electronics

MDPI

Article

Efficiently Estimating Joining Cost of Subqueries in Regular Path Queries

Van-Quyet Nguyen [1,*], Van-Hau Nguyen [1,*], Minh-Quy Nguyen [1], Quyet-Thang Huynh [2] and Kyungbaek Kim [3,*]

[1] Faculty of Information Technology, Hung Yen University of Technology and Education, Hung Yen 160000, Vietnam; quynm@utehy.edu.vn
[2] School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi 100000, Vietnam; thanghq@soict.hust.edu.vn
[3] Department of Artificial Intelligence Convergence, Chonnam National University, Gwangju 61186, Korea
* Correspondence: quyetict@utehy.edu.vn (V.-Q.N.); haunv@utehy.edu.vn (V.-H.N.); kyungbaekkim@jnu.ac.kr (K.K.)

Abstract: Evaluating Regular Path Queries (RPQs) have been of interest since they were used as a powerful way to explore paths and patterns in graph databases. Traditional automata-based approaches are restricted in the graph size and/or highly complex queries, which causes a high evaluation cost (e.g., memory space and response time) on large graphs. Recently, although using the approach based on the threshold rare label for large graphs has been achieving some success, they could not often guarantee the minimum searching cost. Alternatively, the Unit-Subquery Cost Matrix (USCM) has been studied and obtained the viability of the usage of subqueries. Nevertheless, this method has an issue, which is, it does not cumulate the cost among subqueries that causes the long response time on a large graph. In order to overcome this issue, this paper proposes a method for estimating joining cost of subqueries to accelerate the USCM based parallel evaluation of RPQs on a large graph, namely USCM-Join. Through real-world datasets, we experimentally show that the USCM-Join outperforms others and estimating the joining cost enhances the USCM based approach up to around 20% in terms of response time.

Keywords: graph queries; USCM; parallel evaluation; estimating joining cost

Citation: Nguyen, V.-Q.; Nguyen, V.-H.; Nguyen, M.-Q.; Huynh, Q.-T.; Kim K. Efficiently Estimating Joining Cost of Subqueries in Regular Path Queries. *Electronics* **2021**, *10*, 990. https://doi.org/10.3390/electronics10090990

Academic Editors: Fabio Grandi and Domenico Ursino

Received: 28 February 2021
Accepted: 19 April 2021
Published: 21 April 2021

1. Introduction

Regular Path Queries (RPQs) are a powerful way of exploring connections and patterns in a graph database [1]. A number of approaches have been studied for evaluating RPQs [2,3]. However, only a few approaches provide the performance guarantee on computational cost due to the large size of graphs and/or highly complex queries. Therefore, developing efficient approaches for evaluating RPQs with a low computational cost is essentially important in practice [4,5].

Goldman and Widom introduced automata [6] for evaluating RPQs, and it has become a well-known approach. However, the automata-based approach has difficulty when dealing with large graphs which results in the long response time by the mapping of the automaton states onto the graph. Several optimization techniques have been addressed in order to overcome this difficulty. One of the effective approaches is to rewrite RPQs [7–9]. In particular, these researches formulate a given regular expression into another format for the purpose of avoiding the whole graph traversal to reduce the search space. Nonetheless, one might face an obstacle when the RPQs are exceedingly complex (e.g., an RPQ with a modifier operator * over a group of the alternate label).

Alternatively, the approaches based on the threshold rare label have been proposed and have been achieving some success in evaluating RPQs on large graphs by Koschmieder and Leser [10]. The idea of the approaches is based on the fact that, in a graph, not all of

the labels are equally frequent. By using rare labels as the fixed points for graph searching, the authors decomposed a given query into a sequence of RPQs. However, this approach relies on the presence of rare labels, the number of rare labels in the queries, and in the graphs, it could not often guarantee the minimum searching cost.

Recently, Nguyen et al. [11] 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, USCM-Join, for estimating the joining cost of subqueries in order to accelerate the USCM based parallel evaluation of RPQs. We first propose cost functions and algorithms to estimate the result size of a given RPQ, which is in terms of the joining cost of subqueries. Subsequently, we show how to improve the evaluation performance of RPQs by splitting them with a combination of the estimated joining and searching cost. Finally, through both real-world graphs and synthetic graphs, we experimentally demonstrate that the USCM-Join outperforms the original one and other approaches.

The rest of this paper is organized, as follows. Section 2 introduces an overview of related work. Section 3 presents terms, definitions, and splitting RPQs for parallel evaluation using USCM. In Section 4, we propose a method for estimating the result size of RPQs and evaluating RPQs in a parallel manner by exploiting the joining cost. Section 5 conducted the experiments to evaluate the proposed method. Finally, Section 6 concludes with a summary of our proposal.

This paper is an extension of work originally presented in The 9th International Conference on Smart Media and Applications (SMA 2020) [12].

2. Related Work

There have been a lot of studies conducted on RPQs evaluation as well as providing query languages on graph data [1–3,13–15]. 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, and then computes the cross-product of the automaton to find the answer [6]. 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 a long response time. To address this problem, a number of studies have 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 [7,8]. Fernandez Mary et al. [7] presented two optimization techniques based on graph schemas. The first one is query pruning, which is used to rewrite a given regular path expression into another, which helps to reduce substantial search space by avoiding all graph traversal. The second one avoids traversing the whole graph by employing state extents, which transforms the original query into the one that starts searching at the nodes being in *state extents* instead of the root. Calvanese et al. [8] proposed a view-based query rewriting approach for evaluating RPQs in semi-structured data, which guarantees that the new ones contain all the answers of the original ones. Other rewriting approaches for optimizing regular path queries are presented in [16]. However, the query rewriting techniques still have some limitations in dealing 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 [17] or minimizing DFAs [18,19].

Recently, a threshold rare label based approach has been proved to effectively 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. Subsequently, 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. Accordingly, 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.

The efficiency of RPQs evaluation has been intensively studied for distributed graphs. Specifically, there is a survey that depicts the state of the art in query evaluation on distributed graphs [20]. Dan Suciu [21] introduced a distributed query evaluation approach on semi-structured data, which takes a bounded complexity $O(n^2)$ for the volume of data transferred (n stands for the number of cross-edges among sites). Wenfei Fan et al. introduced some efficient algorithms [22], which employed the partial evaluation method to solve three classes of queries (i.e., reachability query, bounded reachability query, and regular reachability query) on distributed graphs. However, their algorithms face a bottleneck issue when the results are gathered together from distributed sources. This issue has also been presented in [23], which motivates the authors to extend RPQs evaluation in [24]. In which, a large number of redundant nodes and edges are found and eliminated before the partial results are gathered together at the coordinate site.

