SCALABLE DISTRIBUTED SUBGRAPH ENUMERATION

AUTHORS: LONGBIN LAI
LU QIN
XUEMIN LIN
YING ZHANG
LIJUN CHANG

OUTLINE

PROBLEM DEFINITION

ALGORITHM FRAMEWORK

TWINTWIG JOIN - VLDB15'

SEED

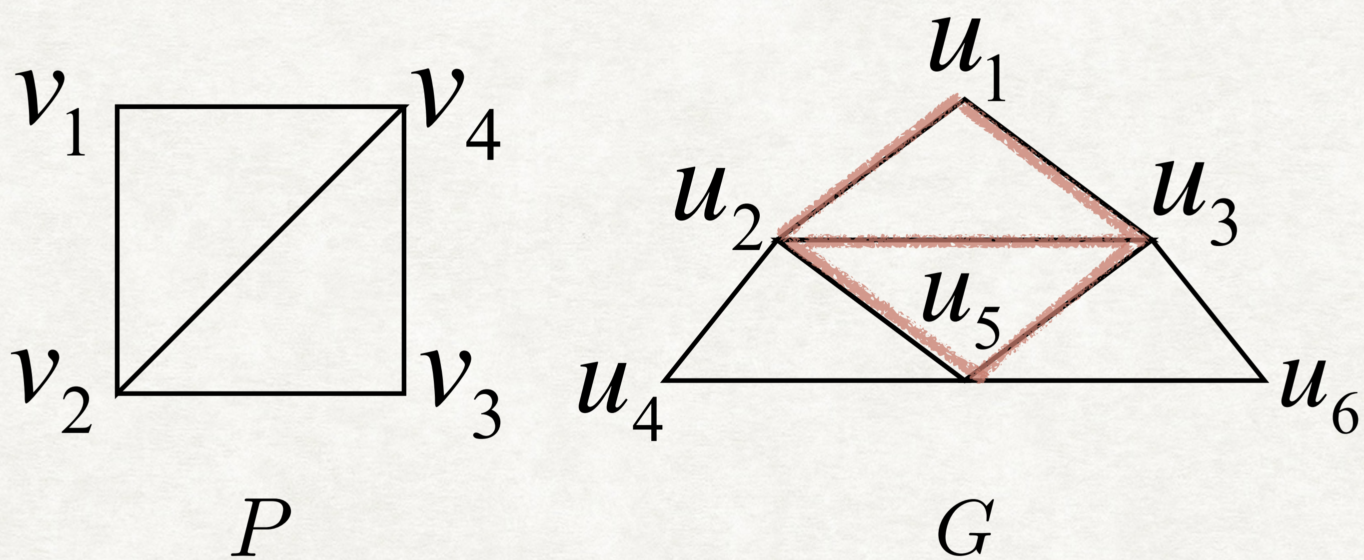
EXPERIMENTS

CONCLUSION

PROBLEM

PROBLEM DEFINTION

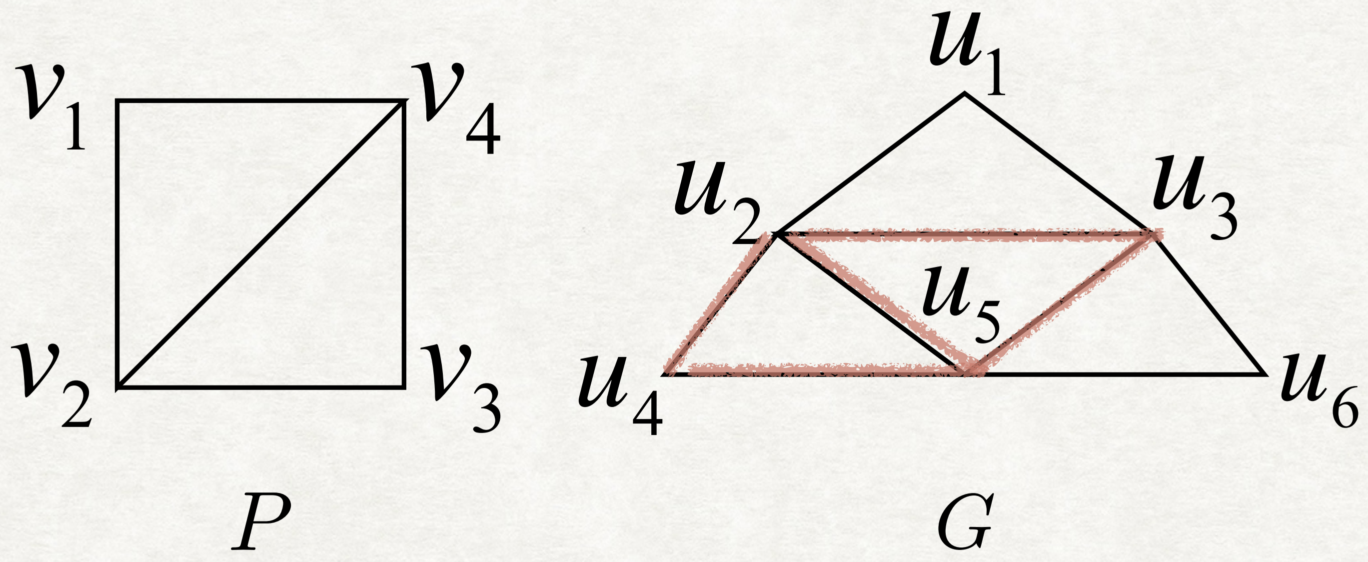
SUBGRAPH ENUMERATION

- Given a data graph G, and a pattern graph P, subgraph enumeration aims to find all subgraphs $g \subseteq G$ (**matches**), that are isomorphic to P.
- 

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 \\ u_1 & u_2 & u_5 & u_3 \end{pmatrix}$$

PROBLEM DEFINTION

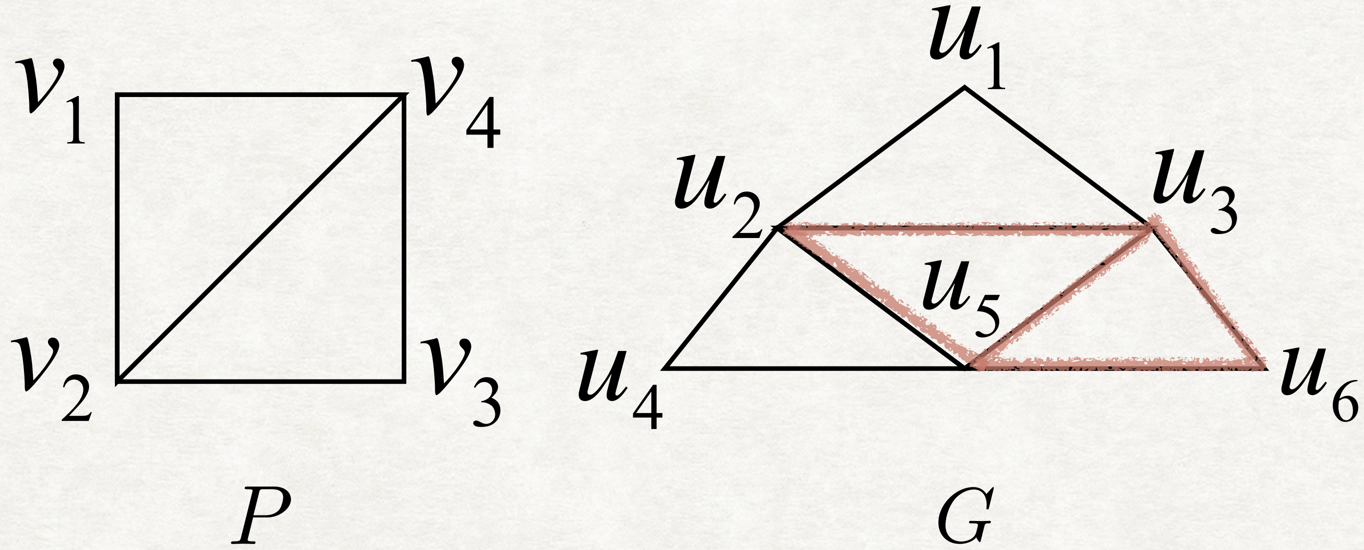
SUBGRAPH ENUMERATION

- Given a data graph G, and a pattern graph P, subgraph enumeration aims to find all subgraphs $g \subseteq G$ (**matches**), that are isomorphic to P.
- 

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 \\ u_4 & u_2 & u_3 & u_5 \end{pmatrix}$$

