IEICE TRANS. INF. & SYST., VOL.E100–D, NO.10 OCTOBER 2017

LETTER

Efficient Regular Path Query Evaluation by Splitting with Unit-Subquery Cost Matrix

Van-Quyet NGUYEN[†a)], ***Nonmember and*** **Kyungbaek KIM**[†b)], ***Member***

SUMMARY A widely-used query on a graph is a regular path query (RPQ) whose answer is a set of tuples of nodes connected by paths corresponding to a given regular expression. Traditionally, evaluating an RPQ on a large graph takes substantial memory spaces and long response time. Recently, several studies have focused on improving response time for evaluating an RPQ by splitting an original RPQ into smaller subqueries, evaluating them in parallel and combining partial answers. In these works, how to choose split labels in an RPQ is one of key points of the performance of RPQ evaluation, and rare labels of a graph can be used as split labels. However there is still a room for improvement, because a rare label cannot guarantee the minimum evaluation cost all the time. In this paper, we propose a novel approach of selecting split labels by estimating evaluation cost of each split subquery with a *unit-subquery cost matrix* (USCM), which can be obtained from a graph in prior to evaluate an RPQ. USCM presents the evaluation cost of a unit-subquery which is the smallest possible subquery, and we can estimate the evaluation cost of an RPQ by decomposing into a set of unit-subqueries. Experimental results show that our proposed approach outperforms rare label based approaches.

key words: *regular path queries, large graphs, graph querying*

1. Introduction

A regular path query (RPQ) is first introduced as part of a query language for graph databases, which are represented as graphs in which nodes are objects and edge labels specify relationships between them [1]. The answer of an RPQ is a set of tuples of nodes that are connected with edge labels in some ways by the paths specified by a regular language. There are a lot of applications using RPQs such as friends recommendations in social networks [2] and detecting signal pathways in protein interaction networks [3]. However, evaluating an RPQ on such large graphs takes substantial memory spaces and long response time. Therefore, in this paper, we focus on finding an efficient regular path queries evaluation on large graphs.

The most common approach for evaluating an RPQ is based on automata. A graph needs to be translated into a NFA (Nondeterministic Finite Automaton), and a regular expression of an RPQ can be converted into an automaton before using it to match paths [4]. However, most modern graph databases contain a huge number of nodes and edges, which leads that the automata based method takes substantial memory spaces and long response time.

Recently, an approach for evaluating an RPQ efficiently on large graphs has been studied [5]. In this approach, an RPQ is decomposed into multiple subqueries by using rare labels which are rarely appeared in a graph, and each subquery is evaluated independently in parallel fashion and the independent partial results are combined into the final answer of an RPQ. Because each subquery can be evaluated in parallel, the evaluation cost, especially the response time, can be reduced.

The rare label based approach adopts the idea that a subquery starting from a rare label takes less response time. However, some subqueries split by rare labels cannot reduce response time if a node with many neighbors is connected to a rare label edge. It is because that the searching space of a subquery split by a rare label is expanded by these many neighbors of the nodes connected to the rare label.

In this paper, we propose a novel approach of selecting split labels for an RPQ by estimating the searching cost of each split subquery with a *unit-subquery cost matrix* (USCM). By choosing split labels which minimize the estimated graph searching cost of split subqueries, the proposed approach can reduce more response time for evaluating a RPQ on large graphs than rare label based approaches.

2. Rationale of the Proposed Approach

An RPQ is evaluated on an edge-labeled directed graph $G = (V, E, \Sigma)$, where V is a finite set of nodes, Σ is a finite set of labels, and $E \subseteq V \times \Sigma \times V$ is a finite set of edges. An edge (v,a,u) denotes a directed edge from node v to u labeled with $a \in \Sigma$. A path ρ between nodes v_0 and v_k in G is a sequence $v_0a_0v_1a_1v_2 \ldots v_{k-1}a_{k-1}v_k$ 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 $L(\rho)$, is the string $a_0a_1 \ldots a_{k-1} \in \Sigma^*$. We also define the *empty* path as (v, ε, v) for each $v \in V$; the label of such a path is the empty string ϵ. An RPQ with a regular expression R is a query of the form $Q(R) = v \xrightarrow{L(R)} u$, where $L(R) \in \Sigma^*$ is a regular language. So, a path ρ satisfies $Q(R)$ on the graph G iff $L(\rho) \in L(R)$, then ρ is the answer of $Q(R)$.

