PRS: Parallel Relaxation Simulation for Massive Graphs

Jianliang Gao, Ping Liu, Xuedan Kang, Lixia Zhang and Jianxin Wang*

School of Information Science and Engineering, Central South University, Changsha, China
*Corresponding author: jswang@csu.edu.cn

Graph pattern matching is becoming important for a variety of emerging applications such as social network analysis. Graph pattern matching is traditionally defined in terms of subgraph isomorphism or graph simulation. These notions, however, often impose too strong constraint to identify meaningful matches. In this paper, we propose a new graph pattern matching based on a notion of relaxation simulation, which extends graph simulation by allowing partially absent vertices. We show that relaxation simulation is able to find significant matches which traditional approaches of graph pattern matching fail to catch. We propose two parallel algorithms to apply relaxation simulation to massive graph since the graph in practice is considerably large. The algorithms are based on Bulk Synchronous Parallel model and can be easily deployed on cloud computing platforms. Finally, we experimentally verify the effectiveness and efficiency of these algorithms, using real-life data and synthesis data. The results suggest that relaxation simulation is a promising framework for real-life massive graph analysis.

Keywords: big data; graph pattern matching; social networks; cloud computing; bulk synchronous parallel

Received 26 January 2014; revised 25 November 2014
Handling editor: Surya Nepal

1. INTRODUCTION

With the rapid development of the Internet, data volumes in many domains have grown dramatically, including social networks, biological networks and World-Wide Web. Such data are often represented as graphs, where nodes are labeled entities and edges represent relations among these entities. To mine the large graph data, graph query is usually used for community discovery and neighbor query in social networks, biological data analysis [1], classification of web documents [2], software plagiarism [3] etc. Graph query is essential for a wide range of emerging applications and has been a research focus in industrial and academic fields.

Various types of queries have been extensively investigated in the past, such as shortest path query, reachability query and pattern matching query. However, the large volume data such as social network data have brought new challenges to the traditional operations. An important issue of graph pattern matching in emerging applications concerns how to efficiently deal with large graphs. For instance, Facebook has more than 1.06 billion users (nodes) with 150 billion links (edges) [4]. The massive graph data are stored in a distributive manner. Therefore, parallel techniques become increasingly important for large graph data analysis.

Graph pattern matching is one of the most widely used operations for graph data analysis. Graph pattern matching tries to find subgraphs of a large data graph that are the same or similar to a given query graph (called pattern graph). Graph pattern matching can be further divided as subgraph isomorphism [5] and graph simulation [6, 7]. Subgraph isomorphism seeks to find the subgraphs that are exactly isomorphic to the query graph. Since subgraph isomorphism is NP-complete and often too restrictive to find meaningful matches in some applications [8], graph simulation has been adopted for graph pattern matching.

Many applications such as social networks and biological networks do not need exact matches. To find out more meaningful matches, some proposals (such as [9, 10]) revised the basic graph simulation by applying new requirements on the match conditions. For instance, bounded simulation [9] extends graph simulation by allowing bounds on the number of hops. However, traditional graph pattern matching methods, including extended graph simulations with additional
Figure 1. An application of graph pattern matching. (a) The relationship among eight roles (a given pattern graph), then we need to find out the feasible matches from data graph. (b) A possible matched subgraph, in which the role BA and DM are replaced by PM and GM, respectively.

conditions, do not allow partially absent nodes in the incomplete matches. In practice, this kind of match is also very significant. For example, some applications of social network analysis need to find out the target team in which partial roles can be absent. The absent roles can be replaced by its direct leaders (parent nodes in graph) and persons under its leadership (child nodes in graph).

Figure 1 shows an example in social networks which allows some absent roles. This example can find out a feasible team to develop a software product, in which partial roles can be absent and replaced by other roles. Figure 1a shows an ideal software development team with the following roles: general manager (GM), project manager (PM), development manager (DM), business analyst (BA), software architect (SA), user interface designer (UD), program developer (PD) and software tester (ST). Node denotes a person labeled with his/her expertise and an edge indicates the cooperative relationship between two persons. Figure 1b shows a possible match, in which nodes BA, DM is missed. Though DM is missed, it can be regarded in real life that its predecessor GM can take the place of DM. That is to say, the task can be executed with the successor set of SD, PD, ST with the cooperation of GM. It is the same for PM to take the place of BA.