PROBLEM DEFINTION

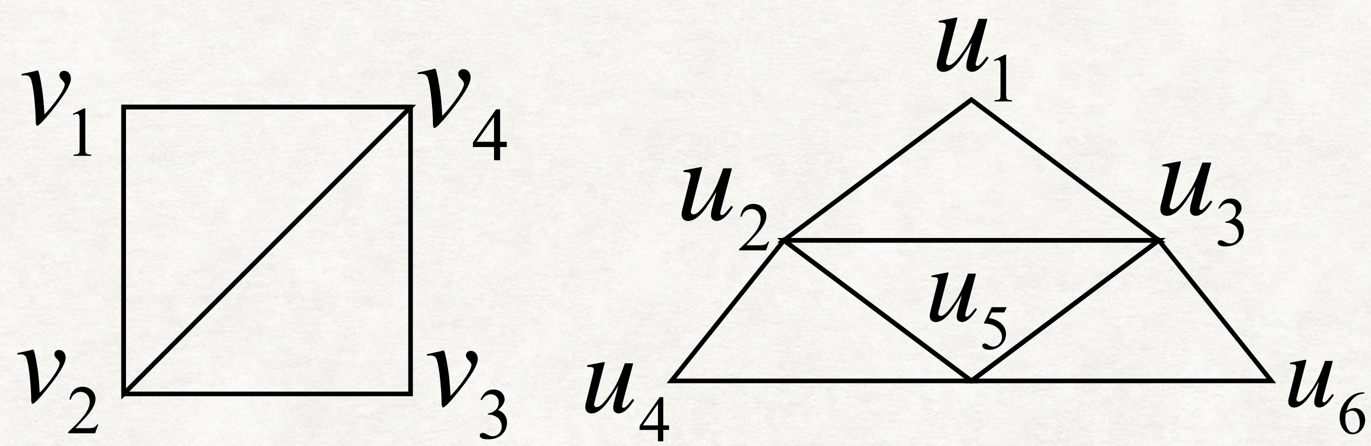
SUBGRAPH ENUMERATION

- Given a data graph G, and a pattern graph P, subgraph enumeration aims to find all subgraphs $g \subseteq G$ (**matches**), that are isomorphic to P.
- 

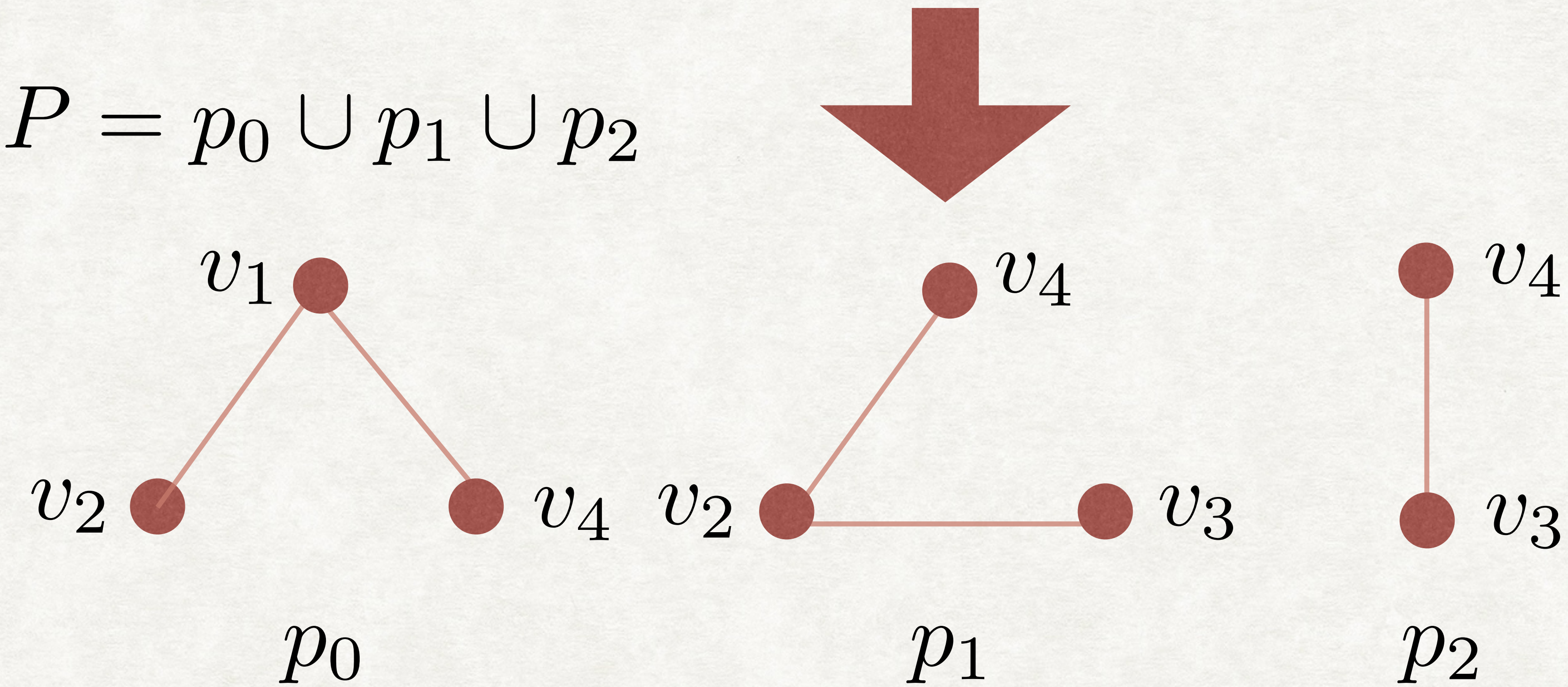
$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 \\ u_6 & u_3 & u_2 & u_5 \end{pmatrix}$$

FRAMEWORK

PATTERN DECOMPOSITION



$$P = p_0 \cup p_1 \cup p_2$$



Join Units

WHAT CAN BE JOIN UNITS

- Graph Storage $\Phi(G) = \{G_u | u \in V(G)\}$
  - Stored as $(u; G_u)$ for each data node
  - G_u: Local Graph of u s.t.
    - (1) Connected
    - (2) $u \in V(G_u)$
    - (3) $\bigcup_{u \in V(G)} E(G_u) = E(G)$

WHAT CAN BE JOIN UNITS

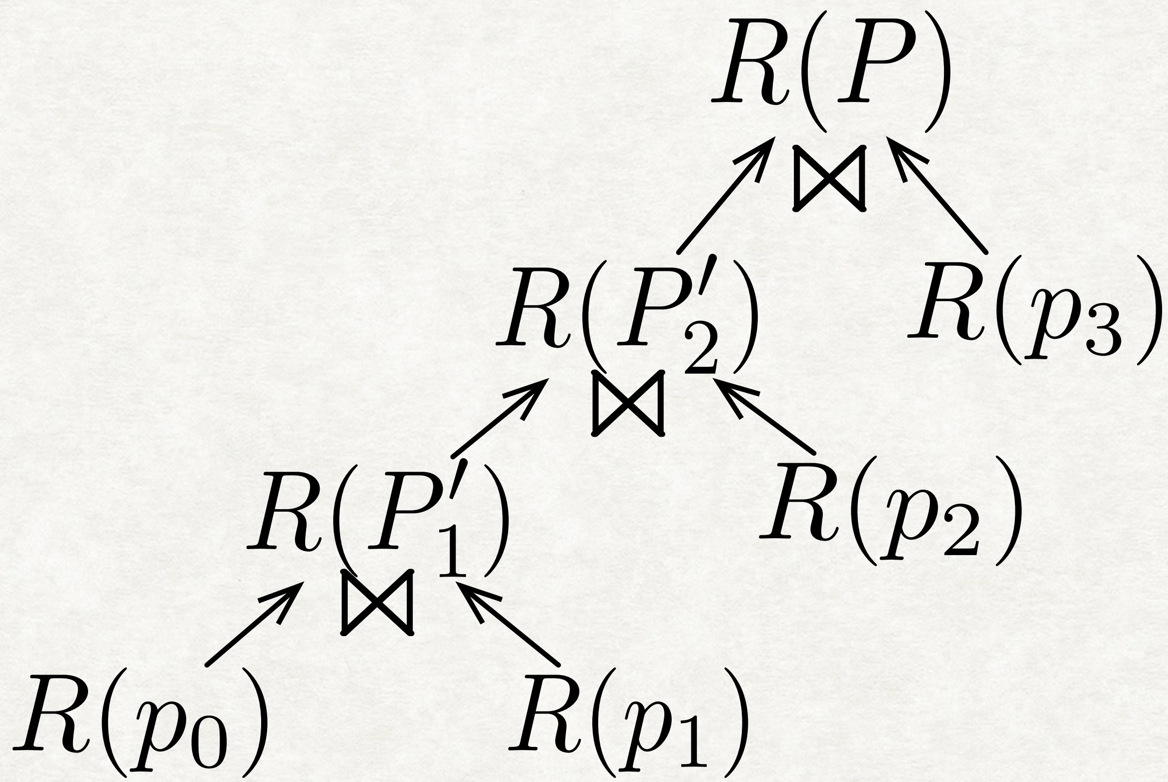
- A structure p can be a join unit iff.

