Accelerating Parallel Evaluation of Regular Path Queries on Large Graphs by Estimating Joining Cost of Subqueries

Van-Quyet Nguyen  
Hung Yen University of Technology  
and Education  
Hung Yen, Vietnam  
quyetict@utehy.edu.vn

Van-Hau Nguyen  
Hung Yen University of Technology  
and Education  
Hung Yen, Vietnam  
haunv@utehy.edu.vn

Huy-The Vu  
Hung Yen University of Technology  
and Education  
Hung Yen, Vietnam  
thevh@utehy.edu.vn

Minh-Quy Nguyen  
Hung Yen University of Technology  
and Education  
Hung Yen, Vietnam  
quynm@utehy.edu.vn

Quyet-Thanh Huynh  
Hanoi University of Science and Technology  
Ha Noi, Vietnam  
thanghq@soict.hust.edu.vn

Kyunbaek Kim  
Chonnam National University  
Gwangju, South Korea  
kyungbaekkim@jnu.ac.kr

ABSTRACT

Using Regular Path Queries (RPQs) is a common way to explore patterns in graph databases. Traditional automata-based approaches for evaluating RPQs on large graphs are restricted in the graph size and/or highly complex queries, which causes a high evaluation cost. Recently, the threshold rare label based approach applied on large graphs has been proved to be effective. Nevertheless, using rare labels in a graph provides only coarse information which could not always guarantee the minimum searching cost. Hence, the Unit-Subquery Cost Matrix (USCM) based approach has been proposed to reduce the parallel evaluation cost by estimating the searching cost of RPQs. However, the previous approach does not take the joining cost among subqueries into account. In this paper, the method of estimating joining cost of subqueries is proposed in order to accelerate the USCM based parallel evaluation of RPQs. Specifically, the proposed method is realized by estimating the result size of subqueries. Through our experiments upon real-world datasets, it is depicted that estimating joining cost enhances USCM based approach up to around 20% in terms of response time.

1 INTRODUCTION

Regular path query (RPQ) is a common class of queries providing a way of finding connections and patterns in a graph database. There have been a number of approaches for evaluating RPQs. However, only a few of these approaches provide the performance guarantee on computational cost due to the large size of graphs and/or highly complex queries. The need for developing an efficient approach with low computational cost is particularly evident for evaluating RPQs, which are most commonly used in practice.

A well-known approach for evaluating RPQs is to exploiting automata. However, this approach has a disadvantage when the applied graph is large: the long response time is triggered by the mapping of the automaton states onto the graph. To deal with this issue, optimization techniques have been studied to reduce the evaluation cost of RPQs. Rewriting RPQs is introduced as the first technique, in which, the original expression of an RPQ is transformed into another one to reduce the search space by avoiding the whole graph traversal. Nevertheless, there is a limitation when the RPQs are highly complex (e.g., an RPQ with a modifier * over a group of alternate label).

Recently, a threshold rare label based approach has been proved to be efficient to evaluate RPQs on large graphs. The basic idea of this method is that the original query is split into multiple subqueries at rare labels, which are used as fixed points for graph searching. However, the threshold rare label based approach has some limitations since it relies on the presence of rare labels, their positions on the queries, and their quantity in the graphs. Our earlier work used Unit-Subqueries Cost Matrix (USCM) to estimate the searching cost of RPQs and obtain the viability of the usage of subqueries in RPQs evaluation. However, this method does not take the joining cost among subqueries into account. This neglect could increase the response time when evaluating RPQs on a large graph.
In this paper, we propose a method of estimating the joining cost of subqueries in order to accelerate the USCM based parallel evaluation of RPQs (called USCM-Join). We first propose cost functions and algorithms to estimate the result size of a given RPQ, which is directly related to the joining cost of subqueries. We then present how to improve the evaluation performance of RPQs by splitting them with a combination of the estimated joining and searching cost. Finally, we conduct extensive experiments on both real-world graphs and synthetic graphs, and the experimental results show that our USCM-Join approach outperforms the original one and other approaches.

