USENIX ATC '23

July 10-12, 2023
BOSTON, MA, USA

Cyclosa: Redundancy-Free Graph Pattern Mining via Set Dataflow

Chuangyi Gui[1,2], *Xiaofei Liao*[1], *Long Zheng*[1,2], *Hai Jin*[1]

[1]Huazhong University of Science and Technology

[2]Zhejiang Lab

Graph Pattern Mining is Ubiquitous

❖Real-world Graphs

Social Network

Chemical Engineering

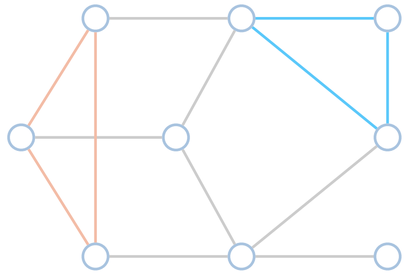
Bio-medicine

...

❖Graph Pattern Mining

Finding subgraphs of interest (patterns) in input graphs

e.g., Finding Triangles



Applications

- Finance: Fraud detection
- Web: Network motifs discovery
- Biology: Drug discovery
- Society: Community detection

Problem: Inherent Redundant Computations

❖Redundancies are prevail in different patterns

- Prior works follow **structural equality** to merge same set formulas[1][2]
- A large number of redundant computations still exist (avg., 60%)

S_1: $v_0 \in V$

S_2: $v_1 \in N(v_0)$

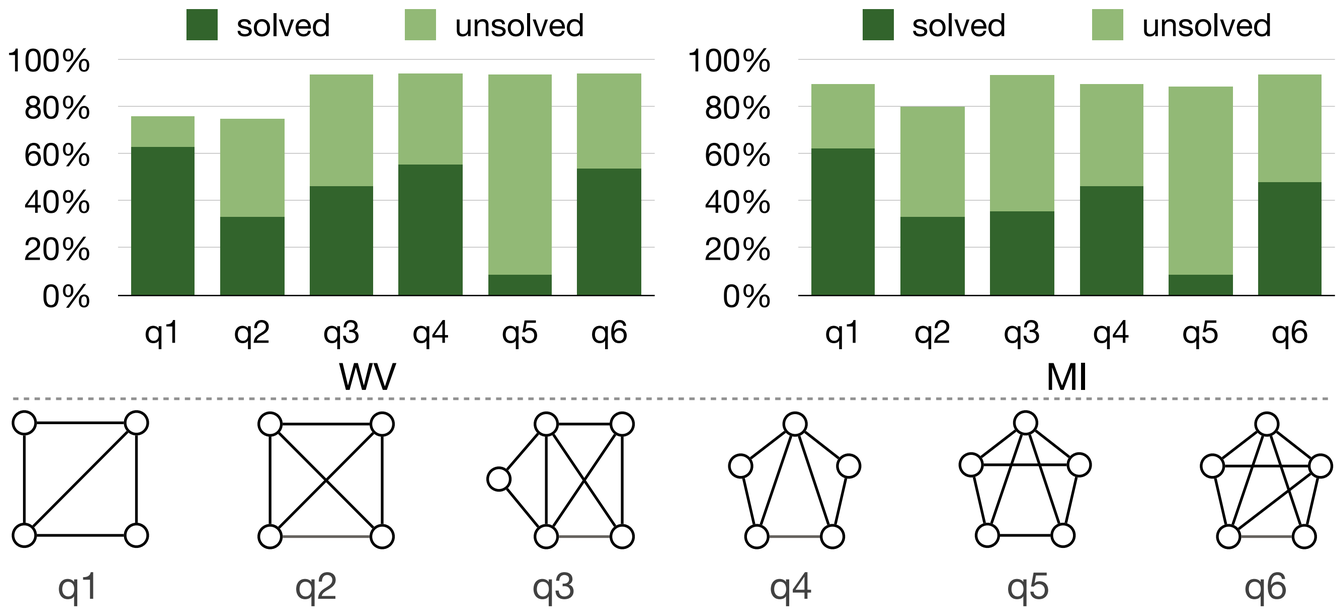
S_3: $v_2 \in N(v_0) \cap N(v_1), v_2 > v_1$

S_4: $v_3 \in N(v_0) \cap N(v_1)$

S_5: $v_4 \in N(v_0) \cap N(v_2)$

$S_3 = S_4 \rightarrow S'$

Rewrite to one



Implicit redundancies occur **during runtime** and are hard to predict

[1] GraphPi [SC'20]

[2] SumPA [PACT'21]

New Opportunity: Computation Similarity

❖A deeper dive: Two types of computation similarities

- **Static Similarity:** finer-grained data reuse possibility at the operands level
- **Dynamic Similarity:** which vertices are more likely to be requested during runtime

S_1: $v_0 \in V$

S_2: $v_1 \in N(v_0)$

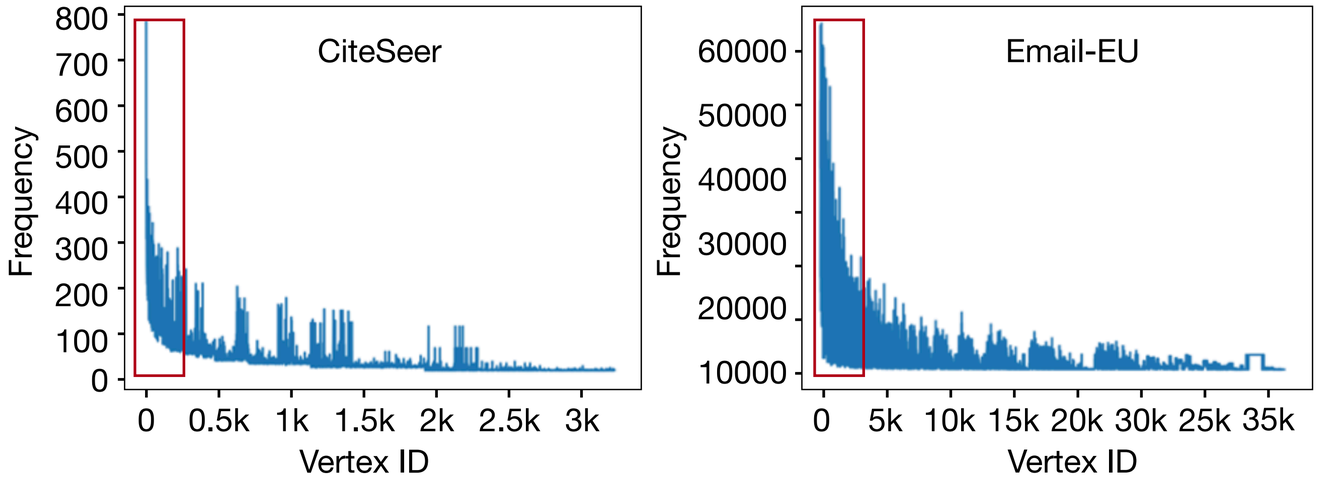
S_3: $v_2 \in N(v_0) \cap N(v_1), v_2 > v_1$