$$R_G(p) = \bigcup_{u \in V(G)} R_{G_u}(p)$$

- $R_{\mathcal{G}}(p)$ stands for the matches of p in $\mathcal{G}$

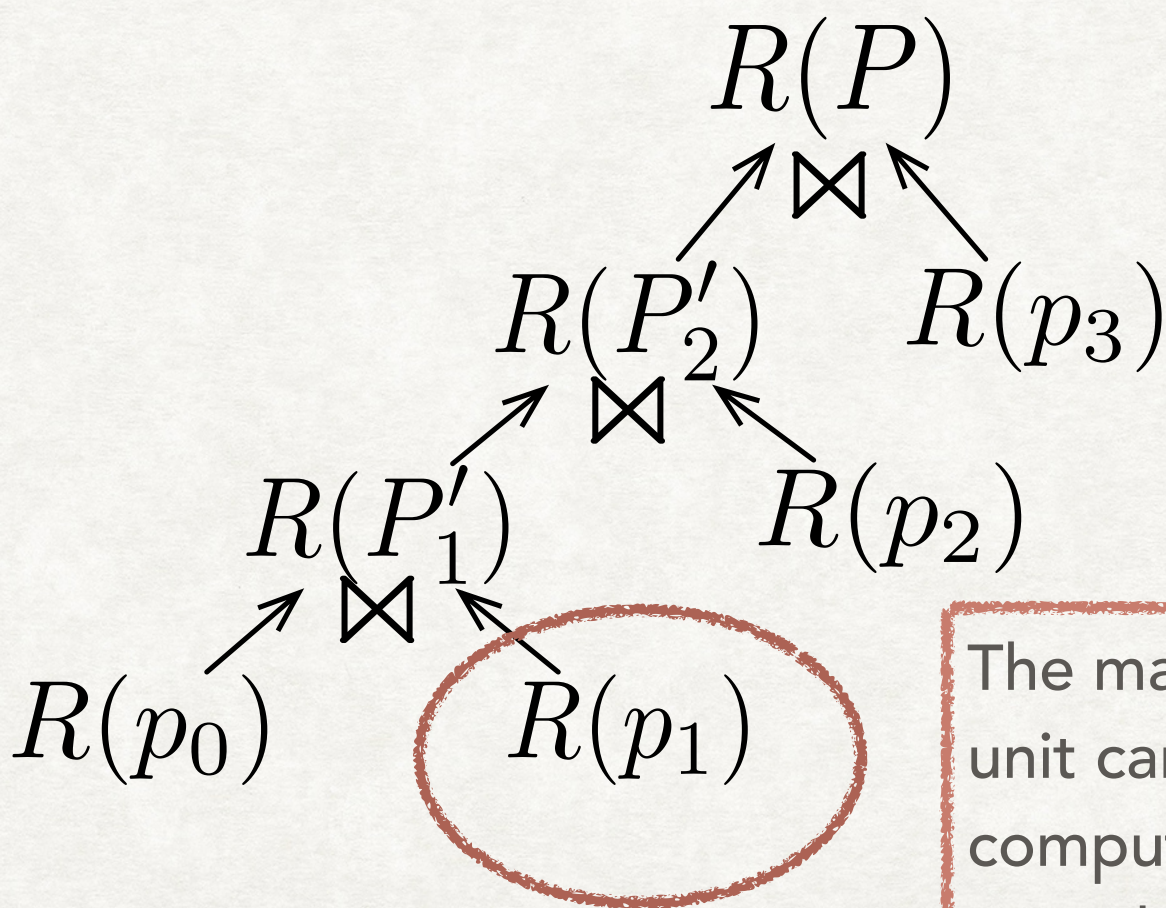
JOIN PLAN (TREE)

- Decomposing $P = p_0 \cup p_1 \cup p_2 \cup p_3$
- Solving: $R(P) = R(p_0) \bowtie R(p_1) \bowtie R(p_2) \bowtie R(p_3)$



JOIN PLAN (TREE)

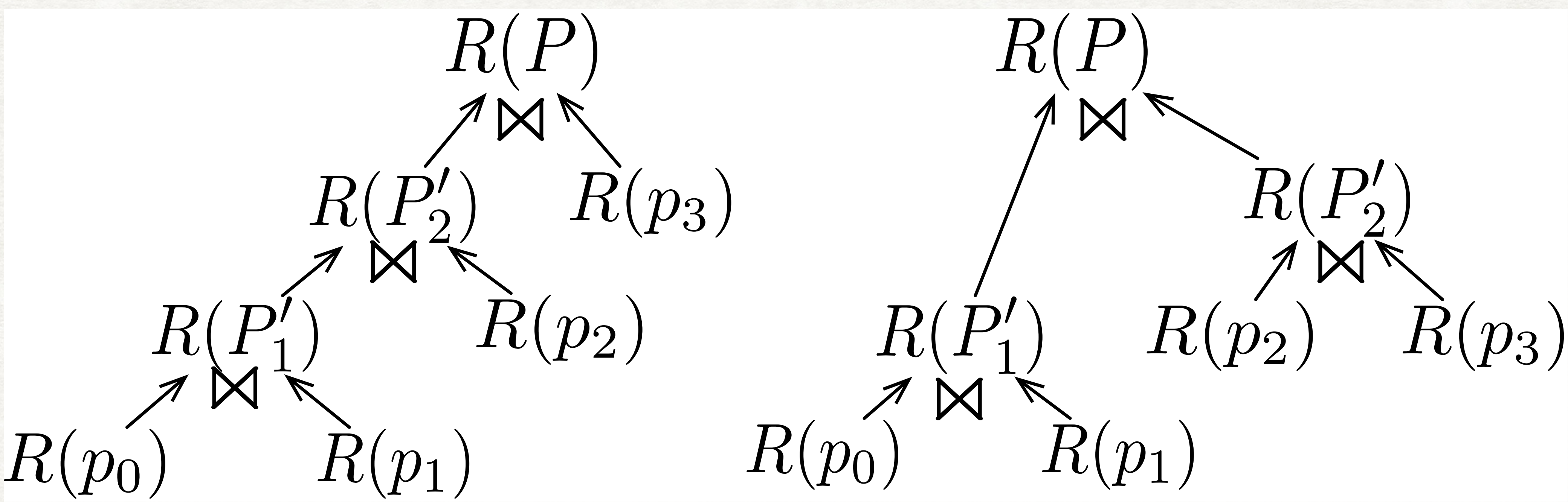
- Decomposing $P = p_0 \cup p_1 \cup p_2 \cup p_3$
- Solving: $R(P) = R(p_0) \bowtie R(p_1) \bowtie R(p_2) \bowtie R(p_3)$



The matches of each join unit can be online computed independently in each local graph

JOIN PLAN (TREE)

- Decomposing $P = p_0 \cup p_1 \cup p_2 \cup p_3$
- Solving: $R(P) = R(p_0) \bowtie R(p_1) \bowtie R(p_2) \bowtie R(p_3)$



Left-deep tree Bushy tree

DESCRIBE THE ALGORITHMS

- Graph Strorage mechanism
  - Determine the join units, thereafter the pattern decomposition
- Join Structure
  - Left-deep tree vs bushy tree

TWINTWIG JOIN - VLDB15'