Let us assume that we have an edge-labeled directed graph G and an RPQ with a regular expression $R = abc(d|c)e$ as shown in Fig. 1. We can define the evaluation cost of an RPQ as the number of traversed edges for searching paths corresponding to the RPQ.

For an automata based approach, there are three edges

Manuscript received March 12, 2017.
Manuscript revised May 24, 2017.
Manuscript publicized July 12, 2017.

[†]The authors are with the Department of Electronics and Computer Engineering, Chonnam National University, Gwangju, Korea.
a) E-mail: quyetict@utehy.edu.vn
b) E-mail: kyungbaekkim@jnu.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2017EDL8060

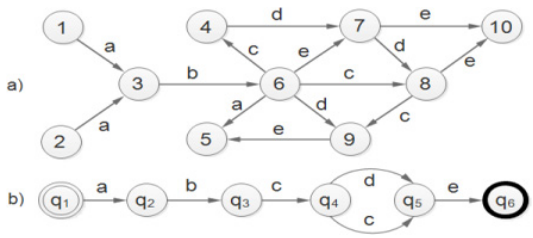

Fig. 1 a) An example of a directed edge-labeled graph; b) a regular path query as an automaton

which have a label and these edges are used as the starting edges of searching the paths. From these edges, we find the nodes for next searching step such as node 3 and node 5. Then, searching process continues from these nodes until no path can be found in the search process. Here, the evaluation cost of automata based approach is fifteen.

For a rare label based approach, because b is only found in the edge (3,b,6), the rarest label b can be used to split an RPQ $Q(abc(d|c)e)$ into two subqueries, $Q(ab)$ and $Q(bc(d|c)e)$. For the subquery $Q(ab)$, there are three starting edges and only one more edges is traversed for searching. For the subquery $Q(bc(d|c)e)$, there is one starting edge and eleven more edges are traversed. If we assume that these two subqueries are evaluated in parallel, the evaluation cost becomes twelve which is the maximum evaluation cost among split subqueries.

In this case, if we use label c as a split label, we may reduce the evaluation cost. By using c as a split label, $Q(abc(d|c)e)$ is split into $Q(abc)$ and $Q(c(d|c)e)$. For the subquery $Q(abc)$, there are three staring edges and six edges are traversed. For the subquery $Q(c(d|c)e)$, there are three starting edges and seven edges are traversed. That is, the evaluation cost can be reduced from twelve to ten if we use the label c as a split label rather than b.

The key observation is that the evaluation cost depends on not only the number of appearance of a label (i.e. rare label) but also the number of outgoing/incoming nodes related to a label.

3. Evaluating RPQ with USCM Based Split

3.1 Unit-Subquery Cost Matrix (USCM)

Intuitively, an RPQ is composed of multiple small subqueries with a few operators such as concatenation, alternation, and Kleene star. Then, we can define a *unit-subquery* as the smallest subquery which concatenated by two labels from Σ; the start label and the end label. For example, in the graph G of Fig. 1, a subquery $Q(ab)$ is a unit-subquery, where a is the start label and b is the end label. In practice, any subquery can be split into multiple unit-subqueries except the subquery with a Kleene star operator. For example, the subquery $Q((a|b)c)$ can be split into two unit-subqueries $Q(ac)$ and $Q(bc)$; meanwhile, the subquery $Q(ab^*c)$ can not split into $Q(ab)$ and $Q(bc)$. Thus, we define a Kleene star unit-subquery as the smallest subquery which concatenated by two labels from Σ; the start label and the end label along with a Kleene star (i.e. $Q(ab^*)$). Now, the subquery $Q(ab^*c)$ can be split into a Kleene star unit-subquery $Q(ab^*)$ and a unit-subquery $Q(bc)$.

