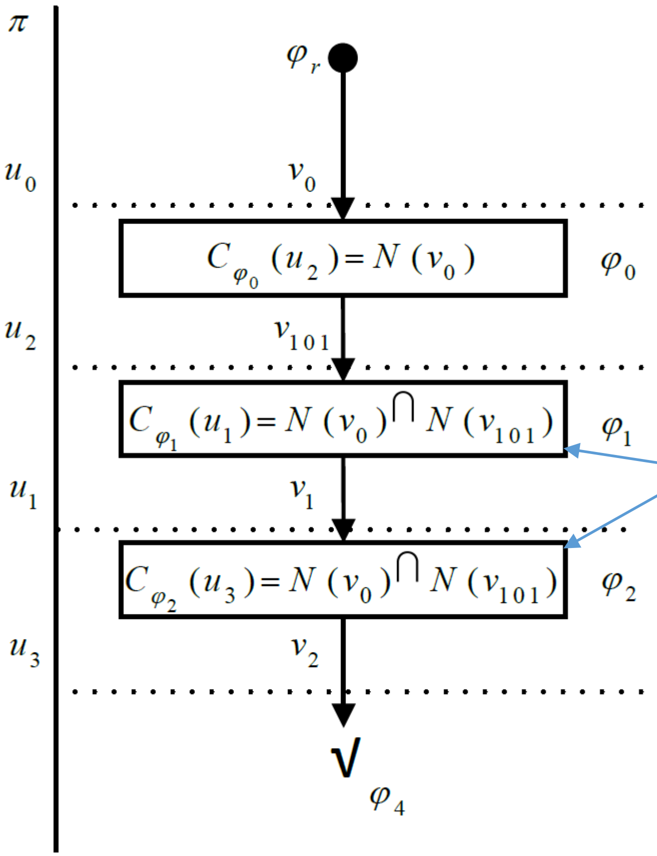
Observation Two



Pattern Graph P.



Data Graph G.



Search Path of SE.

Given partial results φ_1 and φ_2, the same set intersection $N(v_0) \cap N(v_{101})$ is repeated in the computation of candidates of u_1 and u_3.

Outline

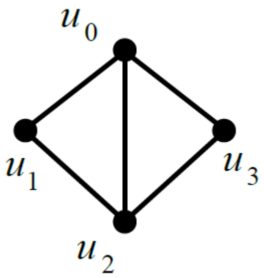
- Background
- Basic Subgraph Enumeration Algorithm
- Lazy Materialization Subgraph Enumeration
- Evaluation
- Conclusions

Lazy Materialization

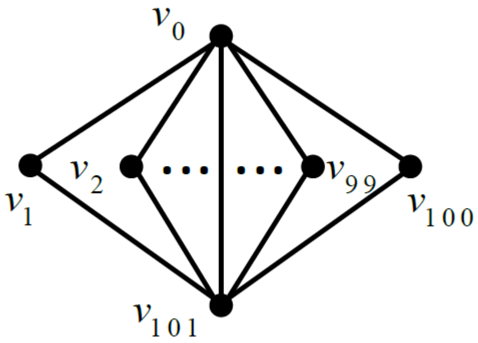
We propose the lazy materialization subgraph enumeration algorithm, called **LIGHT**.

- Separate the computation and the materialization.
- Keep the order of the computation unchanged.
- Delay the materialization until some computation requires it.

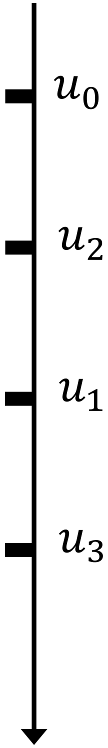
Example of Lazy Materialization



Pattern Graph P.

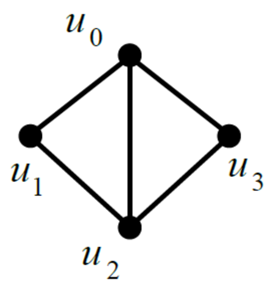


Data Graph G.

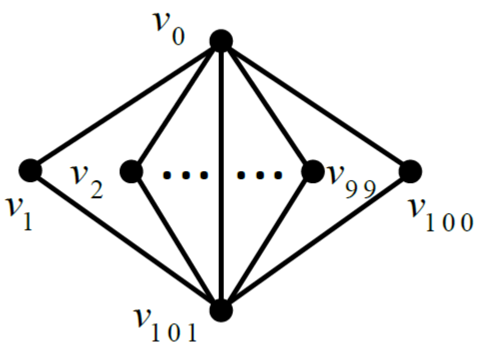


Enumeration Order π.

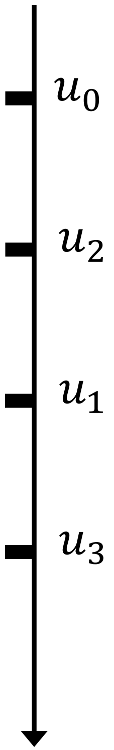
Example of Lazy Materialization



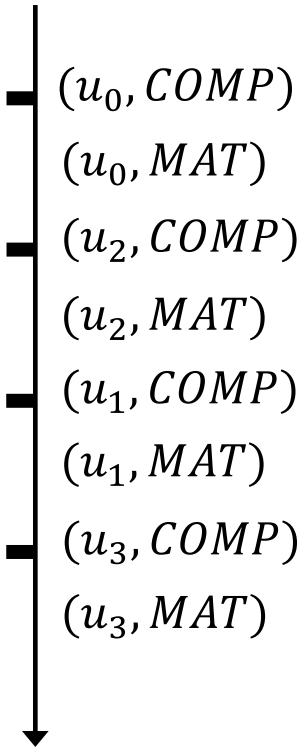
Pattern Graph P.



Data Graph G.

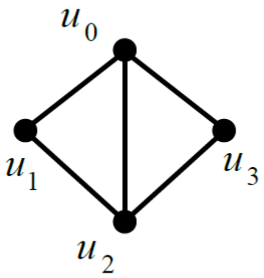


Enumeration Order π.

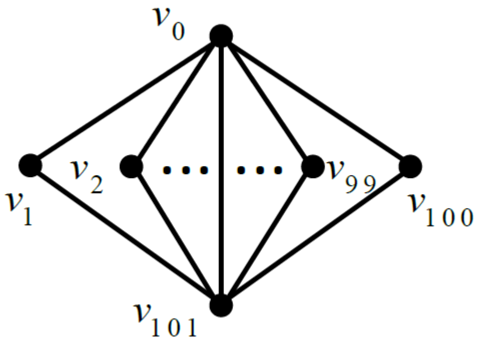


Operation Order of SE.

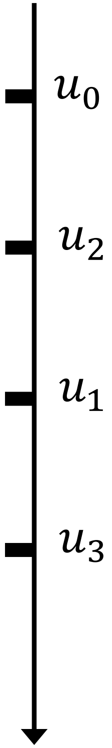
Example of Lazy Materialization



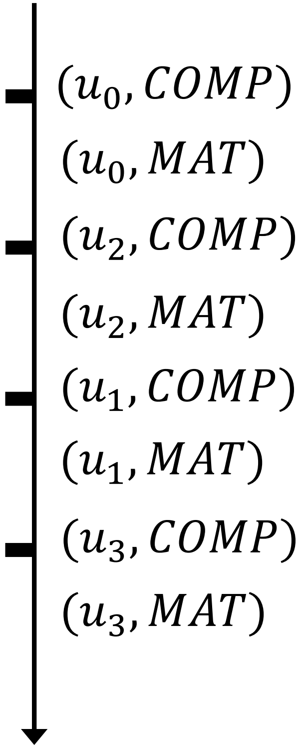
Pattern Graph P.



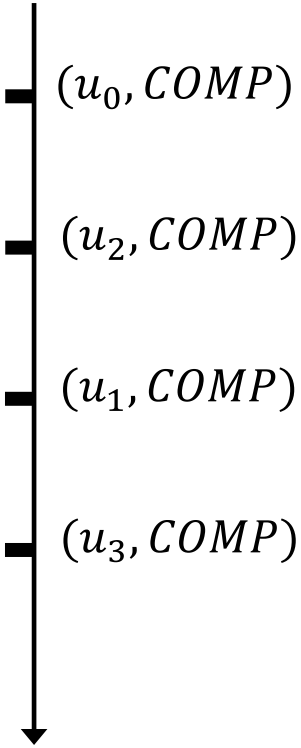
Data Graph G.



Enumeration Order π.

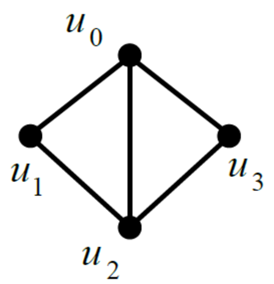


Operation Order of SE.

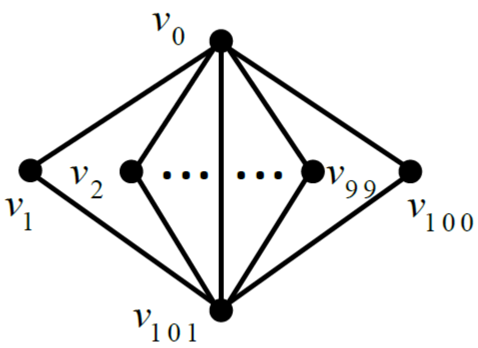


Operation Order of LIGHT.

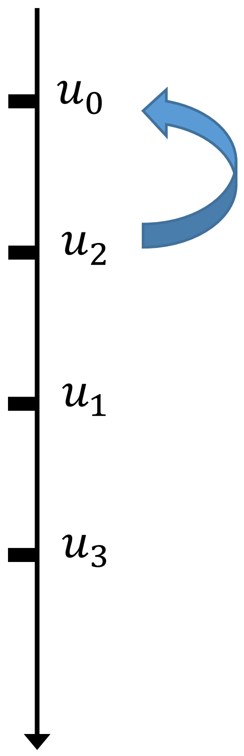
Example of Lazy Materialization



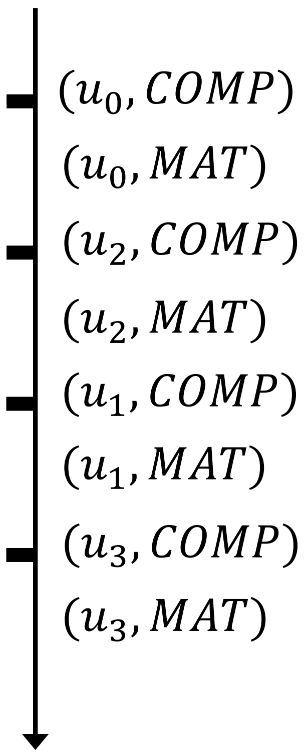
Pattern Graph P.



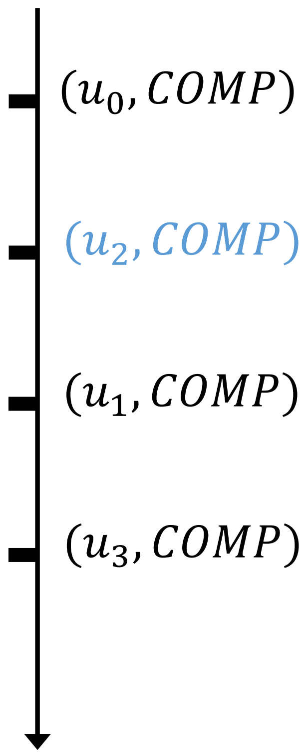
Data Graph G.



Enumeration Order π.

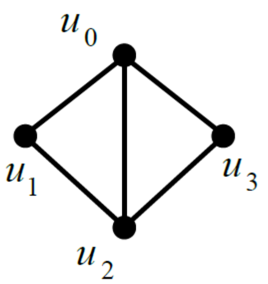


Operation Order of SE.

