Fast Parallel Path Concatenation for Graph Extraction

Yingxia Shao, Kai Lei, Lei Chen, Zi Huang, Bin Cui, Zhongyi Liu, Yunhai Tong, and Jin Xu

Abstract—Heterogeneous graph is a popular data model to represent the real-world relations with abundant semantics. To analyze heterogeneous graphs, an important step is extracting homogeneous graphs from the heterogeneous graphs, called homogeneous graph extraction. In an extracted homogeneous graph, the relation is defined by a line pattern on the heterogeneous graph and the new attribute values of the relation are calculated by user-defined aggregate functions. The key challenges of the extraction problem are how to efficiently enumerate paths matched by the line pattern and aggregate values for each pair of vertices from the matched paths. To address above two challenges, we propose a parallel graph extraction framework, where we use vertex-centric model to enumerate paths and compute aggregate functions in parallel. The framework compiles the line pattern into a path concatenation plan, which determines the order of concatenating paths and generates the final paths in a divide-and-conquer manner. We introduce a cost model to estimate the cost of a plan and discuss three plan selection strategies, among which the best plan can enumerate paths in $O(\log(l))$ iterations, where $l$ is the length of a pattern. Furthermore, to improve the performance of evaluating aggregate functions, we classify the aggregate functions into three categories, i.e., distributive aggregation, algebraic aggregation and holistic aggregation. Since the distributive and algebraic aggregations can be computed from the partial paths, we speed up the aggregation by computing partial aggregate values during the path enumeration.

Keywords—Heterogeneous Graph, Graph Extraction, Path Concatenation, Parallelism

1 INTRODUCTION

Things are connected in the real world. To model the complex connections between different objects, like persons, groups and locations, graph is a natural and friendly option. Given the fact that objects and their relations have diverse semantics in reality, the concept of heterogeneous graph has been proposed to clearly represent the real-world multi-typed relations. Figure 1 shows a heterogeneous scholarly graph which depicts three types of relations among three types of entities (i.e., authors, papers, and venues) in a research community. There are authorBy (dashed-arrow) relations between the Author vertices and Paper vertices; citeBy (dotted-arrow) relations are between Paper vertices; and publishAt (solid-arrow) relations connect the Paper and Venue vertices. Analyzing heterogeneous graphs with these abundant semantics can lead to interpretable results and meaningful conclusions [23].

Fig. 1. A tiny scholarly graph. There are three types of vertices, Author, Paper and Venue, and three types of relations, publishAt, authorBy and citeBy.

Graph extraction is a basic preprocessing step for analyzing the heterogeneous graphs, and it produces a subgraph which only contains single-typed edges and is called edge homogeneous graph. First, the extraction eliminates the gap between heterogeneous graphs and classic graph analysis solutions. Most of previous graph-based algorithms, such as simrank [26], [20], community detection [21], [19], centrality computation [14], etc., focus on such homogeneous graphs. Directly executing them on heterogeneous graphs by simply ignoring the semantics of vertices and edges, the results will lose their values. Second, the extraction can facilitate the new heterogeneous graph analysis approaches to process large graphs as well. A lot of proposed heterogeneous graph analysis approaches [23] select proper features (e.g., linear combination of different relations [2], meta-path [24], [10]) by extracting new relations from the original heterogeneous graphs.

Next we describe several preliminary concepts of the
edge homogeneous graph extraction through an example. Figure 2 lists two possible edge homogeneous graphs extracted from the scholarly graph in Figure 1. Figure 2(a) is a co-author graph, in which the co-author relation is defined as that "two Author vertices have authorBy relations to a same Paper vertex on the scholarly graph". The pattern (right-above figure in Figure 2(a)) to define the new relation is called line pattern. To find the new relations on the scholarly graph, we need to enumerate paths satisfying the line pattern between vertices. This process is called path enumeration. The attribute values of new relation are computed through user-defined aggregate functions. In the co-author graph example, the aggregate functions count the number of paths between a pair of vertices. The value of edge \((a_3, a_4)\) is two, which means there are two different paths satisfying the line pattern. Given that the aggregation is computed between every pair of vertices, we name it as pair-wise aggregation in the graph extraction context. The formal definition of the homogeneous graph extraction problem will be presented in Section 2.

However, existing techniques cannot efficiently process the extraction problem on large heterogeneous graphs. Regular path query (RPQ) [15] is a good option for the path enumeration. But the RPQ doesn’t consider the specific conditions of the edge homogeneous graph extraction problem, it requires the linear number of iterations with respect to the length of line pattern and incurs a large amount of intermediate results. Another possible solution is to use graph databases. But the graph databases are only optimized for querying local graphs, and cannot handle the extraction problem efficiently neither. The other challenge is the computation of pair-wise aggregation. The original evaluation of RPQ doesn’t involve aggregations. Considering there are massive paths during the extraction, the aggregation will be expensive if we simply compute it path by path.

We introduce a parallel graph extraction framework with high performance. Based on the idea that long paths can be generated by concatenating short paths, the input line pattern is compiled into a path concatenation plan. The plan consists of primitive patterns, which are line patterns of length two, and determines the order of concatenating paths. Since the primitive pattern has length two, the two end vertices are both the neighbors of the middle vertex. In accordance with such property, we use the vertex-centric model [12] to iteratively evaluate the path concatenation plan in parallel. After the paths are generated, the pair-wise aggregation can also be computed in the vertex-centric model naturally.

In general, a line pattern corresponds to several path concatenation plans (Figure 5), and different plans lead to different performance. According to the cost model (Section 3.3), both the number of iterations and the number of intermediate paths affect the overall performance of the framework. We design three plan selection strategies. They are iteration optimized strategy, path optimized strategy and hybrid strategy. The best strategy is the hybrid one which generates the minimized number of intermediate paths among all the plans with \(O(\log(l))\) iterations, where \(l\) is the length of a line pattern.

To improve the performance of computing pair-wise aggregation, we analyze the properties of aggregate functions and classify them into distributive aggregation, algebraic aggregation and holistic aggregation. The distributive and algebraic aggregations can be solved in a divide-and-conquer fashion which helps reduce the number of intermediate paths by merging the paths before they are completely enumerated, while holistic aggregation can only be computed in a path-by-path manner and sophisticated techniques are required to achieve high performance.

Finally, to demonstrate the advantages of our parallel homogeneous graph extraction approach, we implemented the prototype on an open-sourced graph computation system, and conducted comprehensive experiments on two real large datasets.

We summarize our contributions as follows.

- We formally proposed the homogeneous graph extraction problem which extracts an edge homogeneous graph based on user-defined relation and aggregate functions.
- We developed an efficient framework to extract homogeneous graph through concatenating paths in parallel.
- We introduced a cost model to estimate the cost of a path concatenation plan and designed three plan selection strategies.
- We categorized the pair-wise aggregations into distributive aggregations, algebraic aggregations and holistic aggregations according to their difficulty. For the distributive and algebraic aggregations, we also introduced the general optimization solutions.

