Distributed Partial Simulation for Graph Pattern Matching

Aissam Aouar[1], Saïd Yahiaoui[2], Lamia Sadeg[1], Nadia Nouali-Taboudjemat[2] Kadda Beghdad Bey[1]

[1]*Ecole Millitaire Polytechnique, BP 17, Bordj el Bahri, Algiers, 16111, Algeria*
[2]*CERIST Research Center on Scientific and Technical Information, Ben Aknoun, Algiers, 16030, Algeria*
Email: aissam.a@gmail.com

Pattern matching in big graphs is important for different modern applications. Recently, this problem was defined in terms of multiple extensions of *graph simulation*, to reduce complexity and capture more meaningful results. These results were achieved through the relaxation of commonly used constraint in *subgraph isomorphism* pattern matching. Nevertheless, these graph simulation variant models are still too strict to provide results in many cases, especially when analyzed graphs contain anomalies and incomplete information. To deal with this issue, we introduce a new graph pattern matching method, called *partial simulation*, capable of retrieving matches despite missing parts of the pattern graph, such as vertices and/or edges. Furthermore, considering the number and inequality of the outputs, we define a relevance function to compute a value expressing how each match vertex respects the pattern graph. Similarly, we define *partial dual simulation* graph pattern matching that returns vertices that satisfy a part of the *dual simulation* constraints and assigns a relevance value to them. Additionally, we provide distributed scalable algorithms to evaluate the proposed partial simulation methods based on the distributed vertex-centric programming paradigm. Finally, our experiments on real-world data graphs demonstrate the effectiveness of the proposed models and the efficiency of their associated algorithms.

Keywords: Big graphs; Graph pattern matching; Graph simulation; Distributed computing; Vertex-centric programming

Received 28 March 2022; revised 28 March 2022

1. INTRODUCTION

Nowadays, graphs are used intuitively to model the relationships between data entities in a wide range of modern applications, including social networks, homeland security, biology, cyber-security and computer networks. These graphs are huge, including billions of vertices and edges, and are distributed across multiple data centers around the globe. For instance, according to Meta's second-quarter 2022 results report, Facebook has approximately 2.93 billion monthly active users [1]. Processing this massive number of linked entities to extract valuable information is a significant challenge in this context, where traditional graph frameworks fall short and new concepts for big graph analysis are needed. Graph Pattern Matching (GPM) is one of the frequent analysis techniques in graph processing. It consists in finding matched subgraphs for a given pattern (query) graph Q in a data graph G and it is generally defined in terms of subgraph isomorphism. The subgraph isomorphism matching model returns the strictest results for GPM in terms of topology [2], but this method comes with an NP-complete complexity [3] which is impractical for big graphs. Moreover, big graphs are often littered with missing or incorrect data and the strictness of an exact match is too rigid to return results. Likewise, information retrieval is an interactive and iterative process and user needs to reformulate the initial query because it is over- or under-specified or simply contains errors [4]. The task is even more complex when a user does not have sufficient knowledge about the data graph [5]. In that case, the exact GPM that can not tolerate query design errors is inappropriate. To address the computational complexity of exact matching, the uncertainty associated with data graphs, and the potential of mistakes in pattern graphs, researchers propose *inexact GPM* approaches. They relax matching conditions in order to find an approximate solution in a reasonable time, owing to the fact that their matched subgraph is less sensitive

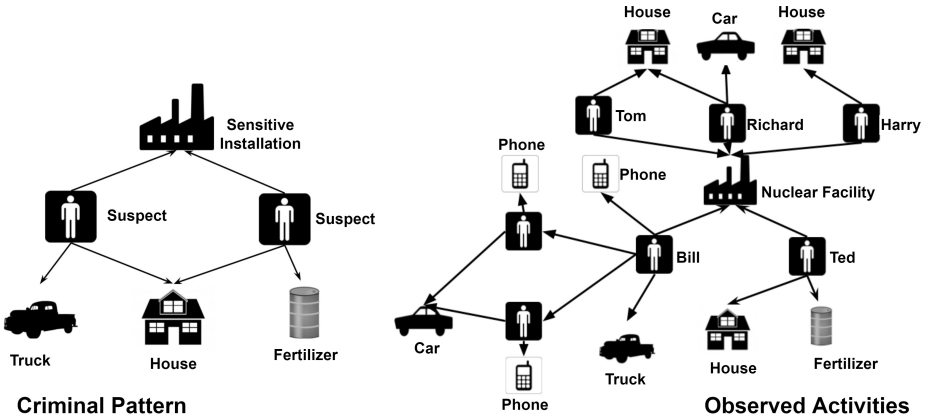


FIGURE 1: Criminal pattern graph and a data graph of observed activities.

to topological rules [6, 7, 8, 9]. The inexact GPM by simulation and its variants come as a practical alternative to subgraph isomorphism and allow matches to be found in polynomial time. Moreover, some researchers believe that graph simulation can be more appropriate than subgraph isomorphism for some modern applications such as social network analysis because it yields matches that are conceptually more intuitive [10, 11, 12]. Although GPM with graph simulation advantages, it still requires that all vertices and edges in the pattern graph have their correspondents in the data graph and does not tolerate any missing. Often, this requirement is too rigorous to obtain results in uncertain data graphs and insufficient to compensate potential design faults in pattern graphs.

The field of information intelligence is an excellent example of how an inexact GPM is more appropriate for extracting the best possible knowledge from the massive of connected data. Indeed, the expansion of information sources such as sensors, metadata, open source information and social networks significantly complicate the fusion and processing of this huge amount of data and makes it a true challenge. When this massive volume of data is combined to create a huge graph, GPM can be applied for situation awareness as well as the detection of criminal organizations [13, 14, 15, 16]. However, every output should be provided in a reasonable time frame and resource consumption. As a result, GPM with graph isomorphism is unfeasible and simulation is applied instead to reduce algorithm complexity and capture more meaningful findings. Besides, being able to find inexact pattern matches is critical, much more when analysts operate in an environment with limited observability and uncertain data. In [17], the authors studied and mapped the 9/11 Al-Qaeda terrorist network characterized by actors that operate in secrecy and minimize interactions. Thus, data about criminals, their contacts and associations are inevitably incomplete, causing missing nodes and links in networks [18]. Foremost, the analyst might need to match a general pattern without knowing all of the details, or may have defined some aspects of the pattern incorrectly. Finding inexact matches alerts about activity that *breaks the mold* of previous threats, and can prevent this kind of surprise for which intelligence agencies have been criticized [19].