The rest of this paper is organized as follows. We introduce an overview of related work in Section 2. We present some terms, definitions, and splitting RPQs for parallel evaluation using USCM in Section 3. Section 4 describes our proposal for estimating the result size of RPQs and evaluating RPQs in a parallel manner by exploiting the joining cost. We conducted experiments to evaluate our method and describe the results in Section 5. Section 6 concludes this paper.

2 RELATED WORK

There have been a lot of studies done on RPQs evaluation as well as providing query languages on graph data [3], [10], [20]. A common way to evaluate an RPQ is using the automata-based approach. This approach converts the graph to a DFA (Deterministic Finite Automaton), and the expression of an RPQ can be translated into an automaton, then computes the cross-product of the automaton to find the answer [9]. However, the limitation of this approach is that every state in automaton needs to be mapped onto the graph, which causes substantial memory space consumption and long response time. To address this problem, a number of studies has been proposed with optimization techniques to reduce the cost of RPQs evaluation.

A strategy for reducing the RPQs evaluation cost is to optimize the RPQs by rewriting them into other ones [8], [5]. Fernandez Mary et al. [8] presented two optimization techniques based on graph schemas. Calvanese et al. [5] proposed a view-based query rewriting approach for evaluating RPQs in semi-structured data which guarantees the new ones contain all the answers of the original ones. However, the query rewriting techniques still have some limitations deal with highly complex RPQs, such as the nested queries with modifier recursion, which leads to state explosion after converting the rewritten query to a DFA for graph searching. Therefore, several techniques have also been proposed for estimating query size [13] or minimizing DFAs [2], [12].

Recently, a threshold rare label based approach has been proved effectively to reduce the search space of RPQ evaluation on large graphs [10]. The authors employ a cost-based technique to determine which labels in a graph and/or a query are considered to be rare. Then, the rare labels are used as start-points, end-points, and way-points in traversal time, which reduce the number of visited nodes as well as the response time. However, the disadvantage of this approach is that the graph search algorithm depends on the presence of rare labels. So, in the case there are poor rare labels on the graph and the RPQs, or long queries, this approach still takes a high evaluation cost.

To the best of our knowledge, although there have been many studies focusing on RPQs evaluation, researches related to evaluation cost estimation of RPQs and its efficiency have not gotten much attention [15]. Silke et al. [18] proposed cost functions to estimate the response time and the result size of reachability path queries. Davoust et al. [7] focused on estimating the volume transmitted through the network while evaluating RPQs to provide appreciated strategies for evaluating them on distributed graphs. Among all of these works, there is no one that issues any cost estimating functions relying on RPQ operators as well as connectivity of labels in the query and the graph. In this work, we propose an efficient method for evaluating an RPQ by splitting it into multiple smaller subqueries based on estimation of their searching cost and joining cost.

3 PRELIMINARIES

3.1 Graph Data and Regular Path Queries

Graph Data. We consider an edge-labeled directed graph $G = (V, E, \Sigma)$, where $V$ is a set of nodes, $\Sigma$ is a set of labels, and $E \subseteq V \times \Sigma \times V$ is a set of edges. In which, an edge $(v, a, u)$ indicates edge direction from node $v$ to node $u$ labeled with $a \in \Sigma$.

Regular Path Queries. An RPQ $Q(R)$ is a regular expression $R$ over some labels in $\Sigma$. Here, $R$ is defined formally by $R = \varepsilon | a | R + R | R \cup R | R^*$, where $\varepsilon$ is an empty value; $a$ is a label in $\Sigma$; $R, R + R, R \cup R,$ and $R^*$ denote concatenation, alternation, and Kleene Star, respectively.