As can be seen from the example, pattern matching of allowing the absence of some nodes is meaningful in real-life applications. Tian et al. [11] presented a technique for approximate matching allowing missing of nodes. However, they aimed to execute query between two graphs with similar size. Their approach is a centralized solution and cannot scale to big graphs. Moreover, the big graph data usually are stored in a distributive manner [12]. Therefore, developing parallel algorithms is critical for the emerging applications of big data.

This paper investigates a new graph pattern matching problem named relaxation simulation, to find significant matches in emerging applications. And we design a parallel treatment of relaxation simulation when the graph is massive. The main contributions of this paper are summarized as follows:

(i) We propose a notion of relaxation simulation, an extension of graph pattern matching. Relaxation simulation allows the absence of child vertices and conducts significant results for emerging applications such as social network analysis.

(ii) We design parallel algorithms, named unidirectional relaxation simulation (URS) and dual relaxation simulation (DRS), respectively. They are efficient vertex-centric distributed algorithms for graph pattern matching, which can be easily parallelized on cloud platforms.

(iii) We present a detailed experimental study with real-life datasets as well as synthetic data. We experimentally verify the effectiveness and scalability of our matching algorithms.

The rest of the paper is organized as follows. We introduce the preliminary concepts and definitions about graph pattern matching in Section 2. We then present relaxation simulation problem including URS and DRS in Section 3. The corresponding distributed algorithms for URS and DRS are given in Section 4 and 5, respectively. Section 6 presents the experimental study. Related work is shown in Section 7. Finally, Section 8 concludes this paper.

2. PRELIMINARIES

We first define data graphs and pattern graphs and then show graph pattern matching.

Data graph. A data graph is a directed graph $G = (V, E, f)$, where (1) $V$ is a finite set of nodes; (2) $E \subseteq V \times V$, in which $(v, v')$ denotes an edge from node $v$ to $v'$; and (3) $f(\cdot)$ is a function on $V$ such that for each node $v$ in $V$, $f(v) = a$ where $a$ is a constant of the attribute. In this paper, the attribute of a node carries the label of the node (adapted from [13]).

Pattern graph. A pattern graph is a directed graph $P = (V_p, E_p, f_p)$, where (1) $V_p$ and $E_p$ are the set of nodes and the set of directed edges, respectively, as defined for data graphs; (2) $f_p$ is a function defined on $V_p$ such that for each node $u$. Intuitively, the predicate $f_p(u)$ of a node $u$ specifies a search condition on labels and data contents. In this paper, our search
In vertex-centric pattern matching, each vertex of the data graph is a computing unit in BSP model. It provides flexible implementation of iterative operations and is of high scalability. Therefore, we also design our algorithms based on BSP model in this paper.

3. RELAXATION SIMULATION PROBLEM

In this section, we introduce the notion of relaxation simulation. First, we consider only child nodes (URS). Then we take both parent nodes and child nodes into consideration (DRS).

**URS.** Consider a data graph $G = (V, E, f)$ and a pattern graph $P = (V_p, E_p, f_p)$. We say that the graph $G$ matches the pattern $P$ via URS, denoted by $P \preceq_{\text{ursim}} G$, if and only if there exists a binary relation $R \subseteq V_p \times V$, if $(u, u') \in R$ then

(i) $f(u) = f_p(u')$

(ii) $\forall (u, v) \in E_P$, then at least one of the following conditions should be met:

(a) $\exists (u', v') \in E$, such that $f(u) = f_p(u')$ and $f(v) = f_p(v')$, or

(b) $\forall (v, w) \in E_P, \exists (u', w') \in E$ such that $f(u) = f_p(u')$ and $f(w) = f_p(w')$

Since URS considers only child nodes, some subgraphs of data graph will be output although they do not match the pattern graph with all nodes. For example, in Fig. 1a, the subgraph consisting of nodes $\{DM, SD, PD, ST\}$ satisfies the definition. The subgraph will be output in URS once it meets the requirement of out edges. To avoid useless results, we present DRS as follows.