Table 1 An example of unit-subquery cost matrix

Label:Count	a	b	c	d	e	Total
a:3	0;1;0	1;1;1	0;1;0	0;1;0	0;1;0	1
b:1	1;5;1	0;5;0	2;9;3	1;6;1	1;7;2	5
c:3	0;4;0	0;4;0	1;5;1	1;8;2	2;4;2	4
d:3	0;5;0	0;5;0	1;6;1	1;7;1	3;5;3	5
e:4	0;2;0	0;2;0	0;2;0	1;4;1	1;2;1	2

The cost of a unit-subquery is defined as the number of edges with the end label, which is connected to the edges with the start label. For example, in the graph G of Fig. 1, the cost of a subquery $Q(ab)$ is one, because there is only one edge $(3, b, 6)$ labeled with b, which is connected to unidirectional edges labeled with a. For a Kleene star unit-subquery, the cost is defined as the total number of edges which can be traversed during searching the end label with a Kleene star (*) from every start label. For example, the cost of $Q(bc^*)$ in the graph G of Fig. 1 is nine.

With the definition of the cost of unit-subqueries, we can generate a *Unit-Subquery Cost Matrix (USCM)* which represents the cost of all possible unit-subqueries from Σ as well as the cost of the Kleene star unit-subqueries. An example of USCM is shown in Table 1. The size of USCM is n by $n+1$ where n is the number of distinct labels in Σ. A cell (i, j) of USCM, except the last column where j is $n+1$, represents three value separated by a semicolon: the first value is *the cost of a unit-subquery*, $Q(a_i a_j)$, whose start label is $a_i \in \Sigma$ and end label is a_j; the second value is *the cost of a Kleene star unit-subquery*, $Q(a_i a_j^*)$; and the third value is *the number of edges labeled with* a_j which can be found from $Q(a_i a_j^*)$. In the last column, a cell (i, j) represents the cost of a unit-subquery $Q(a_i.)$, that is, the summation of the costs of unit-subqueries whose start label is a_i. Additionally, USCM contains the number of edges with a given label (Count) like the first column of USCM.

3.2 Estimation of Evaluation Cost for an RPQ

3.2.1 A String RPQ with Concatenation

In our approach, the evaluation cost of an RPQ is estimated by splitting the original RPQ into multiple unit-subqueries and gathering the cost of each successive unit-subqueries.

Let us assume that there is an RPQ, $Q(R)$, where $R = a_0 a_1 \dots a_n$ as a string that is concatenated by $(n+1)$ labels $a_i \in \Sigma$. Then, $Q(R)$ can be split into $Q(a_0 a_1), Q(a_1 a_2), \dots, Q(a_{n-1} a_n)$, and the evaluation cost for $Q(R)$ is defined as summation of cost of each successive unit-subquery like Eq. (1).

$$C_{Q(R)} = \sum_{i=0}^{n-1} C_{Q(a_i a_{i+1})} = \sum_{i=0}^{n-1} C_i \tag{1}$$

Algorithm 1 *GetBestSetSubquery* with the given k

Require: A regular expression R, a number of subqueries k, U as an USCM of graph G
Ensure: A set of k subqueries $R_1, R_2, \ldots, R_k$
1: bestSetSubquery $\leftarrow \emptyset$; */*list best subqueries with minimum cost*/*
2: minGlobalCost $\leftarrow \infty$;
3: setSubqueries [] $\leftarrow$ GetAllSolutions(R, k);
4: **for each** setSubquery $\in$ setSubQueries **do**
5: maxLocalCost $\leftarrow 0$;
6: **for each** $sb \in$ setSubquery **do**
7: subqueryCost $\leftarrow$ EstimateCost(sb, U);
8: **if** subqueryCost > maxLocalCost **then**
9: maxLocalCost $\leftarrow$ subqueryCost;
10: **if** maxLocalCost < minGlobalCost **then**
11: minGlobalCost $\leftarrow$ maxLocalCost;
12: bestSetSubquery $\leftarrow$ setSubQuery;
13: **return** bestSetSubquery;

For C_0, the evaluation starts from the edge labeled with a_0 and tries to find the path to a_1. Accordingly, C_0 composed of the cost of finding the next search nodes and the cost of searching a_1. That is, $C_0 = \delta(a_0) + \xi(a_0)$, where $\delta(a_i)$ is the number of edges given label a_i which is the Count value for the first column of USCM and $\xi(a_i)$ is the cost of $Q(a_i.)$ which is the value of the last column of USCM.