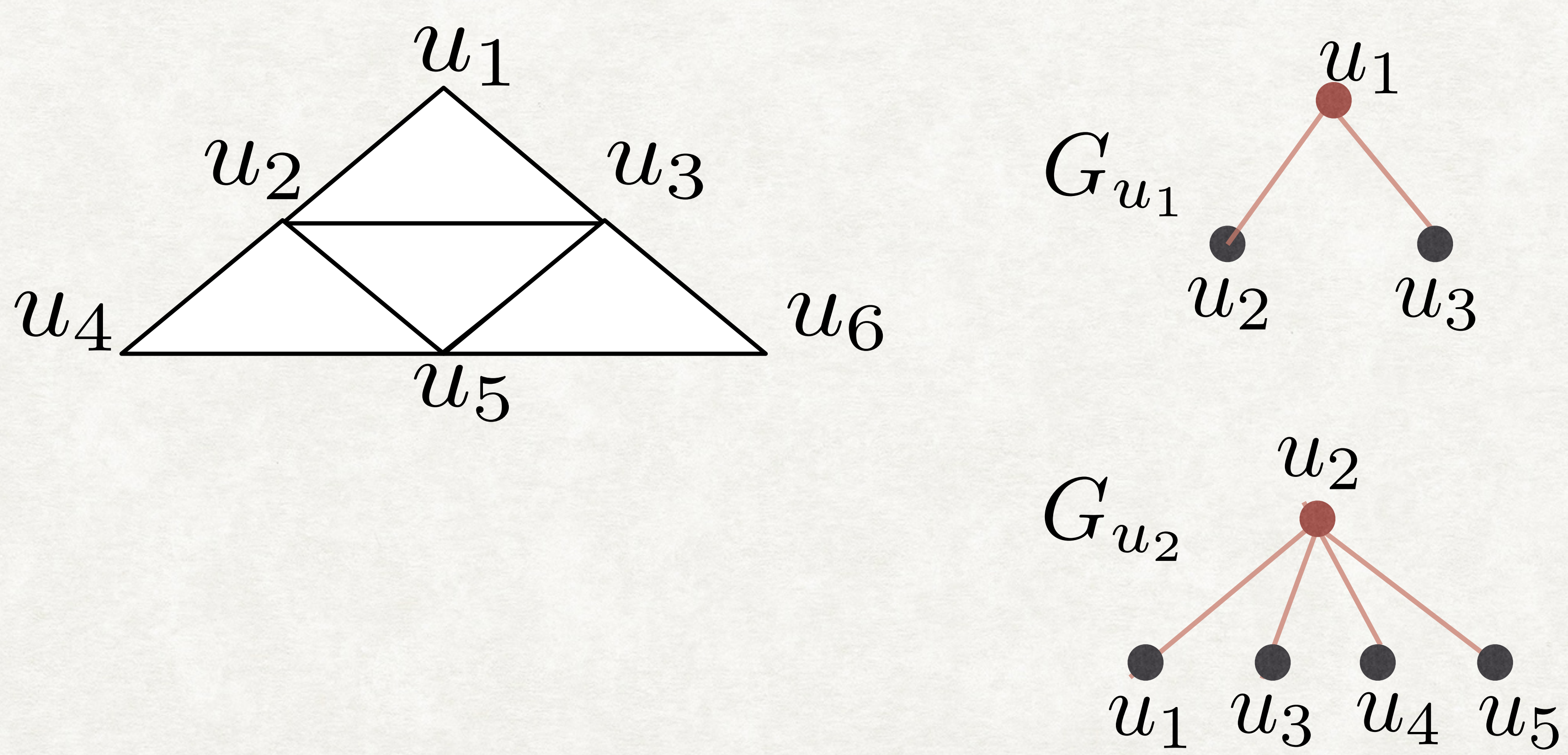
TWINTWIG JOIN - VLDB2015

SIMPLE GRAPH STORAGE

- The simple graph storage, each local graph G_u

$$V(G_u) = \{u\} \cup \mathcal{N}(u)$$

$$E(G_u) = \{(u, u') | u' \in \mathcal{N}(u)\}$$



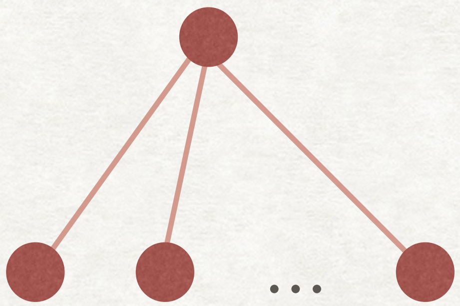
TWINTWIG JOIN - VLDB2015

SIMPLE GRAPH STORAGE

- The simple graph storage, where

$$V(G_u) = \{u\} \cup \mathcal{N}(u)$$

$$E(G_u) = \{(u, u')|u' \in \mathcal{N}(u)\}$$



Star as the join unit

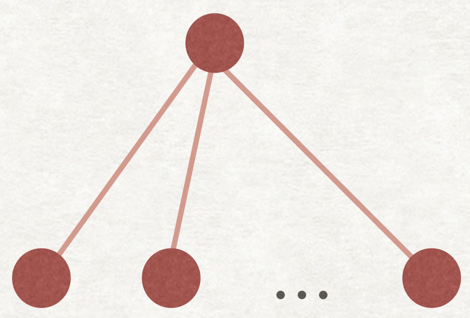
TWINTWIG JOIN - VLDB2015

SIMPLE GRAPH STORAGE

- The simple graph storage, where

$$V(G_u) = \{u\} \cup \mathcal{N}(u)$$

$$E(G_u) = \{(u, u')|u' \in \mathcal{N}(u)\}$$



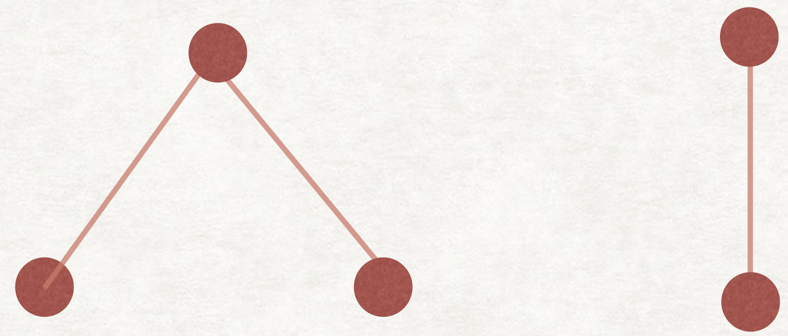
Star as the join unit

A node with degree 1,000,000 will generate 10^{18} 3-stars

TWINTWIG JOIN

SIMPLE GRAPH STORAGE

- Using **twintwigs** as the join units

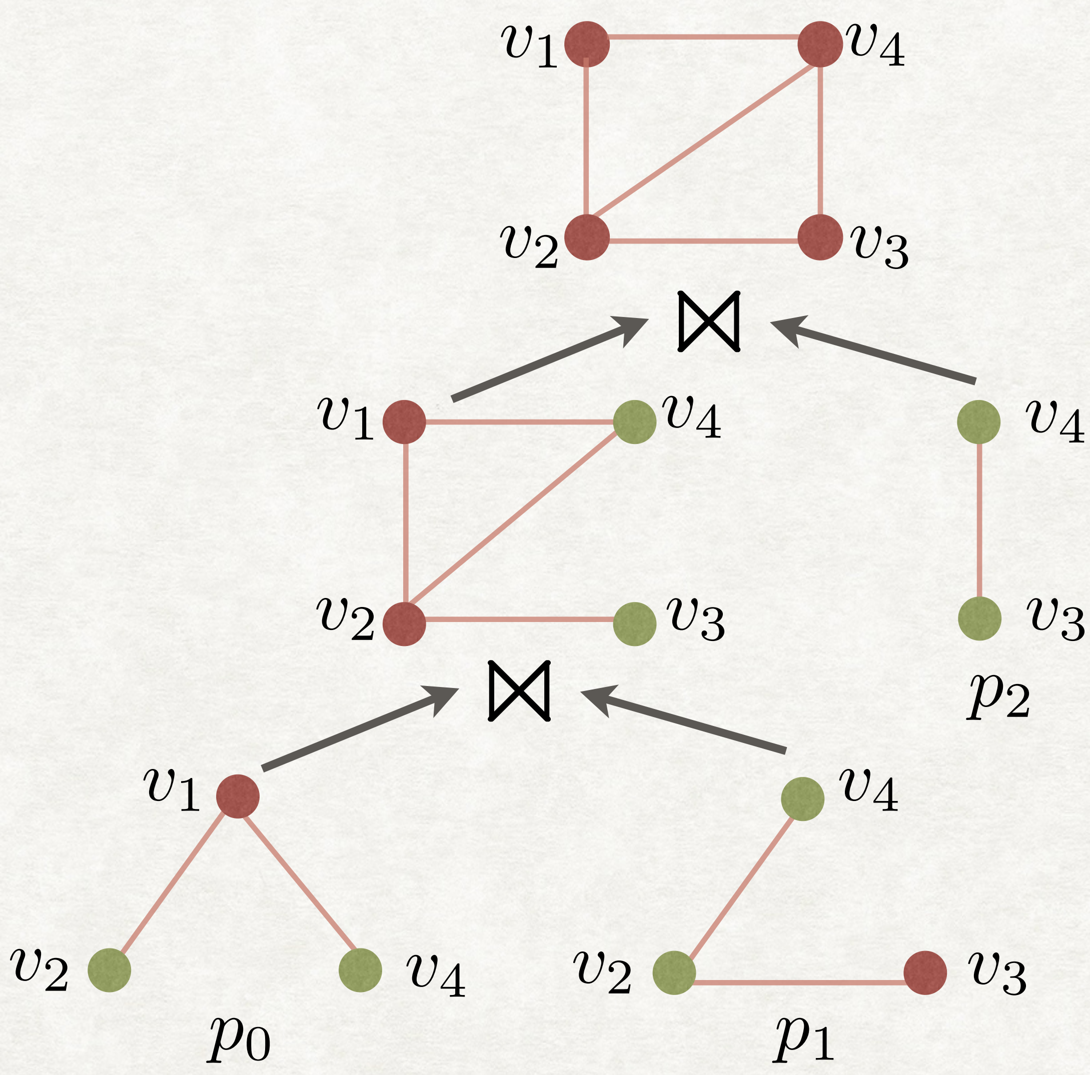


- **Instance Optimality**
  - Given any join plan involving general stars, we can solve it using twintwigs with at most the same (**often much less**) cost