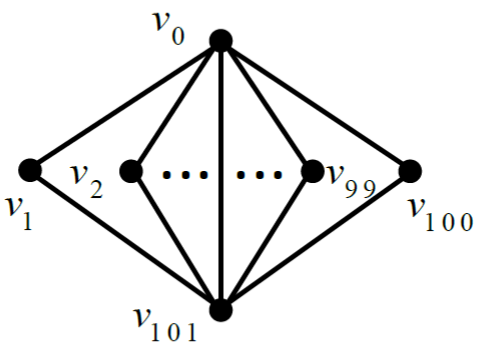


Operation Order of LIGHT.

Example of Lazy Materialization



Pattern Graph P.



Data Graph G.

- u_0
- u_2
- u_1
- u_3

Enumeration Order π.

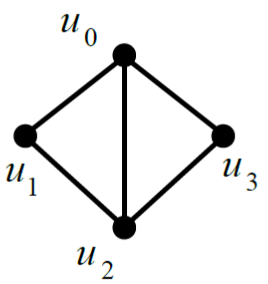
- $(u_0, COMP)$
- (u_0, MAT)
- $(u_2, COMP)$
- (u_2, MAT)
- $(u_1, COMP)$
- (u_1, MAT)
- $(u_3, COMP)$
- (u_3, MAT)

Operation Order of SE.

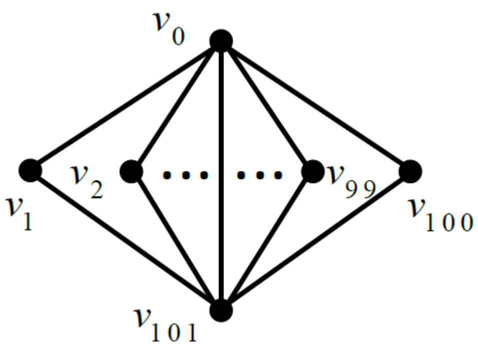
- $(u_0, COMP)$
- (u_0, MAT)
- $(u_2, COMP)$
- $(u_1, COMP)$
- $(u_3, COMP)$

Operation Order of LIGHT.

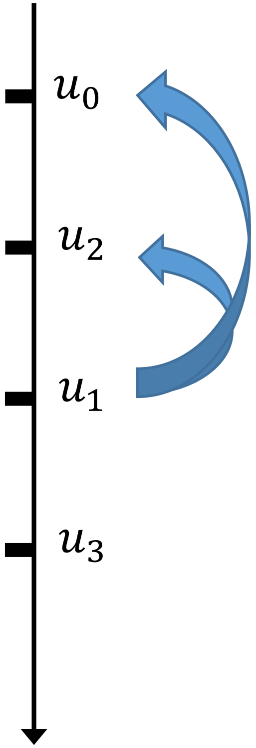
Example of Lazy Materialization



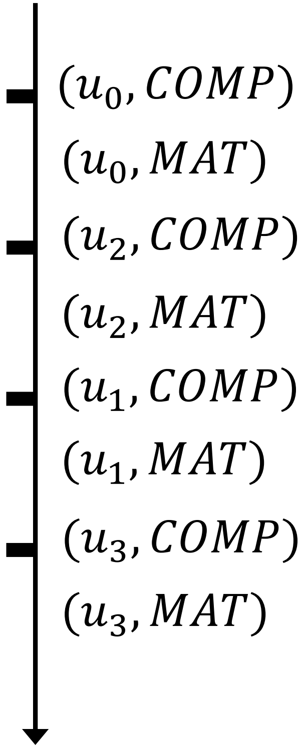
Pattern Graph P.



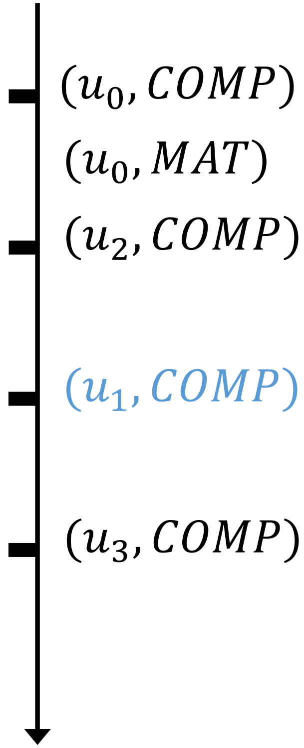
Data Graph G.



Enumeration Order π.

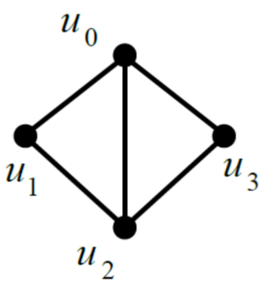


Operation Order of SE.

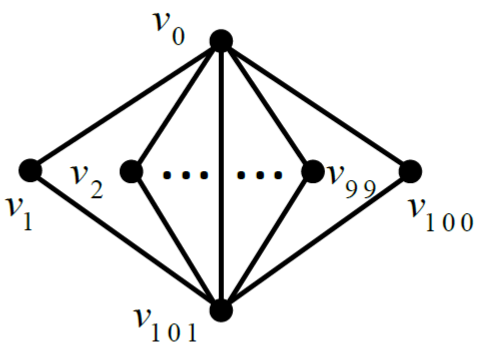


Operation Order of LIGHT.

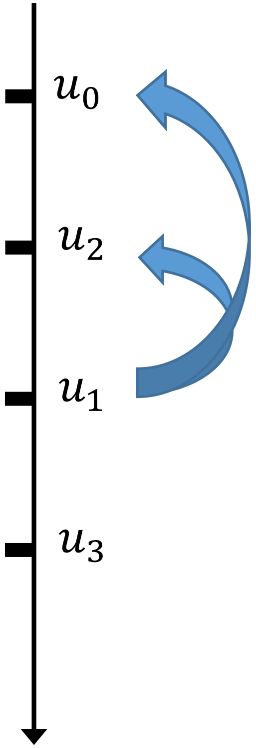
Example of Lazy Materialization



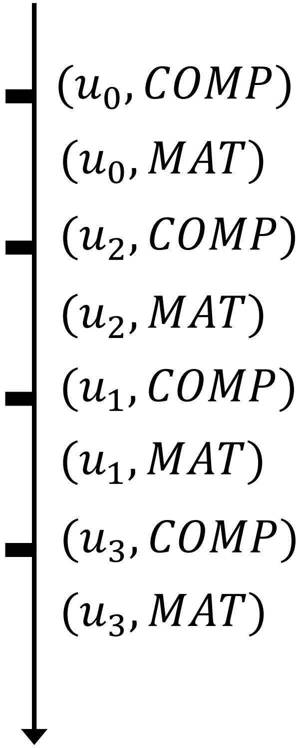
Pattern Graph P.



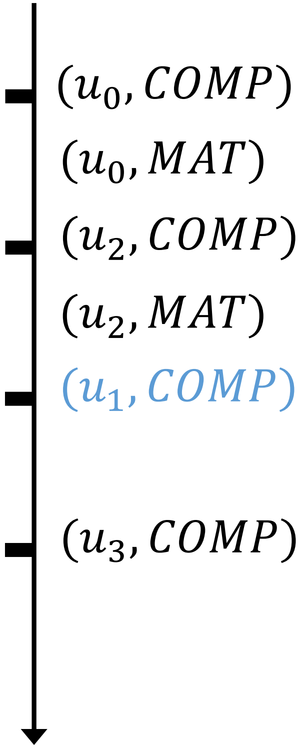
Data Graph G.



Enumeration Order π.

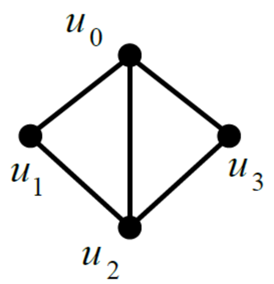


Operation Order of SE.

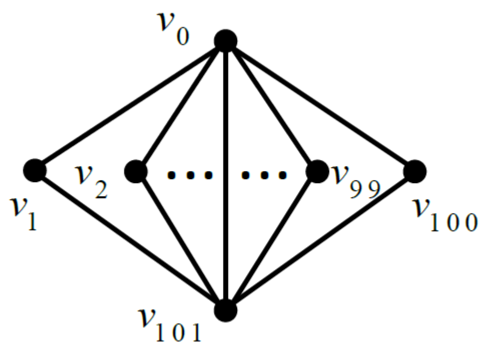


Operation Order of LIGHT.

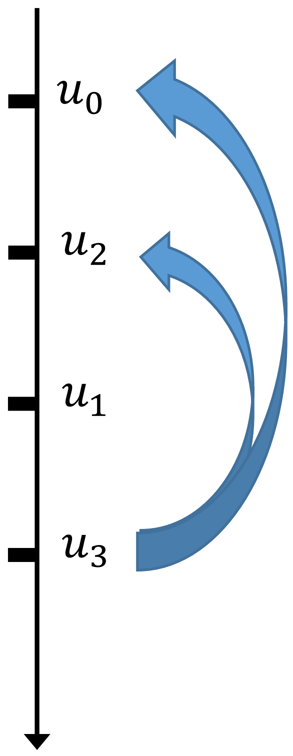
Example of Lazy Materialization



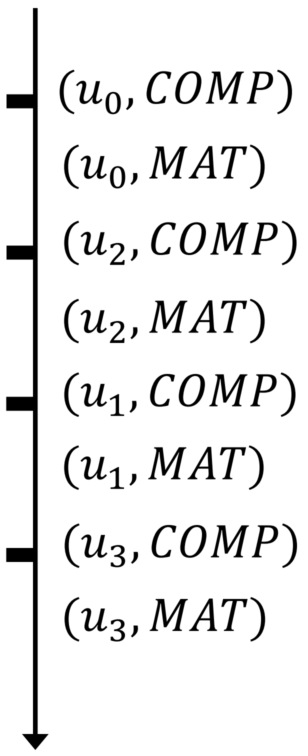
Pattern Graph P.



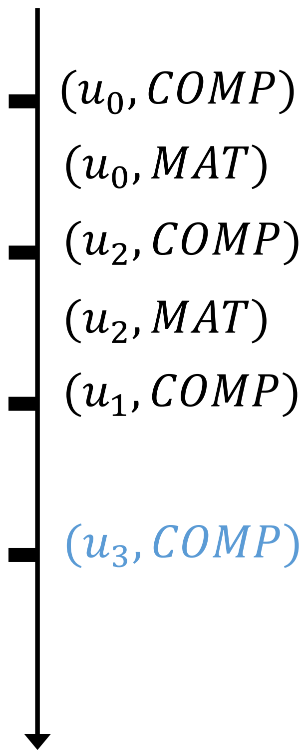
Data Graph G.



Enumeration Order π.

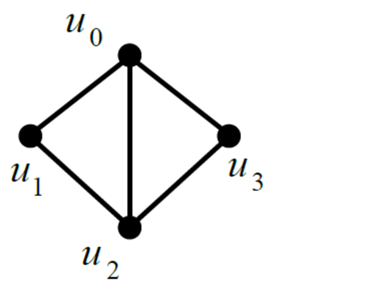


Operation Order of SE.

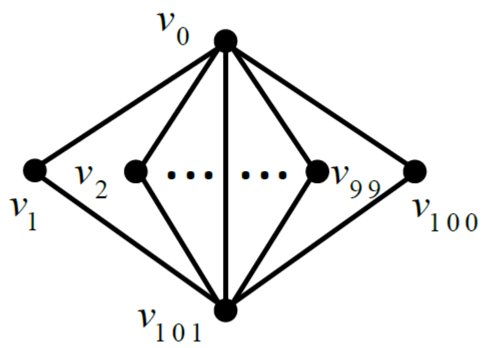


Operation Order of LIGHT.