**DRS.** Consider a data graph $G = (V, E, f)$ and a pattern graph $P = (V_p, E_p, f_p)$. We say that graph $G$ matches pattern $P$ via DRS, denoted by $P \preceq_{\text{drsim}} G$, if and only if there exists a binary relation $R \subseteq V_p \times V$, if $(u, u') \in R$ then

(i) $f(u) = f_p(u')$

(ii) $\forall (u, v) \in E_P$, then at least one of the following conditions should be met:

(a) $\exists (u', v') \in E$, such that $f(u) = f_p(u')$ and $f(v) = f_p(v')$, or

(b) $\forall (v, w) \in E_P, \exists (u', w') \in E$ such that $f(u) = f_p(u')$ and $f(w) = f_p(w')$

(iii) $\forall (t, u) \in E_P$, then at least one of the following conditions should be met:

(a) $\exists (t', u') \in E$, such that $f(t) = f_p(t')$ and $f(u) = f_p(u')$, or

(b) $\forall (s, u) \in E_P, \exists (s', u') \in E$ such that $f(s) = f_p(s')$ and $f(u) = f_p(u')$

From the above definitions, the relation between unidirectional and DRS can be drawn as follows:
Theorem 3.1. For a given pattern graph and data graph, denote the result set of DRS as \( R_{\text{drs}} \) and that of URS as \( R_{\text{urs}} \), then

\[
R_{\text{drs}} \subseteq R_{\text{urs}} 
\]  

(1)

4. PARALLEL URS

In this section, we illustrate our distributed algorithm for the unidirectional relaxation problem. To implement the distributed algorithm, we first design the method of generating condition set from the pattern graph.

4.1. Condition set for URS

Condition set is the set of labels which should be satisfied if two nodes match to each other. According to the definition of URS, the condition set of each node can be calculated as Algorithm 1. The basic idea of Algorithm 1 is that either a child node or grandchild nodes should be added into the condition set for each outgoing edge. First, the subset \( S \) is selected from the existing condition set \( \text{CondSet} \). The operation \( \text{setdiff} \) in Line 4 returns the set difference of two vectors. In the following lines, two sets \( S_1, S_2 \) are added into \( \text{CondSet} \) in Line 8. For each outgoing edge, one exiting subset becomes two subsets, the maximum subset number \( SN_{\text{max}} \) in \( \text{CondSet} \) is:

\[
SN_{\text{max}} = 2^k
\]

(2)

where \( k \) is the out-degree of a node.

Algorithm 1 Generate condition set for \( \forall v_i \in V_p \)

Input: Pattern graph \( G(V_p, E_p, f_p) \), \( v_i \in V_p \)

Output: Condition set: \( \text{CondSet} \)

1. \( \text{CondSet} = \emptyset \); 
2. for each \( v_j \in V_p \) such that \((v_i, v_j) \in E_p\) do 
3. for each subset \( S \in \text{CondSet} \) 
4. \( \text{CondSet} = \text{setdiff}(\text{CondSet}, \{S\}); \)
5. \( S_1 = \text{union}(S, \{v_j\}); \)
6. for each child node \( v_k \) of \( v_j \) do 
7. \( S_2 = \text{union}(S, \{v_k\}); \)
8. \( \text{CondSet} = \text{union}(\text{CondSet}, \{S_1\}, \{S_2\}); \)
9. return \( \text{CondSet} \)

Figure 3 shows an example of the condition set. In the example, node A’s out-degree is three. Three child nodes of it and the corresponding grandchild nodes are shown in Fig. 3b. Note that the out-degree of node D is zero, therefore the corresponding grandchild node set is empty (\( \emptyset \)). From the connection relation, Fig. 3c shows the final condition set of node A, which includes six subsets. That is to say, a node with label A in the data graph matches the node in the pattern graph if it covers any subset shown in Fig. 3c.

