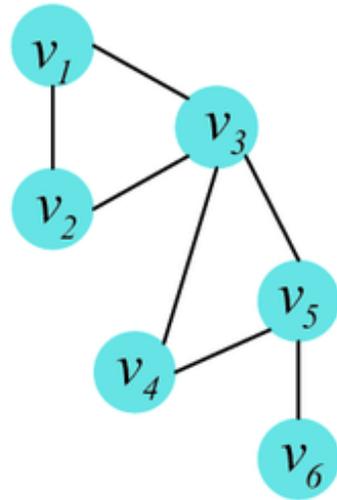


HGMATCH: A Match-by-Hyperedge Approach for Subgraph Matching on Hypergraphs

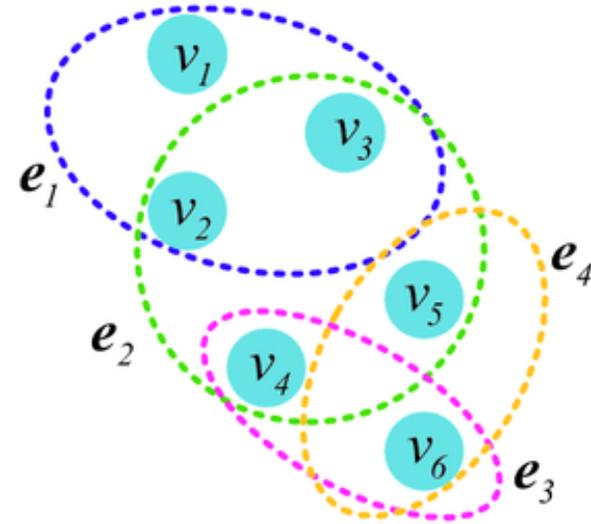
Zhengyi Yang¹, Wenjie Zhang¹, Xuemin Lin², Ying Zhang³, Shunyang Li¹

¹University of New South Wales, ²Shanghai Jiao Tong University, ³University of
Technology Sydney