In the remains of the paper, we define the homogeneous graph extraction problem in Section 2. The parallel graph extraction framework is presented in Section 3. The aggregation and plan selection techniques are elaborated in Section 4 and Section 5 respectively. Experiment
results are discussed in Section 6. We review the related work and conclude this paper in the last two sections.

2 GRAPH EXTRACTION PROBLEM

2.1 Preliminaries

A heterogeneous graph is an abstraction of multi-typed relations among various objects in the real world. Its formal definition is given as follows.

Definition 1: Heterogeneous graph. A heterogeneous graph $G_{he} = (V, E, L_v, L_e, A_v, A_e)$ is a directed, labeled and attributed graph. $L_v$ is the label set of vertices $V$ and $L_e$ is the label set of edges $E$. There are vertex-mapping function $f : V \rightarrow L_v$ and edge-mapping function $g : E \rightarrow L_e$, by which each vertex and edge are mapped to a single label individually, i.e., $f(v) \in L_v$ and $g(e) \in L_e$, where $v \in V$ and $e \in E$. $A_v$ and $A_e$ are the attribute sets of vertices and edges and can be empty.

Homogeneous graph is a special case of heterogeneous graph where $|L_v| = 1$ and $|L_e| = 1$. In this paper, we extend the definition of homogeneous graph and introduce edge homogeneous graph which only has $|L_e| = 1$. Since a relation only connects two vertices, so $|L_v|$ is at most two in an edge homogeneous graph. When context is clear, instead of edge homogeneous graph, we still use homogeneous graph for simplicity.

2.2 Definition of Homogeneous Graph Extraction

Graph extraction is a process of generating graph data which is used for in-depth analysis. In this paper, we focus on extracting edge homogeneous graphs from a heterogeneous graph. The edges of homogeneous graph are created based on a user input pattern on the heterogeneous graph and the attribute values of new edges are computed by user-defined aggregate functions.

Following the intuition that two objects are correlated if there exists a sequence of relations between them, the user input pattern, which defines the new relation, is a line pattern and its formal definition is given as below.

Definition 2: Line pattern. A line pattern $G_p = (V, E, L_v, L_e)$ is a directed labeled connected graph. The graph has exactly two vertices of degree one, which are named start vertex and end vertex respectively, and the other vertices have degree two. $L_v$ and $L_e$ are vertex label set and edge label set respectively. The vertex-mapping function and edge-mapping function are the same as the functions in Definition 1.

The subgraph instance which satisfies the line pattern is called a path in the heterogeneous graph. Because the path is purely determined by the line pattern, it is possible that a vertex in the path has two incoming or outgoing neighbors. Without explicit statement, the “path” in this paper is the one matched by the line pattern.

Next, on basis of line pattern, we give the definition of homogeneous graph extraction problem.

Definition 3: Homogeneous graph extraction problem. Given a $G_{he} = (V, E, L_v, L_e, A_v, A_e)$, a $G_p = (V_p, E_p, L_{vp}, L_{pe})$ and user-defined aggregate functions $\otimes$ and $\oplus$ (refer to Definition 4), generate an edge homogeneous graph $G = (V', E', A'_e)$. Vertex set $V'$ is the union of vertices which match the start vertex $v_s$ and end vertex $v_e$ of $G_p$, i.e., $V' = \{v | f(v) = f_p(v_s)\} \cup \{v | f(v) = f_p(v_e)\}$. Each edge $e = (u, v) \in E'$ indicates there exist at least one path matched by $G_p$ between vertices $u$ and $v$. The attribute $a \in A'_e$ of an edge $e = (u, v)$ is computed from all the paths between vertices $u$ and $v$ by the user-defined aggregate functions $\otimes$ and $\oplus$.

2.3 The Characteristics of Graph Extraction

In this part, we first introduce the two steps for solving homogeneous graph extraction problem. One is path enumeration and the other is pair-wise aggregation. Then we present the hardness of graph extraction problem.

2.3.1 Path Enumeration and Pair-wise Aggregation

Path enumeration is to find out all the paths matched by the line pattern. In graph theory, a path containing no repeated vertices is a simple path, otherwise it is a non-simple path. We will prove that the extraction problem becomes intractable if we target on simple paths in Section 2.3.2.

Pair-wise aggregation computes the attribute values of edges in the extracted homogeneous graph. Each value is aggregated from two levels. First, the value of a path is aggregated from the path’s edges, denoted by $\otimes$; second, the final attribute values are aggregated from the paths’ values, denoted by $\oplus$. Both $\otimes$ and $\oplus$ are binary operators. The operands of $\otimes$ and $\oplus$ are the values of edges and paths respectively. The formal definition of the two-level aggregate model is presented as below.

Definition 4: Two-level aggregate model. Given a heterogeneous graph $G_{he}$, a line pattern $G_p$ and user-defined aggregate functions $\otimes$ and $\oplus$, for each vertex pair $u$ and $v$, $P_{uv}$ denotes the set of paths which satisfy the line pattern $G_p$ between the two vertices $u$ and $v$, the final attribute value of edge $(u, v)$ in the extracted homogeneous graph is computed as following two steps.

1) the value $\text{val}(p)$ of a path $p \in P_{uv}$ is

$$\text{val}(p) = \otimes_{v \in e_i \in p} w(e_i),$$

where $w(e_i)$ is the attribute value of an edge $e_i$ of path $p$.

2) the final attribute value $\text{val}(u, v)$ of the new edge $(u, v)$ is

$$\text{val}(u, v) = \oplus_{p_i \in P_{uv}} \text{val}(p_i).$$
2.3.2 The hardness of Graph Extraction

Next, we show the hardness of homogeneous graph extraction problem. Theorem 1 shows that the problem becomes intractable if path enumeration requires simple paths.

**Theorem 1**: The hardness of graph extraction with enumerating simple path. Assume the homogeneous graph extraction problem enumerates simple paths, it is \#W[1]-complete[6].

**Proof**: Counting simple paths of length \(k\), parameterized by \(k\), on a homogeneous graph \(G\) is \#W[1]-complete, and it can be reduced to the homogeneous graph extraction problem. Given the length \(k\) and a graph \(G\), where each edge has weight 1, we can construct a line pattern of length \(k\) with \(|L_{pv}| = 1\) and \(|L_{pe}| = 1\). The graph \(G\) is a special instance of a heterogeneous graph where \(|L_v| = 1\) and \(|L_e| = 1\). Then we set the user-defined aggregate functions (i.e., \(\otimes\) is set to multiplication and \(\oplus\) is set to addition.) for the homogeneous graph extraction problem to count the number of paths between each pair of vertices. Finally, the global number of paths can be computed from the pair-wise path count in polynomial time. So far the counting simple path problem has been successfully reduced to the homogeneous graph extraction problem.