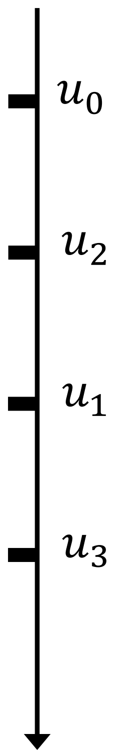
Example of Lazy Materialization



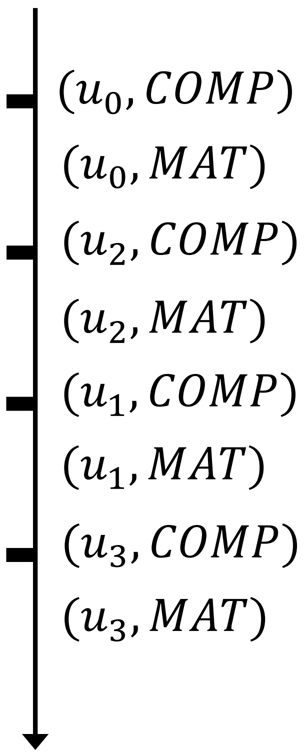
Pattern Graph P.



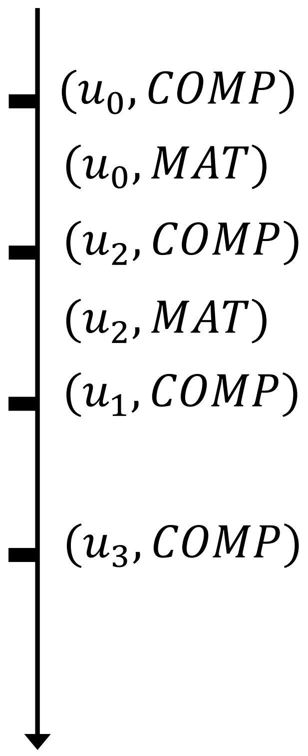
Data Graph G.



Enumeration Order π.

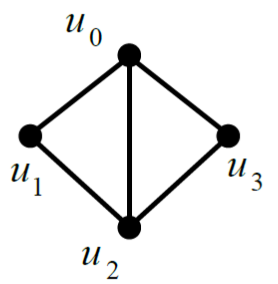


Operation Order of SE.

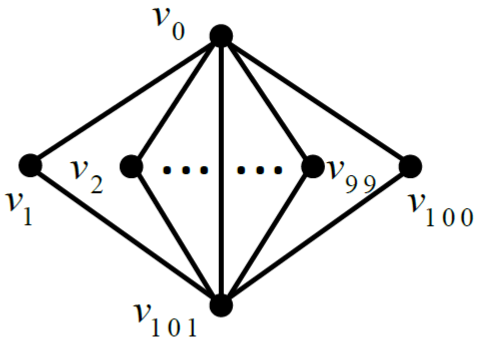


Operation Order of LIGHT.

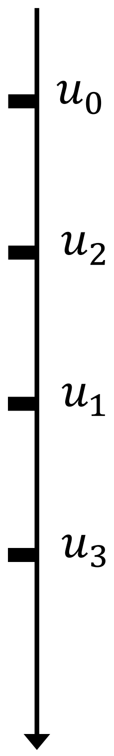
Example of Lazy Materialization



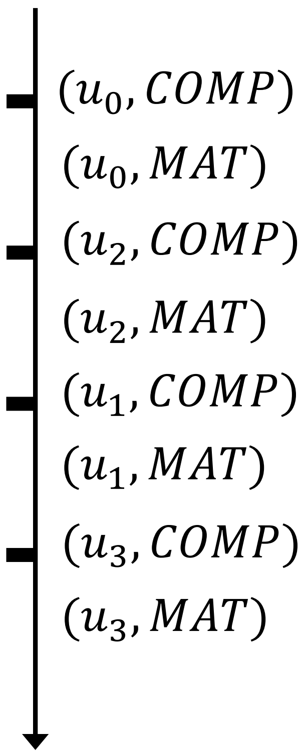
Pattern Graph P.



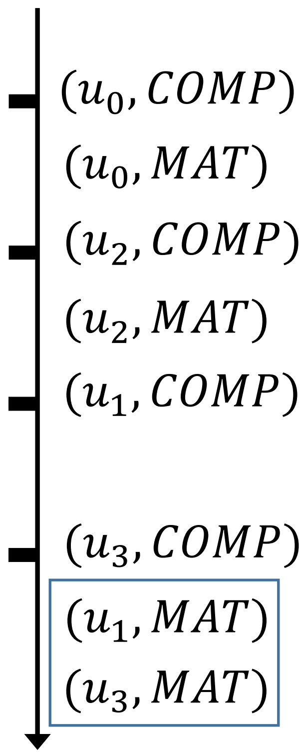
Data Graph G.



Enumeration Order π.

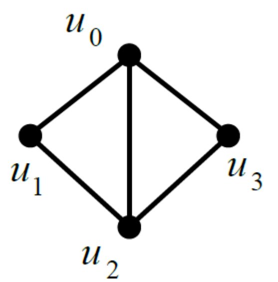


Operation Order of SE.

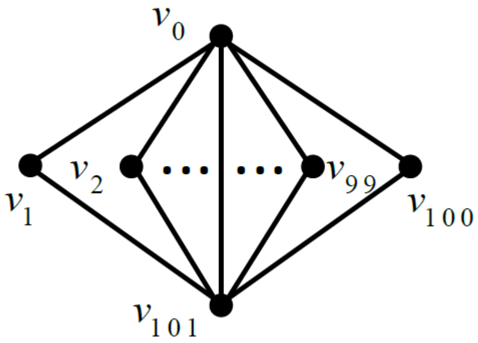


Operation Order of LIGHT.

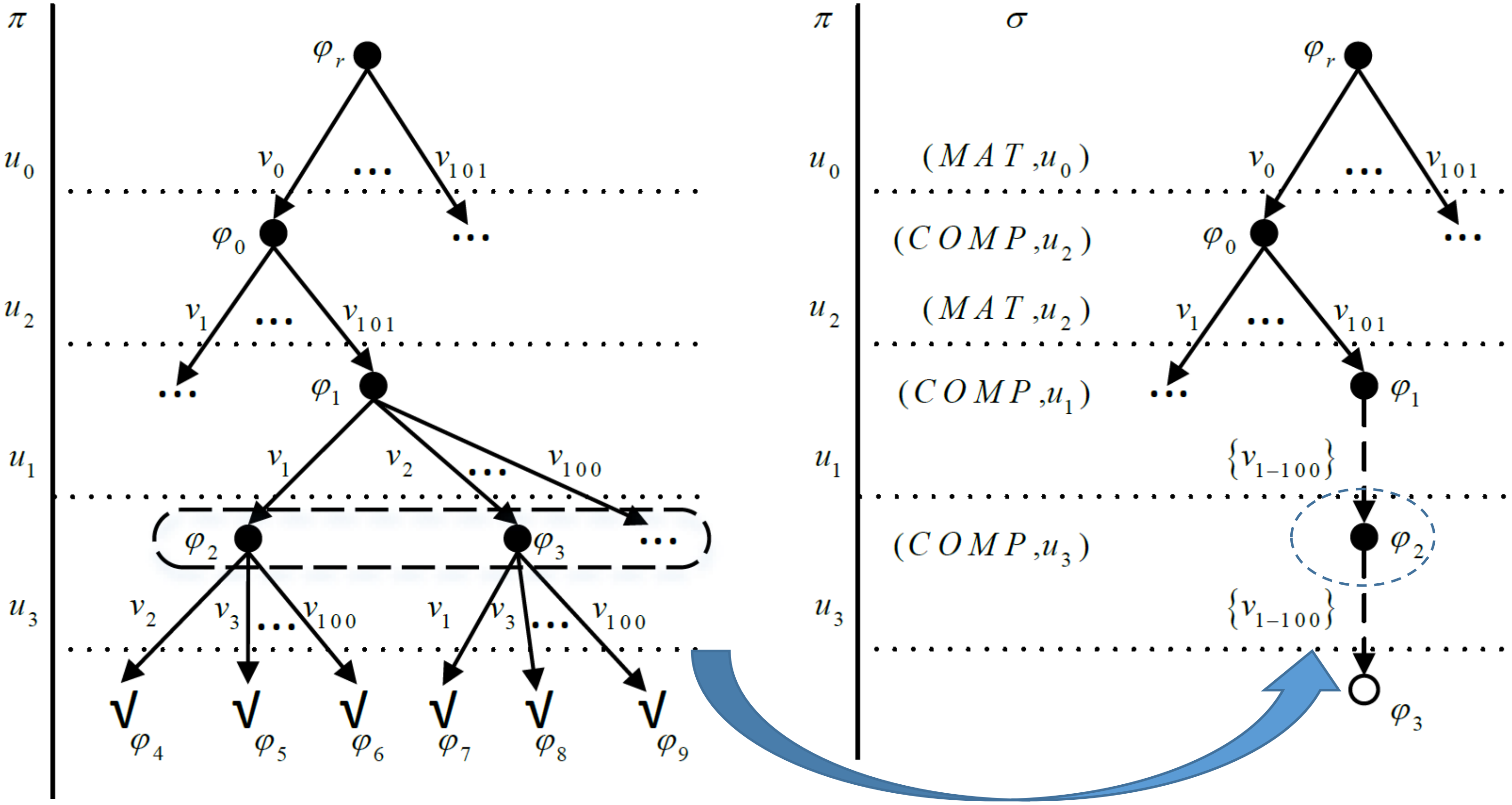
Example of Lazy Materialization



Pattern Graph P.



Data Graph G.



Search Tree of SE.

Search Tree of LIGHT.

MSC based Candidate Sets Computation

Compute the candidate set of $u \in \pi$ by utilizing candidate sets of $u' \in M(u)$ in π.

- Convert it to the minimum set cover (MSC) problem:

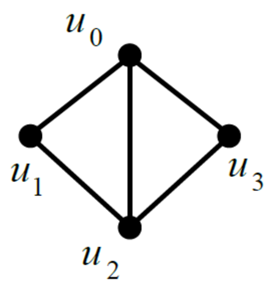
Input: $U = N_+^\pi(u)$, $S = \{\{u'\}|u' \in U\} \cup \{N_+^\pi(u')|N_+^\pi(u') \subseteq N_+^\pi(u) \wedge u' \in M(u)\}$.
Output: The smallest sub-collection S' of S whose union equals U.

Notation:

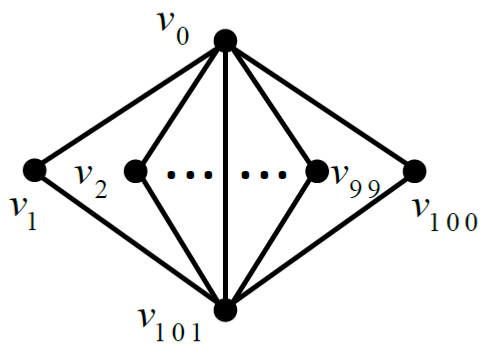
1. The backward neighbors $N_+^\pi(u)$ of u contains the neighbors of u positioned before u in π.

2. $M(u)$ contains all pattern vertices before u in π.

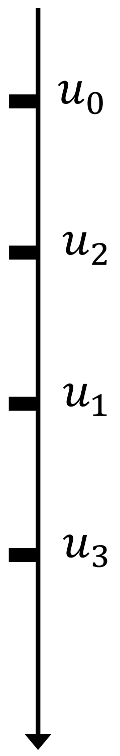
Example of MSC



Pattern Graph P.



Data Graph G.



Enumeration Order π.