To the best of our knowledge, although there have been many studies focusing on RPQs evaluation, researches that are related to evaluation cost estimation of RPQs and its efficiency have not received much attention [17,25–27]. Silke et al. [28] proposed cost functions to estimate the response time and the result size of reachability path queries. Davoust et al. [29] 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 the 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, Σ is a set of labels, and $E \subseteq V \times \Sigma \times V$ is a set of edges. An edge $(v, a, u) \in E$ indicates the edge direction from node v to node u labeled with $a \in \Sigma$.

Regular Path Queries. A regular path query $Q(R)$ is a regular expression R over some labels in Σ. Here, R is defined in formally by

$$R = \epsilon \mid a \mid R \circ R \mid R \cup R \mid R^*,$$

where ϵ is an empty value; a is a label in Σ; $R \circ R$, $R \cup R$, and R^* denote concatenation, alternation, and Kleene Star, respectively.

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

- Concatenation RPQ: $R = a_0 a_1 ... a_n$
- Alternation RPQ: $R = a_0 ... a_{i-1}(a_i|a_{i+1})a_{i+2} ... a_n$
- Kleene Star RPQ: $R = a_0 a_1 ... a_{i-1} a_i^* a_{i+1} ... a_n$
- Highly Complex RPQ: $R = a_0 a_1 ... a_{i-1}(a_i|a_{i+1})^* a_{i+2} ... a_n$

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

To answer an RPQ, $Q(R)$, we need to search all paths in the graph G that satisfy a given regular expression R. Here, a path ρ 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 ρ, denoted as $L(\rho)$, is the string $a_0 a_1 ... a_{k-1} \in \Sigma^*$, where Σ^* is a set of all possible strings over the set of labels Σ. The answer of $Q(R)$ is a set of paths in the form $Q(R) = v \xrightarrow{L(R)} u$, where $v, u \in V$, and $L(R) \subseteq \Sigma^*$ is a regular language. Thus, a path ρ is an answer path of $Q(R)$ iff $L(\rho) \in L(R)$.

Example 1. To illustrate our ideas in this paper, from now on, we use an edge-labeled graph G of a social shopping network, as shown in Figure 1. In which, a node represents an entity, such as a person or a product, and an edge indicates the relationship between two entities with a label, such as $friend$, $follows$, or $purchased$. As a result, we have a set of edge labels

$$\Sigma = \{isLeaderOf, friend, follows, knows, purchased, likes, ownedBy\}.$$

We now consider a simple RPQ with

$$R = isLeaderOf \circ follows \circ likes.$$

This query helps leaders of a company/shop to find out which products are liked by people who are followed by their employees. By using automata-based approach, we can find only one path satisfying the query, which is:

$$v_3 \xrightarrow{isLeaderOf} v_7 \xrightarrow{follows} v_6 \xrightarrow{likes} v_{10}.$$

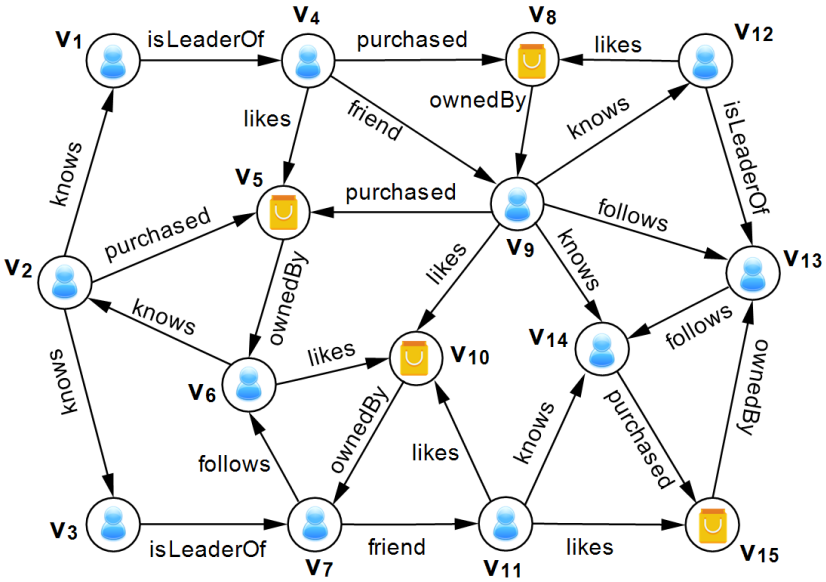

Figure 1. Example of a directed graph representing a social shopping network.

3.2. Uscm-Based Splitting Rpqs for Parallel Evaluation

In the USCM-based approach for parallel evaluation of RPQs [11], 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 $Q(a_i a_j)$ concatenated by a *start label* a_i with an *end label* a_j, where a_i and a_j are elements of the set labels Σ. The number a_i edges connected to a_j edges is defined as the cost of a unit-subquery $\mu(a_i, a_j)$. Table 1 illustrates an example of USCM. For a given RPQ, by estimating the searching cost of every possible set of its subqueries with USCM, the RPQ is split into the best set of subqueries, which has the minimum of estimated searching cost.

Table 1. Example of Unit-Subquery Cost Matrix (USCM) for a graph of social shopping network.

Label:Count	isLeaderOf	Friend	Follows	Knows	Purchased	Likes	ownedBy	Total
isLeaderOf:3	0	2	2	0	1	1	0	6
friend:2	0	0	1	3	1	3	0	8
follows:3	0	0	1	1	1	1	0	4
knows:6	3	0	0	2	3	1	0	9
purchased:4	0	0	0	0	0	0	4	4
likes:6	0	0	0	0	0	0	6	6
ownedBy:4	0	1	3	2	1	2	0	9

4. USCM-Based Parallel Evaluation of RPQs by Estimating Joining Cost

Because the joining cost means the cost of merging results of subqueries, it depends on the result size of the subqueries. Therefore, we first describe how to estimate the result size of a given RPQ as the joining cost. We then present how to split RPQs with combination of the joining and searching cost.

4.1. Estimating Result Size of RPQs with USCM

(1) Concatenation RPQ. The result size is the number of paths on the graph satisfying the query. We denote $\delta(a_i)$ as the number of labels a_i in G. It is undeniable that the result size of an RPQ with length 1, $R = a_0$, equals $\delta(a_0)$. While an RPQ with length 2, $R = a_0a_1$, has a result size being equal to $\mu(a_0, a_1)$. Intuitively, the number of paths satisfying the query with $R = a_0a_1 \dots a_{n-1}a_n$ depends on the number of paths on the graph being matched with regular expression $a_0a_1 \dots a_{n-1}$. We formulate the estimation of result size by the equation below.