Considering that, in real applications with the help of diverse types of vertices and edges, there are many meaningful non-simple paths. For example, when we extract the relations of two authors publishing papers at the same venue, it is reasonable to consider authors of the same paper have such relation. Therefore, in this paper, we enumerate non-simple paths, and the problem is no longer \#W[1]-complete.

However, because the number of paths is exponential to the length of line pattern, the problem is still computation-intensive. But we find that, both the path enumeration and pair-wise aggregation are computed based on paths’ end vertices and all the intermediate paths are independent. This observation indicates that

1. \#W[1]-complete problem does not have fixed-parameter tractable solutions.

3 Parallel Graph Extraction Framework

As discussed previously, the path enumeration and pair-wise aggregation are well suited to the vertex-centric model. Consequently, we propose a parallel graph extraction framework by using popular graph computation systems [12], [7], [22], [16]. The basic assumptions for the framework design are as follows. The heterogeneous graph is partitioned and is stored in distributed memory. The intermediate states are transferred among computing nodes by message passing.

Figure 3 depicts the overview of execution flow in the framework. The left part of the figure shows the logic in the master node. 1) After users submit a line pattern and aggregate functions \(\otimes\) and \(\oplus\), the master compiles the input line pattern into a path concatenation plan (PCP), which defines the order of concatenating short paths to generate the final paths, and sends the plan to the computing nodes. This strategy follows the idea that a long path can be generated by concatenating short paths to reduce the number of iterations. 2) On computing nodes, the framework generates complete paths by iteratively evaluating the plan in a vertex-centric manner. 3) Finally, the framework creates the extracted homogeneous graph by computing the aggregate functions \(\otimes\) and \(\oplus\) following the two-level aggregate model. Since the final matched paths are stored at one of its end vertices as neighbors after the path enumeration, the aggregation can be done in a vertex-centric manner as well.

The performance of the framework is determined by path enumeration and pair-wise aggregation. From Figure 3, we notice that the number of intermediate paths and the number of iterations affect the performance. In the following subsections, we first present concepts of primitive pattern and PCP, then describe the basic
version of PCP evaluation algorithm followed by the cost analysis of the framework. In Section 4, we will introduce partial aggregation to improve the performance of pair-wise aggregation by reducing the number of intermediate paths. In Section 5, we discuss techniques to select a good PCP which reduces both the number of intermediate paths and the number of iterations.

3.1 Primitive Pattern and Path Concatenation Plan

The goal of path concatenation plan is to decide the order of short paths which are concatenated to produce long paths. In a plan, the minimal concatenation operation is that each vertex $v$ generates a new path by connecting one path which ends with the vertex $v$ to another path which starts with the vertex $v$. This logic of concatenation operation is similar to evaluate a line pattern of length two, where the middle vertex concatenates the results matched left side and the results matched right side. Thereby, the minimal concatenation operation can be expressed as a line pattern of length two, which is called primitive pattern. A path concatenation plan consists of a set of related primitive patterns.

Before giving the formal definition of primitive pattern, we explain the concept of extended vertex label set and extended edge label set. Given a heterogeneous graph $G_{he} = (V, E, L_v, L_e, A_v, A_e)$, the extended vertex label set denoted by $L_{ev} = L_v \cup \{id\}$ includes additional vertex labels $\{id\}$ which are the identifications of primitive patterns; and the extended edge label set denoted by $L_{ee} = L_e \cup \{\phi\}$ includes an empty edge label $\phi$ which means no edge matching is required between vertices. To distinguish the new labels $\{\{id\} \text{ and } \phi\}$ from the original labels $(L_v \text{ and } L_e)$, we name the new labels as query labels denoted by $QL$ and the original labels are native labels denoted by $NL$. The following definition describes the concept of primitive pattern.

Definition 5: **Primitive pattern**. The primitive pattern denoted by $P_p = (id, V_{pp}, E_{pp}, L_{ev}, L_{ee})$ is a line pattern of length two with extended vertex and edge label sets, and has a unique identification $id$. The vertex set $V_{pp}$ consists of three vertices, start vertex $v_{ss}$, pivot vertex $v_p$ and end vertex $v_{ee}$, i.e., $V_{pp} = \{v_s, v_p, v_e\}$. The edge set $E_{pp}$ includes a left edge $e_l = (v_s, v_p, \text{dir})$ and a right edge $e_r = (v_p, v_e, \text{dir})$, where dir is the direction (incoming, outgoing or undirected) of the edge, i.e., $E_{pp} = \{e_l, e_r\}$. The mapping functions are similar to the functions in Definition 2.

Fig. 4. Four types of primitive patterns on the scholarly graph. The leftmost figure shows that $v_s$ matches the results of primitive pattern with $id = 1$; $v_p$ matches the results of primitive pattern with $id = 2$; both matched results are concatenated by Venue vertex $v_p$. The other three figures have the similar explanations.

Fig. 5. Five different PCPs of a single line pattern which defines the relation of two authors publishing papers at the same venue.

With respect to the different combinations of vertex labels, there are four types of valid primitive patterns. They are NL-NL pattern, NL-QL pattern, QL-NL pattern and QL-QL pattern. An NL-QL pattern means $f(v_s)$ is a native label (i.e., $L_{ev}, L_r$) and $f(v_e)$ is a query label (i.e., $\{id\}, \phi$). Figure 4 shows four examples of different types of primitive patterns on the scholarly graph.

Now a general line pattern can be compiled into a path concatenation plan expressed by a set of primitive patterns. The formal definition of the plan is stated as below.

Definition 6: **Path Concatenation Plan (PCP)**. PCP is a collection of primitive patterns. The dependence between the primitive patterns forms a binary tree, where each node is a primitive pattern and the leaf nodes are NL-NL patterns. In the tree, a node $X$ is the parent of a node $Y$ only if $Y$’s primitive pattern $id$ is the vertex label in $X$’s primitive pattern.

The height of a PCP tree is denoted by $H$, and from root to leaf, the level number of each node is one to $H$. The node close to the root has small level number. Figure 5 lists five different PCPs of a single line pattern. PCP-1 consists of three primitive patterns and is a binary tree of height 2. The primitive pattern $id = 0$ is the root node with level number 1, and the other two primitive patterns are leaf nodes which are both NL-NL patterns. Except PCP-1, the other four PCPs all have height of three. The following theorem shows the lower bound of the height of a PCP for a line pattern.

**Theorem 2:** A line pattern of length $l$ generates a PCP with $l - 1$ nodes. Because a balanced full binary tree with $n$ nodes has height $\lceil \log_2(n + 1) \rceil$, the height of a PCP of a line pattern of length $l$ is at least $\lceil \log_2(l) \rceil$.