$N_{+}^{\pi}(u_3) = \{u_0, u_2\}$

$M(u_3) = \{u_0, u_1, u_2\}$

MSC Input:

$U = \{u_0, u_2\}$ $\quad N_{+}^{\pi}(u_1)$

$S = \{\{u_0\}, \{u_2\}, \{u_0, u_2\}\}$

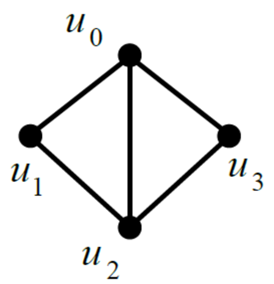
MSC Output:

$S' = \{\{u_0, u_2\}\}$

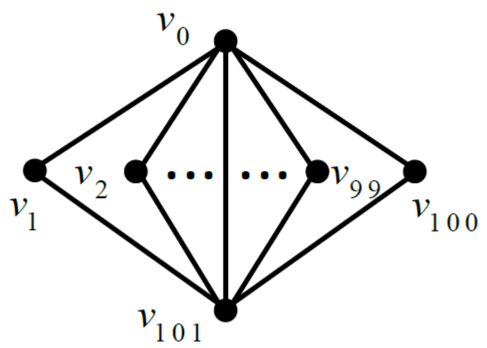
$C_{\varphi}(u_3) = C_{\varphi}(u_1)$

Compute Candidate Set of u_3.

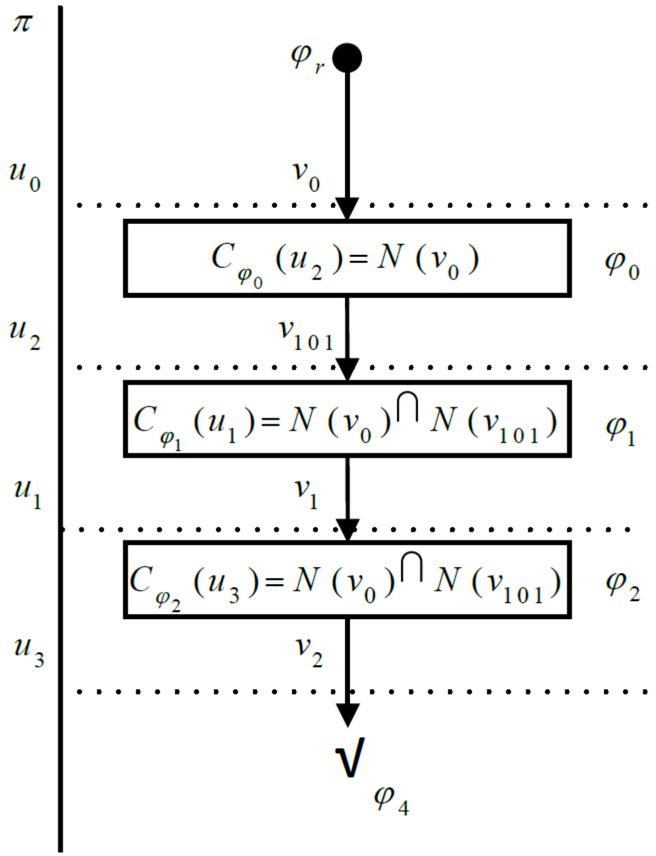
Example of MSC



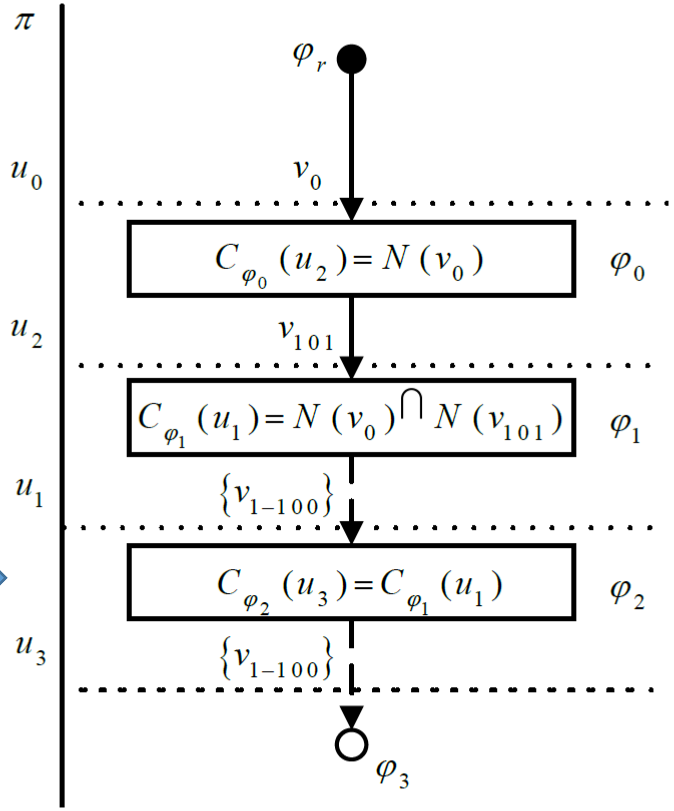
Pattern Graph P.



Data Graph G.



Search Path of SE.



Search Path of LIGHT.

Parallel Implementation

Utilize both vector registers and multiple cores in modern CPUs.

- Parallelize set intersections with SIMD (Single-Instruction-Multiple-Data) instructions.
- Parallelize the exploration of the search tree with multi-threading.

Outline

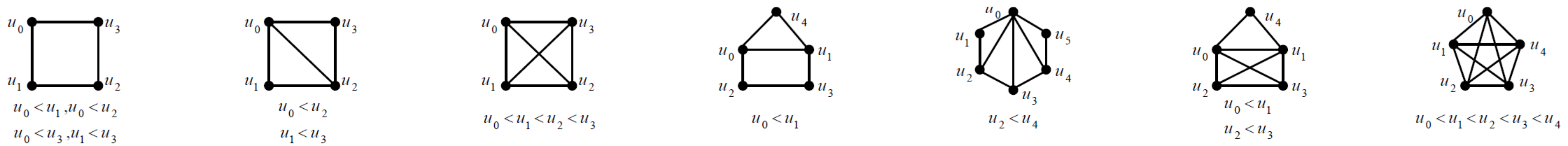
- Background
- Basic Subgraph Enumeration Algorithm
- Lazy Materialization Subgraph Enumeration
- Evaluation
- Conclusions

Datasets

Real-world Datasets.

Dataset	Name	N (million)	M (million)	Memory (GB)
youtube	*yt*	3.22	9.38	0.09
eu-2005	*eu*	0.86	19.24	0.15
live-journal	*lj*	4.85	68.48	0.53
com-orkut	*ot*	3.07	117.19	0.89
uk-2002	*uk*	18.52	298.11	2.30
friendster	*fs*	65.61	1,806.07	13.71

Pattern Graphs.



(a) P_1. **(b)** P_2. **(c)** P_3. **(d)** P_4. **(e)** P_5. **(f)** P_6. **(g)** P_7.

Experimental Environment.

- Implemented in C++ and compiled with icpc 16.0.0.
- A machine equipped with 20 cores (2 Intel Xeon E5-2650 v3 @ 2.30GHz CPUs), 64GB RAM and 1TB HDD.
- Use the AVX2 (256-bit) instruction set and execute with 64 threads.

Comparison with SE

- T_{SE} and T_{LIGHT} are the serial execution time of SE and LIGHT respectively.
- T_{SE+P} and $T_{LIGHT+P}$ are the parallel execution time of SE and LIGHT respectively.
- $Overall\ Speedup = \frac{T_{SE}}{T_{LIGHT+P}}$.

Dataset	*yt*			*lj*		
Pattern	P_2	P_4	P_6	P_2	P_4	P_6
T_{SE}	645	176,181	4,448	677	232,800	34,090
T_{SE+P}	22	4,034	115	15.9	6,949	1,425
T_{LIGHT}	31	3,309	43	26	3,497	285
$T_{LIGHT+P}$	0.3	56	0.9	0.9	80	8.7
Speedup	2,150X	3,146X	4,942X	752X	2,910X	3,918X

Comparison with SE (seconds).

Conclusions

We propose an efficient parallel subgraph enumeration algorithm LIGHT for a single machine.

- Reduce the redundant computation by the lazy materialization and the MSC based candidate sets computation.
- Parallelize LIGHT with both SIMD and multi-threading to fully utilize the parallel computation capabilities in modern CPUs.

Selected References

[1]. F. N. Afrati, D. Fotakis, and J. D. Ullman. Enumerating subgraph instances using map-reduce. In ICDE, 2013.

[2]. Y. Shao, B. Cui, L. Chen, L. Ma, J. Yao, and N. Xu. Parallel subgraph listing in a large-scale graph. In SIGMOD, 2014.

[3]. L. Lai, L. Qin, X. Lin, and L. Chang. Scalable subgraph enumeration in mapreduce. In PVLDB, 2015.

[4]. L. Lai, L. Qin, X. Lin, Y. Zhang, L. Chang, and S. Yang. Scalable distributed subgraph enumeration. In PVLDB, 2016.

[5]. M. Qiao, H. Zhang, and H. Cheng. Subgraph matching: on compression and computation. In PVLDB, 2017.

[6]. K. Ammar, F. McSherry, S. Salihoglu, and M. Joglekar. Distributed evaluation of subgraph queries using worst-case optimal low-memory dataflows. In PVLDB, 2018.

[7]. H. Kim, J. Lee, S. S. Bhowmick, W.-S. Han, J. Lee, S. Ko, and M. H. Jarrah. Dualsim: Parallel subgraph enumeration in a massive graph on a single machine. In SIGMOD, 2016.

[8]. C. R. Aberger, A. Lamb, S. Tu, A. Nötzli, K. Olukotun, and C. Ré. Emptyheaded: A relational engine for graph processing. In TODS, 2017.

[9]. F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang. Efficient subgraph matching by postponing cartesian products. In SIGMOD, 2016.

[10]. J. A. Grochow and M. Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. In Annual International Conference on Research in Computational Molecular Biology, 2007.

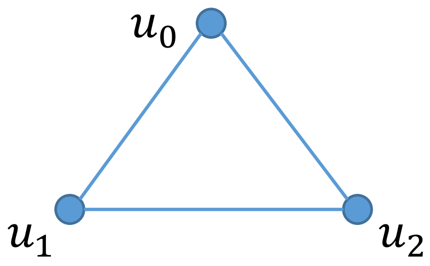
Thanks.
Q&A

Automorphism

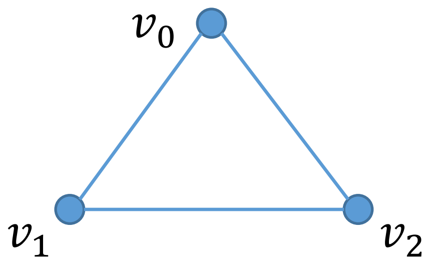
An automorphism of P is a match from P to itself. Because of the automorphisms, a subgraph in G isomorphic to P can result in duplicate matches from P to G.

Automorphism

An automorphism of P is a match from P to itself. Because of the automorphisms, a subgraph in G isomorphic to P can result in duplicate matches from P to G.



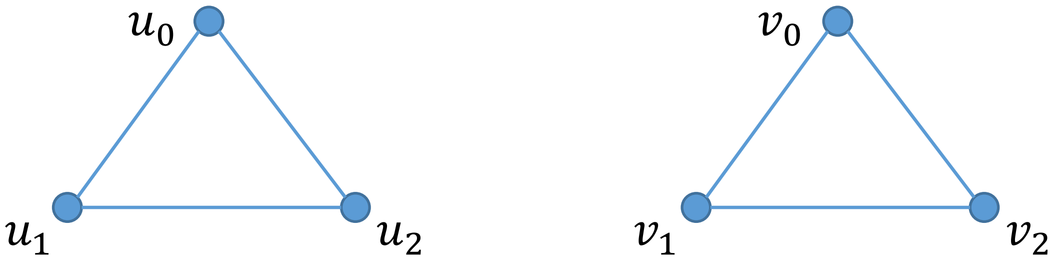
Pattern Graph P.



Data Graph G.

Automorphism

An automorphism of P is a match from P to itself. Because of the automorphisms, a subgraph in G isomorphic to P can result in duplicate matches from P to G.

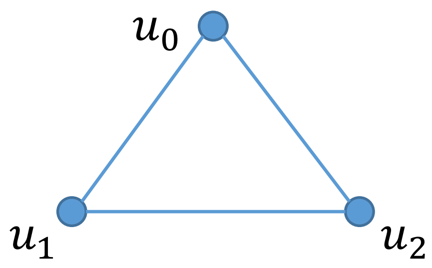


Pattern Graph P. Data Graph G.