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

Example 2. 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

$$R = isLeaderOf \circ follows \circ likes.$$

By using Equation (1), we can estimate the result size of $Q(R)$, as follows.

$$\begin{aligned} P &= \mu(isLeaderOf, follows) \times \frac{\mu(follows, likes)}{\delta(follows)} \\ &= 2 \times \frac{1}{3} \approx 1 \end{aligned}$$

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

(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: $a_0 \dots a_{i-1}a_ia_{i+2} \dots a_n$ and $a_0 \dots a_{i-1}a_{i+1}a_{i+2} \dots a_n$. Specifically, we formulize the estimation by considering 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, as follows.

$$P = \mu(a_0, a_1) \times \frac{\mu(a_1, a_3)}{\delta(a_1)} + \mu(a_0, a_2) \times \frac{\mu(a_2, a_3)}{\delta(a_2)} \tag{2}$$

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

$$\begin{aligned} P = P_{i-1} \times \Bigg(& \frac{\mu(a_{i-1}, a_i)}{\delta(a_{i-1})} \times \frac{\mu(a_i, a_{i+2})}{\delta(a_i)} \\ & + \frac{\mu(a_{i-1}, a_{i+1})}{\delta(a_{i-1})} \times \frac{\mu(a_{i+1}, a_{i+2})}{\delta(a_{i+1})} \Bigg) \\ & \times \frac{\mu(a_{i+2}, a_{i+3})}{\delta(a_{i+2})} \times \dots \times \frac{\mu(a_{n-1}, a_n)}{\delta(a_{n-1})} \end{aligned} \tag{3}$$

where P_{i-1} is the number of paths that are estimated by using Equation (1) on subquery $a_0 a_1 \dots a_{i-1}$.

Example 3. We clarify our idea for estimating the result size of an alternation RPQ by using an example where the regular expression

$$R = isLeaderOf \circ (friend \cup follows) \circ purchased.$$

By using Equation (2), we can estimate the result size of $Q(R)$, as follows.

$$\begin{aligned} P &= \mu(isLeaderOf, friend) \times \frac{\mu(friend, purchased)}{\delta(friend)} \\ &+ \mu(isLeaderOf, follows) \times \frac{\mu(follows, purchased)}{\delta(follows)} \\ &= 2 \times \frac{1}{2} + 2 \times \frac{1}{3} \approx 2 \end{aligned}$$

We can see that the estimated result size equals the number of true paths which includes:

$$v_1 \xrightarrow{isLeaderOf} v_4 \xrightarrow{friend} v_9 \xrightarrow{purchased} v_5$$

$$\text{and } v_{12} \xrightarrow{isLeaderOf} v_{13} \xrightarrow{follows} v_{14} \xrightarrow{purchased} v_{15}.$$

(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 that end at terminal-points, as shown in Figure 2. It includes the number paths with length n (without a_i), P_O, and the number of 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.

$$P_O = P_{i-1} \times \frac{\mu(a_{i-1}, a_{i+1})}{\delta(a_{i-1})} \times \dots \times \frac{\mu(a_{n-1}, a_n)}{\delta(a_{n-1})} \tag{4}$$

$$\begin{aligned} P_K = P_i \times (1 + \omega + \omega^2 + .. + \omega^\gamma) \times \frac{\mu(a_i, a_{i+1})}{\delta(a_i)} \times \dots \\ \times \frac{\mu(a_{n-1}, a_n)}{\delta(a_{n-1})} \end{aligned} \tag{5}$$

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

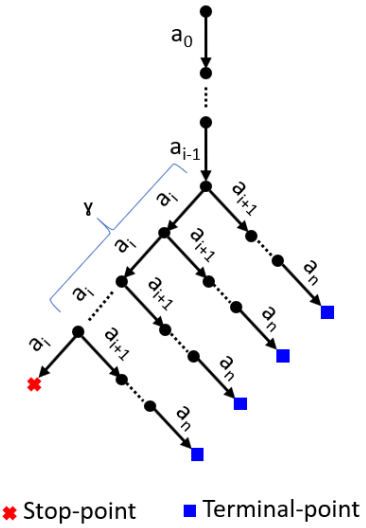

Figure 2. A tree representing all possible paths satisfying a Kleene Star RPQ.

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

$$P_{i+\gamma-1} = P_i \times \omega^{\gamma-1} \tag{6}$$

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

$$\gamma \leq \log_{\omega}(1/P_i) + 1 \tag{7}$$

Example 4. Let us consider an RPQ $Q(R)$, where $R = isLeaderOf \circ follows^* \circ purchased$. The result size in this example is $P = P_O + P_K$, where

$$P_O = \mu(isLeader, purchased) = 1, \gamma \leq \log_{(1/3)}(1/2) + 1 \approx 2, \text{ and}$$
$$P_K = \mu(isLeader, follows) \times (1 + \omega + \omega^2) \times \frac{\mu(follows, purchased)}{\delta(follows)} =$$
$$2 \times (1 + 1/3 + 1/9) \times 1/3 = 26/27 \approx 1$$

Hence, we have $P = 2$, which also equals the number of true paths to this query on the graph, which are:

$$v_1 \xrightarrow{isLeaderOf} v_4 \xrightarrow{purchased} v_8$$
$$\text{and } v_{12} \xrightarrow{isLeaderOf} v_{13} \xrightarrow{follows} v_{14} \xrightarrow{purchased} v_{15}.$$

(4) Highly Complex RPQ. Similar to the result size of an RPQ in case of Kleene Star operator, the result size of a highly complex RPQ is the summation of all paths that satisfy one of all possible paths from start-label a_0 to end-label a_n. However, in the case of a highly complex RPQ, there are multiple stop-points and terminal-points, as shown in Figure 3, so it is difficult to formalize the estimated result size by using equations. To estimate the result size of the highly complex RPQ, $Q(R)$, with $R = a_0 a_1 \cdots a_{i-1}(a_i|a_{i+1})^* a_{i+2} \cdots a_n$, we propose an algorithm that is shown in Algorithm 1. Here, the input arguments (e.g., *pre*, *a*, etc.) of the function *EstimateAlterStar* are extracted from the highly complex RPQ. Initially, we estimate the result size of the concatenation RPQ ($a_0 a_1 a_{i-1} a_{i+2} \cdots a_n$) whose answers have the smallest *path length* and have not included the alternate labels of the complex RPQ (line 1), by using estimation function *EstimateConcat* (not shown). The result is added to variable P, which is the number of paths satisfying R (line 2). Note that the value of P would be increased after calling the recursive function *EstimateAlterStar*. We

then perform estimating the result size of concatenation RPQs having longer path length by adding alternate labels one by one and re-calling *EstimateAlterStar* procedure (line 3–4). The estimation of concatenation RPQs having a longer path length is repeated until the estimated number of paths of every concatenation RPQ is smaller than one (lines 5–8).