Example 1. In Figure 1 inspired from [19], we can observe a pattern graph of suspected people's connections and activities relations, as well as a portion of the data graph. Searching for a subgraph match for the criminal pattern in the data graph gives an empty answer, either with graph isomorphism or graph simulation. This is because the relationship between a suspected person and his residence is not established. Intelligence analysts need a GPM concept that can detect all or a part of potential criminal activity.

Contributions: This paper proposes a new GPM models based on graph simulation to handle situations related to incomplete data graphs, empty answers, errors in pattern graph design and answer with the best results to a general pattern graph.

1. We introduce the notion of partial simulation and partial dual simulation for pattern matching in big graphs, so that we can extract matching vertices that preserve a part of descendant or/and ascendant relations according to the pattern graph.
2. The proposed partial simulation and partial dual simulation GPM are capable of returning the complete match by simulation and dual simulation, respectively if it exists.
3. When an inexact GPM concept is applied, in general, the user is interested in the best results in terms of similarity to the query graph. So, we propose a ranking function that assigns a value to each output result based on how the topology of the pattern graph is preserved.
4. We design distributed algorithms from the point of view of a vertex, featuring the Pregel model, to return the match vertices with the best possible

rank for each one. Also, we provide optimizations to speed up specific types of query graphs, such as acyclic pattern graphs.
5. Using real-life graphs and the GraphX API, we present experimental results that prove the effectiveness of our models and the efficiency of our algorithms.

The rest of the paper is organized as follows. We review the related works in Section 2. Then, we give the preliminary concepts and definitions about graph pattern matching in Section 3. In Section 4, we introduce the new partial simulation GPM model. Section 5 is dedicated to present a ranking method for partial simulation GPM results. Following that, we define the partial dual simulation GPM and its ranking functions in Section 6. Distributed algorithms based on vertex-centric paradigm are explained in Section 7. We cover an experimental study of the given algorithms in Section 8. Finally, Section 9 concludes the paper.

2. RELATED WORK

Exact GPM methods [3, 20] require that all vertices and edges to match exactly the pattern graph. This is costly for big graphs and excessively rigid for large specter of modern applications. Despite the relaxation provided in GPM by simulation and its derived models [21, 11, 22], a match can not ignore a portion of the query graph. In the recent survey [23] we can find a view of different approaches of GPM in massive graphs, algorithms and frameworks. Some of the proposed GPM relaxed models based on graph simulation attempt to capture more significant results that are more consistent with the error and the ambiguity rates found in big graphs. *Bounded Simulation* [10] maps an edge in the query graph to a path in the data graph within a predefined number of hops. [24] adds to graph simulation the possibility to associate an edge in the query graph with a regular expression, specifying the connectivity via a path of certain edge types and of a possibly bounded length in the data graph. Even if these two propositions offer the highest degree of flexibility for expressing the reachability between vertices in a data graph, they do not allow missing relations from the pattern graph. Another solution proposed in [25] is *Parallel Relaxation Simulation* GPM that allows the replacement of missing nodes with only one hop, where a child vertex can be substituted by its own child vertices in the data graph. But this proposal maintains the requirement that all edges in the pattern graph have a map in the matched subgraph and does not distinguish between output vertices that fully match the query graph without substitution and those that do not. [26] proposes *relaxed graph simulation* that adds the multi-hop substitution between connected vertices and propose a ranking function. To allow the missing vertices, this approach still requires a substitution, and the proposed centralized algorithm is a limitation for a wide range of big distributed graphs. In [27], the authors propose a relevance function that favors graph simulation matches that can reach more other matches. In our scenario, the relevance is determined by the number of child edges preserved from the query graph, rather than the number of times they are preserved in the data graph. In Table 1, we give a succinct comparison between the proposed method and related work.

To reduce complexity and response to specific application cases, other works propose efficient graph pattern matching for a particular type of query graphs like Directed Acyclic Graphs (DAG) [28] and trees [29]. Likewise, we propose an optimized algorithms that consider these particular types of pattern graphs to significantly reduce overall query processing times.

3. PRELIMINARIES

In this section, we introduce basic concepts and definitions related to GPM, graph simulation, and graph dual simulation. Throughout this paper, we consider vertex-labeled, directed simple graphs and without loss of generality, we assume that the query graph is connected. Indeed, the result of graph pattern matching for a disconnected query graph is equal to the union of the results for its connected components.

Definition 3.1 (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 (u, u') denotes an edge from node u to u', and*
3. *f is a function, such that for each node u in V, $f(u)$ is a label from an alphabet $\sum$.*

Definition 3.2 (Pattern graph). *A pattern graph is a directed graph $Q = (V_Q, E_Q, f_Q)$, where:*

1. *V_Q is the set of query nodes,*
2. *E_Q is the set of query edges, and*
3. *f_Q is a function defined on V_Q, such that for each node u in V_Q , $f_Q(u)$ is a label from an alphabet $\sum_Q$.*

Definition 3.3 (Child Set). *$C(v)$ is the child set of vertex v such that $v' \in C(v)$ iff $(v, v') \in E$.*

Definition 3.4 (Parent Set). *$P(v)$ is the parent set of vertex v such that $v' \in P(v)$ iff $(v', v) \in E$*

Definition 3.5 (Graph Simulation). *Graph $G = (V, E, f)$ matches pattern graph $Q = (V_Q, E_Q, f_Q)$ via graph simulation, denoted by $Q \prec G$, if there exists a binary match relation $S \subseteq V_Q \times V$ such that:*

1. *for each $(u, v) \in S$, $f_Q(u) = f(v)$, and*
2. *for each node u in V_Q, there exists v in V, such that:*
   a. *$(u, v) \in S$, and*

GPM Method	Simulation	Dual Simulation	Substitute missing edges	Allow missing edges	Ranking	Algorithm
Graph Simulation [27]	✓	×	×	×	✓	Centralized
Bounded Simulation [10]	✓	×	✓	×	×	Centralized
Simulation with regular expression [24]	✓	×	✓	×	×	Centralized
Parallel Relaxation Simulation [25]	✓	✓	✓	×	×	Distributed
Relaxed Graph Simulation [26]	✓	×	✓	×	✓	Centralized
Partial Simulation (this paper)	✓	✓	×	✓	✓	Distributed

TABLE 1: Comparison between the proposed GPM method and related work.