Graphs vs Hypergraphs

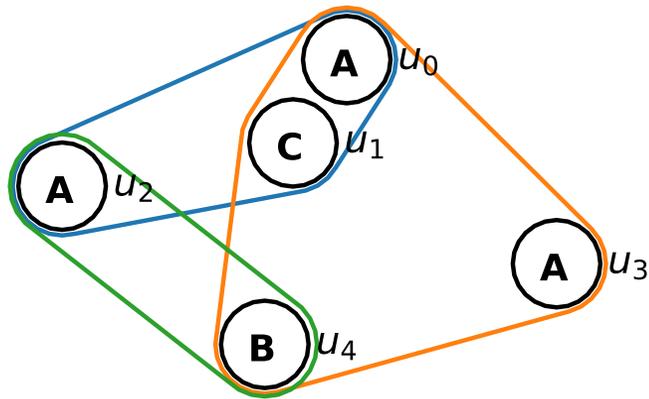


Graph

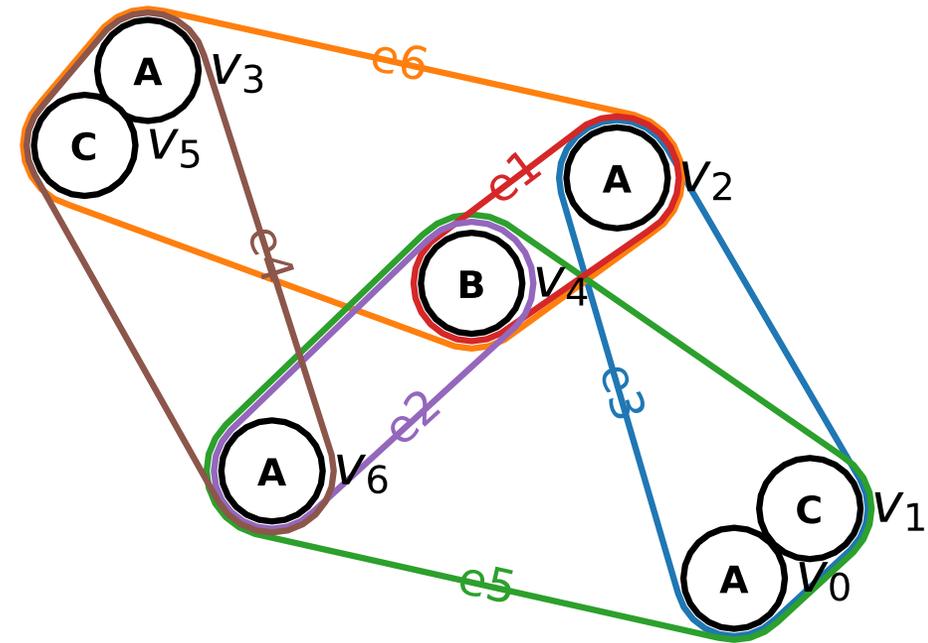


Hypergraph

Subgraph Matching on Hypergraphs

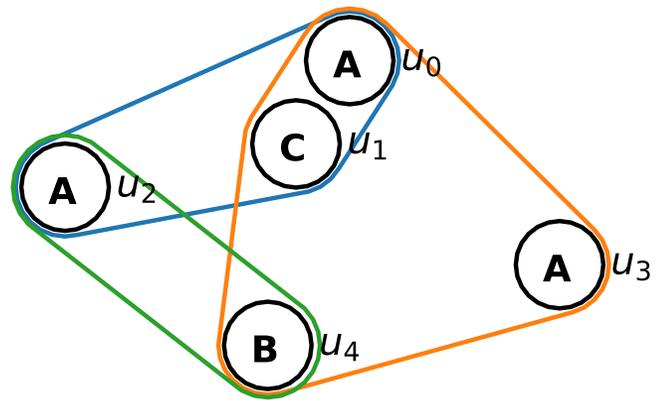


Query Hypergraph

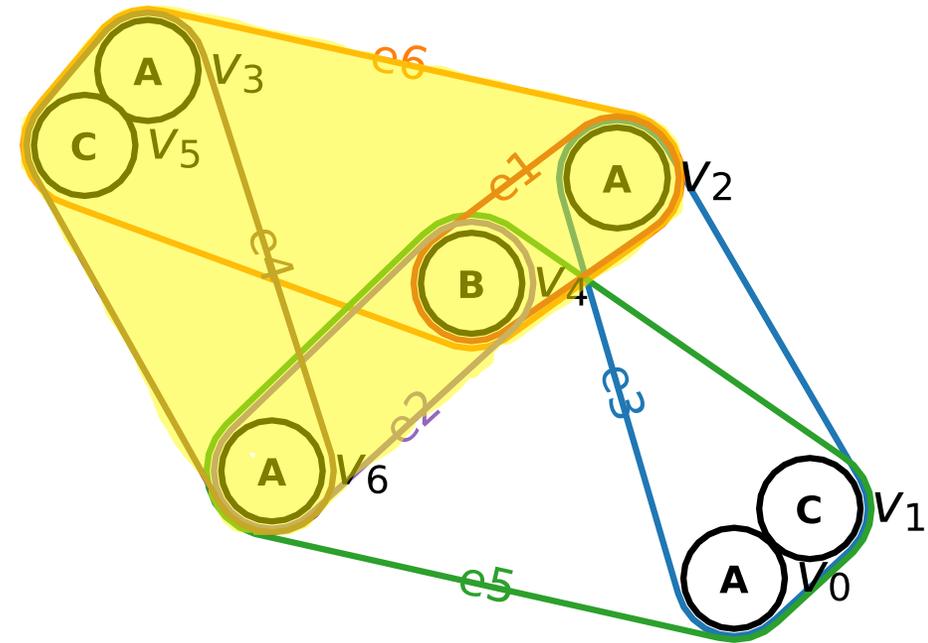
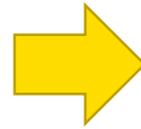


Data Hypergraph

Subgraph Matching on Hypergraphs

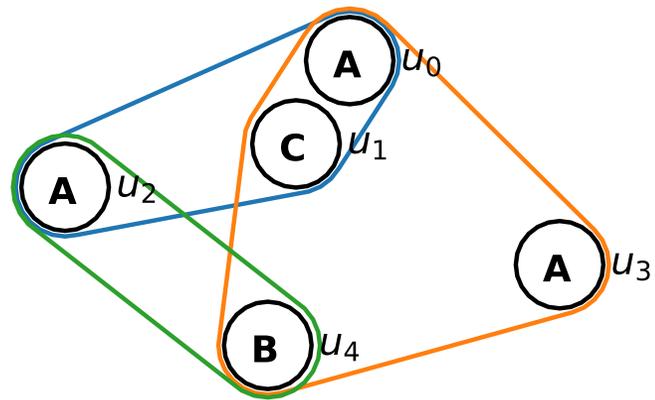


Query Hypergraph

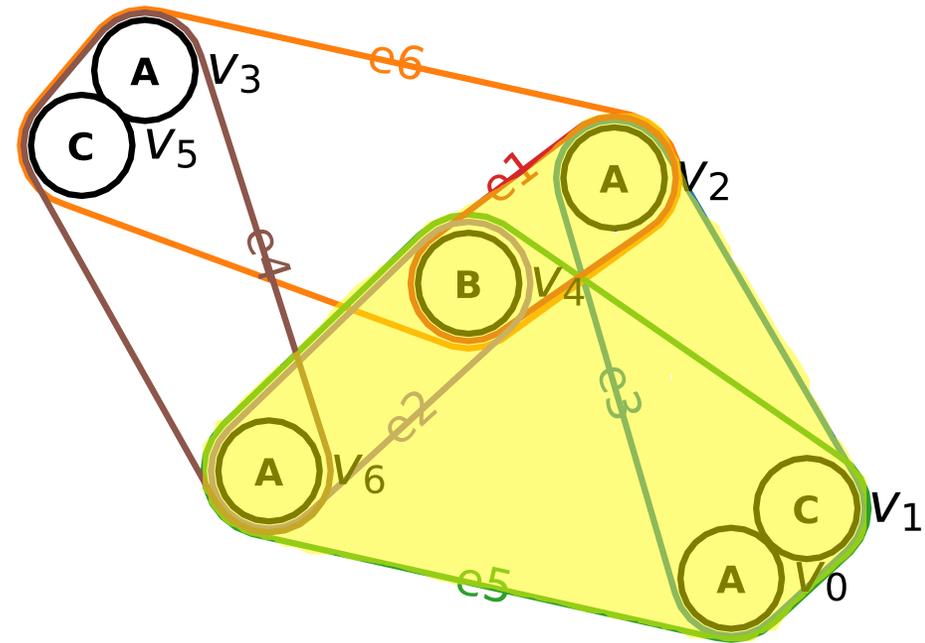
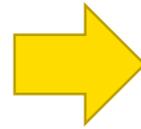