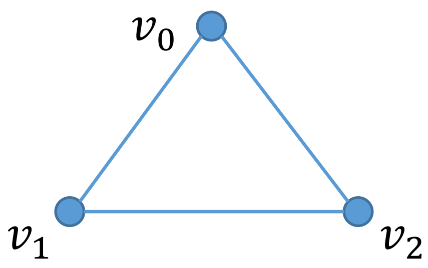
There is only 1 subgraph in G isomorphic to P, while we can find 6 matches from P to G.

Symmetry Breaking

In order to eliminate the duplicate matches, symmetry breaking assigns order $<$ to pattern vertices, and requires the matches φ to satisfy that given $u, u' \in V(P)$, if $u < u'$, then $\varphi(u) < \varphi(u')$.



Pattern Graph P.



Data Graph G.

The orders of P is $u_0 < u_1 < u_2$. There is only one match from P to G that satisfies the constraint of the symmetry breaking, which is $\{(u_0, v_0), (u_1, v_1), (u_2, v_2)\}$.

Problem Definition

Given a data graph G and a pattern graph P, subgraph enumeration finds subgraphs in G that are isomorphic to P.

For the ease of analysis, we assume that there is only one automorphism. Then, the problem is equivalent to finding all matches from P to G.

Basic Subgraph Enumeration Algorithm

Algorithm 1: SE Algorithm

Input: a pattern graph P and a data graph G
Output: all matches from P to G

1 **begin**
2     $\pi \leftarrow$ compute a connected enumeration order of $V(P)$;
3     $i \leftarrow 1$, $\varphi \leftarrow \{\}$;
4   **foreach** $v \in V(G)$ **do**
5   Add $(\pi[i], v)$ to φ;
6   `Enumerate`$(\pi, \varphi, i + 1)$;
7   Remove $(\pi[i], v)$ from φ;
8 **Procedure** `Enumerate`(π, φ, i)
9   **if** $i = |\pi| + 1$ **then** Output φ, **return**;
    `/* The computation phase. */`
10     $C_\varphi(\pi[i]) \leftarrow$ `ComputeCandidates`$(\pi[i], \varphi)$;
    `/* The materialization phase. */`
11   **foreach** $v \in C_\varphi(\pi[i])$ **do**
12   **if** $v \notin \varphi.values$ **then** Same as Lines 5-7;
13 **Function** `ComputeCandidates`(u, φ)
14     $C_\varphi(u) \leftarrow \bigcap_{u' \in N_{+}^{\pi}(u)} N(\varphi(u'))$;
15   **return** $C_\varphi(u)$;

Basic Subgraph Enumeration Algorithm

Algorithm 1: SE Algorithm

Input: a pattern graph P and a data graph G
Output: all matches from P to G

1 **begin**
2 $\quad \boxed{\pi \leftarrow \text{compute a connected enumeration order of } V(P);}$
3 $\quad i \leftarrow 1, \varphi \leftarrow \{\};$
4 $\quad$ **foreach** $v \in V(G)$ **do**
5 $\quad\quad$ Add $(\pi[i], v)$ to φ;
6 $\quad\quad$ `Enumerate`$(\pi, \varphi, i+1)$;
7 $\quad\quad$ Remove $(\pi[i], v)$ from φ;
8 **Procedure** `Enumerate`(π, φ, i)
9 $\quad$ **if** $i = |\pi| + 1$ **then** Output φ, **return**;
$\quad$ `/* The computation phase. */`
10 $\quad C_\varphi(\pi[i]) \leftarrow$ `ComputeCandidates`$(\pi[i], \varphi)$;
$\quad$ `/* The materialization phase. */`
11 $\quad$ **foreach** $v \in C_\varphi(\pi[i])$ **do**
12 $\quad\quad$ **if** $v \notin \varphi.values$ **then** Same as Lines 5-7;
13 **Function** `ComputeCandidates`(u, φ)
14 $\quad C_\varphi(u) \leftarrow \bigcap_{u' \in N_+^\pi(u)} N(\varphi(u'));$
15 $\quad$ **return** $C_\varphi(u)$;

Enumeration order π is a permutation of $V(P)$. $\pi[i]$ is the ith vertex in π.

Basic Subgraph Enumeration Algorithm

Algorithm 1: SE Algorithm

Input: a pattern graph P and a data graph G
Output: all matches from P to G

1 **begin**
2 $\quad$ $\boxed{\pi \leftarrow \text{compute a connected enumeration order of } V(P);}$
3 $\quad$ $i \leftarrow 1$, $\varphi \leftarrow \{\}$;
4 $\quad$ **foreach** $v \in V(G)$ **do**
5 $\quad\quad$ Add $(\pi[i], v)$ to φ;
6 $\quad\quad$ $\texttt{Enumerate}(\pi, \varphi, i+1)$;
7 $\quad\quad$ Remove $(\pi[i], v)$ from φ;
8 **Procedure** $\texttt{Enumerate}(\pi, \varphi, i)$
9 $\quad$ **if** $i = |\pi| + 1$ **then** Output φ, **return**;
$\quad$ `/* The computation phase. */`
10 $\quad$ $C_{\varphi}(\pi[i]) \leftarrow \texttt{ComputeCandidates}(\pi[i], \varphi)$;
$\quad$ `/* The materialization phase. */`
11 $\quad$ **foreach** $v \in C_{\varphi}(\pi[i])$ **do**
12 $\quad\quad$ **if** $v \notin \varphi.values$ **then** Same as Lines 5-7;
13 **Function** $\texttt{ComputeCandidates}(u, \varphi)$
14 $\quad$ $C_{\varphi}(u) \leftarrow \bigcap_{u' \in N_{+}^{\pi}(u)} N(\varphi(u'))$;
15 $\quad$ **return** $C_{\varphi}(u)$;

Enumeration order π is a permutation of $V(P)$. $\pi[i]$ is the ith vertex in π.

Recursively expand the partial result φ by mapping pattern vertices to data vertices along π to find all matches from P to G.

Basic Subgraph Enumeration Algorithm

Algorithm 1: SE Algorithm

Input: a pattern graph P and a data graph G

Output: all matches from P to G

1 **begin**

2 $\quad \boxed{\pi \leftarrow \text{compute a connected enumeration order of } V(P);}$

3 $\quad i \leftarrow 1, \varphi \leftarrow \{\};$

4 $\quad$ **foreach** $v \in V(G)$ **do**

5 $\quad\quad$ Add $(\pi[i], v)$ to φ;

6 $\quad\quad$ `Enumerate`$(\pi, \varphi, i+1)$;

7 $\quad\quad$ Remove $(\pi[i], v)$ from φ;

8 **Procedure** `Enumerate`(π, φ, i)

9 $\quad$ **if** $i = |\pi| + 1$ **then** Output φ, **return**;

$\quad$ `/* The computation phase. */`

10 $\quad C_\varphi(\pi[i]) \leftarrow$ `ComputeCandidates`$(\pi[i], \varphi)$;

$\quad$ `/* The materialization phase. */`

11 $\quad$ **foreach** $v \in C_\varphi(\pi[i])$ **do**

12 $\quad\quad$ **if** $v \notin \varphi.values$ **then** Same as Lines 5-7;

13 **Function** `ComputeCandidates`(u, φ)

14 $\quad C_\varphi(u) \leftarrow \bigcap_{u' \in N_+^\pi(u)} N(\varphi(u'));$

15 $\quad$ **return** $C_\varphi(u)$;

Enumeration order π is a permutation of $V(P)$. $\pi[i]$ is the ith vertex in π.

Recursively expand the partial result φ by mapping pattern vertices to data vertices along π to find all matches from P to G.

The computation phase is to obtain the candidate set $C_\varphi(\pi[i])$ of $\pi[i]$ given φ, and the materialization phase extends φ by mapping $\pi[i]$ to $v \in C_\varphi(\pi[i])$.

Basic Subgraph Enumeration Algorithm

Algorithm 1: SE Algorithm

Input: a pattern graph P and a data graph G
Output: all matches from P to G

1 **begin**
2 $\boxed{\pi \leftarrow \text{compute a connected enumeration order of } V(P);}$
3 $i \leftarrow 1$, $\varphi \leftarrow \{\}$;
4 **foreach** $v \in V(G)$ **do**
5 Add $(\pi[i], v)$ to φ;
6 `Enumerate`$(\pi, \varphi, i + 1)$;
7 Remove $(\pi[i], v)$ from φ;
8 **Procedure** `Enumerate`(π, φ, i)
9 **if** $i = |\pi| + 1$ **then** Output φ, **return**;
/* The computation phase. */
10 $C_\varphi(\pi[i]) \leftarrow$ `ComputeCandidates`$(\pi[i], \varphi)$;
/* The materialization phase. */
11 **foreach** $v \in C_\varphi(\pi[i])$ **do**
12 **if** $v \notin \varphi.values$ **then** Same as Lines 5-7;
13 **Function** `ComputeCandidates`(u, φ)
14 $\boxed{C_\varphi(u) \leftarrow \bigcap_{u' \in N_+^\pi(u)} N(\varphi(u'));}$
15 **return** $C_\varphi(u)$;

Enumeration order π is a permutation of $V(P)$. $\pi[i]$ is the ith vertex in π.

Recursively expand the partial result φ by mapping pattern vertices to data vertices along π to find all matches from P to G.

The computation phase is to obtain the candidate set $C_\varphi(\pi[i])$ of $\pi[i]$ given φ, and the materialization phase extends φ by mapping $\pi[i]$ to $v \in C_\varphi(\pi[i])$.

Compute common neighbors of data vertices mapped to backward neighbors of u where backward neighbors $N_+^\pi(u)$ of u is the neighbors of u positioned before u in π.

Parallelize Set Intersection

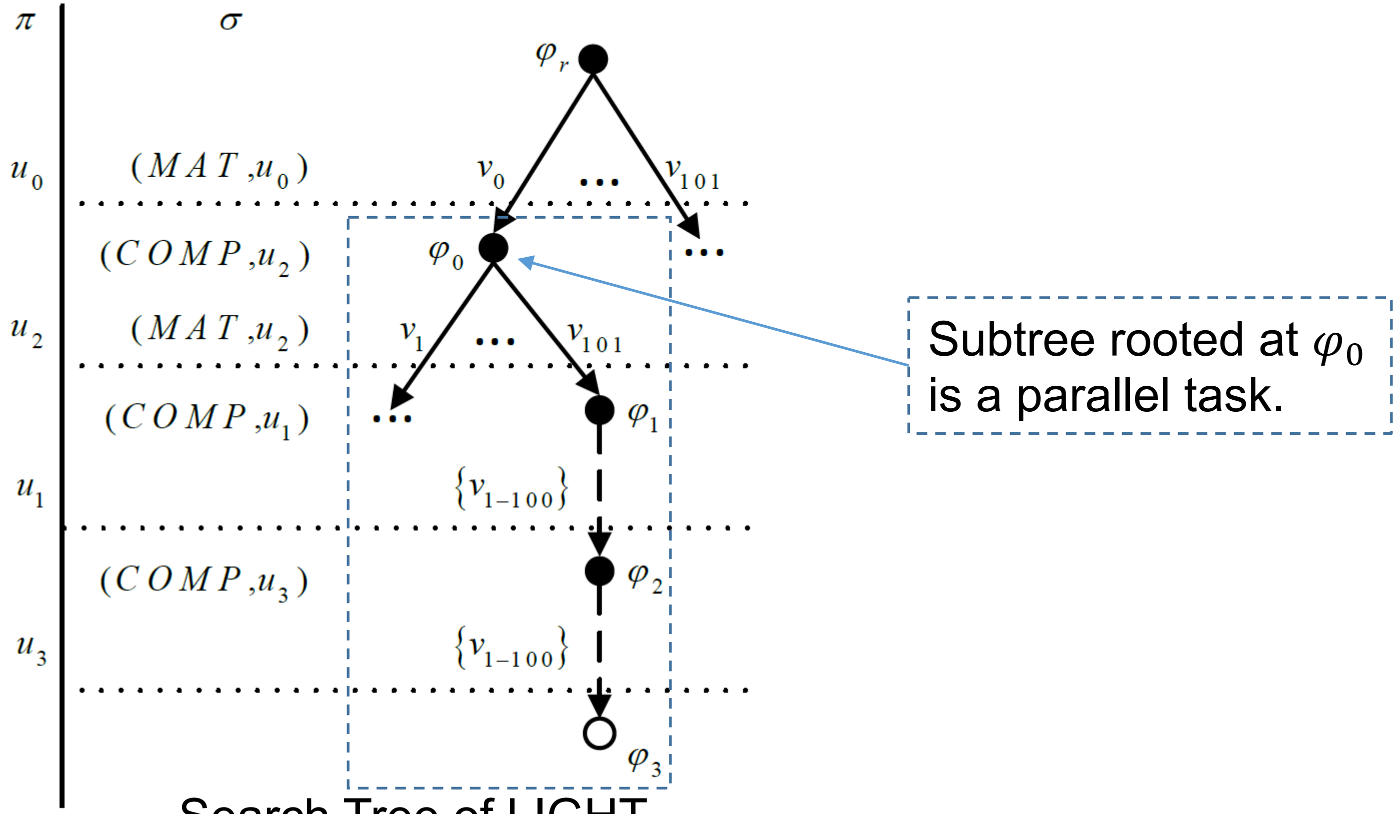
- Given two sets S_1 and S_2, which are stored as sorted arrays, we use SIMD to parallelize the set intersection between S_1 and S_2.
- We use a hybrid set intersection method to handle the size skewness of input sets:

  (1). If the size of S_1 and S_2 is similar, use the merge-based set intersection.

  (2). Otherwise, use the Galloping [1] algorithm.

Parallelize Search Tree Exploration

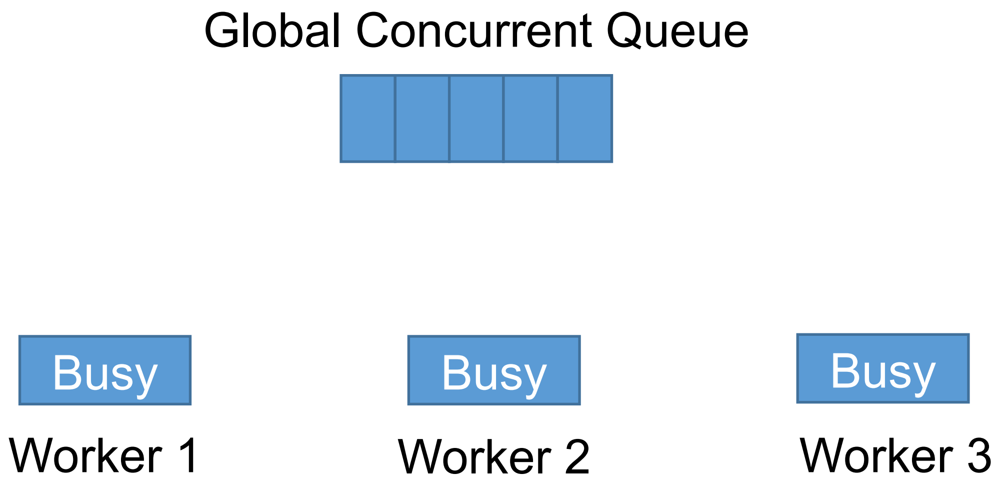
We take the partial results as parallel tasks, and each worker expands the assigned partial results in DFS independently.



Search Tree of LIGHT.

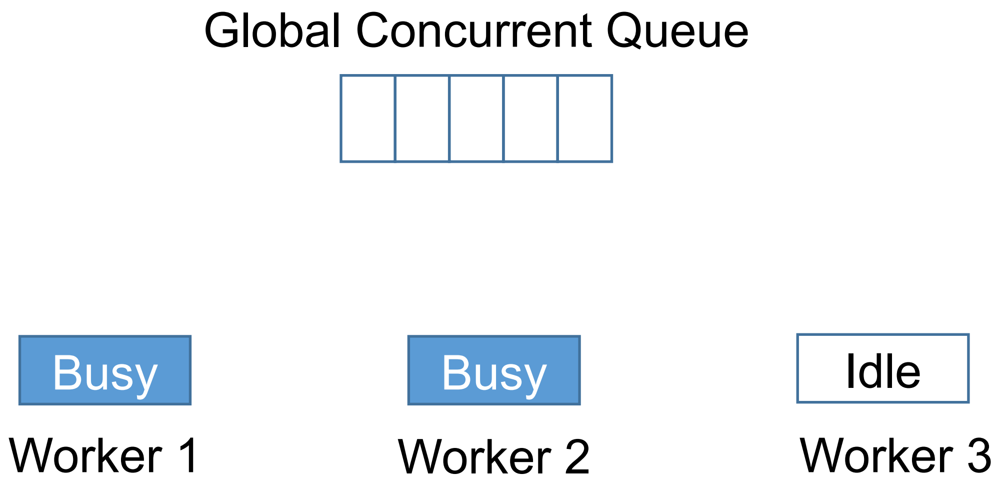
Parallelize Search Tree Exploration

We adopt a sender-initiated method with a global concurrent queue to deliver tasks among workers.



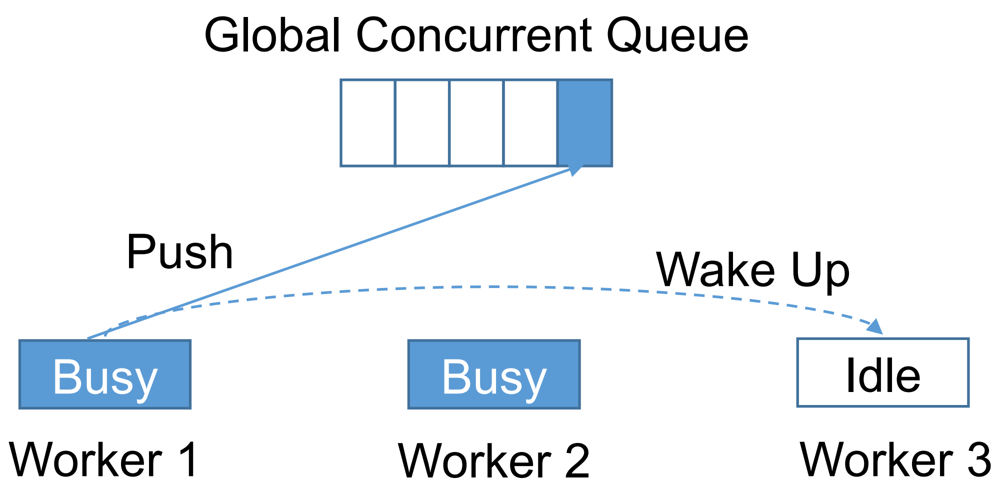
Parallelize Search Tree Exploration

We adopt a sender-initiated method with a global concurrent queue to deliver tasks among workers.



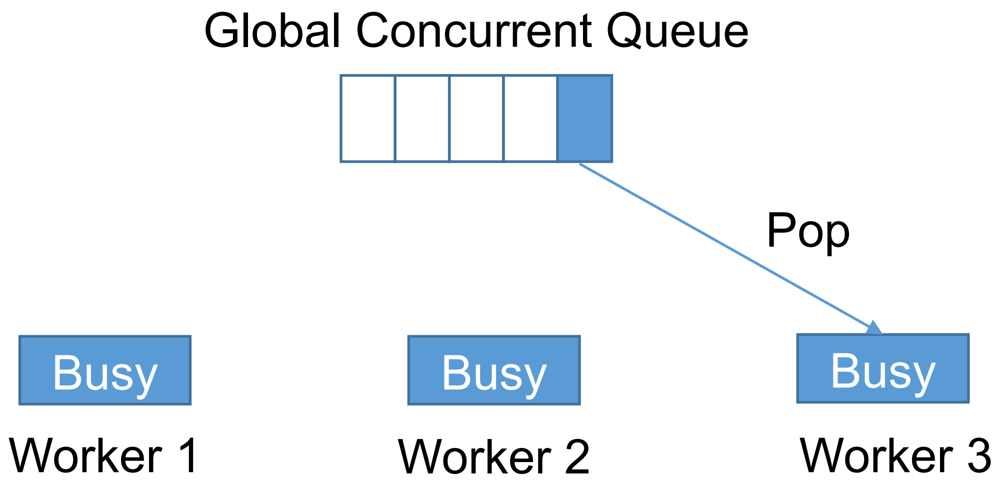
Parallelize Search Tree Exploration

We adopt a sender-initiated method with a global concurrent queue to deliver tasks among workers.



Parallelize Search Tree Exploration

We adopt a sender-initiated method with a global concurrent queue to deliver tasks among workers.



Optimize Enumeration Order

Utilize the ordering method proposed in SEED.

L. Lai, L. Qin, X. Lin, Y. Zhang, L. Chang, and S. Yang. Scalable distributed subgraph enumeration. In PVLDB, 2016.

Experimental Setup

Algorithms Under Study.

- EH [8]: EmptyHeaded, a relational engine for graph processing that answers queries with WCOJ algorithms.
- CFL [9]: the state-of-the-art labeled subgraph enumeration algorithm.
- SE: Algorithm 1, which is the baseline algorithm.
- LM: LIGHT with the Lazy Materialization strategy only.
- MSC: LIGHT with the Minimum Set Cover based candidate set computation method only.
- LIGHT: LIGHT with both the lazy materialization and the minimum set cover based candidate set computation.

Enumeration Order

SE, LM, MSC and LIGHT adopt the same enumeration order.

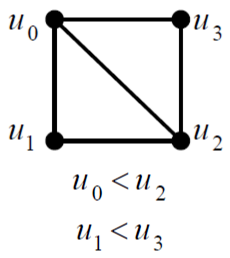
- $\pi(P_2) = (u_0, u_2, u_1, u_3)$, $\pi(P_4) = (u_0, u_1, u_4, u_2, u_3)$, and $\pi(P_6) = (u_0, u_1, u_2, u_3, u_4)$.

The enumeration order of CFL is as follows.

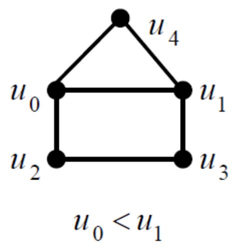
- $\pi(P_2) = (u_0, u_2, u_1, u_3)$, $\pi(P_4) = (u_0, u_2, u_4, u_1, u_3)$, and $\pi(P_6) = (u_0, u_1, u_2, u_3, u_4)$.

The enumeration order of EH is as follows.

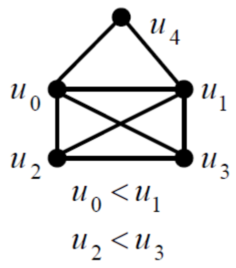
- $\pi(P_2) = (u_1, u_3, u_0, u_2)$
- $\pi(P_4{}') = (u_0, u_3, u_4, u_1)$, and $\pi(P_4'') = (u_0, u_3, u_2)$. Join the matches of $P_4{}'$ and $P_4{}''$.
- $\pi(P_6{}') = (u_0, u_1, u_2, u_3)$, and $\pi(P_6'') = (u_0, u_1, u_4)$. Join the matches of $P_6{}'$ and $P_6{}''$.



P_2.