S_4: $v_3 \in N(v_0) \cap N(v_1)$

S_5: $v_4 \in N(v_0) \cap N(v_2)$

Inputs can also be shared

Spatial perspective



>85% computations include **<15% high-degree** vertices

Temporal perspective

Opportunities for predicting and reusing hard-traced redundancies during runtime

New Abstraction: Set Dataflow

- Need an abstraction to exploit the computation similarity
- **Set Dataflow:** A directed graph indicating how sets are transferred
  - Decouples set operands and set operators for fine-grained analysis
  - Indicates data reuse for redundancy elimination

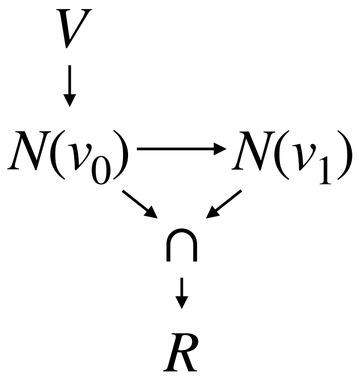
S1: $v_0 \in V$

S2: $v_1 \in N(v_0)$

S3: $v_2 \in N(v_0) \cap N(v_1), v_2 > v_1$

S4: $v_3 \in N(v_0) \cap N(v_1)$

S5: $v_4 \in N(v_0) \cap N(v_2)$



$vertices \rightarrow N(v_k) \rightarrow neighbors$

$[input0, input1] \rightarrow \cap \rightarrow results$

S1 ⟹ **S3**

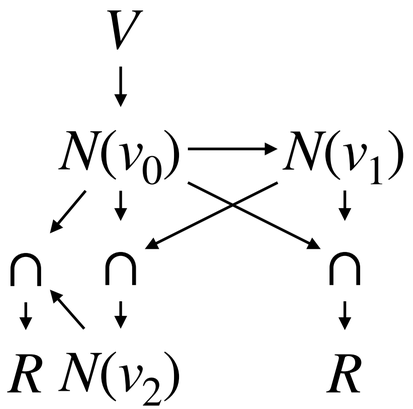
New Abstraction: Set Dataflow

- Need an abstraction to exploit the computation similarity
- **Set Dataflow:** A directed graph indicating how sets are transferred
  - Decouples set operands and set operators for fine-grained analysis
  - Indicates data reuse for redundancy elimination

S1:	$v_0 \in V$
S2:	$v_1 \in N(v_0)$
S3:	$v_2 \in N(v_0) \cap N(v_1), v_2 > v_1$
S4:	$v_3 \in N(v_0) \cap N(v_1)$
S5:	$v_4 \in N(v_0) \cap N(v_2)$

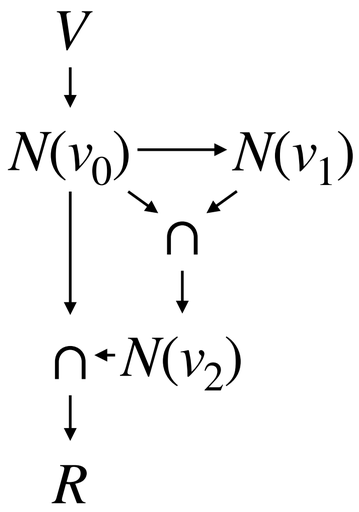
$vertices \rightarrow N(v_k) \rightarrow neighbors$