Let us categorize regular expression $R$ into four types of RPQ as the following:

- Concatenation RPQ: $R = a_0a_1...a_n$
- Alternation RPQ: $R = a_0...a_{i-1}(a_i|a_{i+1})a_{i+2}...a_n$
- Kleene Star RPQ: $R = a_0a_1...a_{i-1}a_i^*a_{i+1}...a_n$
- Highly Complex RPQ: $R = a_0a_1...a_{i-1}(a_i|a_{i+1})^*a_{i+2}...a_n$

where $a_i \in \Sigma$, $0 \leq i \leq n$. For clarity of presentation, we use the symbol $|$ for alternation operator and drop the symbol $\varepsilon$ in terms and equations, but keep them in examples.

To answer an RPQ, $Q(R)$, we need to search all paths in the graph $G$ which satisfy a given regular expression $R$. Here, a path $\rho$ between node $v_0$ and node $v_k$ in $G$ is a sequence

$\rho = v_0 \xrightarrow{a_0} v_1 \xrightarrow{a_1} v_2...v_{k-1} \xrightarrow{a_{k-1}} v_k$

such that each $(v_i, a_i, v_{i+1})$, for $0 \leq i < k$, is an edge. The sequence of labels of a path $\rho$, denoted as $L(\rho)$, is the string $a_0a_1...a_{k-1} \in \Sigma^*$, where $\Sigma^*$ is a set of all possible strings over the set of labels $\Sigma$. The answer of $Q(R)$ is a set of paths in the form $Q(R) = \{ \rho \}$, where $v, u \in V$, and $L(\rho) \subseteq \Sigma^*$ is a regular language. Thus, a path $\rho$ is an answer path of $Q(R)$ iff $L(\rho) \in L(R)$.

3.2 USCM-based Splitting RPQs for Parallel Evaluation

In the USCM-based approach for parallel evaluation of RPQs [16], an original RPQ is split into smaller subqueries based on the estimated searching cost. In this method, a Unit-Subqueries Cost Matrix (USCM) is generated by analyzing the cost of unit-subqueries in a graph. A unit-subquery is a smallest subquery $Q(a_ia_j)$ concatenated by a start label $a_i$ with an end label $a_j$, where $a_i$ and $a_j$ are
The result size is the number of paths on the graph being matched with regular expression \(a_0a_1...a_{n-1}\). We formulate the estimation of result size by the equation below.

\[
P = \mu(a_0, a_1) \times \frac{\mu(a_1, a_2)}{\delta(a_1)} \times \cdots \times \frac{\mu(a_{n-1}, a_n)}{\delta(a_{n-1})}
\]

(1)

**Example 1:** Assuming that there is a graph \(G\) as illustrated in Figure 1, then a recommendation system can help leaders of a company/shop to find out which products are liked by people who are followed by their employees. A regular expression \(R\) representing this finding is \(R = \text{isLeaderOf} \circ \text{follows} \circ \text{likes}\). By using Equation 1, we can estimate the result size of \(Q(R)\) as follows.

\[
P = \mu(\text{isLeaderOf}, \text{follows}) \times \frac{\mu(\text{follows}, \text{likes})}{\delta(\text{follows})}
\]

\[= 2 \times \frac{1}{3} \approx 1 \]

Thus, the estimated result size is one which equals the number of true paths satisfying the query, that is \(v_3 \xrightarrow{\text{isLeaderOf}} v_6 \xrightarrow{\text{follows}} v_7 \xrightarrow{\text{likes}} v_{10}\).