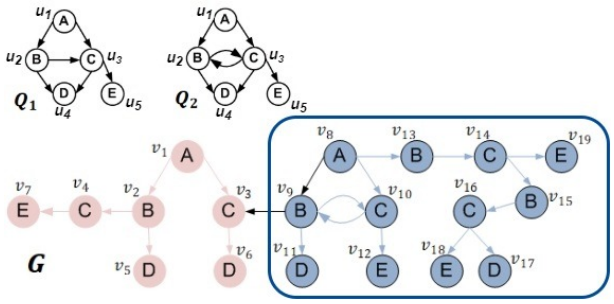


FIGURE 2: Query graphs and a data graph.

b. for each edge $(u, u') \in E_Q$, there exists an edge $(v, v') \in E$, such that $(u', v') \in S$.

While there may be multiple graph simulation relations S in a graph G for a pattern Q, there exists a *unique maximum match*, denoted by $M(Q, G)$, such that for any match S in G for Q, $S \subseteq M(Q, G)$ [10]. We refer to the maximum match as the match in G for Q and $M(Q, G)$ is an empty set if, G does not match Q. The complexity of calculating $M(Q, G)$ is $O((V_Q + V)(E_Q + E))$ and the result is bounded by $|V||V_Q|$ as given in [10]. To preserve query graph topology, pattern matching via dual simulation was proposed by imposing duality constraint to graph simulation and conserving parent relationship as well as the child one [11].

Definition 3.6 (Graph Dual Simulation). *Graph $G = (V, E, f)$ matches pattern graph $Q = (V_Q, E_Q, f_Q)$ via graph dual simulation, denoted by $Q \prec_D G$, if there exists a binary match relation $S^D \subseteq V_Q \times V$ such that:*

1. for each $(u, v) \in S^D$, $f_Q(u) = f(v)$; and
2. for each node u in V_Q, there exists v in V such that:

(a) $(u, v) \in S^D$, and
(b) for each edge $(u, u') \in E_Q$, there exists an edge $(v, v') \in E$ such that $(u', v') \in S^D$
(c) for each edge $(u'', u) \in E_Q$, there exists an edge $(v'', v) \in E$ such that $(u'', v'') \in S^D$

As pattern matching by graph simulation, there may be multiple matches for a query graph Q in a data graph G by dual simulation, but there exists a unique maximum match denoted by $M^D(Q, G)$ [11].

4. PARTIAL SIMULATION

This section proposes a new pattern matching model called *partial simulation*, which allows the retrieval of query graph matches within a data graph without needing all graph simulation constraints to be satisfied. Therefore, partial simulation GPM is defined as a binary relation that does not require that all vertices from Q have a match in G, nor a match in G preserves all the child relations of the query vertex. With this new GPM model, we are able to find results that match just a part of the query graph. It is formally defined as follows.

Definition 4.1 (Partial Simulation). *A graph $G = (V, E, f)$ matches a pattern $Q = (V_Q, E_Q, f_Q)$ with partial simulation, denoted by $Q \prec_P G$, if there exists a nonempty binary match relation $S_P \subseteq V_Q \times V$, such that:*

1. for each $(u, v) \in S_P$, $f_Q(u) = f(v)$,
2. if $(u, v) \in S_P$, for each edge $(u, u') \in E_Q$: $(u', v') \in S_P$, if $f_Q(u') = f(v')$ and $(v, v') \in E$.

If (u, v) and $(u', v') \in S_P$, where $v' \in C(v)$ and $u' \in C(u)$, we call u' a *materialized child* and v' a *relevant child*. We denote the set of relevant child vertices by $C(u, v)$. Also, the edge (u, u') is a materialized child edge and (v, v') edge is a relevant child edge. From the definition of partial simulation, there may be multiple matches S_P in a graph G for a pattern Q, but there exists a *unique maximum match*, denoted by $M_P(Q, G)$, such that for any match S_p in G for Q, $S_P \subseteq M_P(Q, G)$. We refer to $M_P(Q, G)$ as the match in G for Q, if it is an empty set, then G does not match Q.

Example 2. From the query graph Q_1 and data graph G in Figure 2, we can not find a match with graph simulation but we succeed to discern a graph partial simulation relation presented in $S_P = \{(u_1, v_1), (u_2, v_2), (u_3, v_3), (u_3, v_4), (u_4, v_5), (u_4, v_6), (u_5, v_7)\}$. This partial simulation relations occur even if the data nodes v_4 and v_3 have not preserved all child edges of the query vertex u_3. One can propose another partial simulation relation as $S'_P = \{(u_1, v_1), (u_2, v_2), (u_3, v_3), (u_3, v_4), (u_4, v_5), (u_4, v_6)\}$, even if the query node u_5 does not have a match.

Proposition 4.1. *For any graph G and pattern Q, if $Q \prec_P G$, then there is a unique maximum match in*

G for Q.

Proof. Obviously, the maximum match $M_P(Q,G)$ is the union of all n possible partial simulation relations of Q in G: $M_P(Q,G) = \bigcup_{i=1}^{n} S_P^i$. Now, we show the uniqueness of the maximum match by contradiction. That is, if there exist two distinct maximum matches M_1 and M_2 , then $M_3 = M_1 \cup M_2$ is a match that is larger than both M_1 and M_2. □

Proposition 4.2. *If S is a match in G for Q by simulation, it is also a match by partial simulation.*

Proof. If S is a match in G for Q by simulation and $(u,v) \in S$ then $f(u) = f_Q(v)$. This condition is sufficient to make S a partial simulation relation. Any other condition for preserving child relations from the query graph is not mandatory in a partial simulation relation. □

Notice that, pattern matching by graph simulation is a special case of partial simulation GPM, that occurs when each $(u,v) \in S_P$, the set of materialized child vertices is exactly $C(u)$.

Corollary 4.1. *According to Proposition 4.1 and Proposition 4.2 we can conclude that: $M(Q,G) \subseteq M_P(Q,G)$.*

5. RANKING GRAPH PARTIAL SIMULATION MATCHES

As mentioned in [10], the size of a maximum match of a pattern matching by graph simulation $|M(Q,G)|$ is bounded by $|V_Q||V|$ and could be excessively large when G is big. Moreover, we can say from Corollary 4.1 that $|M(Q,G)| \leq |M_P(Q,G)|$. This means that the size of partial simulation maximum match is at least equals to the simulation one and has the same bound. Furthermore, not all matches of a query vertex in different partial simulation relations have necessarily the same set of materialized child vertices. This situation may vary from a data vertex that just match the label of a query vertex, to another one that materializes all child edges, in the same manner as in graph simulation relation. Due to the differences between match vertices of the same query vertex, it will be interesting to focus on the quality of a match in order to select out the most meaningful vertices. To achieve this purpose, we define a *relevance function* that assigns a value to each match vertex in order to determine their ranking. The proposed relevance function is motivated by the fact that in partial simulation GPM, a match data vertex and its descendants have not to preserve all child relations from the query graph. This function will assign a *relevance* to each match vertex in a partial simulation relation depending on its capacity, as well as its descendants, to preserve query graph's child edges.