$[input0, input1] \rightarrow \cap \rightarrow results$



S1 ⇒ **S5**

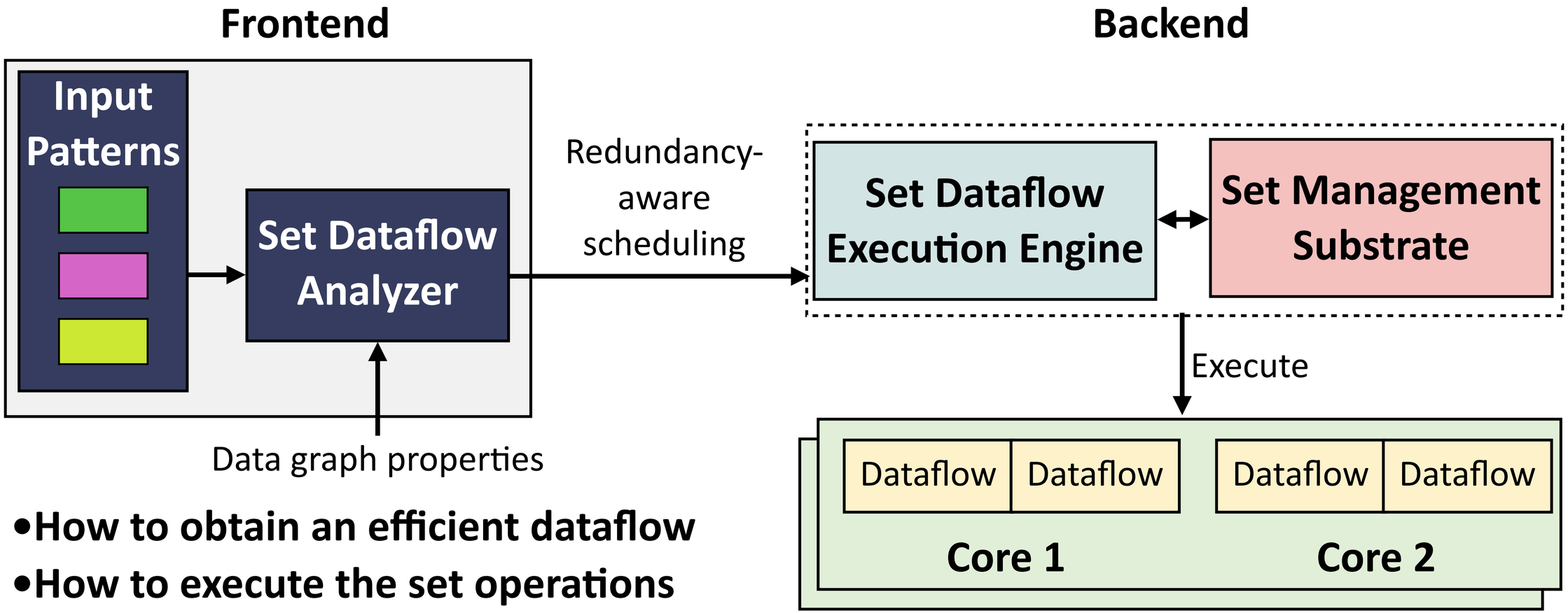
Simplifying



Reduced Operators: 2*∩, 1*$N(v_0)$, 1*$N(v_1)$, 1*$N(v_2)$

Overview of Cyclosa

❖**Cyclosa:** A graph pattern mining framework based on the *set dataflow*

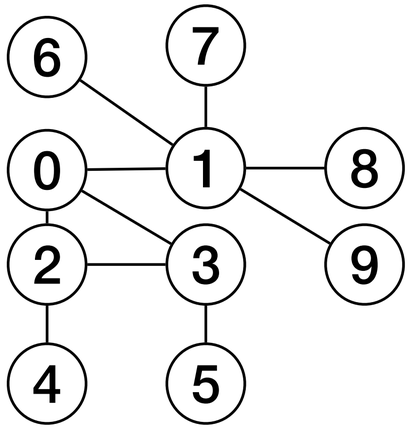


- **How to obtain an efficient dataflow**
- **How to execute the set operations**
- **How to manage the sets data**

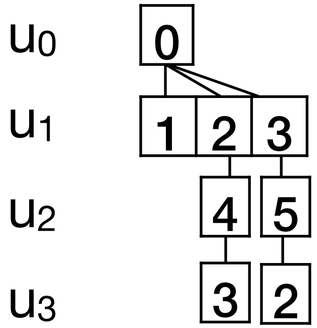
Designing Set Dataflow Analyzer

❖**Challenges:** Embracing small search space and low redundancy

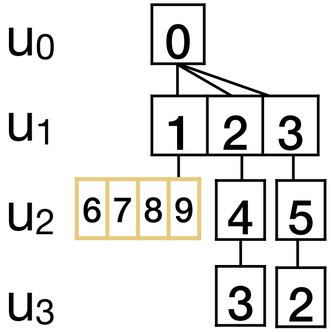
- Various search spaces for different matching orders
- Redundancy is hard to predict

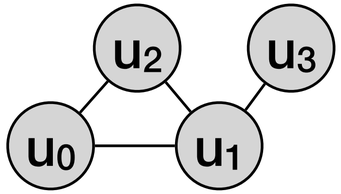


MO₁ [u_0, u_1, u_2, u_3]



MO₂ [u_0, u_1, u_3, u_2]



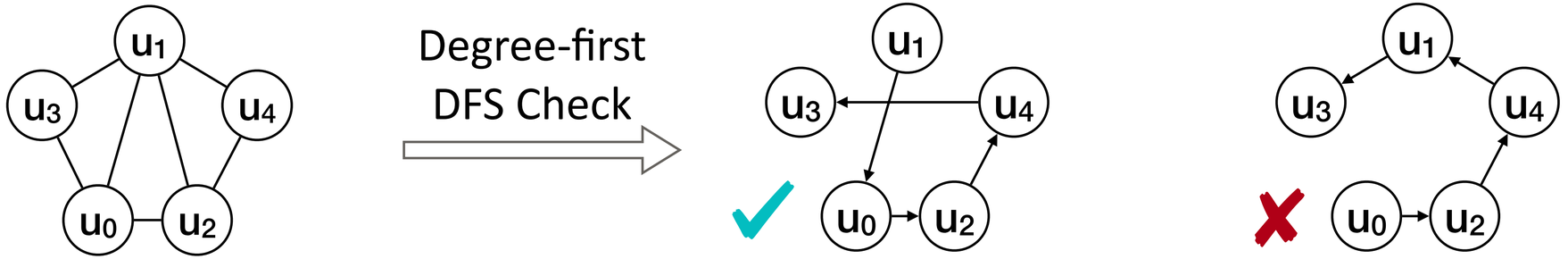


❖**Main idea:** Consider pattern and graph properties at the same time

Redundancy-aware Pattern Analysis

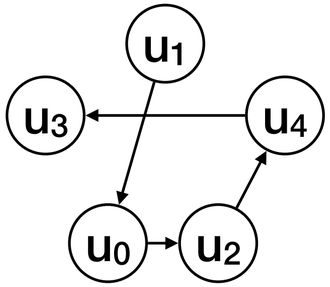
❖**Degree-first DFS Order:** Raising the possibility of results reusing