(2) Alternation RPQ. The result size of \(Q(R)\) in this case can be calculated by summing the number of paths matched two regular expressions: \(a_0...a_{i-1}a_ia_{i+2}...a_{n}\) and \(a_0...a_{i-1}a_{i+2}...a_{n}\). Specifically, we formalize the estimation by consider some specific cases as the following:

- For the simplest case, \(R = a_0(a_1|a_2)a_3\), there is no concatenation sub-query before and after a group of alternate labels. The result size can be estimated like

\[
P = \mu(a_0, a_1) \times \frac{\mu(a_1, a_2)}{\delta(a_1)} + \mu(a_0, a_2) \times \frac{\mu(a_2, a_3)}{\delta(a_2)}
\]

(2)

- For a general case, \(R = a_0...a_{i-1}(a_ia_ia_{i+2})a_{i+2}...a_{n}\), where \(i \geq 2\) and \(i + 3 \leq n\), we estimate the result size of \(Q(R)\) by using the equation below.

\[
P = P_{i-1} \times \left(\frac{\mu(a_{i-1}, a_i)}{\delta(a_{i-1})} \times \frac{\mu(a_i, a_{i+2})}{\delta(a_i)} + \frac{\mu(a_{i+2}, a_{i+3})}{\delta(a_{i+2})} \times \cdots \times \frac{\mu(a_{n-1}, a_n)}{\delta(a_{n-1})}\right)
\]

(3)
with length $P_1$ on subquery $a$ being equal to the summation of all paths satisfying one of all possible paths which end at terminal-points as shown in Figure 2. It includes the number paths formation of all paths satisfying one of all possible paths which end at terminal-points as shown in Figure 2. It includes the number paths containing $\omega$ where $\omega = \frac{\mu(a_i, a_j)}{\delta(a_i)}$ and we assume that $\omega < 1$ as usual; and $\gamma$ is the longest path length of $a_i$ in the result of $Q(R)$ as illustrated in Figure 2, $\gamma \in \mathbb{N}$.

Here, how to specify the upper bound of $\gamma$ for a given RPQ is non-trivial. Fortunately, we can estimate $\gamma$ by using USCM. Obviously, to obtain the paths with $\gamma$ labels $a_i$, the number of path with $\gamma - 1$ labels $a_i$ need to be greater than or equal to one. So, we have the equation

$$P_{i+\gamma-1} \geq P_i \times \omega^{\gamma-1}$$

Then, $P_{i+\gamma-1} \geq 1$ means that

$$\gamma \leq \log_\omega (1/P_i) + 1$$

(3) Kleene Star RPQ. Typically, the result size of an RPQ having Kleene Star operator is much larger than other ones. It is the summation of all paths satisfying one of all possible paths which end at terminal-points as shown in Figure 2. It includes the number paths with length $n$ (without $a_i$), $P_O$, and the remained paths containing at least one label $a_i$, $P_K$. Let $P$ be the result size in this case, we have $P$ being equal to the summation of $P_O$ and $P_K$, where $P_O$ and $P_K$ are estimated by the equations below:

$$P_O = P_{i-1} \times \prod_{i=1}^{n} \frac{\mu(a_i, a_{i+1})}{\delta(a_i)}$$

$$P_K = P_1 \times (1 + \omega + \omega^2 + \ldots + \omega^n) \times \prod_{i=1}^{n} \frac{\mu(a_{i-1}, a_i)}{\delta(a_{i-1})}$$

Algorithm 1 EstimateAlterStar

Require: $pre$, string before alternation operator, $a$: the first label in group of alternation operator, $b$: the second label in group of alternation operator, $suf$: string after alternation operator, and USCM

Ensure: $P$: the number of paths satisfying $R$

1. $P_O \leftarrow$ EstimateConcat($pre + suf$, USCM);
2. $P \leftarrow P + P_O$;
3. $P_1 \leftarrow$ EstimateConcat($pre + a$, USCM);
4. $P_2 \leftarrow$ EstimateConcat($pre + b$, USCM);
5. if $P_1 \geq 1$ then
   6. EstimateAlterStar($pre + a$, $a$, $b$, suf, USCM);
7. if $P_2 \geq 1$ then
   8. EstimateAlterStar($pre + b$, $a$, $b$, suf, USCM);

$P_K = P_1 \times (1 + \omega + \omega^2 + \ldots + \omega^n) \times \prod_{i=1}^{n} \frac{\mu(a_{i-1}, a_i)}{\delta(a_{i-1})}$

where $\omega = \frac{\mu(a_i, a_j)}{\delta(a_i)}$ and we assume that $\omega < 1$ as usual; and $\gamma$ is the longest path length of $a_i$ in the result of $Q(R)$ as illustrated in Figure 2, $\gamma \in \mathbb{N}$.