First, remember that if $(u,v) \in M_P(Q,G)$, then v' is not necessarily unique, such that $v' \in C(v)$ and $(u',v') \in M_P(Q,G)$, where $u' \in C(u)$. In other words, a vertex v matching u by partial simulation may have multiple relevant child vertices for the same materialized child vertex. In our proposed relevance function, we evaluate the completeness of a partial simulation by the number of materialized child vertices and not the number of relevant child vertices like in [27]. For this aim, we start by introducing the notions of *relevant child subset*, *relevant child association*, and *relevant partial simulation match*.

Definition 5.1 (Relevant child subset). *Given $(u,v) \in M_P(Q,G)$, the relevant child subset of a materialized child u', denoted by $C_{u'}(u,v)$, is the subset of relevant vertices $v' \in C(u,v)$ such that: $(u',v') \in M_P(Q,G)$.*

We can describe the relevant child subset $C_{u'}(u,v)$ as a subset of vertices from the relevant child vertices set $C(u,v)$ that match u'.

Definition 5.2 (Relevant child association). *We define relevant child association of v matching u by partial simulation, denoted by $\beta(u,v)$, as a relation in $M_P(Q,G)$ from $C(u)$ to $C(u,v)$, such that for each $u' \in C(u)$, if $C_{u'}(u,v) \neq \emptyset$, it is associated exactly one element from $C_{u'}(u,v)$.*

This relation links each materialized child to a unique relevant child if it exists. We denote by $\beta_{cyc}(u,v) \subseteq \beta(u,v)$ the elements (u',v') where u' is in the same cycle as u and $\beta_{dag}(u,v) \subseteq \beta(u,v)$, the subset of $\beta(u,v)$ that are not in β_{cyc}. The set of all possible relevant child associations when v match u is denoted by $B(u,v)$ where $\forall \beta(u,v)/\beta(u,v) \in B(u,v)$.

Example 3. In Figure 2, all possible relevant child associations of (u_1,u_8) from $M_P(Q_2,G)$ are $B(u_1,v_8) = \{\{(u_2,v_9),(u_3,v_{10})\},\{(u_2,v_{13}),(u_3,v_{10})\}\}$. Also, we have $\beta(u_2,v_9) = \{(u_3,v_3),(u_4,v_{11})\}$ where $\beta_{dag}(u_2,v_9) = \{(u_4,v_{11})\}$ and $\beta_{cyc}(u_2,v_9) = \{(u_3,v_3)\}$.

Definition 5.3 (Relevant partial simulation match). *Relevant partial simulation match (relevant PSM) of v matching u is a relation in $M_P(Q,G)$ denoted by $S_P(u,v)$, such that:*

1. For each $(u_1,v_1) \in S_P(u,v)$ it is associated a $\beta(u_1,v_1)$; and
2. $(u_1,v_1) \in S_P(u,v)$ if $(u_1,v_1) = (u,v)$ or $\exists\ (u_2,v_2) \in S_P(u,v)$ where $(u_1,v_1) \in \beta(u_2,v_2)$;

In other words, for $(u,v) \in M_P(Q,G)$, if we select a relevant child association $\beta(u,v)$ and recursively for each vertex $(u',v') \in \beta(u,v)$, we obtain this new relation called relevant PSM. The relevant PSM for v matching u is not unique, as several relevant child associations may exist.

Example 4. From $M_P(Q_1,G)$ in Figure 2, the relation $S_P(u_1,v_1)$ represented in red is a relevant partial simulation match, where $\beta(u_1,v_1) =$

Match	DAG relevance
(u_1, v_1)	6
(u_2, v_2)	3
(u_3, v_3)	1
(u_3, v_4)	1
(u_4, v_5)	0
(u_4, v_6)	0
(u_5, v_7)	0

TABLE 2: DAG relevance example.

$\{(u_2, v_2), (u_3, v_3)\}$, $\beta(u_2, v_2) = \{(u_3, v_4), (u_4, v_5)\}$, $\beta(u_3, v_3) = \{(u_4, v_6)\}$ and $\beta(u_3, v_4) = \{(u_5, v_7)\}$

Now, we provide a function to determine the relevance of a relevant PSM, indicated by $|S_P(u, v)|$, which evaluates the number of preserved child relations (materialized child edges) when v match u and its descendants in the relevant PSM $S_P(u, v)$. Before we continue, it's essential to remember an important feature of simulation GPM. If a query graph contains a cycle, pattern matching using graph simulation may result in a longer cycle in the data graph. Thus, a query vertex included in a cycle may have matches with different number of descendants [22]. Hence, the relevance function is calculated from two parts to avoid the effect of an unlimited number of possible match repetitions in a cycle. The first is *DAG relevance function*, represented by $|S_P(u, v)|_{dag}$, which is used to estimate the preserved child relations derived from $\beta_{dag}(u, v)$. The second is the *cyclic relevance function*, denoted by $|S_P(u, v)|_{cyc}$, which is determined from $\beta_{cyc}(u, v)$. Finally, we define $|S_P(u, v)|$ as the relevance of $S_P(u, v)$ from all relevant child vertices in beta $\beta(u, v)$.

5.1. DAG relevance function

As stated above, $|S_P(u, v)|_{dag}$ is the DAG relevance of (u, v) in a relevant PSM, and it is based on preserved child relations from relevant child vertices in $\beta_{dag}(u, v)$. This $|S_P(u, v)|_{dag}$ is defined by the addition of the number materialized child vertices directly preserved by v matching u in $S_P(u, v)$, i.e., equal to the size of $\beta_{dag}(u, v)$, and the sum of the PSM relevance $|S_P(u', v')|$ such that $(u', v') \in \beta_{dag}(u, v)$.

$$|S_P(u, v)|_{dag} = |\beta_{dag}(u, v)| + \sum_{\beta_{dag}(u,v)} |S_P(u', v')| \quad (1)$$

Example 5. From Figure 2, the query graph Q_1 is a DAG then $\forall (u, v) \in M_P(Q_1, G), \beta_{cyc}(u, v) = \emptyset$. The relevance of v_1 when match u_1 in the relevant PSM in red color will be: $|S_P(u_1, v_1)|_{dag} = 2 + |S_P(u_2, v_2)|_{dag} + |S_P(u_3, v_3)|_{dag}$ where $|S_P(u_2, v_2)|_{dag} = 2 + |S_P(u_3, v_4)|_{dag} + |S_P(u_4, v_5)|_{dag}$ and $|S_P(u_3, v_3)|_{dag} = 1 + |S_P(u_5, v_6)|_{dag}$ and so on recursively to find at the end the values in the Table 2.