TWINTWIG JOIN

LEFT-DEEP JOIN PLAN

- An optimal **<u>left-deep</u>** join plan with minimum estimated cost



TWINTWIG JOIN

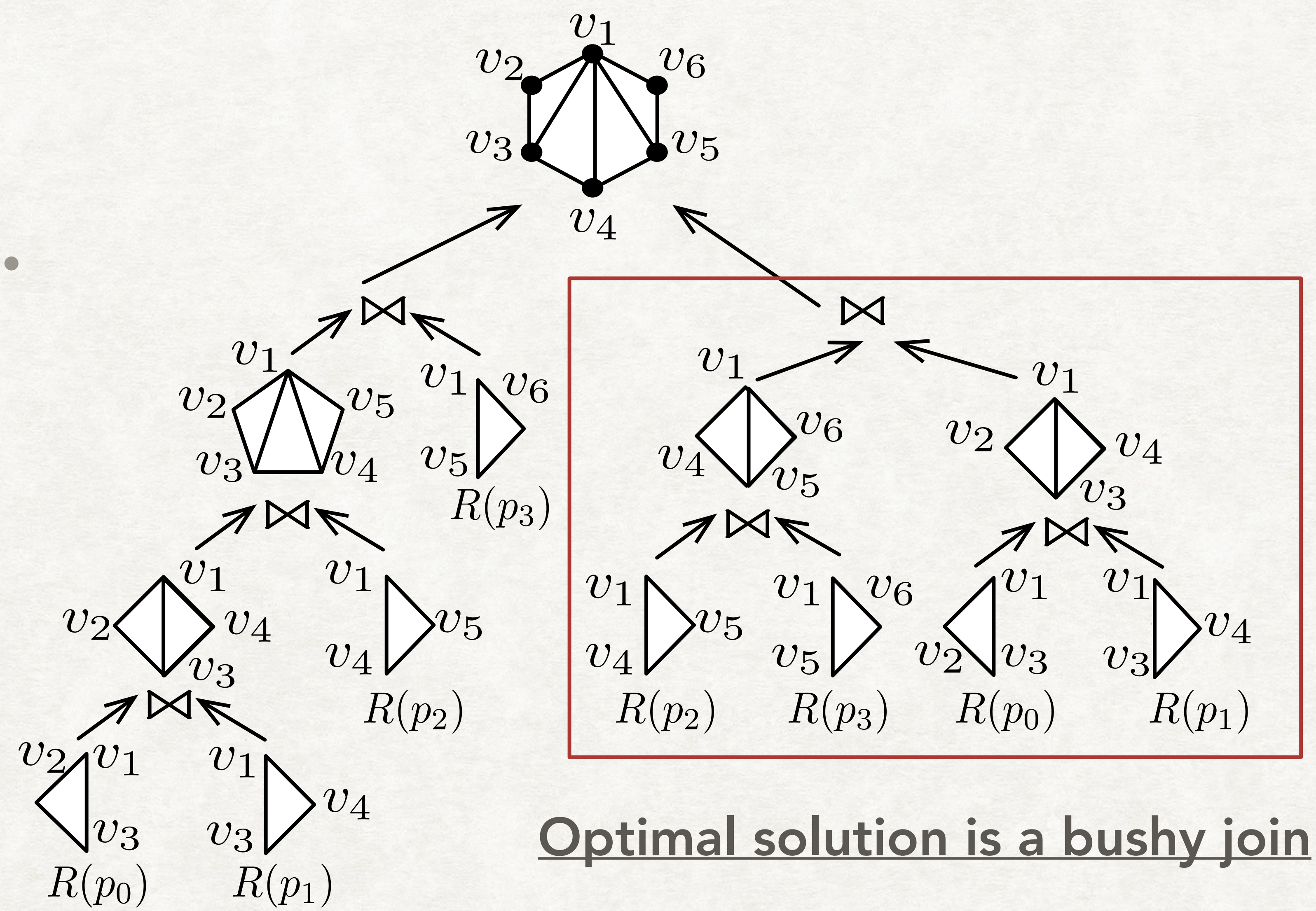
DRAWBACKS

- Simple storage mechanism only support using **star** as join units, too many intermediate results
  - Twintwig: confine to be at most two edges
    - The node with degree 1,000,000 still have 10^{12} two-edge twintwigs
  - Too many execution rounds.
    - A clique of 6 nodes (15 edges): Seven rounds of TwinTwigJoin

TWINTWIG JOIN

DRAWBACKS

- Left-deep join: may result in **sub-optimal** results



Optimal solution is a bushy join

SEED - VLDB17’

MOTIVATIONS

- **S**ubgraph **E**num**E**ration in **D**istributed Context
  - SCP (Star-Clique-Preserved) graph storage: Use star and **clique** as the join units
    - We can avoid using star if clique is an alternative
    - Shorter execution. The 6-clique can now be processed in one single round, instead of 7 rounds in TwinTwigJoin
  - Bushy join plan: Optimality Guarantee
  - Much better performance

SEED

SEED

SCP GRAPH STORAGE

- The SCP Graph Storage, where each local graph G_u^+

$$
\begin{aligned}
V(G_u^+) =& V(G_u) = \{u\} \cup \mathcal{N}(u) \\
E(G_u^+) =& E(G_u) \;\cup \\
& \{(u', u'') | (u', u'') \in E(G) \wedge u', u'' \in \mathcal{N}(u)\}
\end{aligned}
$$

SEED

SCP GRAPH STORAGE

- The SCP Graph Storage, where each local graph G_u^+

$$V(G_u^+) = V(G_u) = \{u\} \cup \mathcal{N}(u)$$

$$E(G_u^+) = E(G_u) \cup \{(u', u'') | (u', u'') \in E(G) \land u', u'' \in \mathcal{N}(u)\}$$

NEIGHBOUR EDGES

SEED

SCP GRAPH STORAGE

- The SCP Graph Storage, where each local graph G_u^+

$$V(G_u^+) = V(G_u) = \{u\} \cup \mathcal{N}(u)$$
$$E(G_u^+) = E(G_u) \cup$$
$$\{(u', u'') | (u', u'') \in E(G) \wedge u', u'' \in \mathcal{N}(u)\}$$

TRIANGLE EDGES

SEED

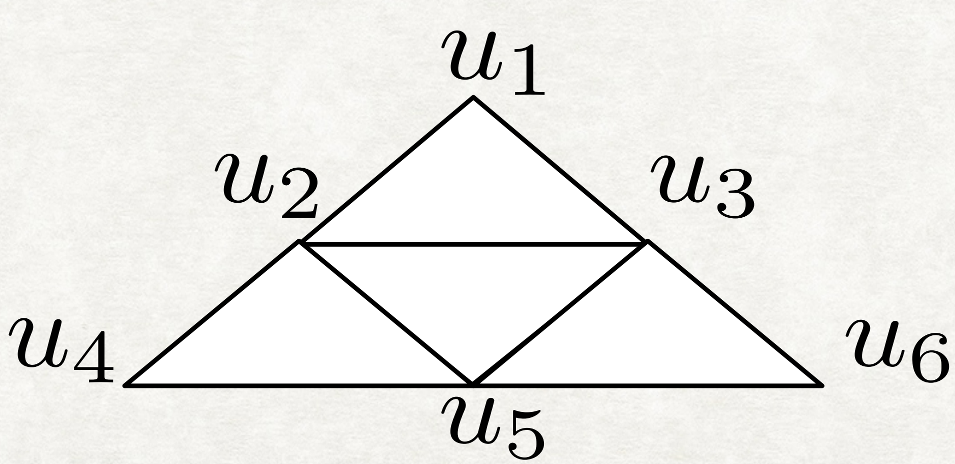
SCP GRAPH STORAGE

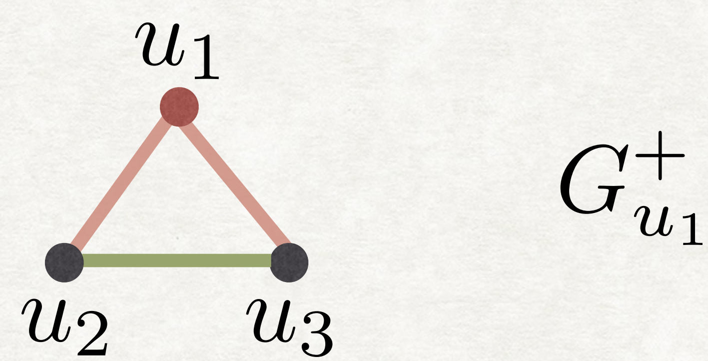
- The SCP Graph Storage, where each local graph G_u^+