Theorem 4.1. In URS, the maximum number of nodes which can be deleted simultaneously is the size of the maximum independent set of pattern graph.

Proof. For pattern graph \( P = (V_p, E_p, f_p) \), denote \( S_{\text{less}}, S_{\text{mis}} \) as the set of the nodes which can be deleted simultaneously and the maximum independent set, respectively. If \( S_{\text{less}} > S_{\text{mis}} \), there are at least two nodes \( v_i, v_j \in S_{\text{less}} \) such that \((v_i, v_j) \in E_p\), which violates the definition of URS (two neighbor nodes cannot be deleted at the same time).

On the other hand, \( \forall v_i, v_j \in S_{\text{mis}}, \) if they cannot be deleted simultaneously, they are child node and the corresponding grandchild node, respectively. Thus, there is one edge between \( v_i, v_j \), which violates the definition of maximum independent set. Therefore, the nodes in the maximum independent set can be deleted simultaneously.

4.2. Parallel algorithm of URS

We implement parallel matching with the BSP model. In the BSP model, the key problem is to design the computing and communication of each superstep. We design the parallel algorithm based on the BSP model, which is shown in Fig. 4. It includes three basic supersteps and one repetitive superstep.

The vertex-centric parallel model is adopted in our algorithm. It means that each vertex executes the supersteps concurrently. First, the vertex compares its label with the label of each node in the pattern graph. We abbreviate this operation as \( \text{equal}(v = v_q) \) in Superstep 1, i.e. for \( v \in V \), \( \exists v_q \in V_p \) such
that \( f(v) = f_p(v_p) \), then \( \text{equal}(v, v_p) \) is true. If there is the same label in the pattern graph, the vertex sends its ID to its child nodes. At the same time, it sets the local variable \( \text{match} \) as true. Otherwise, the local variables \( \text{match} \) and \( \text{inactive} \) are set as false and true, respectively. Note that \( \text{inactive} \) indicates whether the vertex will participate in the following operations or not.

In the second superstep, every matched vertex sends its label to its parent vertices. The IDs of parent vertices are obtained in Superstep 1. After the first two supersteps, every matched vertex has the information of its matched child vertices.

In Superstep 3, every matched vertex compares the labels of its matched child vertices to the condition set which can be obtained via the algorithm in Section 4.1. If the set of the child vertices \( \text{SonList} \) covers any subset of condition set \( \text{CondSet} \), it does nothing. Otherwise, the vertex informs its parents by sending its label.

In the following supersteps, a vertex removes any received label from its child vertex set \( \text{SonList} \). It then judges whether the new \( \text{SonList} \) still satisfies the condition set \( \text{CondSet} \). If not, it will also inform its parent vertex. This process will continue until all vertices become inactive. As can be seen from Fig. 4, the communication direction is from current vertex to its parent nodes except in the first superstep.

The number of supersteps determines the running time. We deduce the maximum number of supersteps \( S_{\text{max}} \) as follows:

\[
S_{\text{max}} = \begin{cases} 
\max(l_p) + 2, & \text{if there is no circle in pattern} \\
|E| + 2, & \text{otherwise}
\end{cases}
\]  

(3)

where \( \max(l_p) \) is the maximum length of paths in the pattern graph, and \( |E| \) is the number of edges in the data graph.

The number of the messages between vertices describes the communication cost. We deduce the number of the messages \( \text{Msg}_{\text{urs}} \) as follows:

\[
\text{Msg}_{\text{urs}} = \sum_{i=1}^{m} (D_{\text{out}_i} + D_{\text{in}_i}) + \sum_{j=1}^{m'} D_{\text{in}_j}'
\]  

(4)

where \( m \) is the number of matched vertices in Superstep 1, \( D_{\text{out}} \) and \( D_{\text{in}} \) are the out-degree and in-degree of the matched vertex, respectively. \( D_{\text{in}}' \) includes only the in-degree of the vertices which the start vertex matches to. \( m' \) is the number of vertices which match the label of the vertex in the first superstep, but become unmatched in the following \( (m' \leq m) \).
5. PARALLEL DRS

DRS is developed from dual simulation, which takes not only child nodes, but also parent nodes into consideration. Both the condition sets and the process will be different.