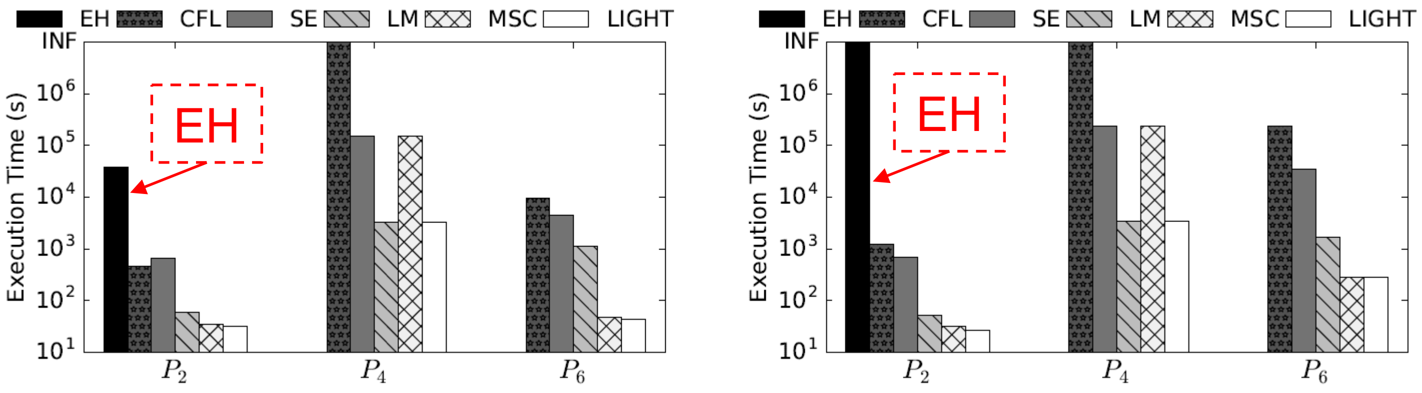


P_4.



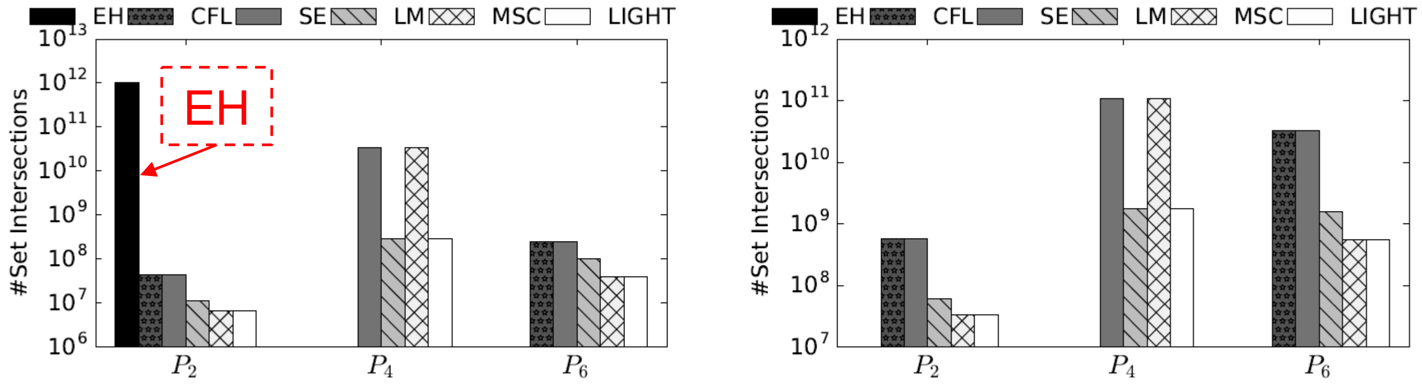
P_6.

Reducing Redundant Computation



(a) *yt* **(b)** *lj*

Comparison of Execution Time.

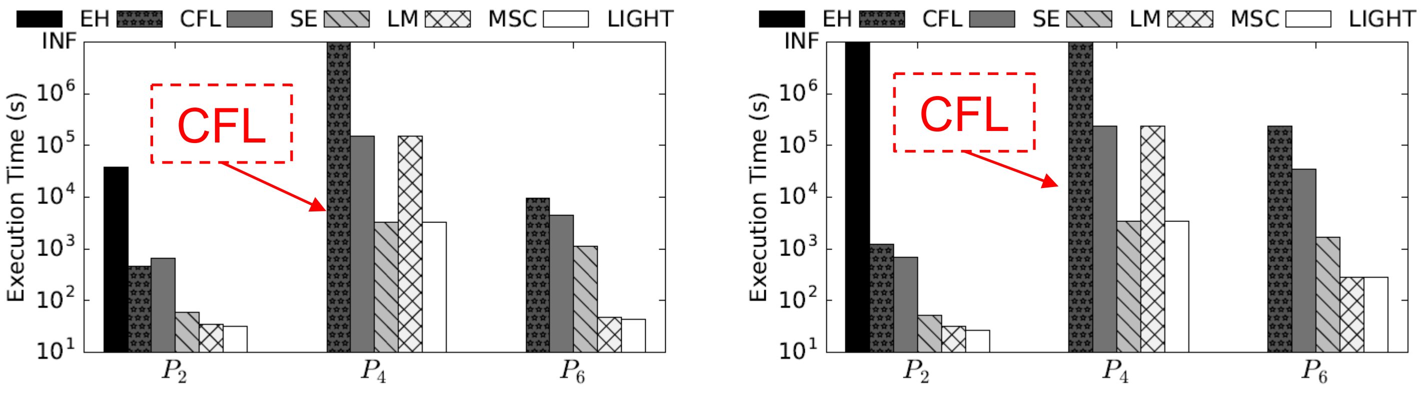


(a) *yt* **(b)** *lj*

Comparison of Number of Set Intersections.

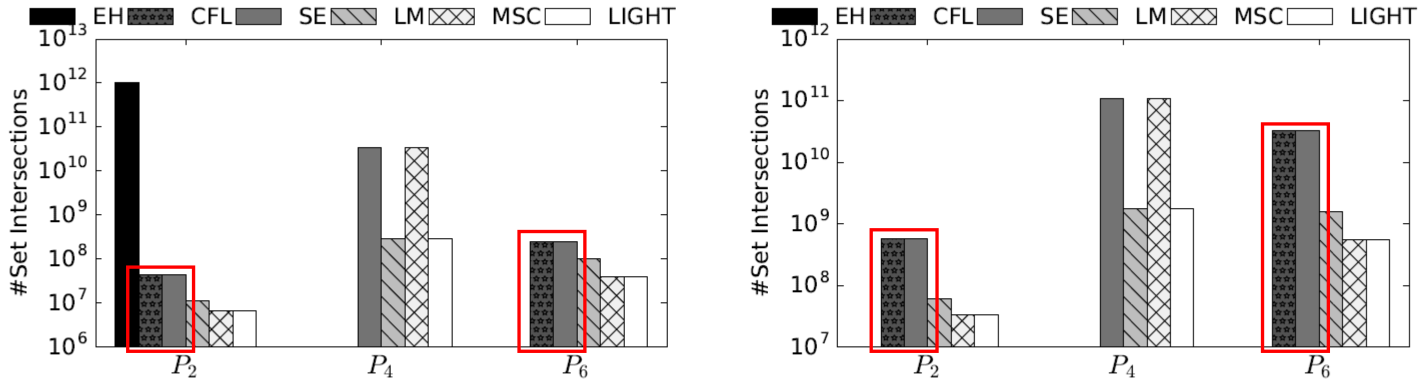
- EH runs slower than other algorithms on P_2, and runs out of memory on P_4 and P_6.

Reducing Redundant Computation



(a) *yt* **(b)** *lj*

Comparison of Execution Time.

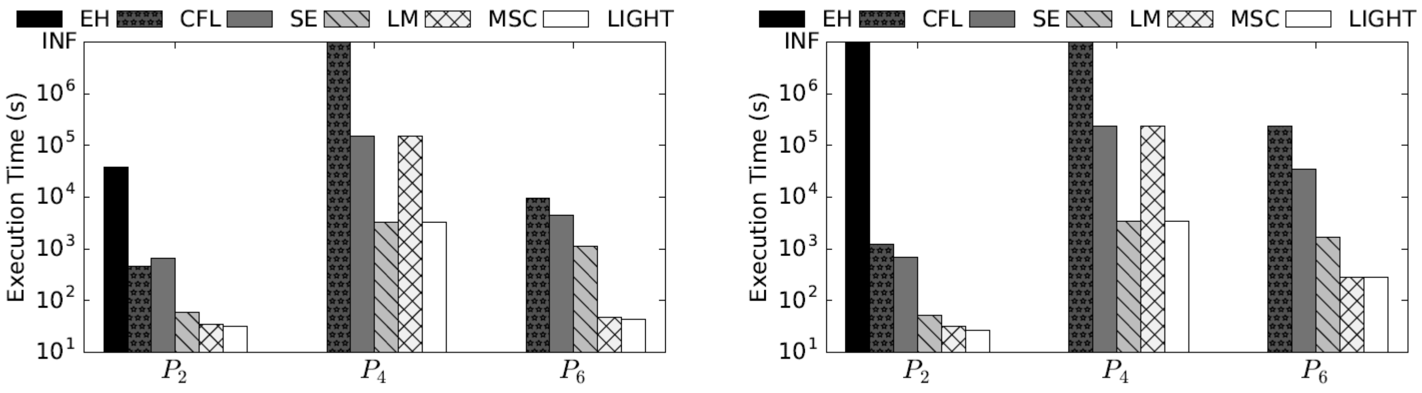


(a) *yt* **(b)** *lj*

Comparison of Number of Set Intersections.

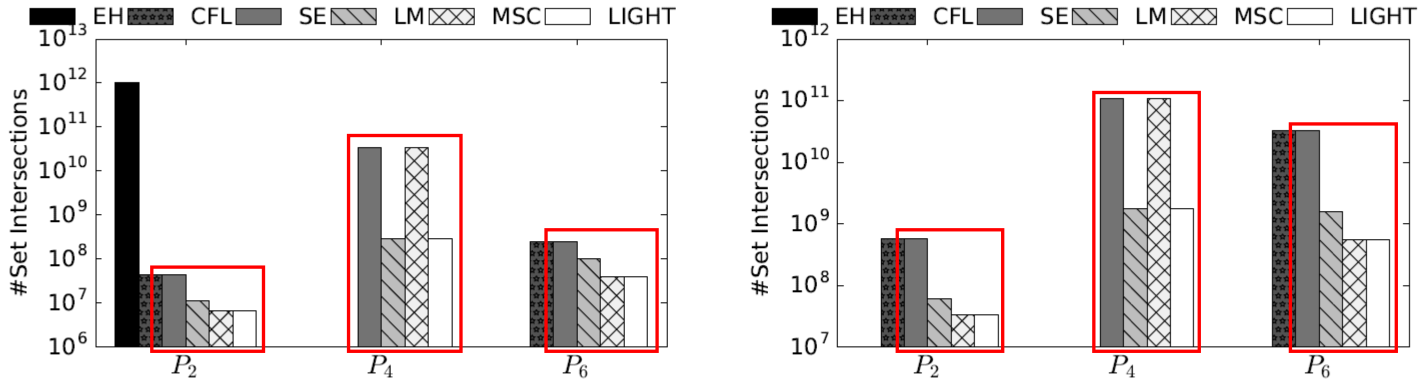
- EH runs slower than other algorithms on P_2, and runs out of memory on P_4 and P_6.
- CFL cannot complete P_4 within the time limit, and performs the same number of set intersections with SE.

Reducing Redundant Computation



(a) *yt* (b) *lj*

Comparison of Execution Time.