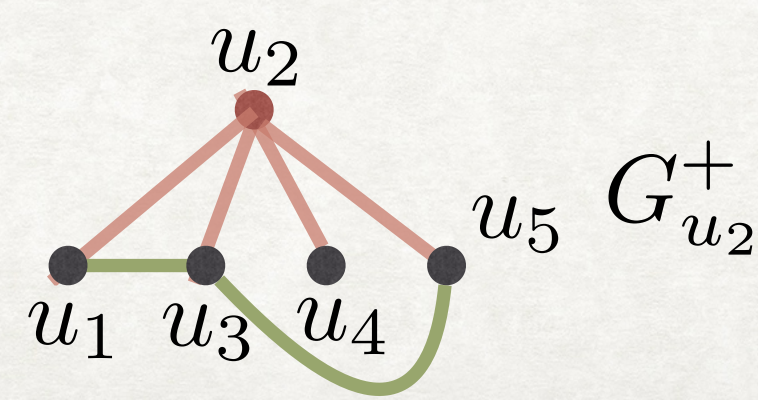
$$V(G_u^+) = V(G_u) = \{u\} \cup \mathcal{N}(u)$$
$$E(G_u^+) = E(G_u) \; \cup$$
$$\{(u', u'') | (u', u'') \in E(G) \wedge u', u'' \in \mathcal{N}(u)\}$$

NEIGHBOUR EDGES

TRIANGLE EDGES







SEED

SCP GRAPH STORAGE

- We show that SCP graph storage supports using both star and clique as the join units

- A more compact version which has **bounded** size for each local graph

SEED

OPTIMAL BUSHY JOIN PLAN

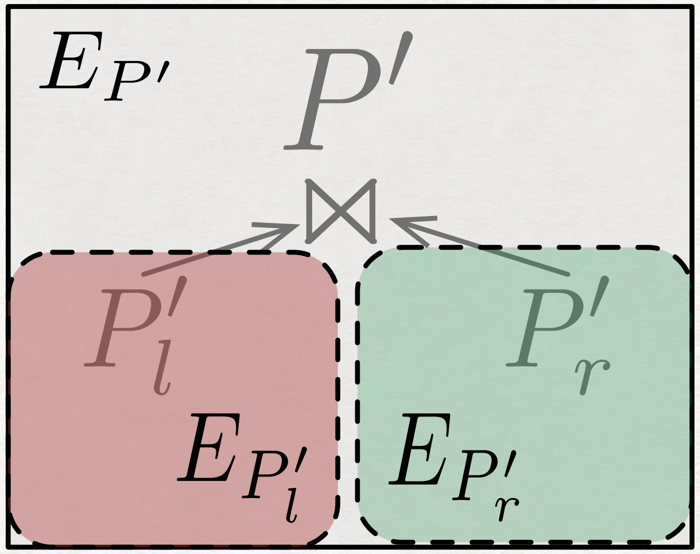
- Notations
  - E_P : The join plan to solve P
  - $C(E_P)$: The cost of the join plan
  - $C(P)$: Estimated # matches of P in G
- We aim at finding a join plan for P , s.t.

$C(E_P)$ **is minimised**

SEED

OPTIMAL BUSHY JOIN PLAN

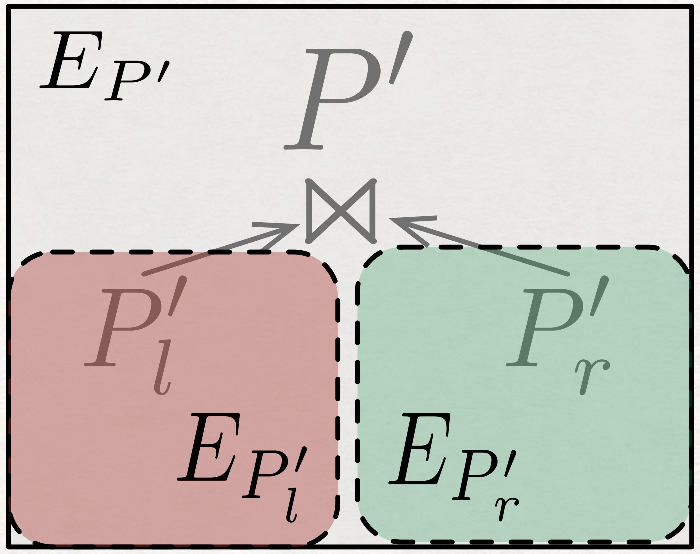
- A dynamic programming transform function
  - e.g. $E_{P'}$
    - (1) $E_{P'_l}$
    - (2) $E_{P'_r}$
    - (3) $R(P') = R(P'_l) \bowtie R(P'_r)$



SEED

OPTIMAL BUSHY JOIN PLAN

- A dynamic programming transform function
  - e.g. $E_{P'}$
    - (1) $E_{P'_l}$
    - (2) $E_{P'_r}$
    - (3) $R(P') = R(P'_l) \bowtie R(P'_r)$



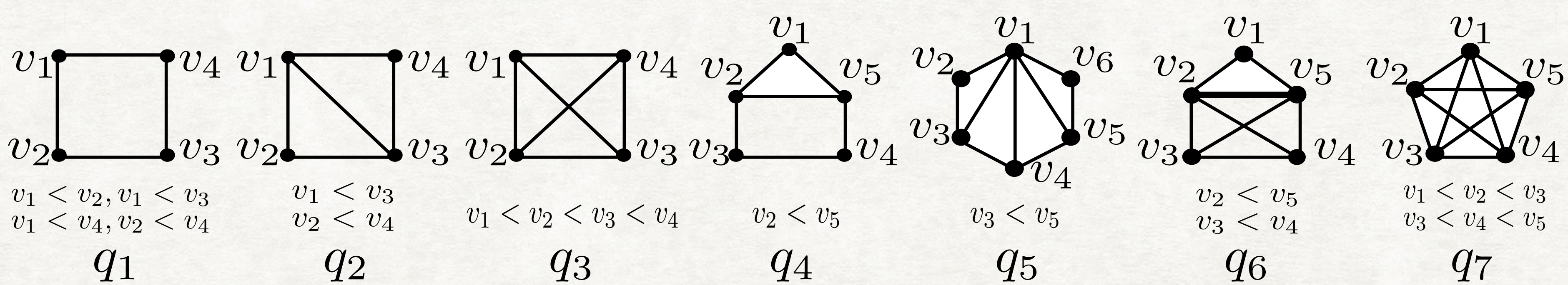
$$C(E_{P'}) = \min_{P'_l \subset P' \wedge P'_r = P' \setminus P'_l} \{C(E_{P'_l}) + C(P'_l) + C(E_{P'_r}) + C(P'_r)\}$$

EXPERIMENTS

EXPERIMENTS

SETUP

- Queries



- Algorithms
  - SEED+O (The most optimised SEED)
  - TT (The most optimised TwinTwigJoin, VLDB 2015)
  - pSgL (Shao et al. Sigmod 2014)