Data Hypergraph

Subgraph Matching on Hypergraphs



Query Hypergraph

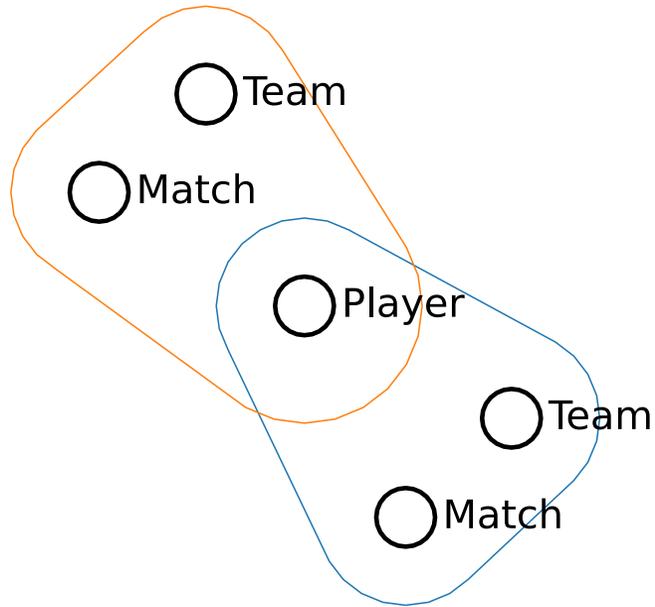


Data Hypergraph

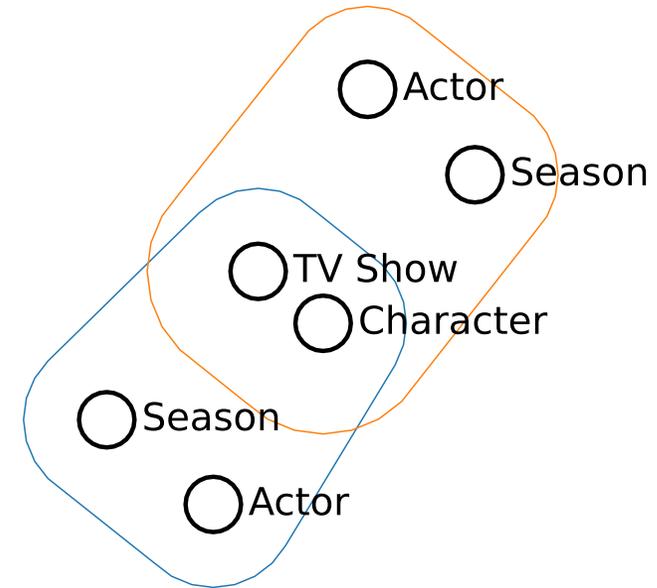
Applications

- Mining Biological Networks
 - e.g., protein interactions, gene interactions
- Querying Hypergraph Databases
 - e.g., AtomSpace, HyperGraphDB, TypeDB
- Pattern Learning in NLP
 - e.g., semantic hypergraphs (each word is a vertex, and each sentence is a hyperedge)
- Q/A over Hypergraph Knowledge Base
 - e.g., JF17K dataset (a subset of non-binary relations extracted from Freebase)

Example Queries for *JF17K* Dataset



Which football players represented different teams in different matches?



Which actors played the same character in a TV show on different seasons?

Strawman Approach

- Convert the hypergraph to a bipartite graph and apply existing subgraph matching algorithms
 - by taking the incidence matrix and treating this as the incidence matrix of a bipartite graph
- Directly extend existing subgraph matching algorithms to the case of hypergraphs
 - recursively expand the partial embedding **vertex-by-vertex** by mapping a query vertex to a data vertex following a given matching order and backtrack when necessary

Motivations

1. The match-by-vertex approach in the strawman approaches generally underutilise high-order information in hypergraphs
 - hyperedges are used as a verification condition in the match-by-vertex framework, which can lead to a huge search space and large enumeration cost
2. It is difficult to compute subgraph matching on massive hypergraphs using sequential algorithms
 - none of the existing subhypergraph matching algorithms supports parallel execution