Condition sets. For each vertex in the pattern graph, there are two condition sets for child nodes and parent nodes, respectively. For child nodes, condition set is the same with URS, as shown in Algorithm 1. To distinguish child condition set and parent condition set, we use CondSetofChild and CondSetofPar to denote them, respectively.

To obtain parent condition set, we reverse the directions of edges in the pattern graph. Then, we calculate the child condition set in the reverse graph by utilizing the method introduced in Section 4.1 to obtain the parent condition set CondSetofPar. Figure 3 shows an example, the CondSetofPar of node A is $\{[E], [B]\}$.

Parallel algorithm of dual relaxation matching. In distributed dual relaxation matching, every vertex manages both child and parent nodes. Child condition set CondSetofChild and parent condition set CondSetofPar are used to judge whether the current vertex matches to the corresponding vertex in the pattern graph or not. Once one vertex does not match, it will inform its child nodes and parent nodes. The process continues until all vertices reach stable status. Figure 5 shows the detail of our algorithm for dual relaxation matching based on the BSP model.

The first superstep is similar to that of URS. When the label of the current vertex $v$ exists in the pattern graph $(equal(v, v_q) = true)$, it sends both ID and label to its child nodes since the label will be used by child nodes in the following comparison.

In the second superstep, every matched vertex saves the received label into the local array ParList. Then, its parent node list ParList is compared with the parent condition set CondSetofPar. If the parent list covers at least one subset of its parent condition set CondSetofPar, it is considered matched temporarily and replies to its parents. Otherwise, it notifies its child nodes to remove itself from the parent list. In this superstep, the current vertex sends message to its children or parents.

For every vertex, the received children are added into local list SonList and the received parents are removed from local parent list ParList. The new lists are then compared with the child condition set CondSetofChild and the parent condition set CondSetofPar, respectively. If not matched, the vertex informs its parents and children to drop itself from their lists.

The following supersteps will repetitively remove the received nodes from the corresponding local lists SonList or ParList. The current vertex processes similarly with Superstep 3. The only difference is that it will drop the received nodes from lists, rather than adding. The process halts when all vertices become inactive.

Different from URS, the number of feedback steps may exceed the length of the longest path in the pattern graph. Figure 6 shows an example of the comparison of feedbacks between URS and DRS. In DRS, the unmatched vertex sends messages to not only its parents, but also its children. For example, vertex E in Fig. 6 will inform vertex A and F in DRS algorithm, while it only need send back to vertex A in URS algorithm. The maximum number of feedback steps determines the number of supersteps in both URS and DRS.

The number of messages between vertices in DRS $Msg_{drs}$ can be formulated as follows:

$$Msg_{drs} = \sum_{i=1}^{m} Dout_i + \sum_{j=1}^{m'} Din'_j + \sum_{k=1}^{m''} (Dout'_k + Din'_k)$$  \hspace{1cm} (5)

where $m$ is the number of matched vertices in Superstep 1, $Dout$ is the out-degree of the matched vertex. $Din'$ and $Dout'$ are the in/out-degrees that the opposite vertex is matched to. $m'$ is the number of vertices that send messages in the second superstep. $m''$ is the number of vertices which match in the second superstep, but become unmatched in the following $(m'' \leq m' \leq m)$.

As can be seen from Formula (5), the number of messages is related to not only out-degree, but also in-degree except in the first superstep. It is different from that of URS.

6. EXPERIMENTAL STUDY

The goal of this experimental study is to evaluate the proposed distributed algorithms including URS and DRS.

6.1. Experimental setting

Platform. The experiments were conducted on a Linux-based cluster of five common machines. The configuration of each machine is 4 GB DDR3 RAM, two 2.7 GHz CPUs, each with 2 cores. These machines are connected with 1 Gigabits Ethernet to form a star-type local area network.