- High-degree vertices are more likely to be involved into redundancies



❖**Graph-aware Cost Estimation:** Minimizing total workloads

- Combines the degree and triangle count of data graph to improve precision



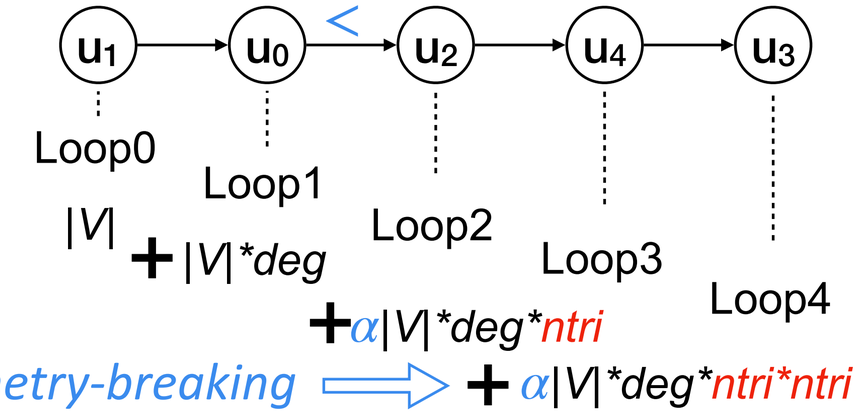
$v_0 \in V$

$v_1 \in N(v_0)$

$v_2 \in N(v_0) \cap N(v_1), v_2 > v_1$

$v_3 \in N(v_0) \cap N(v_1)$

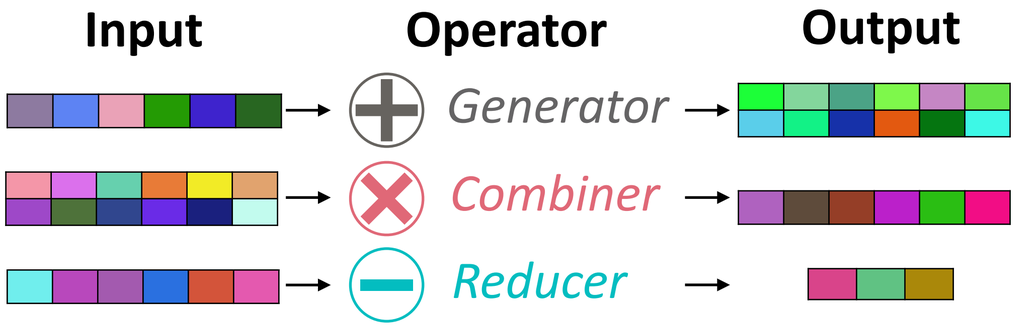
$v_4 \in N(v_0) \cap N(v_2)$

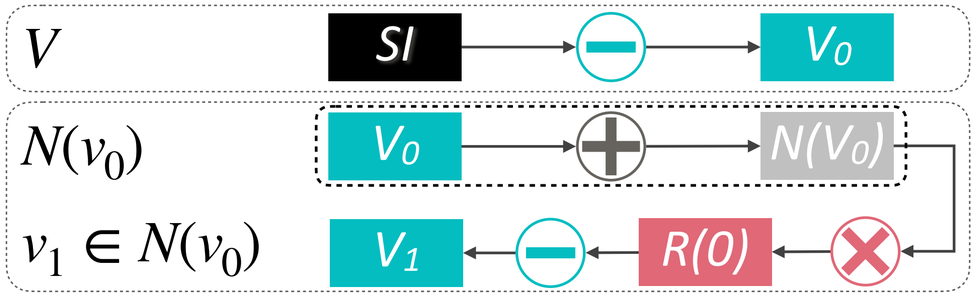


Light-weight Set Dataflow Construction

❖**Set Dataflow Representation:** Decoupled computation and set data

- Extracts basic compute units of graph pattern mining for expressiveness
- Input and output sets are individually identified in the dataflow for data sharing