5.2. Cyclic relevance function

Now we present the cyclic relevance function $|S_P(u, v)|_{cyc}$ for a data vertex v that matches a query vertex u in a relevant PSM $S_P(u, v)$, where u is included in at least one cycle.

Definition 5.4. *A strongly connected component scc of a directed graph is a maximal set of vertices such that any two vertices in the set are mutually reachable.*

This implies that each query vertex in a *scc* is part of a cycle, and that all vertices in this *scc* are descendants of each other. For this case, we define a relation called *relevant strongly connected component*, denoted by $scc_p(u, v)$, which corresponds to the match of the *scc* vertices in $S_P(u, v)$:

Definition 5.5 (Relevant strongly connected component). $(u_1, v_1) \in scc_p(u, v)$ *if :*

1. $(u_1, v_1) = (u, v)$; *or*
2. $\exists\ (u_2, v_2) \in scc_p(u, v)$ *such that* $(u_1, v_1) \in \beta_{cyc}(u_2, v_2)$;

To determine the cyclic relevance for (u, v) in a relevant strongly connected component $scc_p(u, v)$, we are interested in materialized edges from the pattern graph regardless of the size of this relevant scc, because the size of a relevant *scc* may vary from a match to an other as explained above. Further, we can not apply the recursive formula from DAG relevance since there is cycles. Thereby, we determine materialized edges $E_{scc}(u, v)$ and materialized vertices $V_{scc}(u, v)$ of a $scc_p(u, v)$ as follows:

1. $V_{scc}(u, v) = \{u_1 \in V_q \mid (u_1, v_1) \in scc_p(u, v)\}$
2. $E_{scc}(u, v) = \{(u_1, u_2) \in E_Q \mid (u_1, v_1) \in scc_p(u, v)$ and $(u_2, v_2) \in \beta_{cyc}(u_1, v_1)\}$

The $V_{scc}(u, v)$ is all materialized vertices matched in $scc_p(u, v)$ and $E_{scc}(u, v)$ is the set materialized edges from the pattern graph edges. The cyclic relevance function is defined as follows:

$$|S_P(u, v)|_{cyc} = |E_{scc}(u, v)| \quad (2)$$

This function gives an estimation of how a scc in the pattern graph is preserved in the matching $scc_p(u, v)$ by estimating the materialized child edges. This value is independent from the size of $scc_p(u, v)$ and if $S_P(u, v)$ is a graph simulation relation then all data vertices v' such $(u', v') \in scc_p(u, v)$ can reach the same descendants i.e., have the same relevance.

5.3. General relevance function

In general, a query vertex may be included in a cycle all while having child vertices that are not included in the same cycle. Therefore, the general relevance of v matching u in a relevant PSM $S_P(u, v)$ will be measured based on cyclic and DAG relevance values. This means

that we consider the materialized edges of the relevant scc $scc_p(u,v)$ and any DAG relevance of any vertex in it, while avoiding any possible duplication of materialized edges due to the existence of a cycle. The general relevance function is defined as follows:

$$|S_P(u,v)| = |S_P(u,v)|_{cyc} + \sum_{u' \in V_{scc}} \max_{(u',v') \in scc_p(u,v)} |S_P(u',v')|_{dag} \quad (3)$$

This formula gives the general relevance by the addition of *cyclic relevance* and the best *DAG relevance* values between the relevant vertices for each materialized vertex in $V_{scc}(u,v)$. If the query vertex u is not included in a cycle then $\beta_{cyc} = \emptyset$ which means that $|E_{scc}(u,v)| = 0$ and $V_{scc}(u,v) = \{u\}$. The equation 3 becomes $|S_P(u,v)| = 0 + \sum_u \max_{(u,v)} |S_P(u',v')|_{dag} = |S_P(u,v)|_{dag}$ as given in Equation 1. Also, notice that the general relevance is not affected by the size of the match in case where it exists a cycle but just the materialized edges and the best DAG relevances are taken.

Example 6. From $M_P(Q_2,G)$ part in Figure 2, the following relation is a relevant partial simulation match $S_P(u_1,v_8) = \{(u_1,v_8),(u_2,v_9),(u_3,v_{10}),(u_4,v_{11}),(u_5,v_{12})\}$, where $\beta(u_1,v_8) = \{(u_2,v_9),(u_3,v_{10})\}$, $\beta(u_2,v_9) = \{(u_3,v_{10}),(u_4,v_{11})\}$, $\beta(u_3,v_{10}) = \{(u_2,v_9),(u_4,v_{11})\}$. Another relevant PSM can be proposed, if we choose $\beta(u_1,v_8) = \{(u_2,v_{13}),(u_3,v_{10})\}$ as another relevant child association, as shown in Figure 2 in blue line. In this precise case, let us compute the rank of v_{13} when matching u_2. From Q_2 we can say that u_2 is included in the cycle with u_3, so for $\beta(u_2,v_{13}) = \beta_{cyc}(u_2,v_{13}) = \{(u_3,v_{14})\}$ and $scc_p(u_2,v_{13}) = \{(u_2,v_{13}),(u_3,v_{14}),(u_2,v_{15}),(u_3,v_{16})\}$. Form $scc_p(u_2,v_{13})$ we can determine the materialized vertices and edges as follows: $V_{scc}(u_2,v_{13}) = \{u_2,u_3\}$ and $E_{scc}(u_2,v_{13}) = \{(u_2,u_3),(u_3,u_2)\}$. Now, we can apply the Equation 3 : $|S_P(u_2,v_{13})| = |E_{scc}((u_2,v_{13})| + max\,(|S_P(u_2,v_{13})|_{dag}, |S_P(u_2,v_{15})|_{dag}) + max(|S_P(u_3,v_{14})|_{dag}, |S_P(u_3,v_{16})|_{dag}) = 2 + max(0,0) + max(1,2) = 2+0+2 = 4$

5.4. Partial simulation match rank

When a data vertex v matches a query vertex u using partial simulation GPM, the PSM relevance values $|S_P(u,v)|$ may vary according to the chosen relevant PSM. We define the partial simulation rank of $(u,v) \in M_P(Q,G)$, denoted by $\Delta(u,v)$ as the maximum PSM relevance value that can be reached by v when matching u, from any possible relevant PSM $S_P(u,v) \subseteq M_P(Q,G)$.

$$\Delta(u,v) = \max_{S_P(u,v) \subseteq M_P(u,v)} |S_P(u,v)| \quad (4)$$

By associating this rank $\Delta(u,v)$ with each match in $M_P(u,v)$, we can establish a ranking between matches of the same query vertex in term of preserved child edges from the query graph. Moreover, if the relevant PSM is a graph simulation, then the rank is maximal because it materializes all possible child edges of a query vertex.