3.2 PCP Evaluation Algorithm

Intuitively, a PCP should be evaluated from leaf to root. Furthermore, primitive patterns in the same level can be evaluated simultaneously, since they are independent of each other. The core idea of evaluating a PCP is iteratively computing an array of primitive patterns in the
Algorithm 1 PCP Evaluation Pseudocode

Input: PCP plan, The height of PCP H, Aggregate functions ⊕ and ⊗, Heterogeneous graph \( G_{ho} \)
Output: The extracted homogeneous graph \( G_{ho} \)
1: foreach vertex \( v \) in \( G_{he} \) do
2: \( v \) preprocessing of materializing both out/in neighbors of \( v \) at local;
3: endforeach
4: foreach \( h \rightarrow V \) to 1 do
5: \( ppSet \) ← plan.getPPbyLevel(\( h \))
6: foreach vertex \( v \) in \( G_{he} \) do
7: foreach primitive pattern \( pp \) in \( ppSet \) do
8: evaluate \( pp \) on vertex \( v \) by Algorithms 2;
9: endforeach
10: endforeach
11: end foreach
12: foreach vertex \( v_s \) in \( G_{he} \) do
13: \( PS \) ← get paths end with \( v_s \) and index the paths by their start vertices \( v_s \);
14: foreach vertex \( v_s \) do
15: \( path_{v_s,v_k} \) ← get paths start with \( v_s \) from \( PS \);
16: \( aggVal \) ← \( \phi \);
17: foreach path \( pa \) in \( path_{v_s,v_k} \) do
18: \( pathVal \) ← \( \oplus val(e_i) \) \( e_i \) \( \in \) \( pa \);
19: \( aggVal \) ← \( \oplus val \oplus pathVal \);
20: endforeach
21: insert extracted relation \( (v_s, v_e, aggVal) \) into \( G_{ho} \);
22: endforeach
23: end foreach
24: return \( G_{ho} \);

Algorithm 2 Primitive Pattern Evaluation Pseudocode

Input: Vertex \( v \in G_{he} \), Primitive Pattern \( pp \)
1: initialize \( v_s, v_p, v_e, e_i, e_v \) by \( pp \);
2: if \( f(v) = f_p(v) \) then
3: \( leftRes \) ← \( \text{NULL} \); \( rightRes \) ← \( \text{NULL} \);
4: if \( f_p(v_s) \in NL \) then
5: \( leftRes \) ← matching \( e_i \) to the \( v \)'s neighbors in \( G_{he} \);
6: if \( f_p(v_s) \notin NL \) then
7: \( leftRes \) ← retrieving the results of primitive pattern \( id = f_p(v_s) \);
8: endif
9: if \( pp \) is the left child then
10: send the new paths to the end vertex;
11: else
12: \( rightRes \) ← retrieving the results of primitive pattern \( id = f_p(v_s) \);
13: endif
14: concatenate \( leftRes \) and \( rightRes \);
15: if \( pp \) is the left child then
16: send the new paths to the start vertex;
17: else
18: send the new paths to the start vertex;
19: endif
20: endif

same level, and the complete paths will not be generated until the computation reaches the root primitive pattern.

The pseudocode of the evaluation algorithm is illustrated in Algorithm 1. Given the fact that the length of a primitive pattern is two, both the start vertex and end vertex are the neighbors of pivot vertex. This property allows us to concatenate partial paths on pivot vertex by neighbor exploring instead of global searching. Therefore, to evaluate a primitive pattern, we use the vertex-centric model [12] which is easily parallelized. More specifically, in an iteration (Lines 4-11), each vertex in graph \( G_{he} \) generates paths for primitive patterns and sends the paths to the corresponding vertices by calling Algorithm 2. During the concatenation, the partial paths for matching start vertex and end vertex of a primitive pattern come from two different sources in accordance with the labels of vertices. The vertex with \( NL \) label directly matches the data of the heterogeneous graph, while the vertex with \( QL \) label matches the results of previous primitive pattern. For example, in Figure 3, iteration 1 processes NL-NL primitive patterns \( id=1 \) and \( id=2 \), which access the data of \( G_{he} \) (Lines 5 and 10 in Algorithm 2), and the iteration 2 processes QL-QL primitive pattern \( id=0 \), which uses the partial paths generated by primitive patterns \( id=1 \) and \( id=2 \) (Lines 7 and 12 in Algorithm 2).

However, when matching the vertex with \( NL \) label, because the edges in a primitive pattern have directions, the pivot vertex needs to explore both in and out neighbors. Considering graph computation systems only maintain out neighbors for each vertex, we bring in a preprocessing phase (Lines 1-3 in Algorithm 1), in which each vertex materializes its in and out neighbors at local, to truly achieve the neighbor exploration of pivot vertex. Finally, benefit from the vertex-centric model, all the complete paths generated by the PCP evaluation are stored at their end vertices respectively. Therefore, the pair-wise aggregation (Lines 12-23 in Algorithm 1) can be naturally executed in the vertex-centric model as well.

3.3 Cost Analysis

According to Algorithms 1 and 2, the total computation cost is mainly related to the following three parts. First, the cost of accessing each vertex in the heterogeneous graph (Line 6 in Algorithm 1); second, the cost of concatenating paths (Algorithm 2); third, the cost of pair-wise aggregation. Since the preprocessing can be done offline once, we do not count its cost into the total computation cost.

In each iteration, the algorithm scans over all the vertices in the heterogeneous graph once. Therefore, the number of iterations is proportion to the cost of enumerating vertices, i.e., the cost is \( cVH \), where \( c \) is a coefficient and \( V \) is the number of vertices.

The path enumeration cost denoted by \( S_{pcp} \) is simply the sum of the cost of primitive pattern evaluation, i.e.,

\[
S_{pcp} = \sum_{pp_i} S_{pp_i}
\]

where \( S_{pp_i} \) is the cost of evaluating a primitive pattern \( pp_i \). In Algorithm 2, assume the sizes of \( leftRes \) and
right Res are $d_{st}$ and $d_{vr}$ individually with respect to the vertex $v$, then $S_{pp}$, is

$$ S_{pp} = \sum_{v:t(v)=f_p(e_p)} d_{st} \ast d_{vr}. \quad (4) $$

From the above equations, we see that $S_{pp}$ is highly related to the number of intermediate paths. Moreover, the cost of pair-wise aggregation is proportion to the number of final paths as well.

In summary, the overall performance of parallel homogeneous graph extraction framework is highly related to two factors, the number of iterations and the number of intermediate paths. Therefore, we design optimization techniques by carefully considering the two factors.