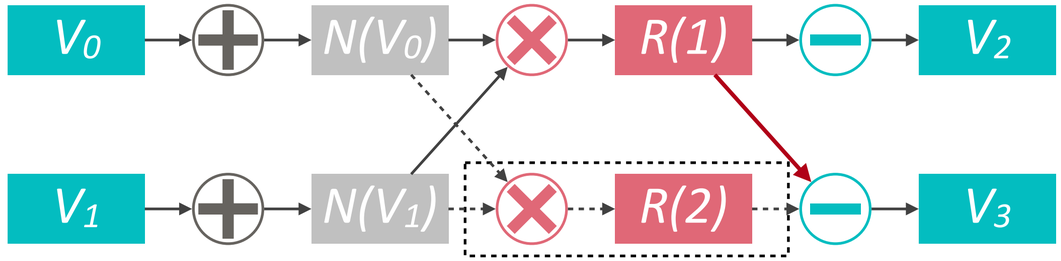




- Eliminates **explicit redundancies** by dataflow cutting

$$v_2 \in N(v_0) \cap N(v_1), v_2 > v_1$$

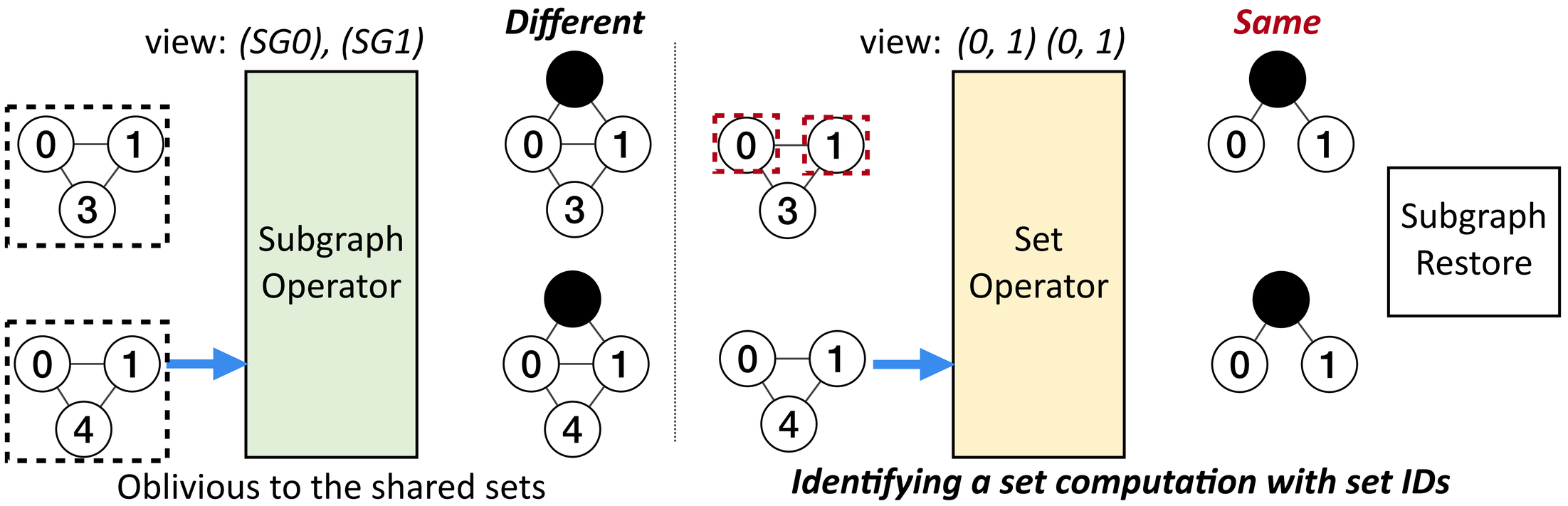
$$v_3 \in N(v_0) \cap N(v_1)$$



Set-centric Dataflow Execution

❖**Set-centric:** Put a set instead of a subgraph as the basic parallel unit

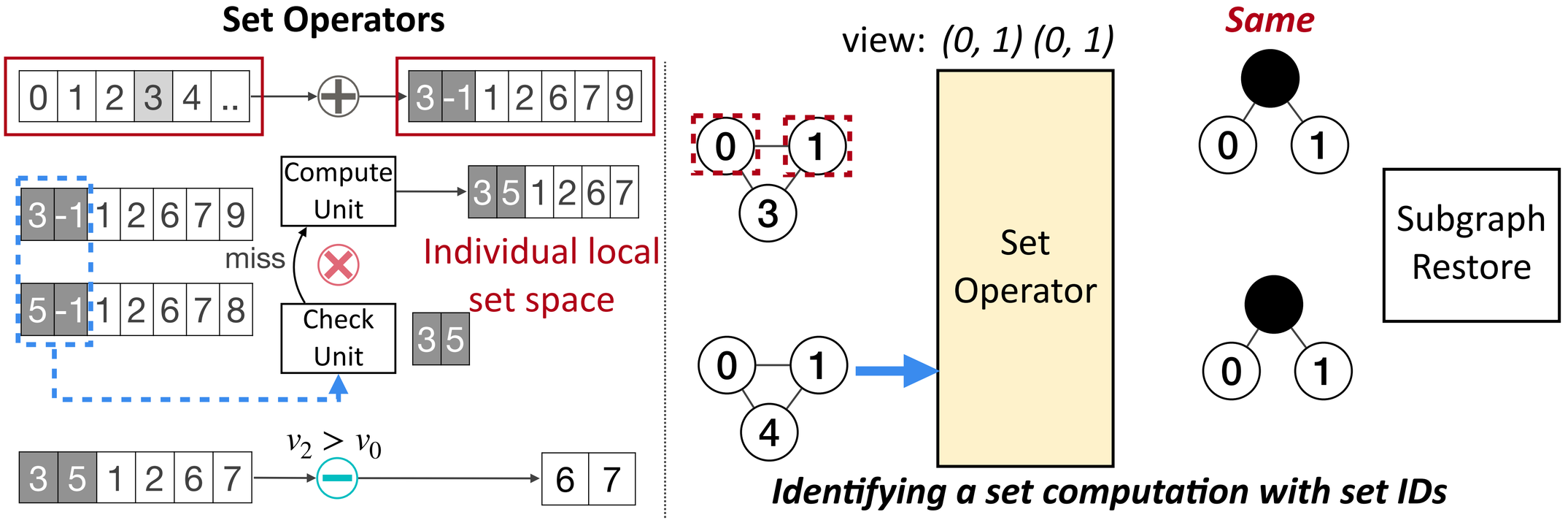
- Subgraphs are oblivious to the shared sets during runtime



Set-centric Dataflow Execution

❖**Set-centric:** Put a set instead of a subgraph as the basic parallel unit

- Subgraphs are oblivious to the shared sets during runtime
- Benefits from the set dataflow scheduling and maintains reusable set space