We use Hama [17] to implement our algorithms. Hama is an Apache top-level open source project, allowing advanced analysis beyond MapReduce [18]. Many data analysis techniques such as graph algorithms require iterative computations, where BSP model can be more effective than ‘plain’ MapReduce. To run such iterative data analysis applications more efficiently, Hama offers pure BSP computing engine, which can make jobs more efficient than running in Hadoop without Hama.

Hama is a general BSP framework on top of Hadoop [19]. As the basic infrastructure, Hadoop is first deployed on our experimental cluster platform. The Hadoop software library is a framework that allows for the distributed processing of large datasets across clusters of computers. In our experimental cluster, one of these machines plays the role of master (master node) and the others are workers (data node).

Data sets. Both real-world datasets and synthesized datasets are used in our experiments. In terms of real-world
datasets [20], we used Twitter data graph with 81 306 vertices and 1 768 149 edges; Amazon data graph with 403 394 vertices and 3 387 388 edges; and Google data graph with 875 713 vertices and 5 105 039 edges. As for synthesized dataset, we used Hama on the experimental cluster to generate large random data. The input parameters of generating graph are the number of vertices 1 000 000, and the number of distinct labels 10. The information of datasets is listed in Table 1.

Pattern graphs have great importance in graph simulations. In our experiments, we generate pattern graphs with various sizes randomly, which consist of various numbers of vertices from 5 to 10.

### 6.2. Experimental results

The most important results achieved from the experiments are categorized in the following groups: running time, supersteps and messages. The results of the proposed algorithms (URS and DRS) are compared with that of SS [15].

---

**FIGURE 5.** Parallel algorithm for DRS.
Feedback in the data graph when unmatched vertex happens. (a) An example pattern graph without circle; (b) in URS, the number of feedback steps is equal to the length of the longest path; (c) in DRS, the number of feedback steps exceeds the length of the longest path.

**FIGURE 6.** Feedback in the data graph when unmatched vertex happens. (a) An example pattern graph without circle; (b) in URS, the number of feedback steps is equal to the length of the longest path; (c) in DRS, the number of feedback steps exceeds the length of the longest path.

**TABLE 1.** Basic statistics on experimental datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Vertices</th>
<th># Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter</td>
<td>81,306</td>
<td>1,768,149</td>
</tr>
<tr>
<td>Amazon</td>
<td>403,394</td>
<td>3,387,388</td>
</tr>
<tr>
<td>Google</td>
<td>875,713</td>
<td>5,105,039</td>
</tr>
<tr>
<td>Synthesized</td>
<td>1,000,000</td>
<td>9,997,782</td>
</tr>
</tbody>
</table>

**Running time:** We examine the running time of the proposed algorithms with different sizes of pattern graphs $|V_p|$. According to the introduction in Section 2, the running time includes the time of local computation, global communication and synchronization time. Figure 7 shows the results of the total running time on four different datasets. It can be seen that the running time increases for all datasets as the pattern graph size $|V_p|$ increases. This is consistent with the analysis of these algorithms. For a given dataset, the increase means that the number of labels in the pattern graph increases during the first two supersteps. In these supersteps, the time of local computation and global communication will increase.

For the proposed URS and DRS, DRS usually spends more time than URS. From Theorem 3.1, DRS can be regarded as further selecting subgraphs from URS. Moreover, bidirectional communication also increases the time for synchronization. The running time is closely related to the number of supersteps and the communication. Therefore, we will discuss supersteps and communication messages in the following.

Compared with the results of SS, the results on running time of URS and DRS are always less with various sizes of pattern graphs. It demonstrates the predominant performance of the proposed algorithms compared with previous work.