Here, how to specify the upper bound of $\gamma$ for a given RPQ is non-trivial. Fortunately, we can estimate $\gamma$ by using USCM. Obviously, to obtain the paths with $\gamma$ labels $a_i$, the number of path with $\gamma - 1$ labels $a_i$ need to be greater than or equal to one. So, we have the equation

$$P_{i+\gamma-1} \geq P_i \times \omega^{\gamma-1}$$

Then, $P_{i+\gamma-1} \geq 1$ means that

$$\gamma \leq \log_\omega (1/P_i) + 1$$
4.2 Parallel Evaluation of RPQs by Exploiting Joining Cost

Let us consider an RPQ, Q(R), which is split into k subqueries. The estimated parallel evaluation cost of Q(R) consists of the estimated searching cost and the estimated joining cost of its subqueries. First, the estimated searching cost $C_S$ of Q(R) can be computed by using Equation 8:

$$C_S = \max [C_{q_1}, C_{q_2}, \ldots, C_{q_k}],$$  \hspace{1cm} (8)

where $C_{q_i}$ is the estimated searching cost of subquery $q_i$, $0 < i \leq k$, which is estimated by using the method in our previous works [16].

Next, we estimate the joining cost of the subqueries. Here, we do not consider methods for optimizing the joining cost such as multiway joins [1] or top-k join queries [19]. We therefore use a join sequence for subqueries’ results. That is, the two first partial answers will be joined, then the result will be used to join with the third partial answer, and so on. In our implementation, a merge-join is used to match two partial answers. Thus, a $O(M \times N)$ merge-like step is performed to determine the matching paths, where $M, N$ are the result sizes of two partial answers. Let us assume that $P_{q_i}$ is the result size of subquery $q_i$, and $C_J$ is the estimated joining cost, so we can formalize $C_J$ by using the equation below:

$$C_J = P_{q_1} \times P_{q_2} + \eta(q_1,q_2) \times P_{q_3} + \cdots + \eta(q_1,q_2,\cdots,q_{k-1}) \times P_{q_k}$$  \hspace{1cm} (9)

where $\eta$ is the function for estimating the result size of the query which is established by concatenation of subqueries.

Finally, to obtain the parallel evaluation cost, denotes as $C_{PE}$, we need to sum up the searching cost $C_S$ and the joining cost $C_J$. However, they are not represented by the same unit, in which, the unit of the searching cost is the number of traversed edges; meanwhile, the unit of the joining cost is the number of operations (merge-like). Therefore, we need to use a conversion to make the units be uniform. To do this, we analyze the searching cost and try to represent it in the number of operations. In practice, in order to find the nodes for the next searching step, each traversed edge is checked whether there exists a matched label with any label of transitions from a current state in the query automaton. We assume that the cost of an operation of matching two labels in the searching step is similar to the one in the joining step. By specifying the searching cost like this way, we can use the number of operations as the unit of the searching cost (from now on). Let us assume that $\beta$ is the average degree of the query automaton, and it can be obtained after reading query. So, we can estimate the parallel evaluation cost in the number of operations as follows:

$$C_{PE} = C_S \times \beta + C_J$$  \hspace{1cm} (10)

Based on estimated evaluation cost, we can reduce the cost of parallel evaluation of an RPQ Q(R) on multiple CPU cores. Let us assume that $k$ is a given number of CPU cores, which can be used to evaluate Q(R). Our algorithm is done in four steps as the following:

Step 1: Split R into every possible set of subqueries so that the number of subqueries in each set is less than or equal to k.