EXPERIMENTS

SETUP

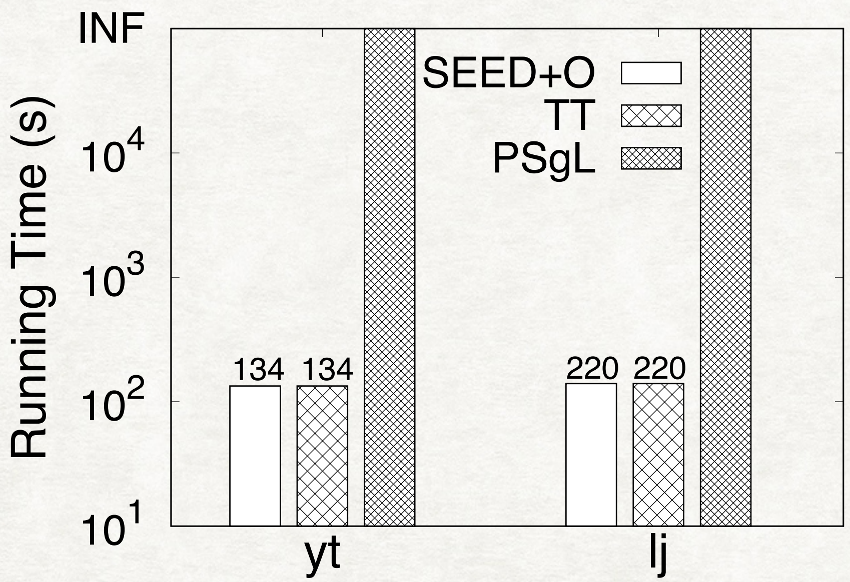
- Cluster
  - Amazon EC2: 1 master node, 10 slave nodes

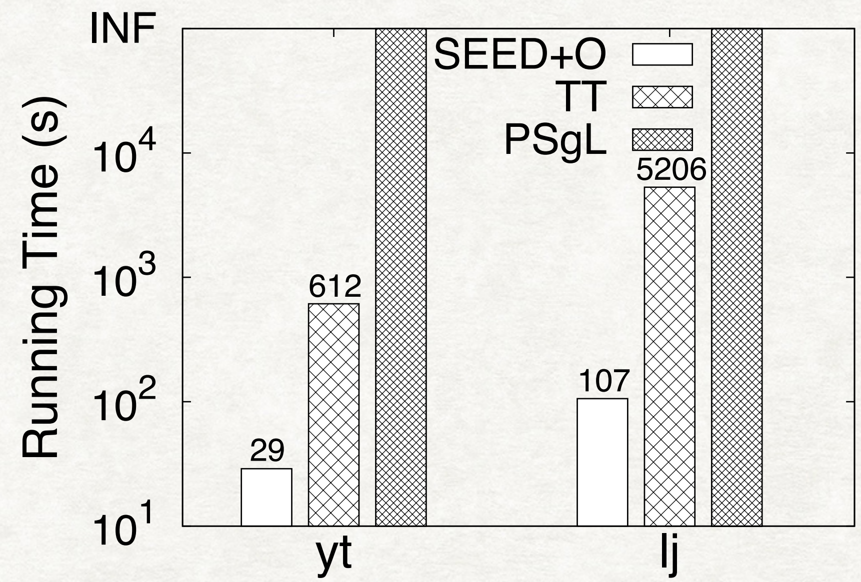
Node	Instance	vCPU	Memory	Disk
master	m3.xlarge	4	15GB	2 x 40GBSSD
slave	c3.4xlarge	16	30GB	2 x 160GB SSD

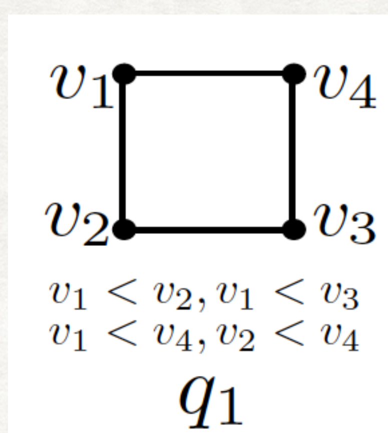
- Hadoop 2.6.2
  - JVM heap space: mapper 1524MB, reducer 2848MB
  - 6 mappers and 6 reducers each machine