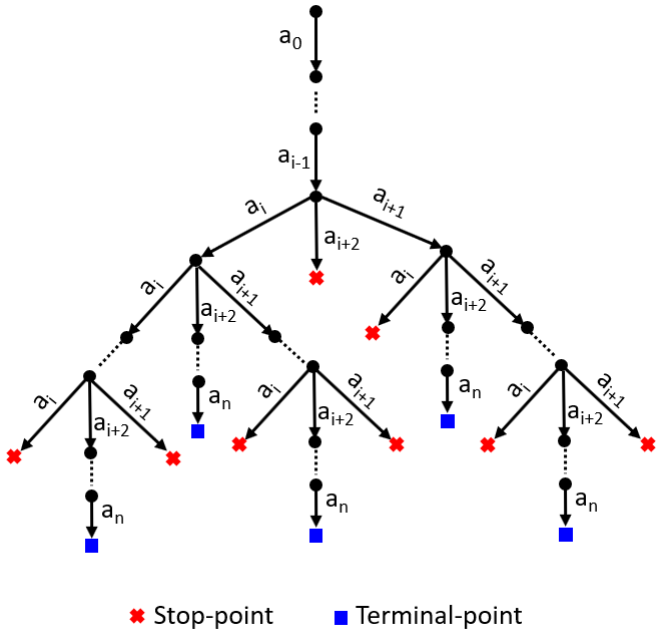

Figure 3. A tree representing possible paths satisfying a complex Regular Path Queries (RPQ).

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*);

4.2. Parallel Evaluation of RPQs by Exploiting Joining Cost

We first describe how to estimate a parallel evaluation cost of an RPQ with a combination of the searching and joining cost. We then present how to split the RPQs into subqueries based on the estimated parallel evaluation cost, and finally how to evaluate the subqueries in a parallel manner.

4.2.1. Estimating Parallel Evaluation 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 using Equation (8):

$$C_S = max[C_{q_1}, C_{q_2}, .., C_{q_k}] \quad (8)$$

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 the previous work [11].

Next, we estimate the joining cost of the subqueries. Here, we do not consider methods for optimizing the joining cost, such as multiway joins [30] or top-k join queries [31]. Therefore, we use a join sequence for subqueries' results. That is, the two first partial answers will be joined, and 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:

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

where η is the function for estimating the result size of the query that is established by concatenation of subqueries.

Finally, we need to sum up the searching cost C_S and the joining cost C_J to obtain the parallel evaluation cost, C_{PE}. 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 likes this way, we can use the number of operations as the unit of the searching cost (from now on). Let us assume that β is the average degree of the query automaton, and it can be obtained after reading query. Accordingly, we can estimate the parallel evaluation cost in the number of operations by using the equation below.

$$C_{PE} = C_S \times \beta + C_J \tag{10}$$

In the next subsection, we will describe how to use the estimated cost for splitting the original query into small subqueries in an efficient way.

4.2.2. Parallel Evaluation of RPQs based on Minimum Estimated Evaluation Cost

We consider reducing 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 $Q(R)$. That means the maximum number of subqueries, which we can split, is not exceed by k. Our algorithm is done in five 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. To do this, we find all possible combinations of sequenced labels.

Step 2: Estimate the evaluation cost of every set of subquery by using our idea in the previous section, in which both the searching cost and joining cost are considered. Subsequently, we only select one set of subquery, S_{best}, which has a minimum estimated evaluation cost for parallel processing in the next step.

Step 3: Evaluate S_{best} in a parallel manner, in which each subquery is separately searched under a CPU core. If no path can be found in any of these search processes, a message is issued to stop the search at every process, and then an empty result is returned. Otherwise, this step will be done when every search process completes the searching.

Step 4: If each search process found at least one path, using the results from *Step 2*, find all of the paths by merging the result of first subquery and the second subquery, and remove all outgoing nodes at split labels that are on no path, as these cannot be on a path

in the final result. This step will be done after $|S_{best}| - 1$ merging times, where $|S_{best}|$ is the number of subqueries of S_{best}.

Step 5: Finally, gather all of the paths satisfying the query and return the final result.

Note that, the searching technique for each subquery in *Step 3* above are recorded from *Step 2*. In which, we will use a backward search algorithm for the subqueries if their estimated searching cost in case of inverse less than the one in normal case. Otherwise, a forward search algorithm is used for evaluating them.

5. Experimental Evaluation

In this section, we conducted two experiments for evaluating the effectiveness of our proposed method. The first one compares the performance of our USCM-Join approach with the USCM-Basic approach (without the consideration of joining cost) and other approaches, including the automata-based approach (AUT) [6] and the threshold rare label based approach (TRL) [10]. The other one is used to assess the accuracy of estimating the result size using our proposed method.

5.1. Evaluation Settings

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

Datasets. We used three real-world graphs: the first one is Yago dataset that is a semantic knowledge-based, derived from Wikipedia, WordNet, and GeoNames [32]; the second one is Freebase dataset that is also a knowledge graph representing the fact around the world [33]; the third one is from biology field (named Alibaba). We also generated an IoT graph dataset (named Smart Building dataset). For deep evaluation, we generated synthetic graphs with a varied graph size as well as the average degree of graph for the evaluation. The usage of datasets is described as the following.

Yago dataset: To verify the adaptability of the proposed method, we used another real-world dataset, Yago, for evaluation. Yago is a huge semantic knowledge base, which extracted and combined entities and facts from 10 Wikipedias in different languages. Currently, to provide knowledge that is based on user's demand, Yago3 dataset (version 3 of Yago) is divided into different portions, and each portion is called a theme [34]. For example, the GEONAMES theme contains data of geographical entities and classes taken from GeoNames; meanwhile, the CORE theme has the main entities of Yago and the facts between entities. We extracted a knowledge graph from CORE theme of Yago to evaluate our proposed method. This graph has 1,756,958 nodes, 3,615,249 edges, and 13 labels. Each node represents an entity, such as a person, an organization, or a city; while, an edge represents the relationship between two entities, and it is assigned by a label as a fact (e.g., hasChild, isLeaderOf, isLocatedIn, etc.). Thus, the Yago dataset is fused by a social graph and a spatial graph, but not things graph.

Freebase dataset: This is a large knowledge graph of the facts around the world, which was developed by Metaweb Technologies company in 2007 and it was acquired by Google Inc. in 2010 [33]. It contains multiple entities, including famous places, people, factories, devices, and so on. The original dataset provides raw data that were dumped in RDF (Resource Description Framework) triples and it has several issues, like name disambiguation and duplicate entities. Therefore, we sought to find a different version of Freebase dataset that had solved the known major issues of the original one. This new dataset is available to download from http://freebase-easy.cs.uni-freiburg.de/dump (accessed on 1 August 2020) [35]. In this dataset, each triple is a fact in the form of $< subject >< predicate >< object >$. It corresponds to an edge on the knowledge graph, in which a $< predicate >$ is considered to be an edge-label. We extracted all if the triples that have predicates representing the relationship between two entities and ignored other ones. For instance, we used a triple $< Ann\ Taylor > < Has\ Child > < Christian\ Noel$

Davis >, but did not use < *Ann Taylor* > < *Weight* >< 57.2 >. Finally, we obtained a Freebase graph with 2,303,121 nodes, 3,224,470 edges, and 16 labels.