Step 2: Estimate the evaluation cost of every set of subqueries by using USCM, in which, both the searching cost and joining cost are considered. Then, we select only one set of subqueries, $S_{best}$, which has a minimum estimated evaluation cost.

Figure 4: Response time comparison on different graphs.

Step 3: Evaluate $S_{best}$ in a parallel manner, in which, each subquery is searched separately under a CPU core.

Step 4: If each search process found at least one path, we merge the results from $|S_{best}|$ subqueries to get the final answer.

5 EXPERIMENTAL EVALUATION

In this section, we conducted experiments for evaluating the effectiveness of our proposed method. We compares the performance of our USCM-Join approach with USCM-Basic approach (without consideration of joining cost) and other approaches including automata-based approach (AUT) [9] and the threshold rare label based approach (TRL) [10].

Environments. We implemented all algorithms in Java. Experiments are conducted on a single personal computer which has 3.60GHz Intel Core i7, 4 CPU cores, and 8.0GB of RAM.

Datasets. We used two real-life graphs: the first one is a protein-protein interactions network named Alibaba [10], which contains 52,050 nodes, 340,775 edges, and 649 labels. The second is a knowledge graph named Yago, which is extracted from the main entities of Yago3 [14] with 1,756,958 nodes, 3,615,249 edges, and 13 labels. We also used a synthetic graph with around 16,000 nodes and 304,000 edges, which is generated by using Gephi [4]. We used 15 distinct labels to annotate edges for these graphs. The occurrence of labels follows the Zipfian distribution.

Query sets. With experiments on Alibaba graph, we used the queries set given by previous research [10]. We analyzed ten thousand queries and found that approximately 87% proportion is simple RPQs, 3% and 10% proportion contain nested RPQs with and without recursive modifiers, respectively. For Yago graph, we created a query set of 40 RPQs with length varied from 4 to 8. The query set has 10 queries for each type of RPQs in Section 4. For the synthetic graphs, we mainly used 1,000 random RPQs with 80% proportion of having concatenation and alternation RPQs, 15% proportion of having Kleene Star RPQs, and 5% proportion of having complex RPQs with recursive modifiers.

Experimental Results. Figure 4 illustrates the average response times of four different approaches on both real-world graphs and the synthetic graph. We observed that USCM-Join approach outperforms other approaches in all the cases. Especially, it reduces the response time around 20% on average compared to the USCM-Basic
approach. To explain how our proposed method reduces response time significantly in detail, we chose 10 queries randomly on Yago graph and shown their evaluation cost in Figure 5. We observed that AUT approach without separating RPQs has no joining cost, but high searching cost. TRL and UCSM-Basic approaches separating RPQs without joining cost consideration sometimes have high joining cost. Meanwhile, USC-MJoin separating RPQs with combination of joining and searching cost achieved the minimum evaluation cost.

6 CONCLUSION

This paper presented a method of estimating the joining cost of subqueries in order to accelerate the USCM based parallel evaluation of RPQs. The proposed method is realized by estimating the result size of subqueries, which are used to estimate the joining cost of the subqueries. Then, the evaluation performance of RPQs is improved by splitting them with combination of the estimated joining and searching cost. Through the experimental results upon real-world datasets, we found that estimating joining cost enhances USCM based approach up to around 20% in terms of response time. In the future, we will evaluate (1) the impact of query types (concatenation, alternation, Kleene Star, and complex RPQs) on the accuracy of result size estimation and (2) the impact of the different number of subqueries on our USCM-Join approach.

ACKNOWLEDGMENTS

This research is funded by Hung Yen University of Technology and Education under the grant number UTEHY.L.2020.07. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2017R1A2B4012559).

REFERENCES


Figure 5: Evaluation cost comparison in detail on Yago graph.