Cyclosa: Redundancy-Free Graph Pattern Mining via Set Dataflow

Chuangyi Gui; Xiaofei Liao; Long Zheng; Hai Jin 2023 USENIX Annual Technical Conference

set operations to guide the exploration and substantially outperform the embedding-centric counterparts that exhaustively enumerate all subgraphs. These systems provide novel specializations to achieve optimum search space, but the inherent redundancies caused by recurrent set intersections on the same or different subgraph instances remain and are difficult to trace, significantly degrading the performance. In this paper, we propose a dataflow-based graph pattern mining framework named Cyclosa to eliminate the above redundancies by utilizing the concept of computation similarity. Cyclosa is characterized by

can elegantly indicate the possibility of redundancies while sustaining the optimal scheduling for high performance. Second, the dataflow-guided parallel execution engine decouples data access and computations to enable efficient results sharing. Third, the memory-friendly data management substrate can automatically manage the computation results with high reuse possibility. Evaluation of different

three features. First, it reorganizes the set operations for a pattern into a set dataflow representation which

patterns demonstrates that Cyclosa outperforms state-of-the-art pattern-centric systems GraphPi and SumPA by up to 16.28× and 5.52×, respectively. **Problem Statement and Research Objectives** Topic: Graph pattern mining systems

order of pattern vertices.

- large number of intermediate partial instances. Recently, advanced graph pattern mining systems have adopted a pattern-centric paradigm to overcome inefficiencies.
 - will not lead to a correct final match. This is achieved by transforming the graph patterns into a series of set operations and executing them in a nested loop following a matching
 - Pattern Graph **Data Graph** Mappings (u₂) Matching Order: (0) 3 $[u_0, u_1, u_2, u_3]$ Constraints: $\{u_1>u_0, u_3>u_2\}$ **Nested Loop Execution** Set Formulas

 $v_3 \in N(v_0) \cap N(v_1) \\ v_4 \in N(v_0) \cap N(v_2)$ Implicit Set Computations Data Graph (3) Goal: Eliminating Redundancies Counts Solved **Original:** 5 $N(1) \cap N(3)$ **Existing:** 4 $N(1) \cap N(3)$ (a) Exp. in One (b) Imp. in One (c) Both Cross Ours: 1 $N(1) \cap N(3)$ Figure 2: An example of redundant computations in graph pattern mining. The black vertices in dashed cycles represent redundant computations $N(1) \cap N(3)$ in different situations. ■ The explicit redundancies : one set intersection can be repeatedly used for computing different pattern vertices connected to the same subgraph instances. ■ The implicit redundancies: the same intersection appears in computing on different

subgraph instances.

View from Set Formulas

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

computations.

It removes explicit and implicit redundancies as follows:

Set Dataflow Analysis Module

Constraints

 $u_3 > u_2$

 $u_1 > u_0$

Constraints

 $u_2 > u_1$

transferred and computed.

are fully shared.

System Overview

Patterns

 $S_1: v_0 \in V$ $S_2: v_1 \in N(v_0)$

 $S_4: v_3 \in N(v_0) \cap N(v_1)$ $\cap +N(v_2)$ $S_5: v_4 \in N(v_0) \cap N(v_2)$ Formula-level Operands-level $S_3 = S_4$ R Computation Similarity Structural Equality

computation. (can only be analyzed after execution.)

Figure 4: The dataflow view for analyzing the mining procedure of p_c in Figure 2. • S_3 and S_4 are reduced. • Results of $N(v_0)$ are reused in two \cap . The static similarity: originates from the operands level of the set operations for a

• We observe two kinds of **computation similarity** in the set operations providing the opportunity to help identify and reuse both explicit and implicit redundancies.

View from Set Dataflow

V

 $N(v_0)$

pattern, exposing the reuse possibility of both inputs and outputs of different

computations, reflecting which vertices are more likely to be requested for

→ In this work, we propose a **set dataflow** to use the computation similarity for redundancy elimination,

as shown in Figure 4. The set dataflow is a directed graph indicating the procedure of how sets are

■ The dynamic similarity: lies in the runtime characteristics of the inputs of occurred

Proposed Method The set dataflow decouples the set formulas into individual operands and operators, and the directed edges represent the transfer relation of the input/output data between different operators.