Alibaba dataset: The Alibaba graph is given by previous research [10]. The graph is a network of protein–protein interactions, which is regularly used in biology systems, for instance, to discover protein functions and pathways in biological processes [36]. This graph has 52,050 nodes, 340,775 edges, and 649 labels.

Smart Building Dataset: For testing with an IoT graph that is fused by all three types of graphs, including things graph, spatial graph, and social graph, we used gMark [37] to generate an edge-labeled graph that represents data objects (e.g., person, device, room) and their relationships in a smart building. Specifically, the graph has 36,000 nodes with nine node types, 273,610 edges, and 19 edge labels [38]. The occurrence of labels follows the given Zipfian or uniform distributions.

Synthetic graphs: To systematically study the adaptability of our proposed method on various parameters, such as graph size (|V| + |E|) or the average degree of the graph (|E|/|V|), we generated the synthetic graphs with different size by using Gephi [39]. Specifically, we varied the graph size from 40 K to 1280 K. In which, the smallest graph has around 2 K nodes and 38 K edges, and the largest one has 64 K of nodes and 1216 K of edges. We used 15 distinct labels to annotate edges for these graphs. The occurrence of labels follows the Zipfian distribution.

Query Sets. For Yago and Freebase graphs, we generated two query sets, each has a set of 40 RPQs with length varying from 4 to 8. The query sets have 10 queries for each type of RPQs in Section 3.1.

For Alibaba graph, we used the queries set that was given by previous research [10]. We analyzed ten thousand queries and found that approximately 87% proportion are simple RPQs, 3% and 10% proportion contain nested RPQs with and without recursive modifiers, respectively. With experiments on the Smart Building dataset, we also created 40 RPQs with length varying from 4 to 8. For the synthetic graphs, we mainly used 1000 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.

5.2. Experimental Results

Exp-1: Efficiency of USCM-Join Approach

Figure 4 illustrates the average response times of four different approaches on both real-world graphs and synthetic graphs. Here, we used the synthetic graph with around 16 K nodes and 304 K edges. We observed that the USCM-Join approach outperforms other approaches in all cases. It reduces the response time around 20% on average as compared to the USCM-Basic approach. Especially, we observed that the USCM-Basic, which considers estimating only the searching cost, reduces the average response time approximately 13%, 56%, 17%, 25%, and 60% when compared to TRL approach with Yago, Freebase, Alibaba, Smart Building, and the synthetic graph 320 K, respectively.

While our approach USCM-Join, which considers both the searching cost and the joining cost, obtained a better performance when this reduction is around 30%, 98%, 50%, 80%, and 110%, respectively.

To explain how our proposed method reduces response time significantly in detail, we chose 10 queries randomly on Yago graph and show their evaluation cost in Figure 5. We observed that the 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 a high joining cost. Meanwhile, USCM-Join separating RPQs with a combination of joining and searching cost achieved the minimum evaluation cost.

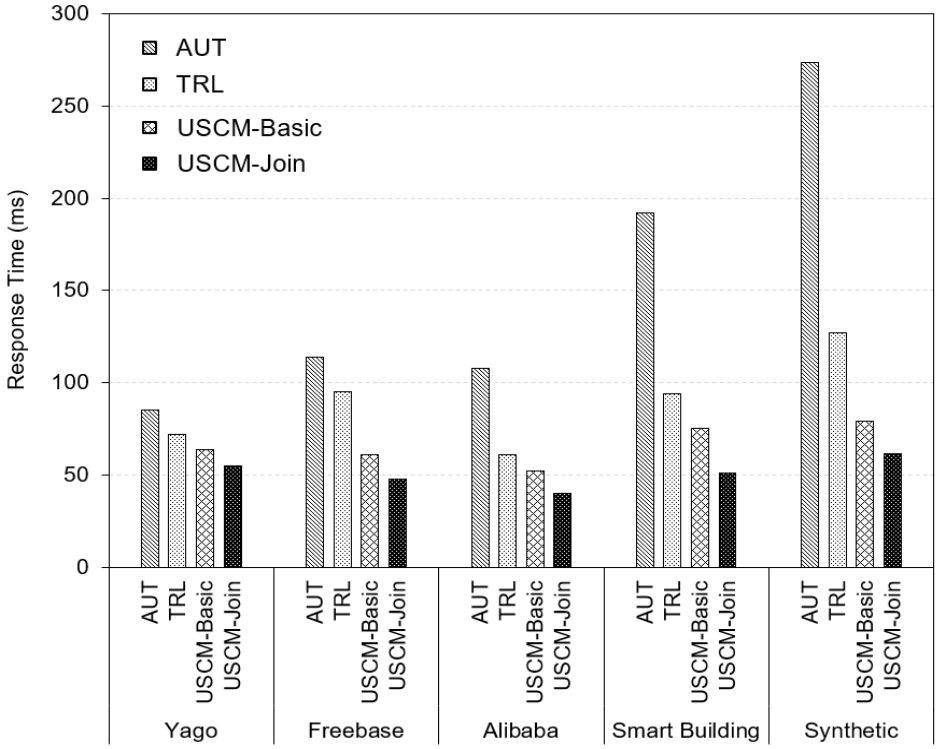

Figure 4. Response time comparison on different graphs.