For C_i, where $i > 0$, we do not consider the searching cost for finding edges with label a_i, because this cost is already considered in the previous step C_{i-1}. So, for C_i, we only consider the searching cost for finding edges with label a_{i+1}. However, this cost is affected by the number of search nodes which are found in the previous step. To consider this effect, we can calculate the probability of how many edges labeled with a_i related to the unit subquery $Q(a_{i-1}a_i)$ are found among the all the edges labeled with a_i, and apply to the searching cost to edges labeled with a_{i+1} from edges labeled with a_i. That is, C_i, where $i > 0$, can be represented like Eq. (2), where $\mu(a_{i-1}, a_i)$ is the cost of unit-subquery $Q(a_{i-1}a_i)$, which is the first value of each cell of USCM.

$$C_i = \frac{\mu(a_0, a_1)}{\delta(a_1)} \times \cdots \times \frac{\mu(a_{i-1}, a_i)}{\delta(a_i)} \times \xi(a_i) \tag{2}$$

3.2.2 An RPQ with Alternation Operator

We assume that an RPQ, $Q(R)$, is defined by a regular expressions $R = a_0a_1 \ldots a_{k-1}(a_k|a_{k+1})a_{k+2} \ldots a_n$, where $a_i \in \Sigma$. Herein, R has an alternation operator between a_k and a_{k+1}. In this case, the original Q(R) can be split into three subqueries $Q(a_0 \ldots a_{k-1})$, $Q(a_{k-1}(a_k|a_{k+1})a_{k+2})$, and $Q(a_{k+2} \ldots a_n)$. For the subqueries $Q(a_0 \ldots a_{k-1})$ and $Q(a_{k+2} \ldots a_n)$, we can estimate their cost by using our method in Sect. 3.2.1.

For evaluating $Q(a_{k-1}(a_k|a_{k+1})a_{k+2})$, we need to consider two different steps; $Q(a_{k-1}(a_k|a_{k+1}))$ and $Q((a_k|a_{k+1})a_{k+2})$. Firstly, $Q(a_{k-1}(a_k|a_{k+1}))$ can be considered into two subqueries $Q(a_{k-1}a_k)$ and $Q(a_{k-1}a_{k+1})$. In here, evaluating the both of subqueries starts from edges labeled with a_{k-1}, and during one time evaluation we can traverse the edges labeled with a_k as well as a_{k+1}. So, the cost of $Q(a_{k-1}(a_k|a_{k+1}))$ can be estimated by the cost of either $Q(a_{k-1}a_k)$ or $Q(a_{k-1}a_{k+1})$. On the other hands, $Q((a_k|a_{k+1})a_{k+2})$ can be decomposed into $Q(a_ka_{k+2})$ and $Q(a_{k+1}a_{k+2})$, and the evaluation process of these subqueries are different to each other. So, C_A, the estimated cost of $Q(a_{k-1}(a_k|a_{k+1})a_{k+2})$ can be estimated by the summation of the costs of $Q(a_{k-1}a_k)$, $Q(a_ka_{k+2})$, and $Q(a_{k+1}a_{k+2})$ like Eq. (3).

$$\begin{aligned} C_A &= C_{Q(a_{k-1}a_k)} + C_{Q(a_ka_{k+2})} + C_{Q(a_{k+1}a_{k+2})} \\ &= \frac{\mu(a_0, a_1)}{\delta(a_1)} \times \cdots \times \frac{\mu(a_{k-2}, a_{k-1})}{\delta(a_{k-1})} \\ &\times \left(\xi(a_{k-1}) + \frac{\mu(a_{k-1}, a_k)}{\delta(a_k)}\xi(a_k) + \frac{\mu(a_{k-1}, a_{k+1})}{\delta(a_{k+1})}\xi(a_{k+1})\right) \end{aligned} \tag{3}$$