6. PARTIAL DUAL SIMULATION

Dual simulation GPM is capable of retrieving more meaningful answers than simulation, because it enforces duality by preserving ascendant relations as well as descendant edges. Simultaneously, more constraints means fewer results when even simulation GPM may be unable to return a match. In that perspective, we introduce *Partial Dual Simulation* as a new model for graph pattern matching that accepts duality but tolerates missing relations and/or vertices in order to catch the most significant results by searching for vertices that verify as much as possible the child and parent relations without being compromised if any condition is missing.

Definition 6.1 (Partial dual simulation). *A graph $G = (V,E,f)$ matches a pattern $Q = (V_Q, E_Q, f_Q)$ with Partial Dual Simulation, denoted by $Q \prec_P^D G$, if there exists a nonempty binary match relation $S_P^D \subseteq V_Q \times V$ such that:*

1. *for each $(u,v) \in S_P^D$, $f_Q(u) = f(v)$,*
2. *if $(u,v) \in S_P^D$, for each edge $(u,u') \in E_Q$: $(u',v') \in S_P^D$, if $f_Q(u') = f(v')$ and $(v,v') \in E$.*
3. *if $(u,v) \in S_P^D$, for each edge $(u',u) \in E_Q$: $(u',v') \in S_P^D$, if $f_Q(u') = f(v')$ and $(v',v) \in E$.*

If (u,v) and $(u',v') \in S_P^D$, where $v' \in P(v)$ and $u' \in P(u)$, we call u' a *materialized parent vertex*, v' a *relevant parent* and we denote the set of all relevant parent vertices by $P(u,v)$. Also, the edge (u',u) is a materialized parent edge and (v',v) is a relevant parent edge. As with the partial simulation GPM, partial dual simulation GPM does not need that all vertices in Q have a match in G, nor does it require that the matches in G preserve all of the query vertex's child and parent relations. There may be multiple matches S_P^D in a graph G for a pattern Q, but it exists a *unique maximum match*, denoted by $M_P^D(Q,G)$, such that for any match S_P^D in G for Q, $S_P^D \subseteq M_P^D(Q,G)$. We refer to the maximum match as the match in G for Q. If $M_P^D(Q,G)$ is an empty set, so G does not match Q. Equally, to partial simulation GPM, If S^D is a match in G for Q by dual simulation, it is also a match by partial dual simulation: $M^D(Q,G) \subseteq M_P^D(Q,G)$.

6.1. Partial dual simulation match rank

As viewed in partial simulation GPM, the size of $M_P^D(Q,G)$ can be huge, larger or equal to $M^D(Q,G)$. Furthermore, a disparity may exist between the matching results. On one hand, we can obtain a data vertex that only matches the label value of the query vertex and, on the other hand, we can obtain

vertices that preserve the label and all descendant and ascendant edges of the query vertex, similarly to a dual simulation relation. Therefore, all these match results with different abilities to preserve child and parent relations are more meaningful for the user if a value is assigned to each match to identify the best results in terms of edge conservation. In this section we will present a ranking function for partial dual simulation pattern matching results by extending definitions presented in the Partial Simulation GPM to be able to assign a value depending on materialized ascendants and descendants relations.

Definition 6.2 (Relevant parent subset). *Given $(u,v) \in M_P(Q,G)$, the relevant parent subset of a materialized parent vertex $u' \in P(u)$, denoted by $P_{u'}(u,v)$, is the set of vertices v' such that:$v' \in P(u,v)$ and $(u',v') \in M_P(Q,G)$.*

Definition 6.3 (Relevant parent association). *We define relevant parents association of v matching u by a partial dual simulation, denoted by $\alpha(u,v)$ as a relation from $P(u)$ to $P(u,v)$ such that for each $u' \in P(u)$, if $P_{u'}(u,v) \neq \emptyset$, it is associated exactly one relevant parent vertex from $P_{u'}(u,v)$.*

Definition 6.4 (Relevant neighbors association). *We define relevant neighbors association of v matching u by a partial dual simulation, denoted by $\Gamma(u,v)$, as a tuple $(\alpha(u,v),\beta(u,v))$ such that $\alpha(u,v)$ and $\beta(u,v)$ are respectively a relevant parent and child associations. We put $(u',v') \in \Gamma(u,v)$ if $(u',v') \in \alpha(u,v)$ or $(u',v') \in \beta(u,v)$ and $|\Gamma(u,v)| = |\alpha(u,v)| + |\beta(u,v)|$.*

Definition 6.5 (Relevant partial dual simulation match). *Relevant partial dual simulation match (PDSM) of v matching u is a relation in $M_P^D(Q,G)$ denoted by $S_P^D(u,v)$, such that for each $(u_1,v_1) \in S_P^D(u,v)$ it is associated a $\Gamma(u_1,v_1)$; and $(u_1,v_1) \in S_P^D(u,v)$ if :*

1. $(u_1,v_1) = (u,v)$; or
2. $\exists\ (u_2,v_2) \in S_P^D(u,v)$ where $(u_1,v_1) \in \alpha(u_2,v_2)$ and $(u_2,v_2) \in \beta(u_1,v_1)$; or $\exists\ (u_2,v_2) \in S_P^D(u,v)$ where $(u_1,v_1) \in \beta(u_2,v_2)$ and $(u_2,v_2) \in \alpha(u_1,v_1)$.