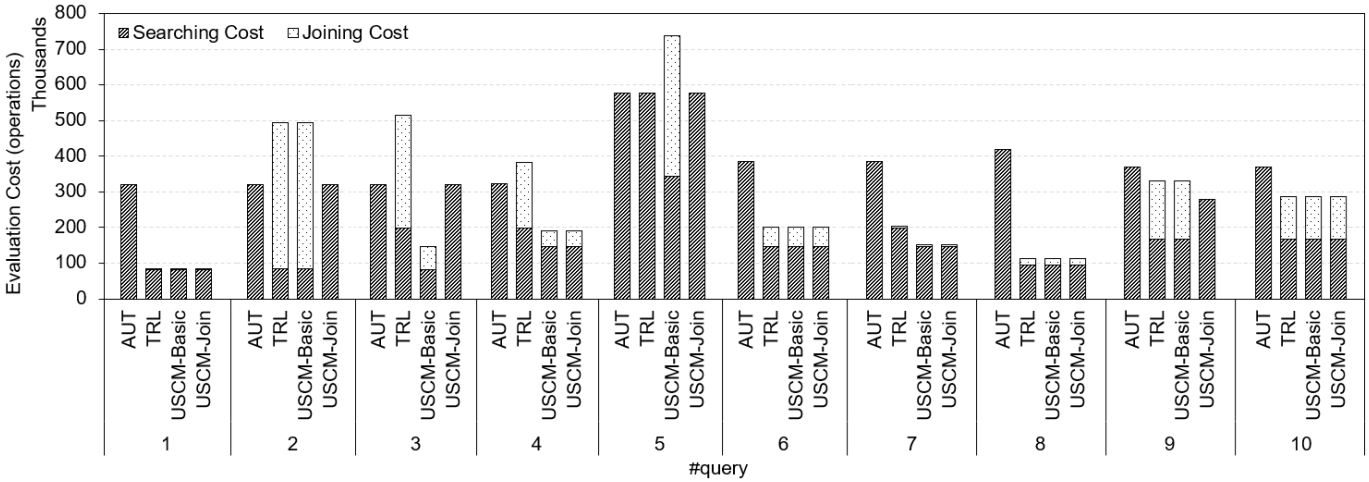

Figure 5. Evaluation cost comparison in detail on Yago graph.

To evaluate the impact of the different number of subqueries on our USCM-Join approach, we created a query set of 1000 queries, each has smallest path length at 5, so that it can be split into four subqueries. Figure 6 shows that a fixed number of subqueries for splitting is not efficient for optimizing RPQs evaluation. Here, splitting RPQs for evaluating them is necessary for reducing the response time, in which the case of splitting an RPQ into two subqueries can reduce the response time significantly, while the best case is the usage of a dynamic number of subqueries by using our estimation method. We can see that a high fixed number of subqueries causes very high joining cost. For example, in the case of four subqueries, the joining cost is twelve times higher than the searching cost on average.

Exp-2: Accuracy of Result Size Estimation

The impact of estimating joining cost to USCM-based RPQs evaluation is related to the accuracy of result size estimation. To understand the accuracy of the proposed method in different situations, we extensively evaluated the accuracy of result size estimation

with various graph parameters (e.g., the size of graph (|V| + |E|), average degree of graph (|E|/|V|)), and query parameters (e.g., length of query, type of query), as shown in Figure 7.

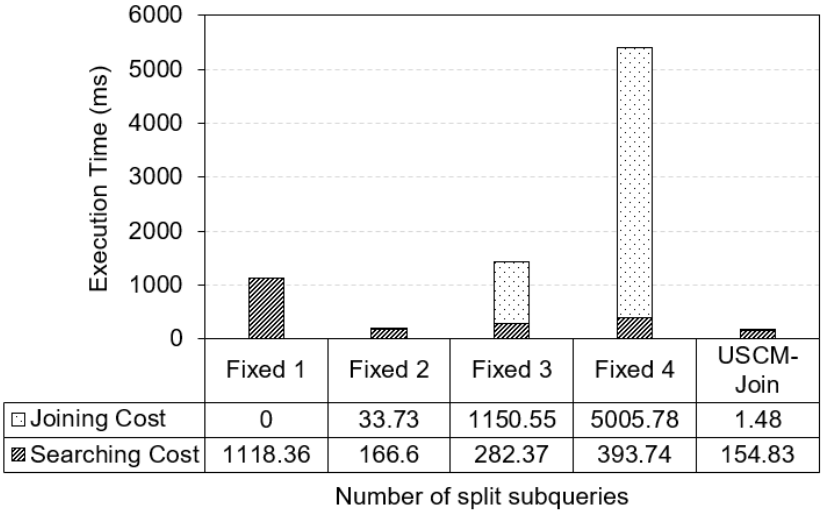

Figure 6. Comparison of response time with varied number of subqueries.

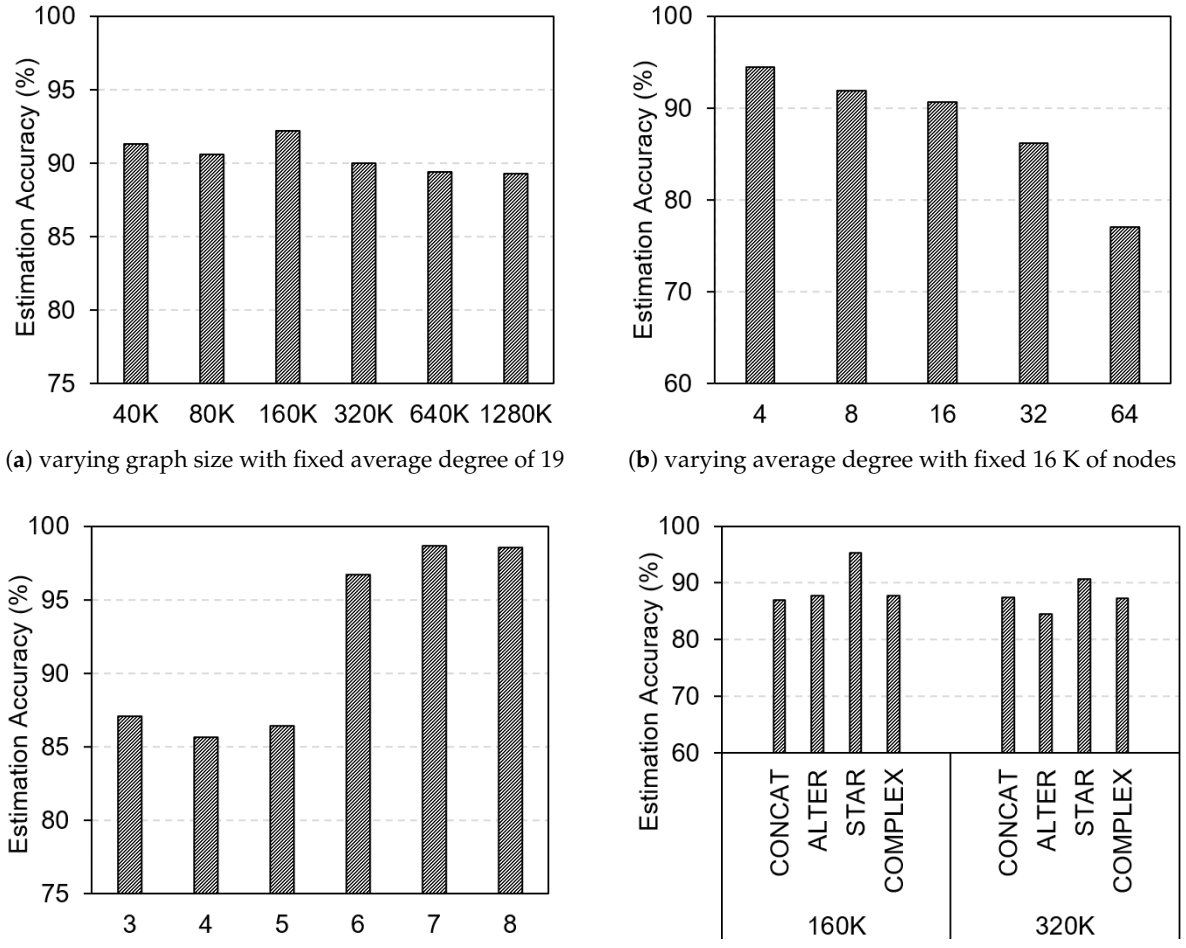

Figure 7. Accuracy of result size estimation with various parameters.