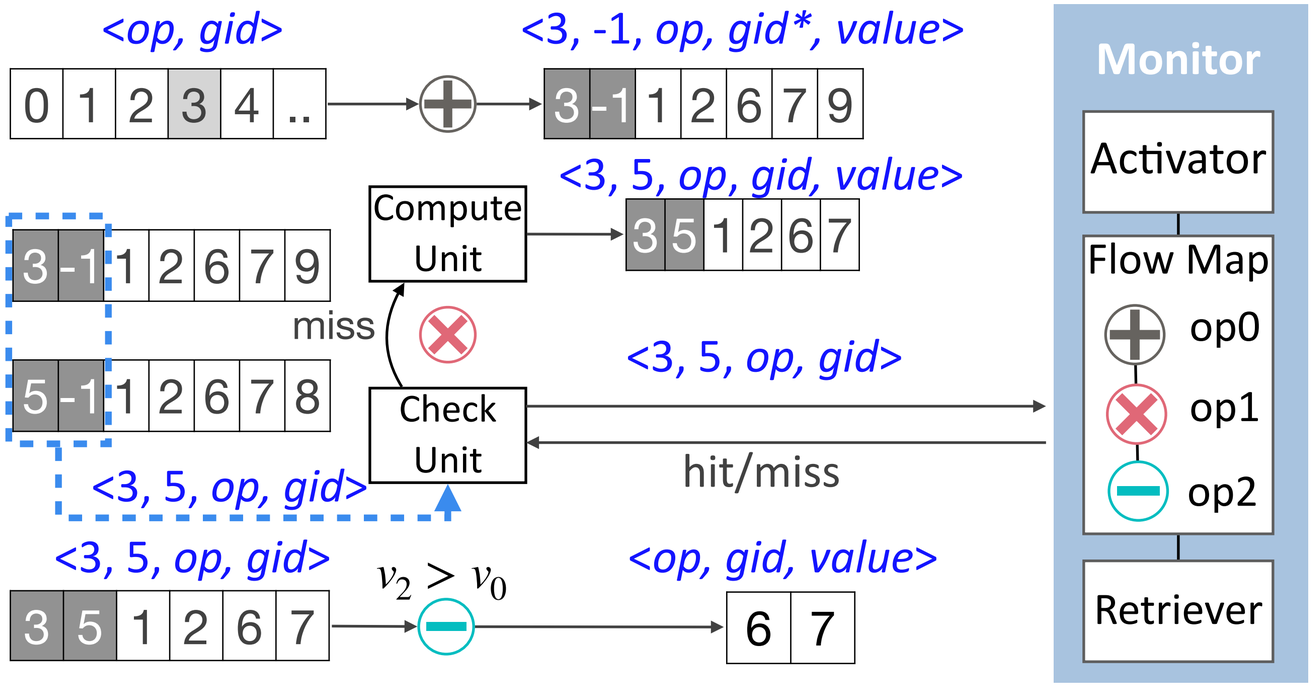


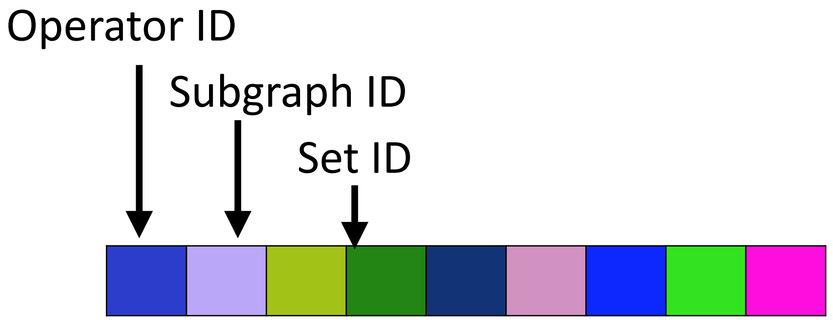
Set-centric Dataflow Execution

❖**Challenges:** Oblivious to the subgraph structure

- Embracing correctness and efficiency is challenging

❖**Solution:** Dual IDs that separate set IDs and scheduling IDs



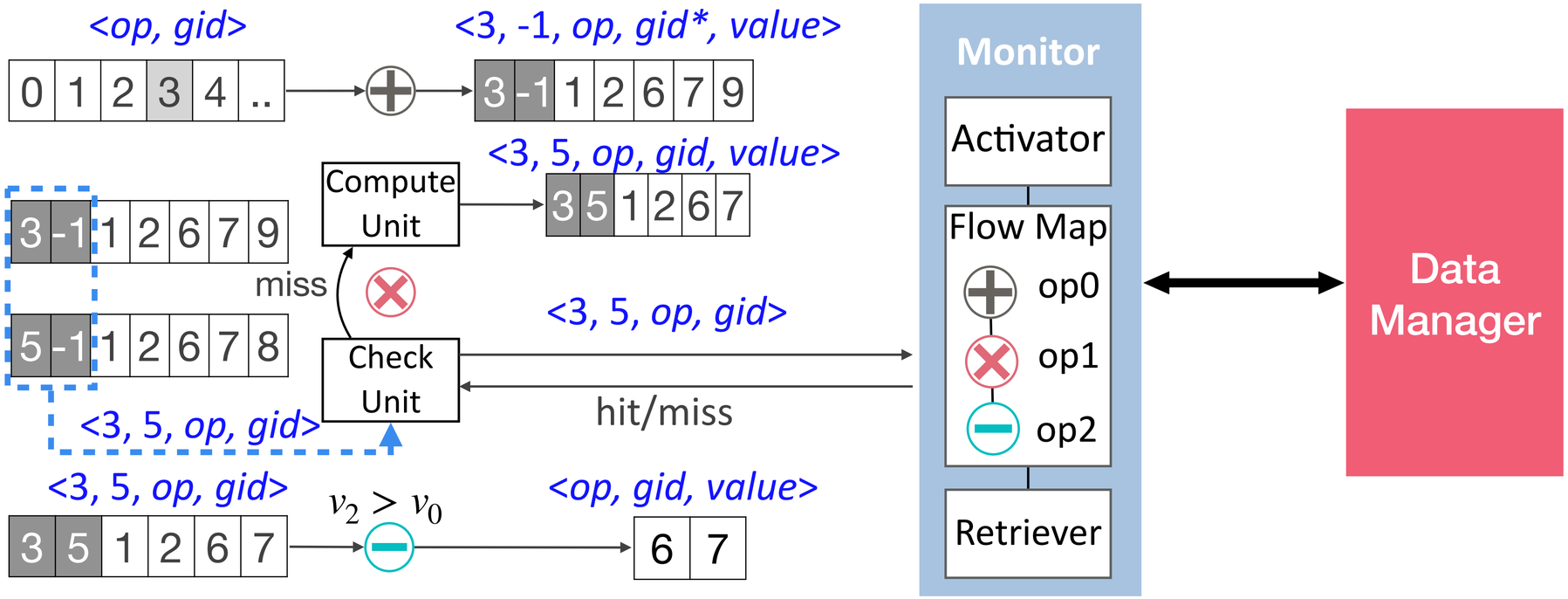


Set-centric Dataflow Execution

❖**Challenges:** Oblivious to the subgraph structure

- Embracing correctness and efficiency is challenging

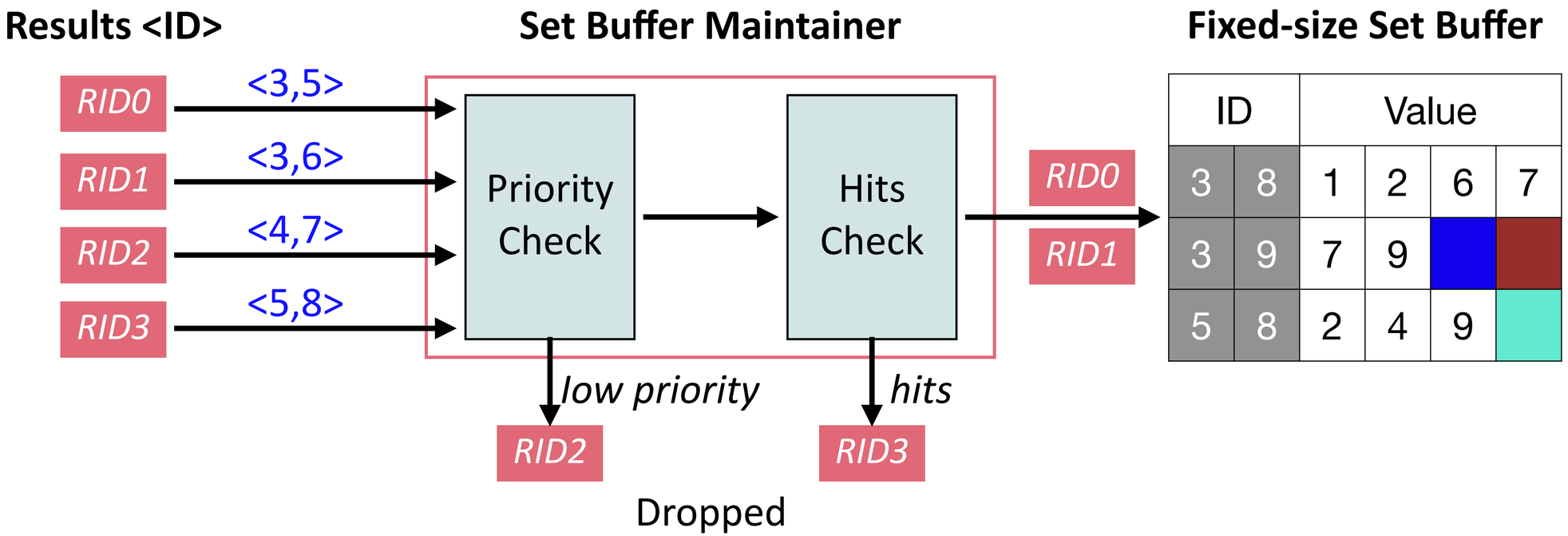
❖**Solution:** Dual IDs that separate set IDs and scheduling IDs