Contributions

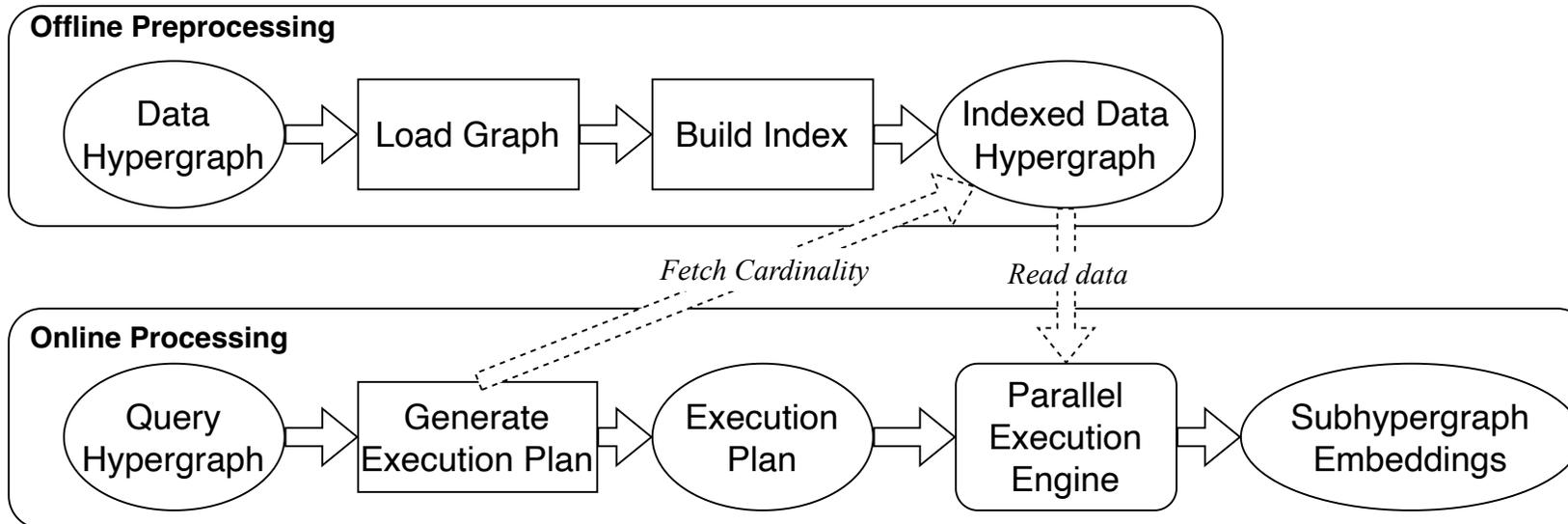
1. **A match-by-hyperedge framework**

- Match the query by hyperedges instead of vertices
- Use set operations to efficiently generate candidates
- Filter out false positives with set comparison

2. **A highly optimised parallel execution engine**

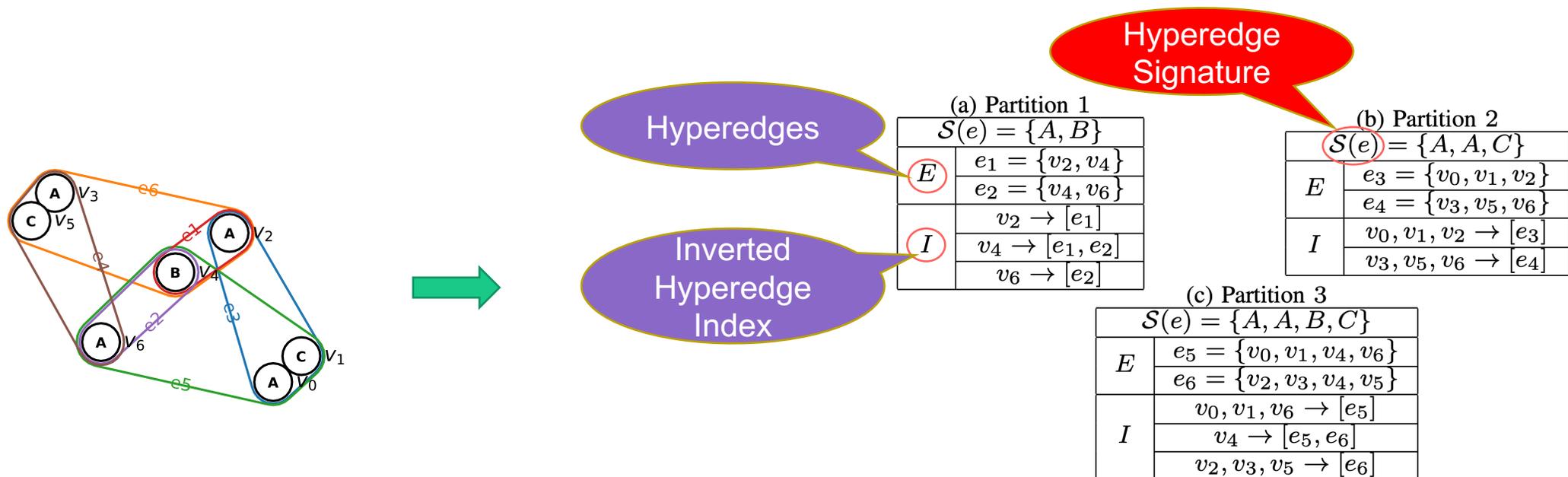
- Adopt the dataflow model for parallelisation
- Bounded memory consumption with our task-based scheduler
- Load balancing with dynamic work-stealing

HGMATCH Overview

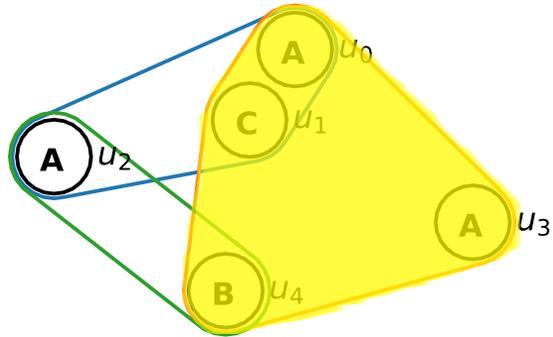


Hypergraph Data Layout

- Hypergraphs are stored as **hyperedge tables** with **inverted hyperedge index**
 - Hyperedge Signature**: a multiset of all vertex labels contained in a hyperedge



Match-by-Hyperedge Framework



Suppose partial result $m = (e_1, e_3)$, we want to match $\{u_0, u_1, u_3, u_4\}$ the next data hyperedge e .