**Supersteps:** Figure 8 shows the results of supersteps. It is obvious that supersteps are correlated to the pattern graph greatly. For most of $|V_p|$, the supersteps of DRS are more than that of URS. On the one hand, one leaf node may match in URS, but not in DRS. The failed match will be fed back along the temporary match path, which increases the number of supersteps. On the other hand, DRS requires dual transmission when a vertex fails to match. The feedback path in DRS may exceed the maximum path in the pattern graph. While the feedback path is shorter than the maximum path in URS if there is no circle in the pattern graph, as described in Formula (3). This will be reflected on the number of supersteps.

In Fig. 8, one obvious observation is that the numbers of supersteps do not always increase as the pattern graph size $|V_p|$ increases. The matched vertices in each superstep affect the number of supersteps greatly. Therefore, as shown in Fig. 9, we further analyze the active vertices in each superstep with Google datasets. Figure 9a and b shows the results of active vertices in DRS and URS, respectively. It can be seen that, the smaller pattern graph is (such as $|V_p| = 5$), the quicker the number of active vertices decreases. It leads to less supersteps. Therefore, if the matched vertices are less, the supersteps can also be less even though the pattern graph size is bigger.

When comparing with SS, the supersteps of URS and DRS are less on various datasets. Since SS includes two steps: dual graph simulation and finding matches in the results of dual graph simulation under distance limitation. Just for the first step, the supersteps of dual graph simulation usually are more than that of URS and DRS. When the match results of the first step are very few, the second steps need less supersteps to processing the less intermediate results, as shown in Fig. 8d.

**Messages:** Figure 10 shows the results of communication messages between vertices. It can be seen that messages are correlated to the sizes of pattern graphs. The bigger pattern graph will generate the more temporary matched vertices, which will consumes communication time when they become unmatched. In our experiments, we only compare the total number of messages sent between vertices. If a vertex send a message, we increases the number of messages with 1. As can be seen from Fig. 10, the number of sent messages correlates not only the pattern graph, but also datasets. For instance, in Fig. 10a, the number of messages in DRS is more than that of URS. But it is reverse in Fig. 10b. It is because
Twitter data graph is denser than the other three datasets. Because lack of the status of parent nodes in URS, the vertices will send messages to all its parent nodes only if they are matched in the first superstep. While every vertex in DRS knows both parents and children’s status, thus it only need send messages to matched parents and children. But it sends messages bidirectionally which may cause more messages. Therefore, the number of messages in URS will exceed that in DRS if the data graph is denser. On the contrary, the number of messages in DRS will be more because its bidirectional messages overrun the messages to inactive parents in URS.

Compared with previous method SS, the proposed DRS and URS both need fewer numbers of messages. One interesting case can be observed in Fig. 10b and c that the numbers of messages are all close when the size of the pattern graph $V_p$ is 10. It is because the number of labels inserted in the data graph is 10. Most vertices are matched firstly, but become inactive quickly for their parent or child requirements. Therefore their numbers of messages are similar in this situation.

**Scalability:** To illustrate the scalability, running time is evaluated with various numbers of machines in cluster system. In the experiment, we use the Amazon dataset and the pattern graph is randomly generated with seven vertices. As shown in Fig. 11, the results of running time with various number of machines from 2 to 5. It can be seen that the running time is correlated to the number of machines greatly. As the increase of the number of machines, the proposed algorithms (both DRS and URS) consume less running time. The results demonstrate the scalability and performance of the proposed distributed algorithms.

7. RELATED WORK

In this section, we survey related work in two categories: graph pattern matching algorithms and distributed graph processing techniques.

7.1. Graph pattern matching algorithms

In graph pattern matching, subgraph isomorphism [5] is NP-complete and often too restrictive to find meaningful matches in the application with large scale such as social networks [9, 21]. Therefore, graph simulation [6] and approximate matching have been developed in recent years.

**Graph simulation.** Graph simulation defines graph matching as a relation between nodes in the pattern graph and in the
FIGURE 8. Supersteps of the parallel algorithms for (URS), dual relaxation simulation (DRS) and strict simulation (SS). (a) Twitter, (b) Amazon, (c) Google and (d) Synthesized.