Redundancy-aware Set Management

❖Automatically results caching for set computations

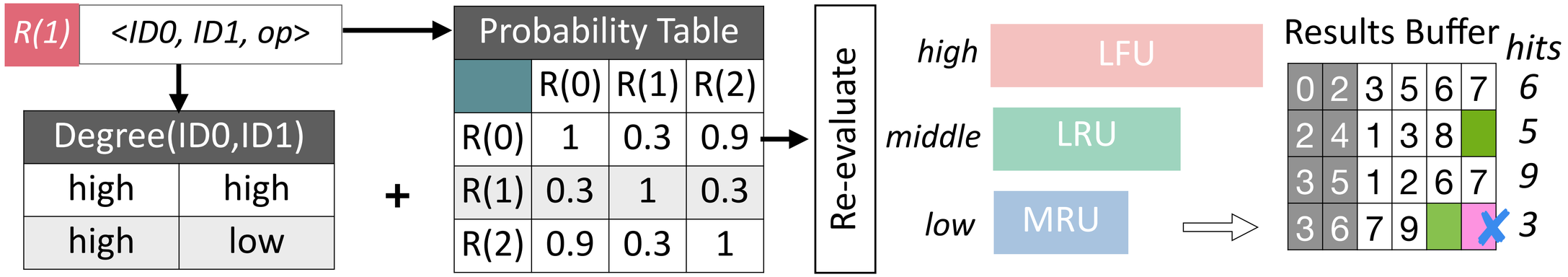
- **Challenges:** Storing all results in limited memory space is impossible
- **Solution:** Selectively storing results with high reuse possibility



Redundancy-aware Set Management

❖Degree-guided Caching Mechanism

- Storing computed results from **high-degree** vertices first (previous observation)
- Priority decision based on the **Combiner position** in the set dataflow



Evaluated Benchmarks

❖Diverse graph patterns
- Single vs. multiple patterns
- Sparse vs. dense patterns
- Patterns with different sizes

(*k*-Cliques, Pseudo Cliques, *k*-Motifs, …)

❖Real-world graphs
- Graphs with 100K to Billion edges

Graphs	|V|	|E|	Size
WikiVote (WV)	7.1K	100.8K	0.81MB
MiCo (MI)	96.6K	1.1M	8.24MB
WikiTalk (WT)	2.39M	5.02M	40.16MB
Patents (PA)	3.8M	16.5M	0.12GB
LiveJournal (LJ)	4.0M	34.7M	0.26GB
Orkut (OR)	3.1M	117.2M	0.87GB
Friendster (FR)	65.6M	1.8B	13.46GB

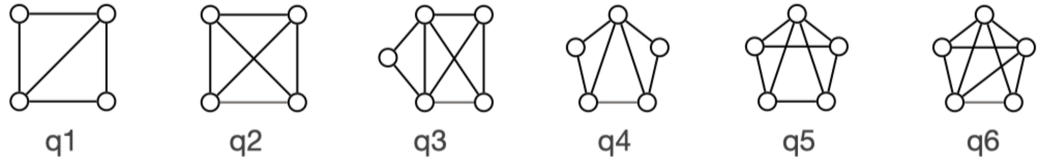
❖State-of-the-arts systems
- **GraphPi:** Best search space
- **SumPA:** Most related

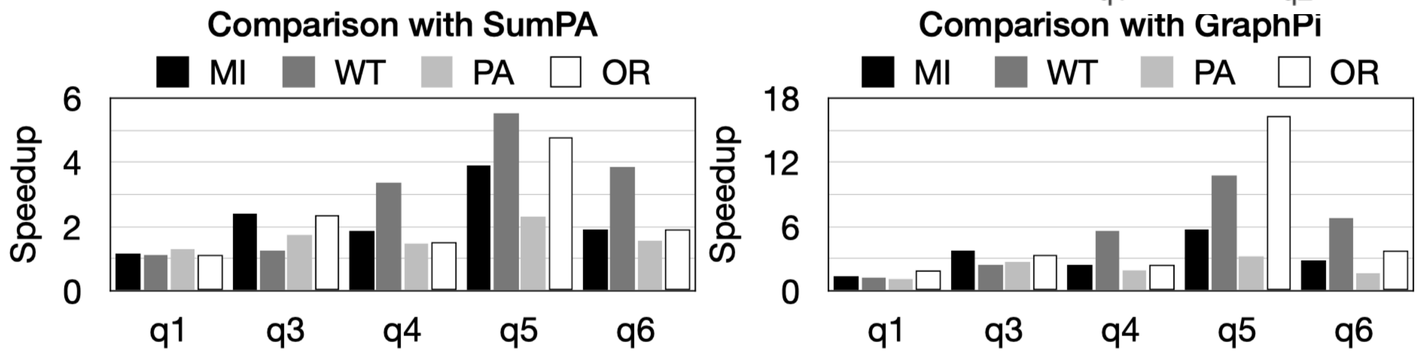
❖Experimental Platform
- 2x14-core E5-2680v4 CPU
- 256GB RAM, 512GB SSD
- 64bit Ubuntu 18.04

Performance Comparison

❖Comparison with state-of-the-arts

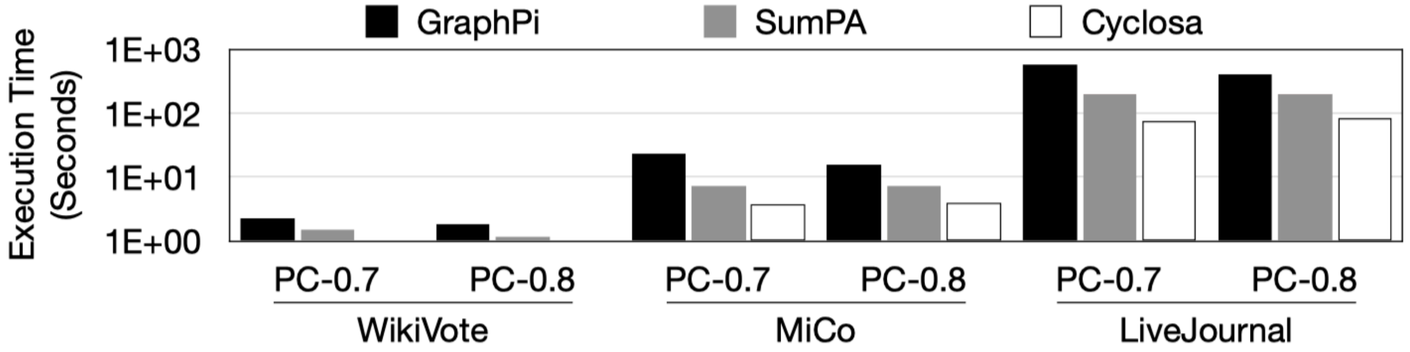
- Subgraph listing for single patterns





- Yielding 1.19× to 16.28× speedup over GraphPi
- Handles both explicit and implicit redundancies well