4 Aggregation in Homogeneous Graph Extraction

The performance of pair-wise aggregation is proportion to the number of paths. To reduce the number of paths, we borrow the idea of classifying the aggregate functions in traditional OLAP [9], i.e., the pair-wise aggregations is divided into three types, including distributive aggregations, algebraic aggregations and holistic aggregations. We first present the criteria to distinguish three types of aggregations; then we introduce optimization techniques for the distributive and algebraic aggregations.

4.1 Distributive, Algebraic and Holistic Aggregation

Most of aggregation strategies can be represented by the two-level aggregate model (in Section 2.3). As illustrated in Algorithm 1, a universal solution for the pair-wise aggregation first exhaustively enumerates all the paths, and then applies the aggregate model on the paths. However, the number of paths is exponential to the length of line pattern, and hence it is inefficient to aggregate the values by naïve enumeration.

We classify the aggregations into three different types by analyzing the properties of aggregate functions $\otimes$ and $\oplus$. They are distributive aggregations, algebraic aggregations and holistic aggregations. The distributive aggregation can be solved in a divide-and-conquer fashion which means the final aggregation can be computed from several sub aggregations that are calculated based on partial paths. The algebraic aggregation can be solved by maintaining several distributive aggregations. A holistic aggregation needs to enumerate all the paths. In practice, the distributive aggregation is more attractive and is possible to be solved efficiently. The following theorem gives the condition to be a distributive aggregation.

Theorem 3: Distributive Aggregation Condition. If $\otimes$ is distributive over $\oplus$, then the two-level aggregation is the distributive aggregation. In other words, the equation below holds.

$$ val_i(u, v) = \bigoplus_{v \in V} (val_{i-1}(u, v) \otimes val_i(v, v)), \quad (5) $$

Algorithm 3 Primitive Pattern Evaluation With Partial Aggregation

**Input:** Vertex $v \in G_{pp}$, Primitive Pattern $pp$

1: Lines 1-13 in Algorithm 2.
2: /*Because of partial aggregation, each element in left Res and right Res only has three fields, matched start vertex $v'$ and partial aggregate value $val_i* /
3: leftAggMap $\leftarrow \phi$, rightAggMap $\leftarrow \phi$.
4: foreach $p_i = (v_i, u_i, v_i')$ in leftRes do
5: leftAggMap$[v_i'] \leftarrow$ leftAggMap$[v_i']$ $\oplus$ val_{i-1}$
6: endforeach
7: foreach $p_i = (v_i', u_i, v_i)$ in rightRes do
8: rightAggMap$[v_i'] \leftarrow$ rightAggMap$[v_i']$ $\oplus$ val_{i-1}$
9: endforeach
10: foreach $val_i$ in leftAggMap do
11: foreach $v_i'$ in rightAggMap do
12: $nVal \leftarrow$ leftAggMap$[v_i']$ $\otimes$ rightAggMap$[v_i']$.
13: new partial path $nPath \equiv (v_i', nVal, v_i')$.
14: Lines 15-19 in Algorithm 2.
15: end foreach
16: end foreach

where $val_i(u,v)$ is the aggregate value from paths of length $l$ between vertices $u$ and $v$, $t$ $<$ $l$ and $v_i$ is the $i^{th}$ vertex in a path of length $l$.

Proof: Equation 5 can be deduced as follows by the property that $\otimes$ is distributive over $\oplus$.

$$ val_i(u, v) = \bigoplus_{v \in V} (\bigotimes_{p_i \in P_{uv}} e_{ij} \bigoplus_{p_i \in P_{uv}} w(e_{ij})) : $ $Definition 4.

$$ = \bigoplus_{v \in V} (\bigotimes_{p_i \in P_{uv}} e_{ij} \bigoplus_{p_i \in P_{uv}} w(e_{ij})) : $ $Definition 4.

$$ = \bigoplus_{v \in V} (\bigotimes_{p_i \in P_{uv}} e_{ij} \bigoplus_{p_i \in P_{uv}} w(e_{ij})) : $ $Definition 4.

$$ = \bigoplus_{v \in V} (\bigotimes_{p_i \in P_{uv}} e_{ij} \bigoplus_{p_i \in P_{uv}} w(e_{ij})) : $ $Definition 4.

$$ \bigotimes_{p_i \in P_{uv}} e_{ij} \bigoplus_{p_i \in P_{uv}} w(e_{ij})) : $ $Definition 4.

$$ = \bigoplus_{v \in V} (w_i(u, v_i) \otimes w_{l-t}(v_l, v)). \quad (6) $$

According to Theorem 3, a lot of common aggregate functions are distributive aggregations. Name a few, multiplication-addition ($\otimes$+$\oplus$), min-max, max-min, addition-max, sum-min etc. The computation of these aggregate functions can be optimized by aggregating during the path enumeration.

Since algebraic aggregation is a combination of distributive aggregation, it can be benefit from Theorem 3 as well. The holistic aggregations are still computation and space expensive. In this work, we focus on optimizing the computation of distributive and algebraic aggregations, because the majority of analysis scenarios [10, 24, 3, 27, 25] handle these two kinds of aggregations. Furthermore, the optimization of holistic aggregation requires special designs according to its concrete aggregate functions [13].
4.2 Optimization with Partial Aggregation

Theorem 3 points out distributive aggregations can be aggregated ahead with partial paths. In other words, distributive aggregations can be computed through partial aggregation. The partial aggregation merges intermediate paths with the same start and end vertices by the aggregate functions before the complete paths are found. As a result, the number of intermediate paths is reduced and the overall performance is improved.

The algorithm with partial aggregation is illustrated in Algorithm 3. Before generating new paths, we first use $\oplus$ to merge the paths with the same start and end vertices from different workers (Lines 3-9 in Algorithm 3). In Lines 10-15, when creating the new paths, we also use $\odot$ to compute the partial aggregated values for the new partial paths. With above optimizations, the number of intermediate paths in each iteration is for the new partial paths. With above optimizations, we also use $\otimes$ in Algorithm 3. Before generating new paths, we first aggregated ahead with partial paths. In other words, distribution of intermediate paths and edges are uniformly distributed over the vertices of heterogeneous graph, we can extend Equation 4 into the following model to estimate the cost of a $pp_i$ in a PCP,

$$S_{pp_i} = \begin{cases} \frac{V_p \times S_{pp_i}}{L \times S_{pp_i}} & QL - QL pattern \\ \frac{V_p \times S_{pp_i}}{S_{pp_i} \times R} & NL - QL pattern \\ \frac{V_p \times S_{pp_i}}{L \times R} & NL - NL pattern \end{cases}$$

where $V_p$ is the vertex set in which each vertex matches the pivot vertex of $pp_i$. According to Equation 3, the estimated cost of a PCP ($S_{pp}$) can be computed by summing up Equation 7 of different $pp_i$s.

Moreover, a sophisticated distribution assumption (e.g., power law or normal distributions) can be used to increase the accuracy of the estimation. However, similar to the work [17], it requires other techniques to estimate the parameters of the distribution, which is not the focus of our work. Through the experiments, we can observe that the uniform distribution assumption is fair enough to help us select a good plan.