FIGURE 9. Active vertices in each superstep of unidirectional relaxation simulation (URS) and dual relaxation simulation (DRS) with Google Dataset. Pattern graph size $|V_p| = 5, 7, 9$, respectively. (a) Active vertices in DRS, (b) Active vertices in URS.

data graph. Prior works such as [9, 10, 22] usually generate candidate sets for each vertex in the pattern graph based on their labels, and then filter the mismatched vertices according to the edges in the pattern graph.

Graph simulation is less restrictive than subgraph isomorphism, for example, bounded simulation [9], strong simulation [10]. Bounded simulation extends graph simulation for graph pattern matching by allowing bounds on the number of hops. To extend [9], a proposal incorporated regular expressions as edge constraints on the pattern graph [23]. Strong simulation imposes additional constraints (duality and locality) on graph simulation, and it introduced an approach based on the ball
FIGURE 10. Messages of the parallel algorithms for unidirectional relaxation simulation (URS), dual relaxation simulation (DRS) and strict simulation (SS). (a) Twitter, (b) Amazon, (c) Google and (d) Synthesized.

FIGURE 11. Scalability evaluation.

[10]. It executes dual simulation on each ball for vertex in the data graph and retains the same cubic-time complexity of previous extensions. Top-k graph pattern matching is studied in [24]. By introducing relevance functions and distance functions to rank the matches, they proposed an approximation algorithm and a heuristic algorithm to output the first k matches with the highest scores.

Approximate matching. Graph simulation relaxes the requirement of edge-to-edge, i.e. it allows less edges between vertices. Approximate matching extends mappings from edge-to-edge to edge-to-path. In other words, approximate matching pays attention to the connectivity between the nodes. Approximate matching is of a large search space since it allows additional vertices between two matched nodes. Most approaches use index to accelerate approximate matching [7, 8, 14, 25–28]. Cheng et al. [7, 28] and Zou et al. [25] used 2-hop label scheme as index. While a signature representing neighborhoods within a given distance for each vertex is used as index in [27]. Zhang et al. [26] recorded the distance of each pair of vertices in the graph. They calculated different distances between a pair of vertices in the pattern graph and its counterpart in the data graph. Then the results are used to decrease the match requirement, which is different from our relaxation simulation.

7.2. Distributed graph processing techniques

Distributed query processing has been studied extensively for relational data [29] and XML [30, 31]. There are also some works on distributed graph processing to manage large-scale graphs [31, 32].
Fan et al. [33] proposed a distributed approach to query the reachability between nodes in a large graph. They evaluated reachability queries as a series of boolean equations. The equations form a boolean equation system and the reachability result is obtained from the boolean equation system. In [22], a distributed algorithm for graph simulation is proposed. Every worker computes with local data, and then sends the local match segments which contain boundary nodes to a coordinator. The coordinator is responsible for the assignment scheduling to further evaluate graph simulation. A distributed vertex-centric approach based on BSP for pattern matching is proposed in [15]. Message is sent between the neighboring nodes, and the matches are obtained when there is no active vertex. Instead of relying on huge size indices, Sun et al. [14] used graph exploration and massive parallel computing for query processing. They presented a subgraph isomorphism algorithm for graphs deployed on a distributed memory store Trinity [34]. To avoid the construction time and the storage capacity for complicated indices, they only use a simple string index which maps node labels to node IDs.

8. CONCLUSION

We have proposed a revision of graph pattern matching, named relaxation simulation. By allowing partially absent nodes, it yields more sensible matches in emerging fields such as social network analysis. Moreover, we have also designed parallel algorithms for both URS and DRS. These algorithms are based on the BSP model and can be deployed on cloud computing platforms. Our experimental results have verified the scalability and effectiveness of the proposed parallel algorithms using real-life and synthetic data.

FUNDING

This work was supported by the National Natural Science Foundation of China [grant numbers 61106036, 61173169, 61272147], the Science and Technology Plan of Hunan Province [2013RS4046] and the Major Science & Technology Research Program for Strategic Emerging Industry of Hunan [2012GK4054].

REFERENCES