1. The explicit redundancies can be removed by cutting and maintaining unique operators, e.g., only single $N(v_0)$ and $N(v_1)$ exist. Original two $N(v_0) \cap N(v_1)$ are thus reduced to one, and the set operands

2. The implicit redundancies between operators are indicated by overlapped inputs, e.g., the results of

Dataflow Execution Engine

Dataflow

Schedule

Results

Managei

Data Management Substrate

(c) Combining and Simplifying Set Dataflow

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

☐ Cyclosa

MiCo

Graph Retriver

Reordered Graph

Set Processor

Set Buffer

 $N(v_0) \cap N(v_1)$ and $N(v_0) \cap N(v_2)$. Based on the dynamic similarity, we can heuristically cache the computation results of high-degree vertices for reusing.

Set Dataflow

Figure 6: Overview of Cyclosa Set Dataflow Analysis Module

The set dataflow of input patterns is constructed in this module. Each pattern is first analyzed to generate a reuse-aware matching order and constraints with data graph properties. Then, based on the matching order, a set operation analyzer generates a redundancy-reduced set dataflow by

 $v_0 \in V$

(a) Core Components of Set Dataflow

 $v_1 \in N(v_0)$

Data

(b) Generating Sub-dataflows

Dataflow Construction

Cost Estimation

Loop2

Loop3 $+\alpha |V|^*deg^*ntri$

(u₁

Loop0

|V| + |V|*deg

ntri: average number of triangles per edge

Unit

Check

 $v_2 > v_0$

-1 > for the neighboring set of vertex 3.

 $ID \rightarrow req < 3, 5, op, gid >$

hit/miss

(a) The Workflow of Set Dataflow Execution Engine

6

related neighboring sets. → nei < 3, -1, op, gid*, value >

called to generate a new output. → res < 3, 5, gid, value >

2 6 7 9

1 2 6 7 8

<3, 5, op, gid>

<3, 5, op, gid>

3 5 1 2 6 7

Generator Module

Combiner Module

 $[u_2, u_1, u_3, u_0, u_4]$ $[u_{3}, u_{0}, u_{1}, u_{2}, u_{4}]$ **Load Estimation**

Orders

[U₀,U₁,U₂,U₃]

 $[u_0, u_2, u_1, u_3]$

 $[u_2,u_1,u_0,u_3]$ $[u_2, u_3, u_0, u_1]$

Orders

[U₀,U₁,U₂,U₃,U₄] ✓ $[U_0, U_3, U_1, U_2, U_4]$

keeping unique operands.

(u₃)

Order Enumeration

Figure 8: An example of constructing the set dataflow given Figure 7: The procedure of finding an appropriate reuse-aware four set formulas. The sub-dataflow of each set formula is first constructed with three operators, and the input/output sets matching order for a pattern by **1** generating constraints, **2** enumerating valid DFS orders by degrees, @ estimating cost of each operator are uniquely assigned and identified. The for each order with graph information, and 4 selecting the sub-dataflows are then combined into a final set dataflow by • reducing inputs/outputs and • simplifying set operators. The Generator consumes a valid candidate set of a pattern vertex to generate the neighboring sets. • The Combiner receives two sets and outputs a single set. • The Reducer checks a result set and selects valid candidates following filtering rules to produce a new candidate set for certain pattern vertices. Dataflow Execution Engine: Figure 10(a) nei<3, -1, op, gid*, value nei<..> res<.. 1 2 6 7 9 3 req<..> ret<..> <3, 5, op, gid, value> Activator

> Flow Map op0

> > op1

op2

Each set contains two parts for identification: the set ID and the elements value, e.g., ID < 3,

It traverses each vertex element from the input candidate set or the initial set to generate

The check unit first queries whether there are already computed results with the same

If the request hits, then the computation is omitted. Otherwise, the compute unit is