- Counting multiple pseudo cliques

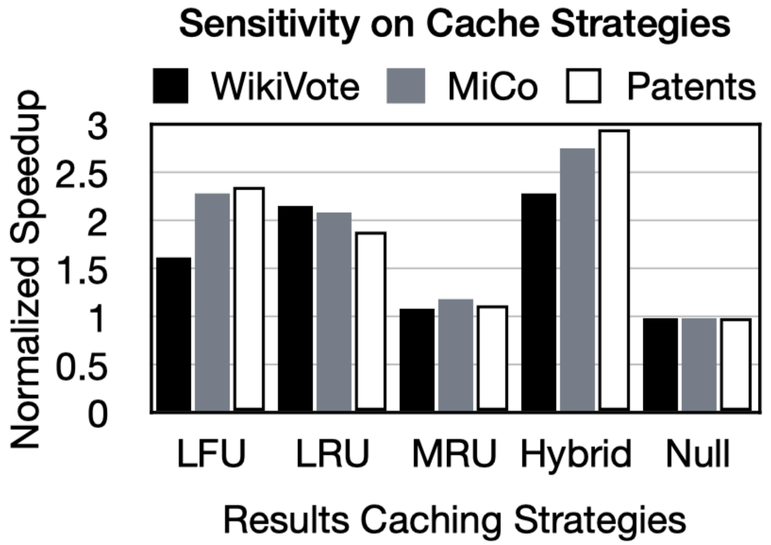


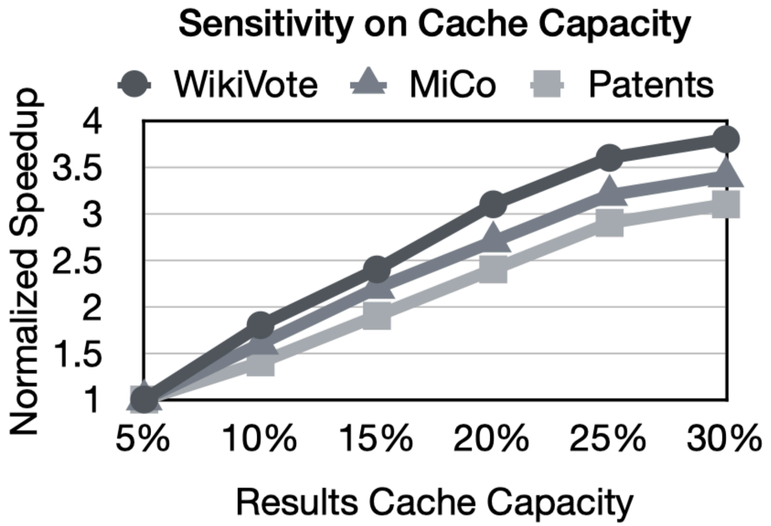
- Yielding 4.01× ~ 7.52× speedup over GraphPi
- Redundancies among multiple patterns are well reduced by set dataflow

Evaluation of Results Caching

❖Efficiency of results caching

- Achieves avg. 2.53x better performance than non-caching
- Reduces more redundancies than existing systems, especially for implicit ones
- Efficient in a limited memory space

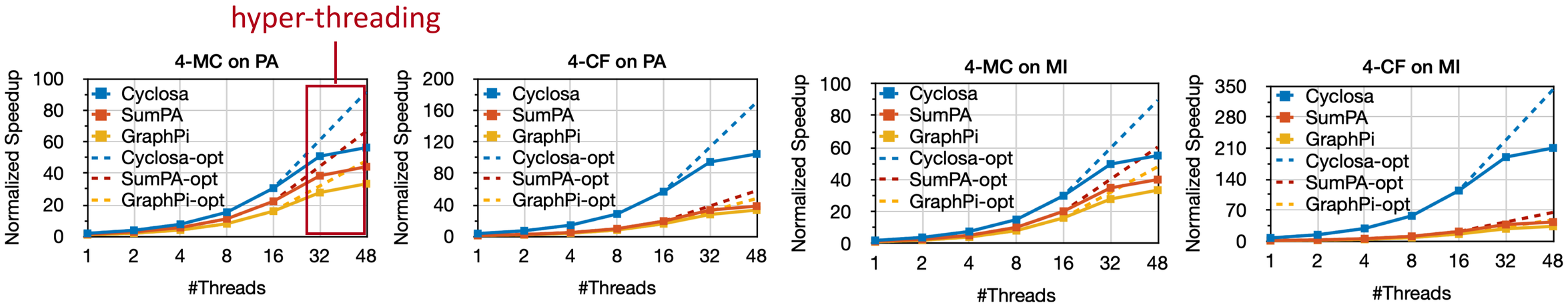




Evaluation of Scaling Threads

❖Scalability on the number of threads

- Scaling well with # threads because of multi-level parallelisms in the set dataflow
- Increasingly-saturated performance improvement due to memory access contention



Overhead

- Memory consumption
  - Relatively small memory footprint with a DFS-style execution mode
  - Set space allocated for each node of the set dataflow is reused for computation
  - Smart caching strategies explore the trade-off between capacity and reusability

Table 5: Memory Consumption on Orkut with 28 Threads

Systems	4-CF	4-MC	5-MC	PC-0.8
Cyclosa	2.69GB	2.75GB	2.96GB	2.94GB
GraphPi	3.81GB	3.83GB	4.16GB	3.97GB

- Dataflow construction time
  - Limited construction overhead because graph patterns are usually small

Table 6: Time for Constructing Set Dataflow

App.	WikiVote	Patents	Orkut
5-CF	1.74ms	1.72ms	1.69ms
5-MC	9.32ms	9.44ms	9.37ms
PC-0.8	3.16ms	3.09ms	3.22ms

Conclusion

- ❖Identifying explicit and implicit redundancies in graph pattern mining
  - •Opportunities from static and dynamic computation similarity
- ❖We propose the concept of set dataflow to remove redundancies
  - •Cost-efficient and redundancy-aware set dataflow analysis
  - •Set-centric dataflow execution model
  - •Memory-friendly results caching and reusing mechanisms
- ❖Advanced performance improvement
  - •Outperforming state-of-the-arts by up to 16.28x