(a) Partition 1

| $S(e) = \{A, B\}$ | |
|-------------------|------------------------------|
| E | $e_1 = \{v_2, v_4\}$ |
| | $e_2 = \{v_4, v_6\}$ |
| I | $v_2 \rightarrow [e_1]$ |
| | $v_4 \rightarrow [e_1, e_2]$ |
| | $v_6 \rightarrow [e_2]$ |

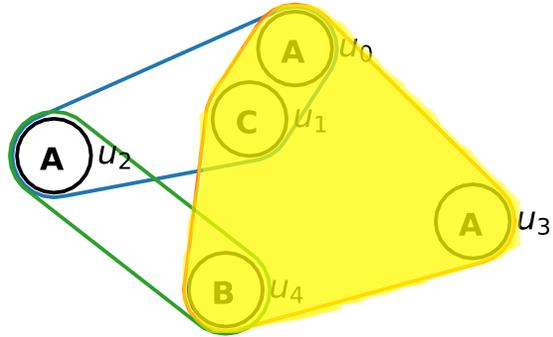
(b) Partition 2

| $S(e) = \{A, A, C\}$ | |
|----------------------|-----------------------------------|
| E | $e_3 = \{v_0, v_1, v_2\}$ |
| | $e_4 = \{v_3, v_5, v_6\}$ |
| I | $v_0, v_1, v_2 \rightarrow [e_3]$ |
| | $v_3, v_5, v_6 \rightarrow [e_4]$ |

(c) Partition 3

| $S(e) = \{A, A, B, C\}$ | |
|-------------------------|-----------------------------------|
| E | $e_5 = \{v_0, v_1, v_4, v_6\}$ |
| | $e_6 = \{v_2, v_3, v_4, v_5\}$ |
| I | $v_0, v_1, v_6 \rightarrow [e_5]$ |
| | $v_4 \rightarrow [e_5, e_6]$ |
| | $v_2, v_3, v_5 \rightarrow [e_6]$ |

Match-by-Hyperedge Framework



Suppose partial result $m = (e_1, e_3)$, we want to match $\{u_0, u_1, u_3, u_4\}$ the next data hyperedge e .

- e must have the same signature with the query hyperedge

(a) Partition 1

| $S(e) = \{A, B\}$ | |
|-------------------|------------------------------|
| E | $e_1 = \{v_2, v_4\}$ |
| | $e_2 = \{v_4, v_6\}$ |
| I | $v_2 \rightarrow [e_1]$ |
| | $v_4 \rightarrow [e_1, e_2]$ |
| | $v_6 \rightarrow [e_2]$ |

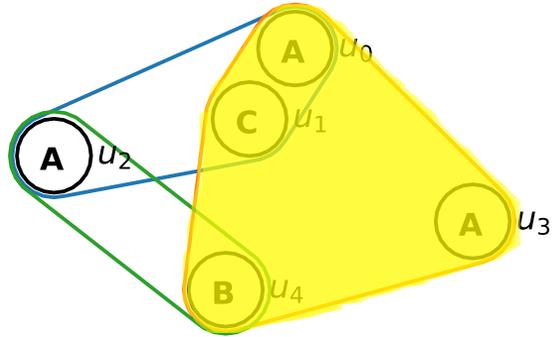
(b) Partition 2

| $S(e) = \{A, A, C\}$ | |
|----------------------|-----------------------------------|
| E | $e_3 = \{v_0, v_1, v_2\}$ |
| | $e_4 = \{v_3, v_5, v_6\}$ |
| I | $v_0, v_1, v_2 \rightarrow [e_3]$ |
| | $v_3, v_5, v_6 \rightarrow [e_4]$ |

(c) Partition 3

| $S(e) = \{A, A, B, C\}$ | |
|-------------------------|-----------------------------------|
| E | $e_5 = \{v_0, v_1, v_4, v_6\}$ |
| | $e_6 = \{v_2, v_3, v_4, v_5\}$ |
| I | $v_0, v_1, v_6 \rightarrow [e_5]$ |
| | $v_4 \rightarrow [e_5, e_6]$ |
| | $v_2, v_3, v_5 \rightarrow [e_6]$ |

Match-by-Hyperedge Framework



Suppose partial result $m = (e_1, e_3)$, we want to match $\{u_0, u_1, u_3, u_4\}$ the next data hyperedge e .

- e must have the same signature with the query hyperedge
- e must be incident to $v_4 \in e_1$ and $v_0, v_1 \in e_3$

(a) Partition 1

| $S(e) = \{A, B\}$ | |
|-------------------|------------------------------|
| E | $e_1 = \{v_2, v_4\}$ |
| | $e_2 = \{v_4, v_6\}$ |
| I | $v_2 \rightarrow [e_1]$ |
| | $v_4 \rightarrow [e_1, e_2]$ |
| | $v_6 \rightarrow [e_2]$ |

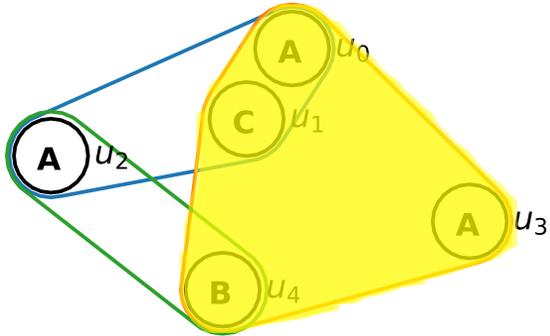
(b) Partition 2

| $S(e) = \{A, A, C\}$ | |
|----------------------|-----------------------------------|
| E | $e_3 = \{v_0, v_1, v_2\}$ |
| | $e_4 = \{v_3, v_5, v_6\}$ |
| I | $v_0, v_1, v_2 \rightarrow [e_3]$ |
| | $v_3, v_5, v_6 \rightarrow [e_4]$ |

(c) Partition 3

| $S(e) = \{A, A, B, C\}$ | |
|-------------------------|-----------------------------------|
| E | $e_5 = \{v_0, v_1, v_4, v_6\}$ |
| | $e_6 = \{v_2, v_3, v_4, v_5\}$ |
| I | $v_0, v_1, v_6 \rightarrow [e_5]$ |
| | $v_4 \rightarrow [e_5, e_6]$ |
| | $v_2, v_3, v_5 \rightarrow [e_6]$ |

Match-by-Hyperedge Framework



Suppose partial result $m = (e_1, e_3)$, we want to match $\{u_0, u_1, u_3, u_4\}$ the next data hyperedge e .