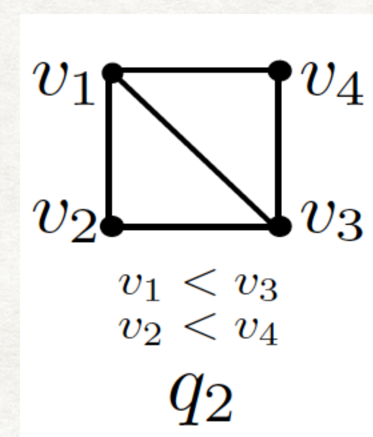
EXPERIMENTS

RESULTS



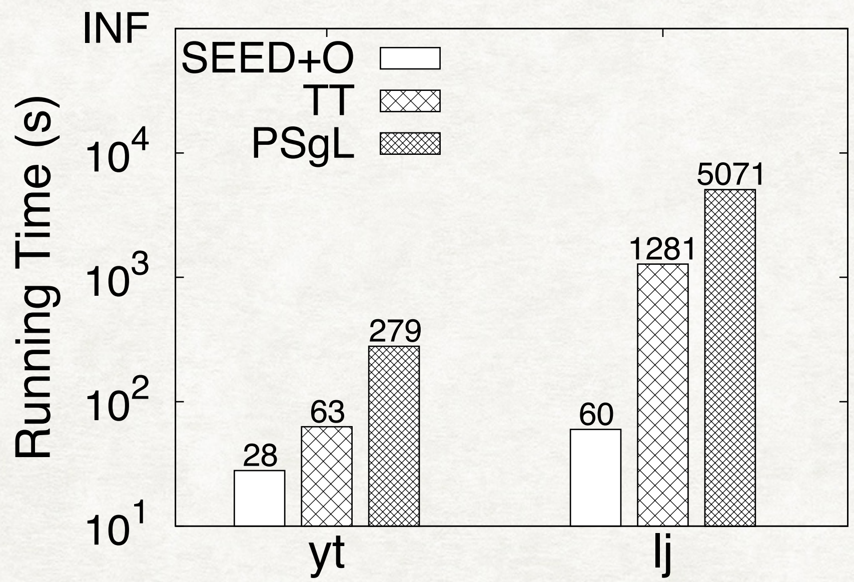


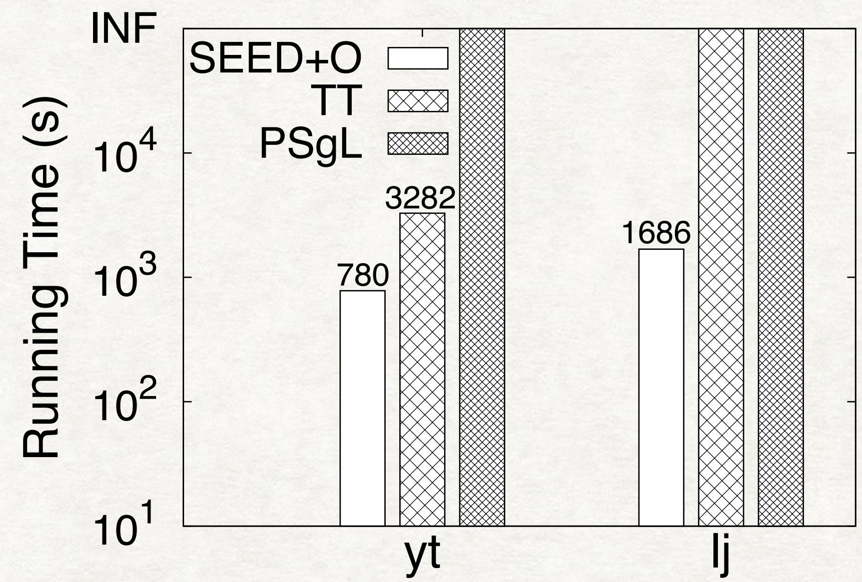


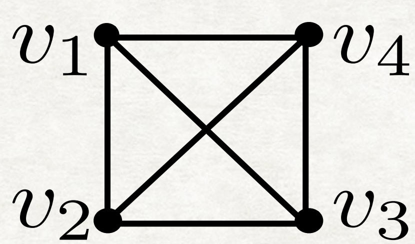


EXPERIMENTS

RESULTS

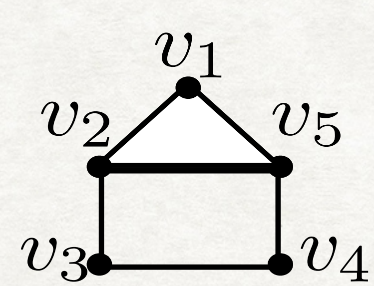






$v_1 < v_2 < v_3 < v_4$

q_3

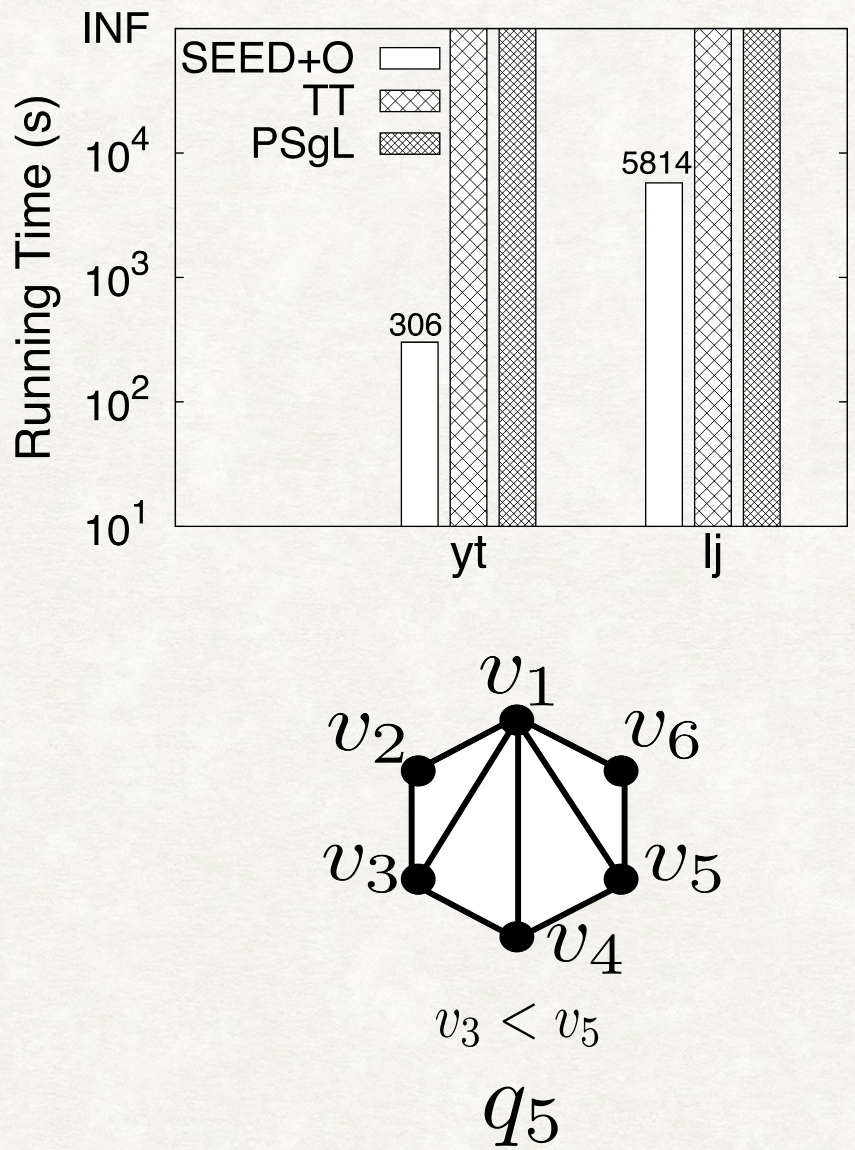


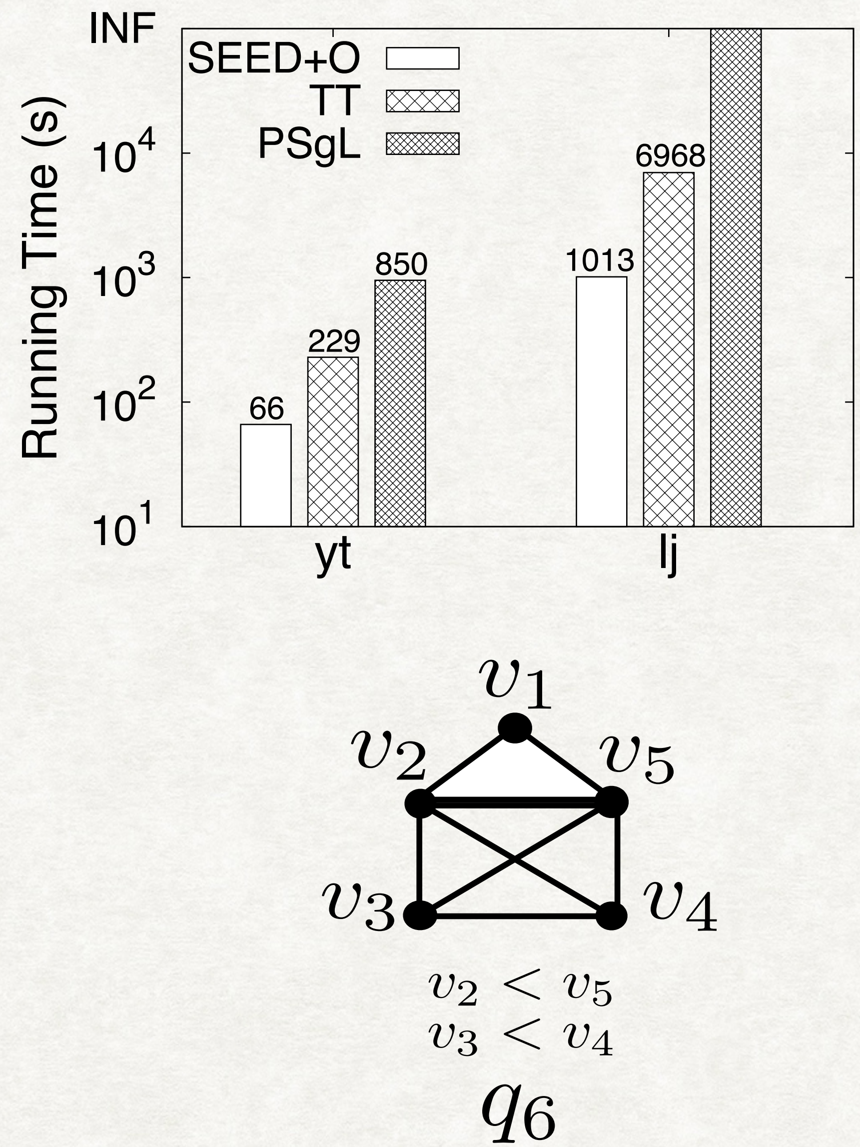
$v_2 < v_5$

q_4

EXPERIMENTS

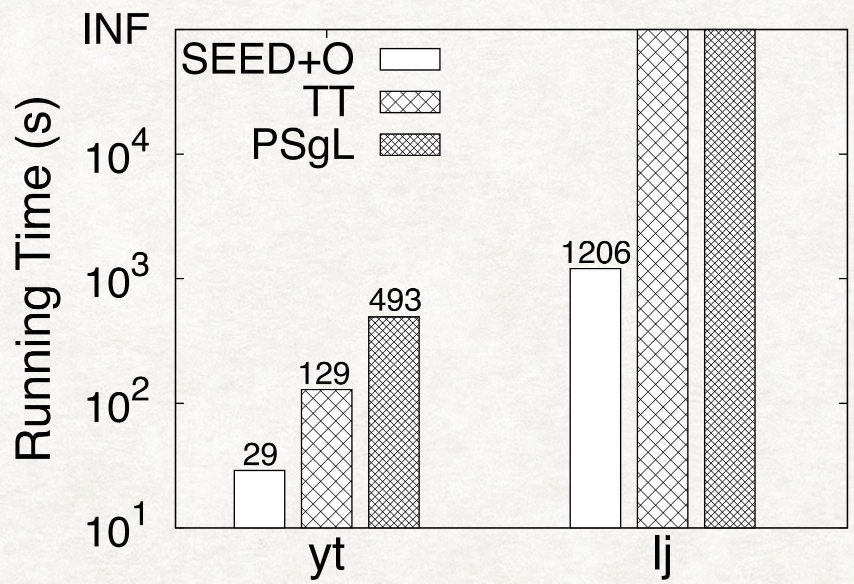
RESULTS

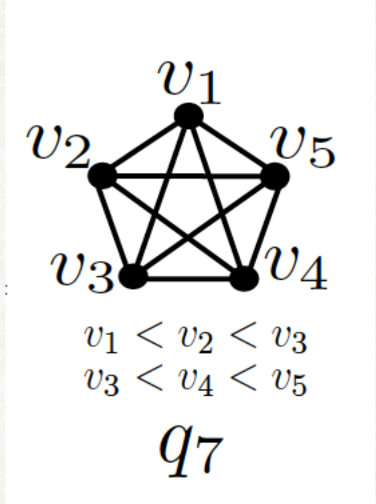




EXPERIMENTS

RESULTS





CONCLUSION

- A general decompose-and-join framework to solve subgraph enumeration
- TwinTwigJoin = Simple graph storage (twintwigs as the join units) + Optimal left-deep join
- SEED = SCP graph storage (star and clique as the join units) + Optimal bushy join

Q & A
THANK YOU!