This relevant PDSM represents a subgraph where the match tuple (u,v) has a unique relevant neighbor association $\Gamma(u,v)$ and recursively for each $(u',v') \in \Gamma(u,v)$. A match tuple $(u,v) \in M_P^D(Q,G)$ is given a relevance value based on a relevant PDSM and indicates the preserved relations in that PDSM. For the same reasons as in partial simulation, the relevance function for relevant PDSM is determined by whether or not neighbor vertices are included within a cycle. However, in order to recognize cycles in a query graph that can be matched by partial dual simulation, the query graph is considered as an undirected graph because of the duality dictated by dual simulation. Remember that, if the query graph Q is considered as an undirected graph and there is no cycle in it, then Q is a tree. Thus, the *tree relevance*‘ $|S_P^D(u,v)|_{tree}$ of a relevant PDSM is calculated from neighbors $(u',v') \in \Gamma_{tree}(u,v) = (\alpha_{tree}(u,v),\beta_{tree}(u,v))$ such that u' is not in the same undirected cycle as u and given as follows:

$$|S_P^D(u,v)|_{tree} = |\Gamma_{tree}(u,v))| + \sum_{(u',v') \in \Gamma_{tree}(u,v)} |S_P^D(u',v')|_u \quad (5)$$

In this Equation 5, the tree relevance of a relevant PDSM, $|S_P^D(u,v)|_{tree}$, is given by the addition of the number of edges directly materialized by tree neighbor vertices in given by $|\Gamma_{tree}(u,v))|$ and the sum of their relevances $|S_P^D(u',v')|_u$ which stand to the relevance of v' matching u' in S_P^D and its neighbors except match of u. This condition is needed to ensure that materialized edges are counted only once in a $S_P^D(u,v$. The cyclic relevance of a relevant PDSM $|S_P^D(u,v)|_{cyc}$, when u may be included in a cycle in the undirected query graph, is estimated according to the relation in the relevant PDSM called relevant dual connected component, denoted by $dcc_p(u,v)$, such that $dcc_p(u,v) \subseteq S_p^D(u,v)$ and $(u_1,v_1) \in dcc_p(u,v)$ if :

1. $(u_1,v_1) = (u,v)$; or
2. $\exists\ (u_2,v_2) \in dcc_p(u,v)$ such that $(u_1,v_1) \in \Gamma_{cyc}(u_2,v_2')$;

Form this last definition, we determine materialized edges $E_{dcc}(u,v)$ and materialized vertices $V_{dcc}(u,v)$ of an $dcc_p(u,v)$ as follows:

1. $V_{dcc}(u,v) = \{u'/(u',v') \in dcc_p(u,v)\}$
2. $E_{dcc}(u,v) = \{(u',u'')/(u',v') \in dcc_p(u,v)$ and $(u'',v'') \in \Gamma_{cyc}(u',v')\}$

The cyclic relevance of a relevant PDSM $S_P^D(u,v)$ is given as the number of materialized edges in the relevant dcc, given by:

$$|S_P^D(u,v)|_{cyc} = |E_{dcc}(u,v)| \quad (6)$$

The general relevance of a relevant PDSM, represented by $|S_P^D(u,v)|$, specifies the materialized edges preserved by v when matching u and its relevant neighbors, which match query vertices regardless of whether they are in the same undirected cycle as u.

$$|S_P^D(u,v)| = |S_P^D(u,v)|_{cyc} + \sum_{u' \in V_{dcc}} \max_{(u',v') \in dcc_P(u,v)} |S_P^D(u',v')|_{tree} \quad (7)$$

6.2. Partial dual simulation rank

From all the possible PDSM relevance values of $(u,v) \in M_P^D(Q,G)$ according to different relevant PDSM, we call *Partial dual simulation rank*, denoted by $\Delta^D(u,v)$, the maximum possible relevance and given by:

$$\Delta^D(u,v) = \max_{S_P^D(u,v) \subseteq M_P^D(u,v)} |S_P^D(u,v)| \quad (8)$$

7. DISTRIBUTED ALGORITHMS FOR GRAPH PARTIAL (DUAL) SIMULATION PATTERN MATCHING

Next, we propose distributed algorithms to calculate the maximum match of a query graph in a data graph by partial simulation and partial dual simulation with a rank for each match vertex. These algorithms follow the vertex-centric programming model introduced in [30]. Here, the algorithms are presented as a function from the point of view of a vertex that has the possibility to send messages to neighbor vertices. Generally, the algorithm is executed in a Bulk Synchronous Parallel (BSP) framework which ensures that vertex-functions and message passing are executed synchronously. Hence, the distributed algorithms terminate when all vertices vote to halt and there is no message to be processed. The inputs of our algorithms are a query graph Q and a data graph G and the output is the maximum match of Q in G by partial simulation (or partial dual simulation) with the rank of each vertex. We start by giving the *PSDAG* algorithm dedicated to the simple case where the pattern graph is acyclic. Next, we present the general algorithm *PS* for any type of query graphs and we finish by algorithms for partial dual simulation, one for tree query graphs *PDSTree* and a second for general patterns *PDS*.

7.1. PSDAG: Partial simulation GPM distributed algorithm for DAG query graphs

Since the pattern graph is a DAG, the vertex function in PSDAG algorithm is based on Equations 1 and 4, where a data vertex rank is the maximal DAG relevance.

$$\Delta(u,v) = \max_{S_P(u,v) \subseteq M_P(u,v)} |S_P(u,v)|_{dag}$$

To do so, we propose the next data structures representing the status of a data vertex v identified by vID:

- *matchSetRanks*: $Map\,[uID \rightarrow rank]$: to save the set of matches with their respective rank obtained in each superstep. uID is the identifier of the matched query vertex and $rank$ is the maximum relevance of the vertex vID matching uID by partial simulation at a specified superstep.
- *childRanks*: $Map\,[uID' \rightarrow rank]$: to store of the best relevance for each materialized child vertex identified by uID' from all possible relevant child vertices of vID.

In the first superstep of Algorithm 1, each vertex initiates its *matchSetRanks* by comparing its label with those of the query vertices. Whenever a label equivalence is found, it assigns the value "0". Then, if at least one match exists, the vertex sends a message to its parents containing its *status* represented by *matchSetRanks*. At the end of the superstep the data vertex votes to halt. The second superstep concerns the vertices that receive messages from their child vertices, i.e. those which have at least one match. Then, for each match uID in *matchSetRanks*, it filters the received matched vertices uID' according to the expected set of child vertices of uID in Q. After calculating the set of materialized child vertices, the new rank can be defined as the size of its this set $|\beta_{dag}(u,v)|$. The set of materialized child vertices can be determined by the intersection between the set of matched nodes uID' received from child vertices and the set of child vertices $C(uID)$ of the query vertex in the pattern graph. If there is any change in the ranks of vID, the vertex informs its parents by sending its new status. Besides that, the vertex initializes the *childRanks* map during this superstep. It adds any materialized child vertex used to calculate the partial simulation rank of a match by their identifiers uID' with the highest relevance