5.2 PCP Selection

In this subsection, we elaborate three strategies of PCP selection. The first strategy is iteration optimized strategy which selects a PCP with the minimized number of iterations. The second strategy is path optimized strategy which selects a PCP with minimized size of intermediate paths. The third one is a hybrid strategy which selects a PCP with minimized size of intermediate paths from PCPs with minimized number of iterations.

5.2.1 Iteration Optimized Strategy

Theorem 2 shows that the PCPs of a line pattern of length $l$ have heights at least $\lceil \log(l) \rceil$. This lower bound is achieved when PCP is a balanced full binary tree. Thereby, we define the iteration optimized strategy as below.

Definition 7: Iteration Optimized Strategy. Given a line pattern $P$ of length $l$, the goal of this strategy is to select out a PCP which has $\lceil \log(l) \rceil$ height.

The main procedure for generating such a PCP is that we use divide-and-merge framework to recursively divide the line pattern into two sub line patterns with the same length until each sub line pattern has length less than three; then merge these sub line patterns to form a PCP. During the division phase, there might be multiple vertices which divide the line pattern into two sub line patterns with the same length. In such cases, we randomly pick one. The procedure is finished in $O(l)$ time. This strategy only reduces the number of iterations without considering the size of intermediate paths.

5.2.2 Path Optimized Strategy

The goal of path optimized strategy is to find out the best PCP which generates the minimized number of intermediate results. The optimization problem is presented as below.
Definition 8: Path Optimized Strategy. Given a line pattern $P$ and a heterogeneous graph $G_{he}$, the goal of the problem is to find a PCP satisfying that

$$\min \{ S_{pcp} \}.$$ 

Each PCP forms a binary tree (Definition 6), different PCPs may share the same subtrees. For example, in PCP-3 and PCP-4 shown by Figure 5, the left vertices in their primitive pattern $id = 0$ are corresponding to the results from the same sub pattern, which consists of the first four vertices of the original line pattern. This observation leads to the optimal substructure for the path optimized strategy, which is formalized by the following equations. Given a line pattern of length $l$, the vertices are sequentially numbered from left to right, then

$$S_{pcp}[i, j] = \begin{cases} \min_{0 \leq k < j} \{ S_{pcp}[i, k] + S_{pcp}[k, j] + S_{pp_k} \} & j - i > 2 \\ 0 & j - i \leq 2 \end{cases}$$

where $i, j, k$ are the ids of vertices in the line pattern and $S_{pcp}[i, j]$ is the cost of a sub pattern of line pattern between vertices $i$ and $j$. In addition, $k$ implies that the vertex $k$ to be a pivot vertex in $pp_k$. Consequently, the best PCP has the cost $S_{pcp}[1, l]$. The above equations can be solved in $O(l^3)$ by the dynamic programming technique. This strategy only optimizes the size of intermediate results without considering the number of iterations.

5.2.3 Hybrid Strategy

As mentioned at the beginning, the number of iterations and the size of intermediate paths do not have positive correlation. In practice, the number of iterations has more influence than the size of intermediate paths on the overall performance. This is because the distribution of intermediate paths is hard to be predicted, and they cannot be processed by the workers in balance. Therefore, increasing the number of intermediate paths to some extend may improve the utility of workers, and the overall performance will not degrade significantly. In contrast, by applying the hash partition schema, the vertices of heterogeneous graph can be evenly distributed over the workers, increasing the number of iterations certainly degrades the overall performance.

Based on above observations, the PCP selection strategy should give priority to reduce the number of iterations. More specifically, a PCP should have the number of iterations as small as possible before reducing the size of intermediate paths. So we design a hybrid strategy by using both iteration optimized strategy and path optimized strategy. We use the iteration optimized strategy to construct a balanced binary tree, but compared to the previous random selection of pivot vertices, we choose the pivot vertices according to the cost model with dynamic programming technique. The new dynamic programming equations are presented as below.

$$S_{pcp}[i, j] = \begin{cases} \min_{k \in \{\frac{j-s}{l}, \frac{j-2+s}{l+1}\}} \{ S_{pcp}[i, k] + S_{pcp}[k, j] + S_{pp_k} \} & j - i > 2 \\ 0 & j - i \leq 2 \end{cases}$$

Compared to the path optimized strategy, the above equations only enumerate pivot vertices ($k$) around the middle index in a line pattern to guarantee the final PCP has the minimized number of iterations.

6 EXPERIMENTAL STUDY

In this section, we first demonstrate the effectiveness of plan selection algorithm and show the benefit of partial aggregation techniques. Then we compare our solution with three different approaches, they are based on graph database, regular path query (RPQ) and matrix model respectively. Finally we show the scalability of our solution with respect to the number of workers, the size of datasets and the length of line pattern.

6.1 Experiment Settings

The prototype of parallel graph extraction framework was implemented on top of Giraph\textsuperscript{2}. The framework reads a line pattern from HDFS and the line pattern is represented by a line graph. The user-defined aggregate functions are two new abstract methods (i.e., $\otimes$ and $\oplus$) in the vertex program of primitive pattern evaluation, which uses Algorithm 2 for the basic extraction solution, and Algorithm 3 for the extraction solution with partial aggregation. Besides, the framework invokes a preprocessing phase as well. The preprocessing aims to materialize in and out neighbors of each vertex with the corresponding labels. This can be finished in three iterations and only conducted once.

The experiments were conducted on a cluster with 22 physical nodes, who have 48G memory and a 2.6Ghz CPU individually. We use two datasets, dblp-2014\textsuperscript{3} and us-patent\textsuperscript{4}, for the experiments. The schemas of dblp-2014 and us-patent are shown in Figure 6(a) and Figure 7(a).

Line patterns. The design of line pattern requires domain knowledges, because different analysis applications have different demands on the extracted graph and aggregate functions. However, according to many existing applications and literatures [24], [10], the line pattern can be still classified into two categories.

- Bipartite line pattern (BP for short): the pattern defines new relations between the same type of vertices.
- Symmetry line pattern (SP for short): the pattern defines new relations between the same type of vertices.

3. http://dblp.uni-trier.de/xml/
On dblp-2014 dataset, we define three symmetry patterns and one bipartite pattern. They are explained as follows.

- dblp-BP1 (Figure 6(b)): the pattern extracts publish relation between authors and venues.
- dblp-SP1 (Figure 6(c)): the pattern extracts co-authorship among authors.
- dblp-SP2 (Figure 6(d)): the pattern extracts the relation of authors who publish papers on the same venue.
- dblp-SP3 (Figure 6(e)): the pattern extracts the relation of venues where papers of the same author are published.

On us-patent dataset, we define three symmetry patterns and two bipartite patterns on us-patent. The details are listed as below.