To do this, we defined the accuracy of result size estimation by the closeness between the estimated result size and the actual result size, which is calculated by the fraction C_R^e/C_R^t in the case of $C_R^e \leq C_R^t$, otherwise it is C_R^t/C_R^e, where C_R^e is the average estimated result size and C_R^t is the average actual result size.

Firstly, we evaluate the accuracy of our estimation method on six synthetic graphs with different sizes that vary from 40 K to 1280 K, but the same average degree of 19. The results are illustrated, as in Figure 7a. From the result, we can see that the accuracy that is brought by our estimating proposal is high, which is, at most, 92% with different graphs' size.

Secondly, in order to evaluate the impact of the average degree of graph on the estimation accuracy, we generated five synthetic graphs whose number of nodes is kept at 16 K and number of edges is varied in the range between 64 K and 1024 K. As a result, the average degree of the graphs is varied between 4 and 64, corresponding to the graph size. From the result in Figure 7b, we observed that the accuracy of result size estimation for the graphs having average degree less than or equal to 16, which is mostly around 90%, are higher than those in the cases having an average degree of 32 and 64. Specifically, the estimation accuracy of result size estimation is reduced around 5% and 15% when we increase the average degree of graph from 16 to 32 and 64, respectively.

Thirdly, we evaluate the impact of query path length on the estimation accuracy. We randomly generate 1200 RPQs with only concatenations operator. The length of RPQs (query path length) is varied from 3 to 8. Accordingly, we have six subsets of queries with a different path length, and each has 200 queries. Figure 7c shows that the accuracy of result size estimation slightly decreases from 87% to 85% when we increase the length of query from 3 to 4, but the accuracy is increased if the length of query greater than or equal to 5. Especially, in the case of the length of query being greater than or equal to 6, the accuracy obtained is, at most, 98%. The results are reasonable because almost all of the long queries have no path on the graph satisfying them. Therefore, it is easy to correctly estimate the result size of such queries by exploiting our proposal.

Last but not least, we evaluate the impact of query types (concatenation, alternation, Kleene Star, and complex RPQs) on the accuracy of result size estimation. We use two synthetic graphs: 160 K and 320 K of nodes and edges. For the queries set, we generate 1000 queries, where each type of query has 250 queries. The queries are defined by format: *abc*, *a(b|c)d*, *ab*c*, and *a(b|c)*d* corresponding to four types of query above, respectively; where *a*,*b*,*c*, and *d* are labels in the graph. Figure 7d illustrates the estimation accuracy of result size, in which all types of query have accuracy over 85%, and the estimation accuracy of Kleene Star RPQs is the highest one, around 90%.

In short, from the results shown in Figure 7, we observed that the accuracy of our proposed estimating method is mostly over 85%. Especially, the accuracy is over 90% in the cases of graphs with 16 average degree or less (Figure 7b) or queries with length more than 6 (Figure 7c).

6. Conclusions

This paper proposed a method of estimating the joining cost of subqueries in order to accelerate the USCM based parallel evaluation of regular path queries (RPQs), namely USCM-Join. The proposed method is realized by estimating the result size of subqueries, which are used to estimate the joining cost of the subqueries. Subsequently, the evaluation performance of RPQs is improved by splitting them with a combination of the estimated joining and searching cost. To evaluate the effectiveness of the proposed method, we conducted experiments: (1) comparison of the average response times of RPQs among four methods and (2) the impact of query types (concatenation, alternation, Kleene Star, and complex RPQs) on the accuracy of result size estimation and on our USCM-Join approach. Through the experimental results upon real-world datasets and synthetic datasets, we found that our USCM-Join approach outperformed others and estimating the joining cost enhances the USCM based approach up to around 20% in terms of response time.

Author Contributions: Conceptualization, V.-Q.N. and K.K.; methodology, formal analysis, and writing—original draft preparation, V.-Q.N.; resources, visualization, writing—review, and editing, V.-H.N., Q.-T.H. and M.-Q.N.; supervision, project administration, and funding acquisition, V.-Q.N. and K.K. All authors have read and agreed to the published version of the manuscript.