The output of this module is a set with valid candidates for a pattern vertex. → ret < op, gid,

- Dataflow Monitor Data Management Substrate: Figure 10(b) req ret Updater (res <3,5> Degree Reordering 0 1 1 1 1 1 Maintainer hit updat 2 3 Value **CSR Data Layout** 2 6 7 9 1 2 6 7 Edge _____ 6 7 7 9 Results Buffer Data Graph Candidates Buffer (b) The Workflow of Data Management Substrate
 - Sensitivity on Cache Strategies Sensitivity on Cache Capacity ■ WikiVote ■ MiCo □ Patents Speedup 3.5 3
- Normalized Speedup 3 2 5 2 5 0 0 0 0 Normalized 15% 20% 25% LFU LRU MRU Hybrid Null 10% Results Caching Strategies Figure 15: Normalized speedups under various settings of the cache capacity and caching strategy for 4-MC 4-CF on PA 200 Cyclosa Cyclosa

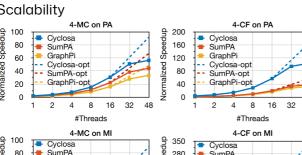
Figure 16: The normalized speedups with various number of

Typical graph pattern mining applications include subgraph matching, clique finding, and motifs

counting. Despite the prevalence of graph pattern mining applications, they have high

computational complexity and usually need hours or even days to complete.

1E+01 PC-0.8 0 q3 WikiVote q4 q6 q1 q3 q4 q5



Reducer Module

80 SumPA 280 SumPA GraphP 60 210

threads for different applications Overhead: Memory Consumption / Time for Constructing Set Dataflow

Scalability Normalized Normalized Speedup Normalized 40 140 GraphPi-op 20 70 0

8 16

Sensitivity on Cache

Notes

Parallel For v_0 in V_2 $v_0 : C(u_0) = V$ For v_1 in $N(v_0) \& v_1 > v_0$: $v_1 : C(u_1) = N(v_0), v_1 > v_0$ For v_2 in $N(v_0) \cap N(v_1)$: $v_2 : C(u_2) = N(v_0) \cap N(v_1)$ For v_3 in $N(v_0) \cap N(v_1) \& v_3 > v_2$: $v_3: C(u_3) = N(v_0) \cap N(v_1), v_3 > v_2$ Output(v_0, v_1, v_2, v_3) Figure 1: The example of mining a diamond pattern in a pattern-centric system The redundant computations usually cost more than 80% of the runtime and severely degrade the performance. (1) Patterns and Set Operations (2) Redundancy Examples $v_1 \in N(v_0), v_1 > v_0$ Implicit $v_2\in N(v_0), v_2>v_1$ $v_3\in N(v_1)\cap N(v_2), v_3>v_0$

Merge & Rewrite p_b Cross Patterns $v_1 \in N(v_0)$ $v_2 \in N(v_0) \cap N(v_1), v_2 > v_1$

 $v_1\in N(v_0), v_1>v_0$ $v_2 \in N(v_0) \cap N(v_1)$ Explicit $v_3 \in N(v_0) \cap N(v_1), v_3 > v_2$

Searching Graphs

 A common approach is to enumerate all the subgraphs, usually under a certain depth, to check whether the subgraphs satisfy the pattern constraints, which is called the embedding-centric paradigm. • This approach is easy to develop and parallelize. o However, it results in high memory consumption and wasted computing resources due to a The main idea is to use the structure information of graph patterns to filter intermediates that

https://www.usenix.org/conference/atc23/presentation/gui **Abstract** Graph pattern mining is an essential task in many fields, which explores all the instances of userinterested patterns in a data graph. Pattern-centric mining systems transform the patterns into a series of

Figure 10: The workflows of set dataflow execution engine and data management substrate Results Maintainer Candidates Updater Data Graph Fetcher **Evaluation and Results** Performance Comparison: Single Pattern Query / Multi-Pattern Query (Seconds) 1E+02 Execution 12 Figure 13: Performance comparison on listing single patterns Figure 14: Performance on finding pseudo cliques