- e must have the same signature with the query hyperedge
- e must be incident to $v_4 \in e_1$ and $v_0, v_1 \in e_3$

(a) Partition 1

| $S(e) = \{A, B\}$ | |
|-------------------|------------------------------|
| E | $e_1 = \{v_2, v_4\}$ |
| | $e_2 = \{v_4, v_6\}$ |
| I | $v_2 \rightarrow [e_1]$ |
| | $v_4 \rightarrow [e_1, e_2]$ |
| | $v_6 \rightarrow [e_2]$ |

(b) Partition 2

| $S(e) = \{A, A, C\}$ | |
|----------------------|-----------------------------------|
| E | $e_3 = \{v_0, v_1, v_2\}$ |
| | $e_4 = \{v_3, v_5, v_6\}$ |
| I | $v_0, v_1, v_2 \rightarrow [e_3]$ |
| | $v_3, v_5, v_6 \rightarrow [e_4]$ |

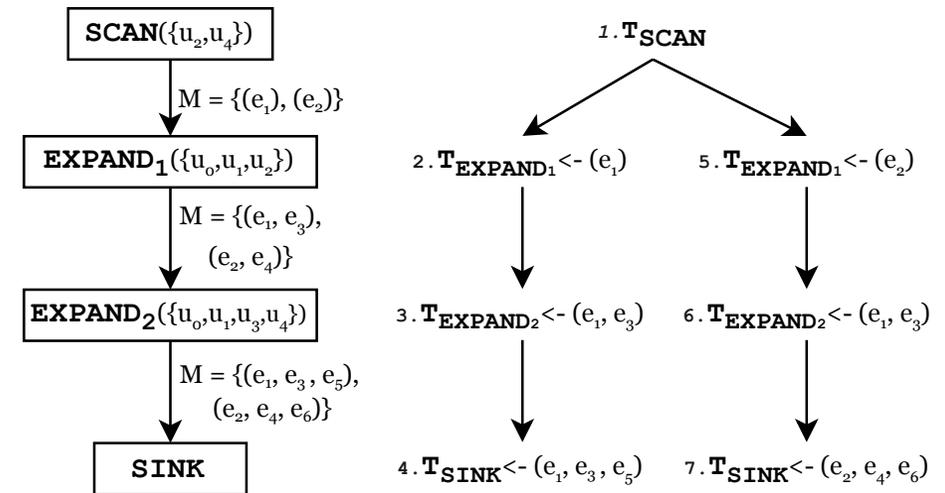
(c) Partition 3

| $S(e) = \{A, A, B, C\}$ | |
|-------------------------|-----------------------------------|
| E | $e_5 = \{v_0, v_1, v_4, v_6\}$ |
| | $e_6 = \{v_2, v_3, v_4, v_5\}$ |
| I | $v_0, v_1, v_6 \rightarrow [e_5]$ |
| | $v_4 \rightarrow [e_5, e_6]$ |
| | $v_2, v_3, v_5 \rightarrow [e_6]$ |

$$\Rightarrow C(e) = \{e_5\} \cap \{e_5\} \cap \{e_5, e_6\} = \{e_5\}$$

Parallel Execution

- Dataflow Model
 - We designed three operators: SCAN, EXPAND, SINK
- Task-based Scheduler
 - Computation are broken down into tasks and scheduled in LIFO order to bound memory
- Dynamic Work Stealing
 - Idle worker will steal tasks from others for load balancing