3.2.3 An RPQ with Kleene Star Operator

Let us assume that there is an RPQ, $Q(R)$, where $R = a_0a_1 \ldots a_{k-1}a_k^*a_{k+1} \ldots a_n$ with a Kleene star operator. To estimate the cost of $Q(R)$, we can split this query into three subqueries including $Q(a_0 \ldots a_{k-1})$, $Q(a_{k-1}a_k^*a_{k+1})$, and $Q(a_{k+1} \ldots a_n)$, then the evaluation cost for $Q(R)$ is defined as summation of cost of each subquery. We can estimate the costs of the subqueries $Q(a_0 \ldots a_{k-1})$ and $Q(a_{k+1} \ldots a_n)$ by using the proposed method described in Sect. 3.2.1. The subquery $Q(a_{k-1}a_k^*a_{k+1})$ can split into $Q(a_{k-1}a_k^*)$ (a Kleene star unit-subquery) and $Q(a_ka_{k+1})$ as we described in Sect. 3.1. So, the estimated cost of $Q(a_{k-1}a_k^*a_{k+1})$, C_K, is defined as summation of the estimated cost of $Q(a_{k-1}a_k^*)$ and $Q(a_ka_{k+1})$, as shown in Eq. (4).

$$\begin{aligned} C_K &= \frac{\mu(a_0, a_1)}{\delta(a_1)} \times \cdots \times \frac{\mu(a_{k-2}, a_{k-1})}{\delta(a_{k-1})} \\ &\times \left(\varphi(a_{k-1}, a_k^*) + \frac{\mu'(a_{k-1}, a_k)}{\delta(a_k)}\xi(a_k)\right) \end{aligned} \tag{4}$$

where $\varphi(a_{k-1}, a_k^*)$ is the cost of a Kleene star unit-subquery $Q(a_{k-1}a_k^*)$, which is the second value in the corresponding cell of USCM; $\mu'(a_{k-1}, a_k)$ is the number of the edges labeled with a_k found in all possible results of $Q(a_{k-1}a_k^*)$, which is the third value in the corresponding cell of USCM.

Note that our approach can also be used for other modifiers (+ and ?). That is, the estimated cost of a unit-subquery $Q(a_{k-1}a_k+)$ equals to the estimated cost of $Q(a_{k-1}a_ka_k^*)$, while the estimated cost of $Q(a_{k-1}a_k?)$ equals to the estimated cost of $Q(a_{k-1}a_k)$.

3.3 Splitting Query Based on Cost Estimating

If there are k CPUs to evaluate an RPQ, we can split the RPQ into k subqueries to minimize the evaluation cost by estimating the evaluation cost of subqueries with USCM, as shown in Algorithm 1. The proposed algorithm consists of three main steps: (1) Find all possible sets of k subqueries (lines 1-3); (2) For each set of k subqueries, estimate the

evaluation cost with USCM to find the local maximum of the evaluation cost (line 4-9); (3) Compare the evaluation cost of each set of k subqueries and find the set of k subqueries with the minimum of the evaluation cost (line 10-13).

In Algorithm 1, procedure *GetAllSolutions* (line 3) returns all of the possible sets of k subqueries. That is, it finds all of the possible combination of sequenced labels, and this procedure takes polynomial time. Here, we consider only the labels as the split labels if it is not at the position of the labels with Kleene star operator or inside a bracket of an alternation operator.

3.4 Handling Highly Complex RPQs

Our approach is not only effective with simple RPQs but also some levels of complexity. To estimate the cost of a complex RPQ, we need to represent an RPQ into language equivalent deterministic finite-state automata (DFA), then find out sub-queries which can be used to estimate the cost. First, USCM can be used to estimate complex RPQs which contain modifiers on groups of alternate expressions. For example, an RPQ, Q(R), with $R = a(b|c)^*d$ can be estimated by $C_{Q(R)} = C_0 + C_1 + C_2$, where $C_0 = \delta(a) + \xi(a)$; $C_1 = C_{Q(bb^*)} + C_{Q(bc^*)} + C_{Q(cc^*)} + C_{Q(cb^*)}$; and $C_2 = C_{Q(bd)} + C_{Q(cd)}$. The right-hand side of C_0, C_1, and C_2 can be estimated by using our method in Sect. 3.2. Second, in case of evaluating highly complex RPQs, for instance, nested RPQs which contain a group expression inside another one, our method can evaluate the nested RPQs without recursive modifiers (e.g. Q(R) with $R = ((a(b|c)^*d)|e)fg$) in optimized cost. In particular, evaluating a nested RPQ with recursive modifiers (e.g. Q(R) with $R = (a(b|c)^*de)^*fg$) on a large graph will take very high cost. Therefore, it does not take advantages of splitting query for evaluating in parallel fashion because of unbalancing evaluation cost among sub-queries. For such query, our approach does not split the query, but it is evaluated as a single query by using automata-based approach. It is the same strategy of previous related works on splitting RPQ [5]. Thus, the optimization for evaluating highly complex RPQs with recursive modifiers is outside the scope of this paper.