- patent-BP1 (Figure 7(e)): a bipartite pattern extracts relation between locations and categories of the patents.
- patent-BP2 (Figure 7(f)): a bipartite pattern extracts the two-hop citation relation between the inventor and patents.
- patent-SP1 (Figure 7(b)): the pattern extracts co-inventor relation among inventors.
- patent-SP2 (Figure 7(c)): a symmetry pattern extracts the citation relation among different locations.
- patent-SP3 (Figure 7(d)): a symmetry pattern extracts the citation relation among inventors.

According to the size of results of each pattern, we also divide above patterns into light patterns and heavy patterns which are listed in Table 1.

**Aggregate Functions.** Since different instances of distributive and algebraic aggregations have the similar computation patterns, in the experiments, we use path counting as a representative.

**Baselines.** We compared our prototype with three different kinds of approaches. First one is graph database-based approach. We use graph database Neo4j\(^5\) to store the heterogeneous graphs, and ran the database on a Linux server with 96GB memory. Since graph database provides the interface of querying paths from a certain vertex. By using the graph database to answer a query, we first retrieve vertices matched by the start vertex of the input pattern; then we query the paths and aggregate them for each retrieved vertex.

Second one is matrix-based solution which is introduced by Rodriguez [18]. The path enumeration and pair-wise aggregation processing can be converted into matrix multiplications. A heterogeneous network is transformed into a set of matrixes and vertex mappings. To answer a query, we first retrieve related matrixes and do the matrix multiplications, then transform the results into a subgraph in the original heterogeneous network by using the vertex mappings. This solution is implemented based on the sparse matrix multiplication in SciPy\(^6\).

Third one is RPQ-based solution which is implemented following the idea in work [15]. Each experiment is ran five times, and the average performance and error-bar are reported.

**6.2 Effectiveness of Partial Aggregation Technique**

We first demonstrate the effectiveness of partial aggregation technique through comparing the basic and optimized graph extraction solutions. The basic solution first enumerates all the paths that matched by the line pattern and then computes the aggregate functions. The optimized solution aggregates partial paths during the path enumeration. Both solutions use hybrid strategy to select a PCP. As representatives, we show the results of running dblp-SP3 and dblp-BP1 on dblp-2014 dataset and patent-SP3 and patent-BP2 on us-patent dataset with ten workers.

Figure 8 shows the runtime and the number of intermediate paths for each pattern. As discussed in Section 3.3, the larger number of intermediate paths leads to poorer overall performance. Since the optimized solution reduces the number of intermediate paths with

---

6.3 Comparison of Different Plans

According to the cost analysis in Section 3.3, the performance of homogeneous graph extraction solution is related to the number of intermediate paths and the number of iterations. We evaluate four types of plan selection strategies, line strategy (line), iteration optimized strategy (iterOPT), path optimized strategy (pathOPT) and hybrid strategy (hybrid). The line strategy indicates that we enumerate paths by expanding the pattern from one end vertex to the other end vertex in sequence. The other three strategies have been introduced in Section 5. In addition, the pathOPT and hybrid strategies select PCPs according to the estimation model proposed in Section 5.1. The following experiments are conducted by using the partial aggregation technique with ten workers.

Figure 9 presents the performance of three patterns (patent-SP2, patent-BP1 and dblp-SP3) on us-patent and dblp-2014 datasets as representatives. Other patterns have similar results to the one of dblp-SP3. It is clear to see from Figure 9(a) that the hybrid strategy has the best performance across different patterns and datasets. This is because the hybrid strategy synthetically considers the number of iterations and the size of intermediate paths when selecting a plan. For example, when running patent-SP2, the hybrid strategy and pathOPT strategy select the same PCP, and they have similar performance. The iterOPT strategy has the same number of iterations (Figure 9(b)), but has larger number of intermediate paths (Figure 9(c)), so it performs a bit worse compared to the hybrid and pathOPT strategies. When running patent-BP1, though line and pathOPT strategies lead to small size of intermediate paths (Figure 9(b)), they lose high performance because of the large number of iterations (Figure 9(c)). In addition, for other patterns used in experiments, different plan selection strategies choose the same PCP, so the performance is similar. The result of dblp-SP3 is visualized as a representative.

From all the experiments, the line strategy always has the worst performance. This indicates that, without optimizing the number of iterations and the size of intermediate paths, the solution cannot be efficient. In general, the hybrid strategy is the best option to efficiently execute the extraction framework. Moreover, in Figure 9(b), we noticed that pathOPT and hybrid usually generates PCPs with the size of intermediate paths smaller compared to other approaches, which confirms the effectiveness of our estimation model.

6.4 Comparison of Standalone Solution

Here we present the results of comparing our parallel graph extraction solution (PGE for short) with two standalone solutions, graph database based solution and matrix-based solution in Table 2. We run PGE with hybrid plan in a single worker.

Comparing with the graph database-based solution, we clearly see that, even with a single worker, the PGE has a better performance. This is because, the graph database is only optimized for querying local graph data,
and it cannot achieve high performance when handling global graph data.

The performances of Matrix-based solution and PGE may vary for different queries. After profiling, we found that when the final matrix is small or sparse, matrix-based solution performs better, otherwise PGE wins. First, this is because the time cost of transforming the final matrix into the subgraph with original vertex ids is proportional to the number of edges in the matrix. Second reason is that, PGE is ran in a parallel framework with a single worker, the overhead is much larger than the matrix-based solution.

### 6.5 Comparison of RPQ-based Solution

Now we show the results of comparing our parallel graph extraction solution to the RPQ-based solution [15] in Table 3. For the parallel graph extraction solution, we use the hybrid plan. And both solutions are run in parallel with ten works. The results show that current RPQ solution has good performance when the extraction workload is light and it performs bad when the workload increases. For example, when running dblp-SP3, our RPQ solution has good performance when the extraction time ranges from 0.01 to 0.1 times the hybrid plan. And both solutions are run in parallel with ten works. The results show that current RPQ solution has good performance when the extraction workload is light and it performs bad when the workload increases. For example, when running dblp-SP3, our RPQ solution has good performance when the extraction time ranges from 0.01 to 0.1 times the hybrid plan.

### 6.6 Scalability

Finally, we conduct experiments to evaluate the scalability of the parallel homogeneous graph extraction approach. The experiments demonstrate that the approach scales well with increasing the number of workers, increasing the size of dataset or increasing the length of line pattern.

#### Scalability with varying the number of workers.

Figure 10(a) shows the runtime of executing dblp-SP2 with different number of workers. The results reveal that the solution almost scales linearly when the number of workers increases. For example, the dblp-SP2 spends about 140 seconds in extracting the homogeneous graph when 20 workers are provided; and the cost is reduced to about 93 seconds with doubling the number of workers (i.e., 40).