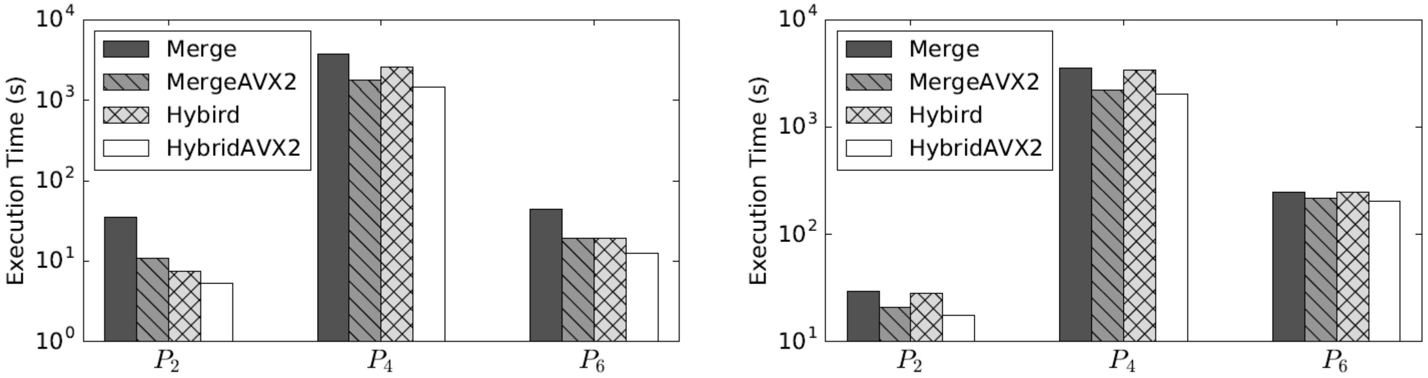


(a) *yt* (b) *lj*

Comparison of Number of Set Intersections.

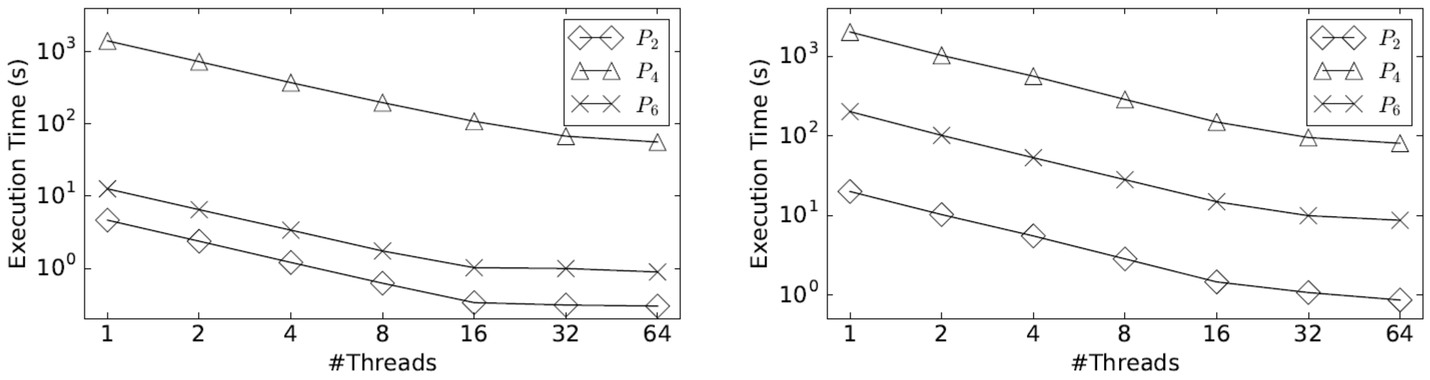
- EH runs slower than other algorithms on P_2, and runs out of memory on P_4 and P_6.
- CFL cannot complete P_4 within the time limit, and performs the same number of set intersections with SE.
- LIGHT significantly reduces the number of set intersections compared with SE, and outperforms the other algorithms.

Parallelization



(a) *yt* **(b)** *lj*

Execution Time with Different Set Intersection Methods.

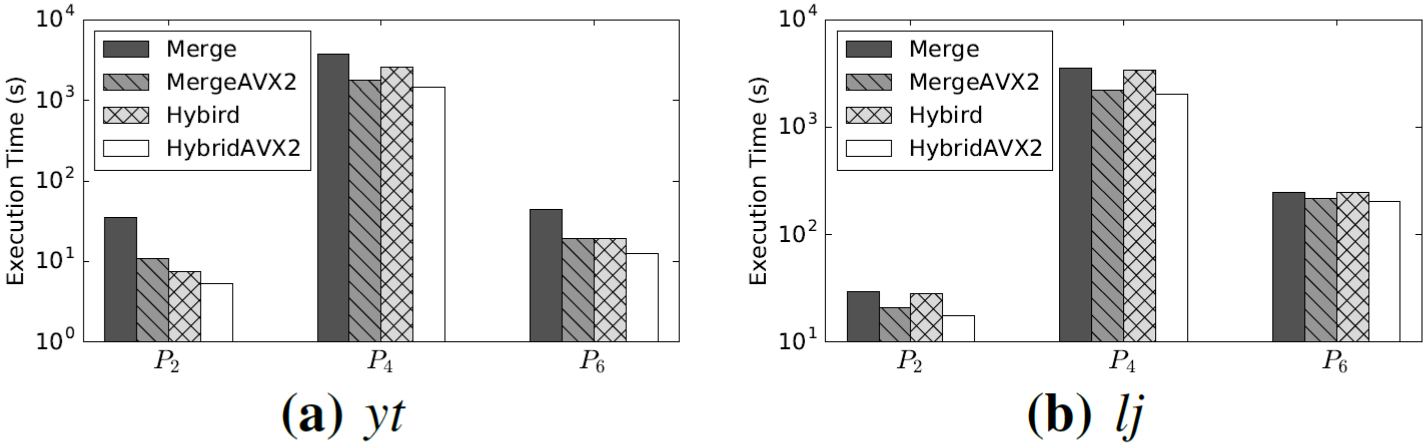


(a) *yt* **(b)** *lj*

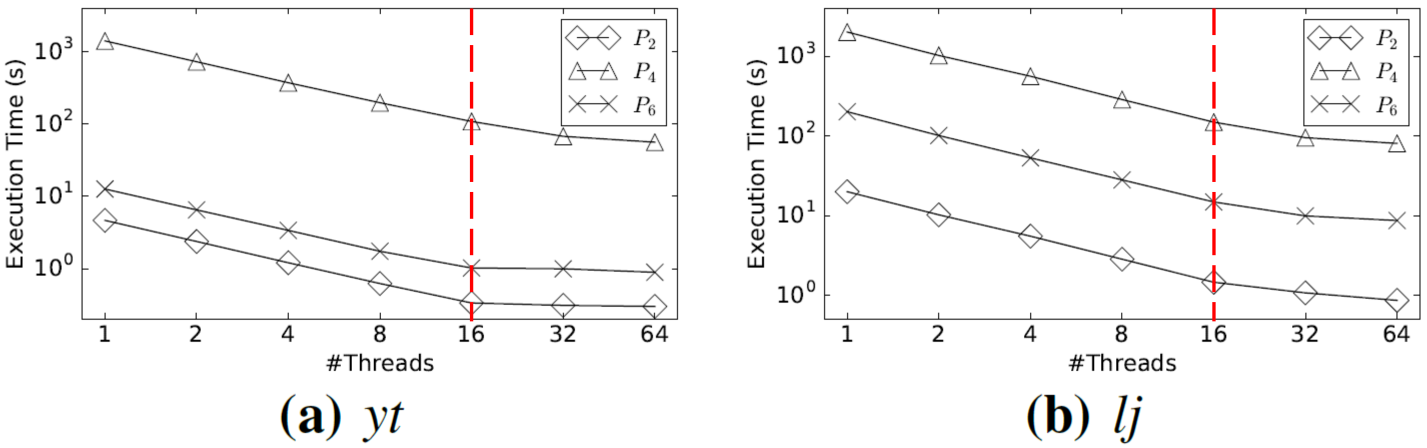
Execution Time with the Number of Threads Varied.

- HybridAVX2 runs 1.2-6.5X times faster than Merge.

Parallelization



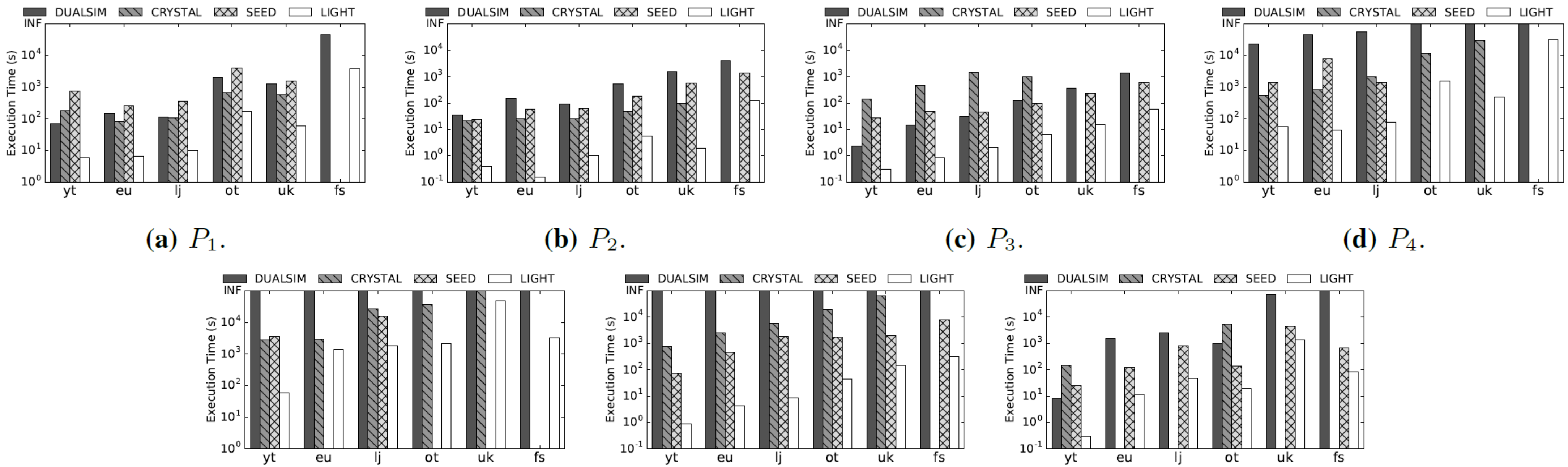
Execution Time with Different Set Intersection Methods.



Execution Time with the Number of Threads Varied.

- HybridAVX2 runs 1.2-6.5X times faster than Merge.
- LIGHT achieves almost linear speedup, when #threads varies from 1 to 16.

Comparison with Existing Algorithms



(a) P_1. (b) P_2. (c) P_3. (d) P_4.

(e) P_5. (f) P_6. (g) P_7.

Execution Time of LIGHT, DUALSIM, SEED and CRYSTAL on the Real-world Datasets.

Backup

Dataset	*yt*	*eu*	*lj*	*ot*	*uk*	*fs*
Memory (GB)	0.123	0.090	0.022	0.048	0.239	0.008

Memory consumption of candidate sets on P_5.

Dataset	*yt*			*lj*		
Pattern	P_2	P_4	P_6	P_2	P_4	P_6
Percentage	34.8%	35.9%	8.1%	1.1%	2.1%	0.7%

Percentage of the Galloping search.

Backup

Dataset	lj	ot	uk	fs
p_0	1.78×10^8	6.28×10^8	2.22×10^9	4.17×10^9
p_1	2.64×10^{10}	1.28×10^{11}	9.15×10^{11}	4.66×10^{11}
p_2	3.95×10^{10}	6.71×10^{10}	1.11×10^{12}	1.85×10^{11}
p_3	5.22×10^9	3.22×10^9	1.07×10^{11}	8.96×10^9
p_4	2.62×10^{13}	4.97×10^{13}	9.42×10^{14}	5.47×10^{13}
p_5	7.38×10^{15}	4.01×10^{15}	6.13×10^{17}	1.34×10^{15}
p_6	9.56×10^{12}	2.60×10^{12}	4.01×10^{14}	3.18×10^{12}
p_7	2.46×10^{11}	1.58×10^{10}	1.16×10^{13}	2.17×10^{10}

The Number of Matches (P_0 represents the triangle).