4. Evaluation

To evaluate the effectiveness of our proposed approach (USCM), we compare the proposed approach to the automata-based approach (AUT) [4] and the threshold-rare label based approach (TRL) [5], in the aspect of the average response time for answering RPQs on large graphs. We also compare the proposed approach to K-rare label based approach (KRL), which chooses *K-1* rarest labels among the labels of an RPQ.

In our experiments, we have used the graph and the queries set given by previous research [5]. The graph is a network of protein-protein interactions which is used regularly in systems biology, for instance, to discover of protein functions and pathways in biological processes [6]. This

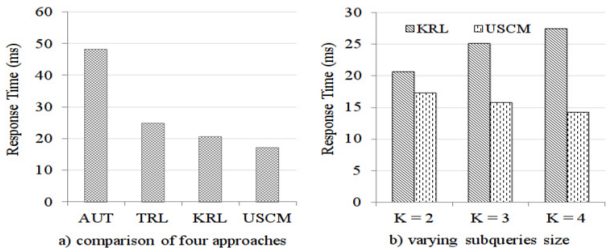

Fig. 2 Performance evaluation

graph has 52.050 nodes, 340.775 edges, and 649 labels. We analyzed 10.000 queries in the queries set and found the following properties. The queries set has around 87% proportion of having simple RPQs, 3% proportion of having nested RPQs without recursive modifiers, and 10% proportion of having nested RPQs with recursive modifiers.

Figure 2a illustrated the average response times of four different approaches. For KRL and USCM, the number of subqueries, K, is set to 2 for both graphs, and the threshold value sets to 12 for TRL. We observed that approaches of splitting an RPQ outperform AUT. That is, splitting an RPQ is necessary to reduce the evaluation cost of an RPQ. We also observed that USCM reduced the average response time around 45% and 20% than TRL and KRL respectively. To observe the difference between KRL and USCM in detail, we compared the average response time of evaluating RPQs with different the number of subqueries. As shown in Fig. 2b, we observed that USCM is around two times faster than KRL for 4 subqueries. Especially, our method is obtained better performance when the number of split subqueries is increased, while the performance of KRL is decreased over the number of split subqueries. This result proved the effectiveness of estimating evaluation cost of each spit subquery with USCM.

5. Conclusion

We proposed a novel approach of splitting an RPQ into subqueries by estimating evaluation cost of each possible subquery with a unit-subquery cost matrix (USCM). By minimizing the estimated evaluation cost of separated subqueries, our USCM based approach can reduce more response time for evaluating RPQs on a large graph than previous approaches.

Acknowledgments

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2016-0-00314) supervised by the IITP (Institute for Information & communications Technology Promotion). 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

[1] A.O. Mendelzon and P.T. Wood, "Finding regular simple paths in graph databases," SIAM Journal on Computing, vol.24, no.6, pp.1235–1258, 1995.

[2] I. Konstas, V. Stathopoulos, and J.M. Jose, "On social networks and collaborative recommendation," Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, pp.195–202, ACM, 2009.

[3] J. Scott, T. Ideker, R.M. Karp, and R. Sharan, "Efficient algorithms for detecting signaling pathways in protein interaction networks," Journal of Computational Biology, vol.13, no.2, pp.133–144, 2006.

[4] R. Goldman and J. Widom, "Dataguides: Enabling query formulation and optimization in semistructured databases," Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97), pp.436–445, Aug. 1997.

[5] A. Koschmieder and U. Leser, "Regular path queries on large graphs," Scientific and Statistical Database Management, vol.7338, pp.177–194, Springer, 2012.

[6] J. Zahiri, J.H. Bozorgmehr, and A. Masoudi-Nejad, "Computational prediction of protein–protein interaction networks: algorithms and resources," Current genomics, vol.14, no.6, pp.397–414, 2013.