#### Scalability with varying the size of dataset.

Figures 10(b) and 10(c) illustrate the results of running dblp-SP2 on datasets with varying the number of vertices from 1M to 10M. Since the original dblp-2014 only contains about 4M vertices, we generate a set of synthetic datasets in different size as follows. The datasets whose number of vertices is less than 4M are generated by randomly sampling vertices from the original dblp-2014. We generate datasets having more than 4M vertices by adding new fake venues, which are randomly sampled from the existing venues. From Figure 10(b), we can see that the runtime increases when the size of dataset becomes large, but the increase is not linear to the size of dataset. As analyzed previously, the performance of our solution is affected by the number of intermediate paths when the number of iterations is the same. We further profiled the size of intermediate paths for different datasets. And the sizes of intermediate paths on different datasets are normalized to the size of intermediate paths on dataset with 1M vertices. So does the runtime. Figure 10(c) illustrates the normalized values (or ratios), and it shows that the degradation of the performance is proportional to the number of intermediate paths. This demonstrates that the solution still scales well to the size of dataset implicitly.

#### Scalability with varying the length of line pattern.

Increasing the length of line pattern, the number of intermediate paths increases exponentially. However, by using partial aggregation technique, the actual size of intermediate paths is polynomial. Figure 10(d) shows the results of running line patterns of different lengths with 40 workers on us-patent dataset. Here we use the citeBy relation to generate line patterns of different lengths. A line pattern of length $L$ means that the line pattern consists of $L$ citeBy relations. The results demonstrate that when the length of line pattern increases at the beginning, the performance degrades fast because of the fast increasing of the number of intermediate paths; but when the length of line pattern exceeds a certain threshold, like nine in the experiment, the decrease of the performance becomes slightly. This is due to the benefit brought by the partial aggregation technique.

### 7 Related Work

To run existing graph algorithms on heterogeneous graphs, Rodriguez [18] introduced path algebra, which is
a variety of matrix algebra, for mapping multi-relational networks to single-relational networks. His work only focused on theoretical part of designing path algebra operations to represent the various semantics, and didn’t address any practical challenges. Our work formalizes the homogeneous graph extraction problem on top of the graph model, and centers on developing efficient evaluation techniques for the problem.

In the graph model, the homogeneous graph extraction has two sub tasks, path enumeration and aggregation of paths. The path enumeration is an evaluation of path query on graph data. Cruz et. al [4] first introduced a general path query expressed by regular expression to support recursive queries. This kind of path query is called regular path query (RPQ) and RPQ can selects paths in arbitrary length. In the past years, the RPQ has attracted many attentions [11], [15], [5]. L. Libkin introduced that regular path query with graph data has NLOGSPACE data and PSPACE combined complexity [11]. A similar query in RDF domain is property path. The work [1] has shown that the counting version of the property path is intractable. In this work, the line pattern is a special type of RPQ and the line pattern requires to select out paths in fixed-length. Besides, the definition of meta-path [24] is similar to the one of line pattern, but focus on different problems.

Aggregation is a common routine to summarize a graph (or network). The basic flow is that we first create a temporary graph based on some filters or criteria; then compute the aggregate metrics. In graph OLAP applications [3], [27], [25], the filters are always related to the attributes of vertices and edges, and the attributes are used to generate groups of vertices and edges. The aggregation can be simple statistical functions, like MAX, MIN, COUNT, etc., and it can be complex metrics (e.g., maximum flow [3]) as well. Compared to the aggregation schema of graph OLAP, in this work, the aggregation schema uses the line patterns to produce elements for aggregating and involves a two-level aggregate model.

Finally, we discuss the dynamic programming technique for PCP selection. Because a line pattern on a heterogeneous graph is similar to a chain join in relation databases, to some extend, the PCP and classical join ordering (e.g., left-deep tree and bushy tree) [8] are similar. Both the PCP and join ordering form a tree structure, and they can be selected by using dynamic programming technique. However, the PCP has its own characteristics. First, PCP is a plan for path concatenation with graph exploration. The primitive pattern (node) in a PCP is evaluated by graph exploration. Second, PCP is built on top of graph data model. Third, the cost model of PCP is to estimate the size of intermediate paths, not the number of I/O disks. To summarize, PCP has its own approach for finding a good plan.

8 Conclusion

Heterogeneous graph is a natural model to describe the complex relations in the real world. We focused on studying the problem of extracting homogeneous graph from a heterogeneous graph based on the user input pattern. To efficiently solve the problem, we proposed a parallel graph extraction algorithm with the help of path concatenation strategy. Demonstrated by the extensive experiments, the proposed solution indeed extracted homogeneous graphs efficiently compared to graph-based and matrix-based solutions.

Acknowledgements

This research is funded by 973 program (No. 2014CB34040), China Postdoctoral Science Foundation (No. 2017M610020), National Natural Science Foundation of China (Nos. 61572039, 61472009 and U1536201), Shenzhen Gov Research Project (No. CYJ20151014093505032) and Alibaba-PKU joint Program. Zi Huang’s work is supported by Australian Research Council (No. FT130101530). Kai Lei is the corresponding author.

References


Kai Lei is an Associate Research Professor and Assistant Dean, School of Electronics and Computer Engineering (ECE), Shenzhen Graduate School, Peking University, Shenzhen, China. He received his B.Sc. in Computer Science from Peking University in 1998 and M.Sc in Columbia University in 1999. In 2015, he got his Ph.D. in Computer Science, at Peking University.

Lei Chen is a Professor in Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He received his BS degree in Computer Science at Tian Jin University, P.R.China(BS 94), and an MA degree in computer science at Asian Institute of Technology (AIT) Asian Institute of Technology (MS 97). He received a Ph.D. degree in Computer Science at University of Waterloo.

Zi Huang is an ARC Future Fellow in School of ITEE. The University of Queensland. She received her BSc degree from Tsinghua University, and her PhD from The University of Queensland. Dr. Huang's research interests mainly include multimedia indexing and search, social data analysis and knowledge discovery.

Bin Cui is a Professor in the School of EECS and Institute of Network Computing and Information Systems, at Peking University. His research interests include database systems, and data mining. Prof. Cui has published more than 100 research papers, and is the winner of Microsoft Young Professorship award (MSRA 2008), and CCF Young Scientist award (2009).

Yunhai Tong is currently an Associate Professor of Department of Machine Intelligence and the State Key Laboratory on Machine Perception (School of Electronics Engineering and Computer Science) at Peking University, China. He received his B.Sc. and M.S. in Computer Science from Petroleum University of China, in 1993 and 1996 respectively, and the Ph.D. degrees in Computer Science, from Peking University, in 2002.

Jin Xu is currently a Professor of Department of Computer Science at Peking University, China. His current research interests include system optimization and algorithm, biological and artificial neural networks, and graph theory and network optimization design.