Example Dataflow Graph and Task Tree

Experimental Setup

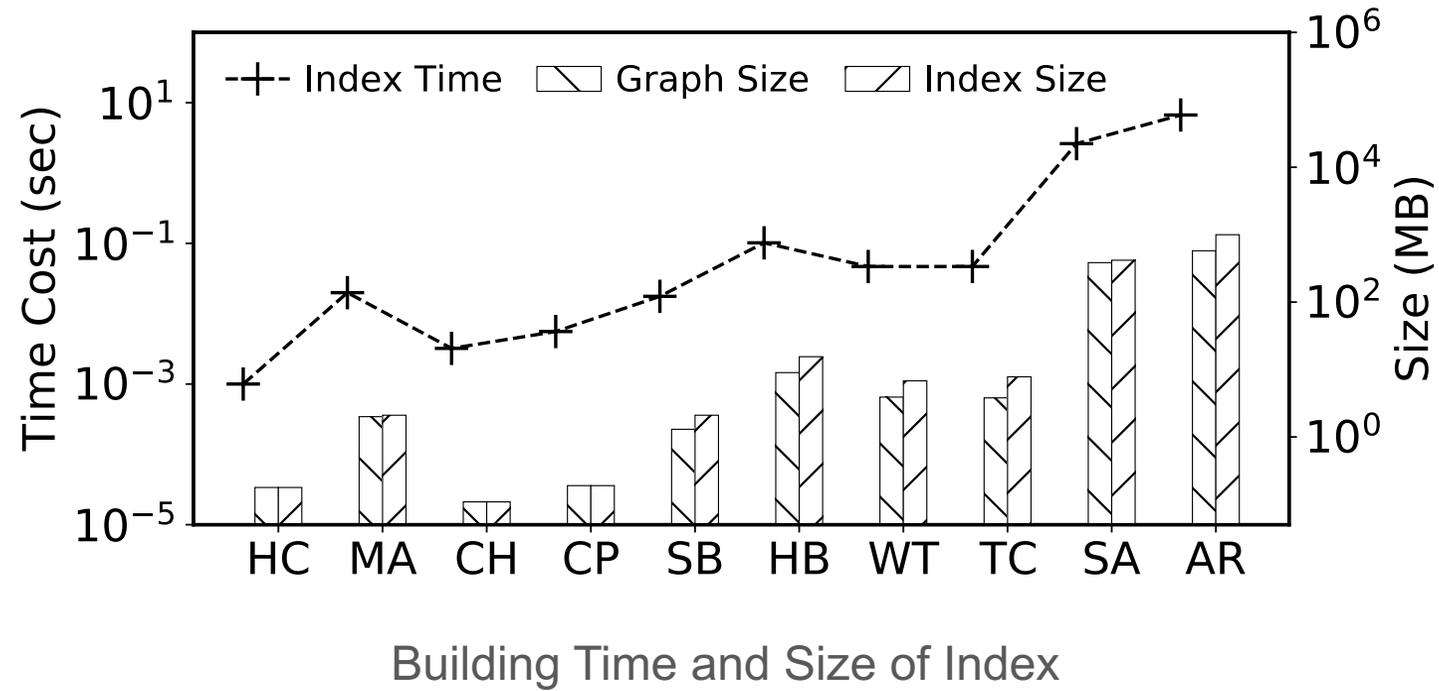
- **Hardware:** a server with two 20-core Xeon E5-2698 V4 CPU and 512G of memory
- **Baselines:** we propose a generic framework to extend existing subgraph matching algorithms to the case of hypergraphs
 - We compared the extended version of *CFL* (SIGMOD16), *DAF* (SIGMOD19), *CECI* (SIGMOD19), and *RapidMatch* (VLDB20)
- **Queries:** randomly sample subhypergraphs from the data hypergraphs with given number of hyperedges and vertices

Datasets

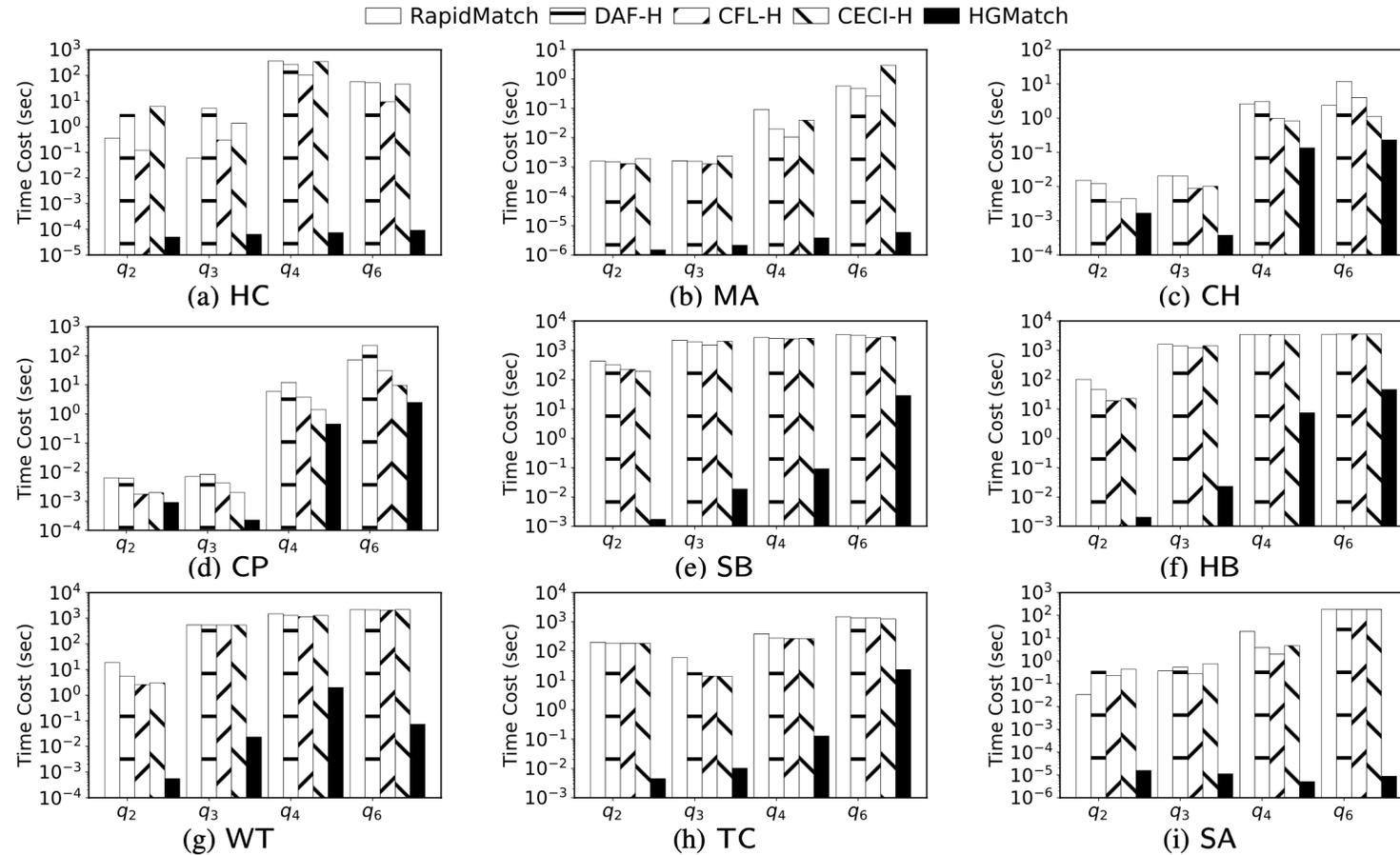
- **Datasets:** we use 10 real-world hypergraphs as data hypergraphs