Funding: 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 the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2021-2016-0-00314) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Libkin, L.; Vrgoč, D. Regular path queries on graphs with data. In Proceedings of the 15th International Conference on Database Theory, Berlin, Germany, 26–28 March 2012; pp. 74–85.
2. Barceló Baeza, P. Querying graph databases. In Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, New York, NY, USA, 22–27 June 2013; pp. 175–188.
3. Yakovets, N.; Godfrey, P.; Gryz, J. Query planning for evaluating SPARQL property paths. In Proceedings of the 2016 International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016; pp. 1875–1889.
4. Scott, J.; Ideker, T.; Karp, R.M.; Sharan, R. Efficient algorithms for detecting signaling pathways in protein interaction networks. *J. Comput. Biol.* **2006**, *13*, 133–144. [CrossRef] [PubMed]
5. Konstas, I.; Stathopoulos, V.; Jose, J.M. On social networks and collaborative recommendation. In Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, Boston, FL, USA, 19–23 July 2009; pp. 195–202.
6. Goldman, R.; Widom, J. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB'97, Athens, Greece, 25–29 August 1997; pp. 436–445.
7. Fernandez, M.; Suciu, D. Optimizing regular path expressions using graph schemas. In Proceedings of the 14th International Conference on Data Engineering, Orlando, FL, USA, 23–27 February 1998; pp. 14–23.
8. Calvanese, D.; De Giacomo, G.; Lenzerini, M.; Vardi, M.Y. Rewriting of regular expressions and regular path queries. In Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Philadelphia, PA, USA, 31 May–2 June 1999; pp. 194–204.
9. Calvanese, D.; De Giacomo, G.; Lenzerini, M.; Vardi, M.Y. Rewriting of regular expressions and regular path queries. *J. Comput. Syst. Sci.* **2002**, *64*, 443–465. [CrossRef]
10. Koschmieder, A.; Leser, U. Regular path queries on large graphs. In *Scientific and Statistical Database Management*; Springer: Chania, Greece, 2012; pp. 177–194.
11. Nguyen, V.Q.; Huynh, Q.T.; Kim, K. Estimating searching cost of regular path queries on large graphs by exploiting unit-subqueries. *J. Heuristics* **2018**. [CrossRef]
12. Nguyen, V.Q.; Nguyen, V.H.; Nguyen, H.-T.; Nguyen Nguyen, M.Q.; Huynh, Q.T.; Kim, K. Accelerating Parallel Evaluation of Regular Path Queries on Large Graphs by Estimating Joining Cost of Subqueries. In Proceedings of the Ninth International Conference on Smart Media and Applications, Jeju Island, Korea, 17–19 September 2020.
13. Pacaci, A.; Bonifati, A.; Özsu, M.T. Regular path query evaluation on streaming graphs. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, Portland, OR, USA, 14–19 June 2020; pp. 1415–1430.
14. Wadhwa, S.; Prasad, A.; Ranu, S.; Bagchi, A.; Bedathur, S. Efficiently answering regular simple path queries on large labeled networks. In Proceedings of the 2019 International Conference on Management of Data, Hong Kong, China, 10–13 June 2019; pp. 1463–1480.
15. Trißl, S. Cost-based optimization of graph queries. In Proceedings of the SIGMOD/PODS PhD Workshop on Innovative Database Research (IDAR), Beijing, China, 10 June 2007.
16. Grahne, G.; Thomo, A. Query containment and rewriting using views for regular path queries under constraints. In Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, San Diego, CA, USA, 9–12 June 2003; pp. 111–122.
17. Liu, T.; Liu, A.X.; Shi, J.; Sun, Y.; Guo, L. Towards fast and optimal grouping of regular expressions via DFA size estimation. *IEEE J. Sel. Areas Commun.* **2014**, *32*, 1797–1809. [CrossRef]
18. Almeida, J.; Zeitoun, M. Description and analysis of a bottom-up DFA minimization algorithm. *Inf. Process. Lett.* **2008**, *107*, 52–59. [CrossRef]
19. Liu, D.; Huang, Z.; Zhang, Y.; Guo, X.; Su, S. Efficient Deterministic Finite Automata Minimization Based on Backward Depth Information. *PLoS ONE* **2016**, *11*, e0165864.
20. Kossmann, D. The state of the art in distributed query processing. *ACM Comput. Surv. (CSUR)* **2000**, *32*, 422–469. [CrossRef]
21. Suciu, D. Distributed query evaluation on semistructured data. *ACM Trans. Database Syst. (TODS)* **2002**, *27*, 1–62. [CrossRef]

22. Fan, W.; Wang, X.; Wu, Y. Performance guarantees for distributed reachability queries. *Proc. VLDB Endow.* **2012**, *5*, 1304–1316. [CrossRef]
23. Nguyen, V.Q.; Tung, L.D.; Hu, Z. Minimizing data transfers for regular reachability queries on distributed graphs. In Proceedings of the Fourth Symposium on Information and Communication Technology, Da Nang, Vietnam, 5–6 December 2013; pp. 325–334.
24. Tung, L.D.; Nguyen, V.Q.; Hu, Z. Efficient query evaluation on distributed graphs with Hadoop environment. In Proceedings of the Fourth Symposium on Information and Communication Technology, Da Nang, Vietnam, 5–6 December 2013; pp. 311–319.
25. Martens, W.; Trautner, T. Evaluation and Enumeration Problems for Regular Path Queries. In Proceedings of the 21st International Conference on Database Theory (ICDT 2018), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Vienna, Austria, 26–29 March 2018.
26. Abul-Basher, Z.; Yakovets, N.; Godfrey, P.; Ghajar-Khosravi, S.; Chignell, M.H. TASWEET: Optimizing disjunctive regular path queries in graph databases. In Proceedings of the EDBT/ICDT 2017 Joint Conference 20th International Conference on Extending Database Technology, Venice, Italy, 21–24 March 2017.
27. Fletcher, G.H.; Peters, J.; Poulovassilis, A. Efficient regular path query evaluation using path indexes. In Proceedings of the 19th International Conference on Extending Database Technology, Bordeaux, France, 15–16 March 2016; pp. 636–639. [CrossRef]
28. Trißl, S.; Leser, U. Estimating Result Size and Execution Times for Graph Queries. In Proceedings of the ADBIS (Local Proceedings), Novi Sad, Serbia, 20–24 September 2010; pp. 11–20.
29. Davoust, A.; Esfandiari, B. Processing Regular Path Queries on Arbitrarily Distributed Data. In *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*; Springer: Rhodes, Greece, 2016, pp. 844–861.
30. Afrati, F.N.; Ullman, J.D. Optimizing multiway joins in a map-reduce environment. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 1282–1298. [CrossRef]
31. Wu, M.; Berti-Equille, L.; Marian, A.; Procopiuc, C.M.; Srivastava, D. Processing top-k join queries. *Proc. VLDB Endow.* **2010**, *3*, 860–870. [CrossRef]
32. Suchanek, F.M.; Kasneci, G.; Weikum, G. Yago: A core of semantic knowledge. In Proceedings of the 16th International Conference on World Wide Web, Banff, AB, Canada, 8–12 May 2007; pp. 697–706.
33. Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; Taylor, J. Freebase: A collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, Vancouver, BC, Canada, 9–12 June 2008; pp. 1247–1250.
34. Mahdisoltani, F.; Biega, J.; Suchanek, F.M. Yago3: A knowledge base from multilingual wikipedias. In Proceedings of the CIDR, Asilomar, CA, USA, 6–9 January 2013.
35. Bast, H.; Bäurle, F.; Buchhold, B.; Haußmann, E. Easy access to the freebase dataset. In Proceedings of the 23rd International Conference on World Wide Web, Seoul, Korea, 7–11 April 2014; pp. 95–98.
36. Zahiri, J.; Hannon Bozorgmehr, J.; Masoudi-Nejad, A. Computational prediction of protein–protein interaction networks: Algorithms and resources. *Curr. Genom.* **2013**, *14*, 397–414. [CrossRef] [PubMed]
37. Bagan, G.; Bonifati, A.; Ciucanu, R.; Fletcher, G.H.; Lemay, A.; Advokaat, N. gMark: Schema-driven generation of graphs and queries. *IEEE Trans. Knowl. Data Eng.* **2017**, *29*, 856–869. [CrossRef]
38. Nguyen, V.Q.; Bui, T.X.L.; Nguyen, V.H. An efficient graph modeling approach for storing and analyzing heterogeneous IoT data. *UTEHY J. Sci. Technol.* **2020**, *27*, 21–27.
39. Bastian, M.; Heymann, S.; Jacomy, M. Gephi: An open source software for exploring and manipulating networks. *ICWSM* **2009**, *8*, 361–362.