| Dataset | $ V $ | $ E $ | $ \Sigma $ | a_{max} | \bar{a} | $ Index $ |
|---------|------------|-----------|------------|-----------|-----------|-----------|
| HC | 1,290 | 331 | 2 | 81 | 34.8 | 178KB |
| MA | 73,851 | 5,444 | 1,456 | 1,784 | 24.2 | 2.1MB |
| CH | 327 | 7,818 | 9 | 5 | 2.3 | 109KB |
| CP | 242 | 12,704 | 11 | 5 | 2.4 | 190KB |
| SB | 294 | 20,584 | 2 | 99 | 8.0 | 2.1MB |
| HB | 1,494 | 52,960 | 2 | 399 | 20.5 | 15.5MB |
| WT | 88,860 | 65,507 | 11 | 25 | 6.6 | 6.8MB |
| TC | 172,738 | 212,483 | 160 | 85 | 4.1 | 7.8MB |
| SA | 15,211,989 | 1,103,193 | 56,502 | 61,315 | 23.7 | 419.7MB |
| AR | 2,268,264 | 4,239,108 | 29 | 9,350 | 17.1 | 998.6MB |

Index Building

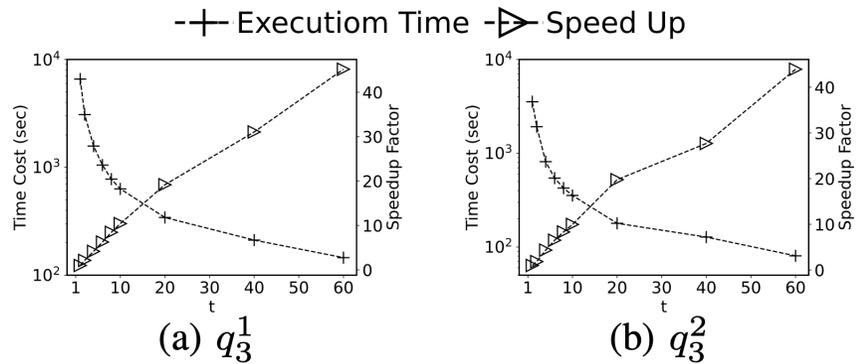


Single-thread Comparisons

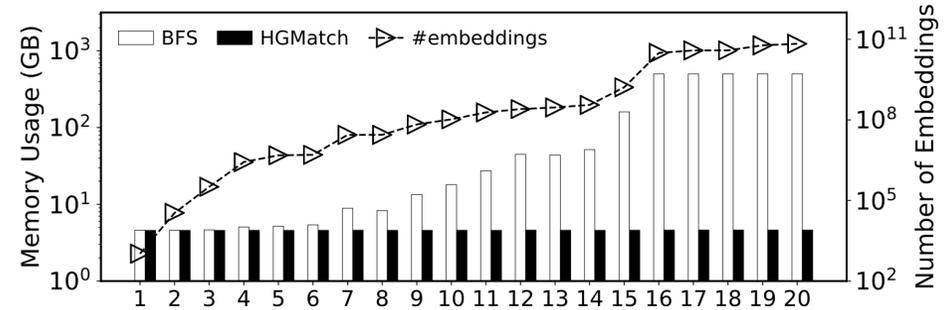


Execution Time for each Query Set

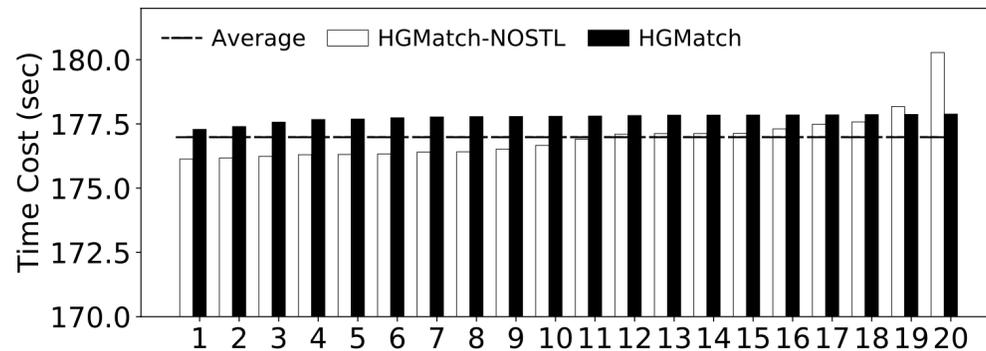
Parallel Comparisons



Vary Number of Threads



Task-based Scheduling



Work Stealing

Thank you!



UNSW
SYDNEY