Algorithm 1: PSDAG: Partial simulation pattern matching for DAG

```
   Input: Q = (V_Q, E_Q, f_Q)
   Output: matchSetRanks
 1 Superstep 1:
 2   foreach uID ∈ V_Q do
 3     if f(vID) = f_Q(uID) then
 4       add [uID → 0] to matchSetRanks
 5   if matchSetRanks is nonempty then
 6     send matchSetRanks to P(vID)
 7   vote to halt
 8 Superstep 2:
 9   if matchSetRanks is nonempty then
10     foreach uID' ∈ (messages ∩ C(uID)) do
11       add [uID' → 0] to childRanks
12     update matchSetRanks with: uID →
         sizeOf(messages ∩ C(uID))
13     if any change in matchSetRanks then
14       send matchSetRanks to P(vID)
15   vote to halt
16 Superstep 3:
17   if matchSetRanks is nonempty then
18     foreach uID' ∈ (messages ∩ C(uID)) do
19       if messages(uID') > childRanks(uID')
           then
20         add [uID' → messages(uID')] to
             childRanks
21     foreach uID ∈ matchSetRanks do
22       foreach uID' ∈ C(uID) do
23         matchSetRanks(uID) +=
             childRanks(uID')
24     if any change in matchSetRanks then
25       send matchSetRanks to vID parents
26   vote to halt
```

	Supersteps			
Match	**1**	**2**	**3**	**4**
(u_1, v_1)	0	2	5	6
(u_2, v_2)	0	2	3	3
(u_3, v_3)	0	1	1	1
(u_3, v_4)	0	1	1	1
(u_4, v_5)	0	0	0	0
(u_4, v_6)	0	0	0	0
(u_5, v_7)	0	0	0	0

TABLE 3: Evolution of vertices' rank during different supersteps.

(which at this step is equal to 0). This map enables the vertex to reevaluate the rank of any match in following supersteps if one or more child vertices modify their rank. The third superstep and beyond concerns vertices that receive a message from child vertices that have changed the rank of one of their matches during the previous superstep. In this case, the vertex updates the *childRanks* map if there is any new value superior to the previous one, and reevaluates the ranks of its matches.

Example 7. In Table 3, we can see the evolution of ranks of each data vertices from the partial simulation relation presented in Example 2.

7.2. PS: Partial simulation distributed algorithm for generic query graphs

Basically, Algorithm 2, which treats the case of generic query graphs PS is similar to Algorithm 1. The only difference is that we must consider the materialized edges of the relevant *scc* for data vertices that match a query vertex involved in at least one cycle in Q. Otherwise, if a query vertex is not included within any circle, the vertex behaves identically to the *PSDAG* algorithm. Coming back to Equations 3 and 4, we can deduce the compute part of a matching data vertex. Actually, a data vertex behavior is defined by the process of finding the relevant *scc* that ensures the maximal size of materialized edges set $E_{scc}(u, v)$ with the maximum sum of DAG relevance values of the relevant vertices $V_{scc}(u, v)$. To achieve that, each vertex needs to keep trace of different possible relevant *scc* in order to select the rank corresponding to the highest relevance. Moreover, each vertex needs to send to its parents all possible relevant *scc* and not only the best one. This is necessary since the elected relevant *scc* for a vertex is not always the best for its parent, especially if we want to maximize the materialized edges set size. To do that, we start in the first superstep by constructing the initial match set as in *PSDAG* and for each match uID we identify its materialized child vertices included or not in the same *scc*. In the next sections we just present the data structure and supersteps of vertex that has a match included in cycle Q.

- *matchSetRanks* : $Map[uID \rightarrow Set\{SCCMatch\}]$, where $SCCMatch$ = [*sccEdges: Set of edges*; *dagRanks* : $Map\,(uID' \rightarrow rank)$]
- *ChildMatches* : $Map[uID' \rightarrow Set\{SCCMatch\}]$: to save the $SCCMatch$ data structures for each materialized child from all possible relevant child vertices.

Algorithm 2: PS: Partial simulation pattern matching

```
   Input: Q = (V_Q, E_Q, f_Q)
   Output: matchSetRanks
1  Superstep 1:
2      foreach uID ∈ V_Q do
3          if f(vID) = f_Q(uID) then
4              add
                [uID → Set(SCCMatch(set.empty, map.empty))]
               to matchSetRanks;
5      if matchSetRanks is nonempty then
6          send matchSetRanks to vID parents;
7      vote to halt;
8  Superstep 2:
9      foreach uID ∈ matchSetRanks do
10         foreach uID' ∈ Messages do
11             if uID' ∈ SCC and uID' ∈ C(uID)
                then
12                 add (uID, uID') to
                    SCCMatch.sccEdges
13             else if uID' ∈ C(uID) then
14                 increment uID →
                    SCCMatch.dagRanks(uID) + 1
15             update ChildMatches
16         if any change in matchSetRanks then
17             send matchSetRanks to P(vID)
18     vote to halt;
19 Superstep 3:
20     foreach uID ∈ matchSetRanks do
21         Calculate DAG Rank from child
            uID' ∉ SCC as PSDAG
22         if uID ∈ SCC then
23             construct different possible associations
                of SCCMatch form child matches
                uID' ∈ SCC
24             Optimize set of SCCMatch in
                MatchSetRanks(uID)
25         if any change in matchSetRanks then
26             send matchSetRanks tovID parents;
27     vote to halt;
```

The new *matchSetRanks* data structure maps each match uID to a set of *SCCMatch* elements, each of which represents a unique relevant *scc*. A set data structure called *sccEdges* is used to store materialized edges within an *SCCMatch*. If the matched cycle is longer than the original in the pattern graph, this data structure type prevents repetition of materialized edges.

Also, there is a map *dagRanks* used to keep track of the best DAG relevance of each materialized vertex in V_{scc}. The rank of vID that matches uID at a specific superstep corresponds to the *SCCMatch* that maximizes the size of the set *SCCMatch* with the sum of DAG relevance values from *dagRanks* as in Equations 4 and 3.

The difference with Algorithm 1 starts in the second superstep when a vertex receives the first messages from its child vertices. At this stage each match uID has exactly one *SCCMatch* representing edges directly materialized by the match tuple (u, v). Each time there is a match uID' from child vertices in the same *scc*, we add the edge (uID, uID') to *sccEdges* set. The *dagRanks* map, contains one tuple $(uID \rightarrow rank)$, where the $rank$ is equal to the number of materialized child vertices that are not in the same *scc* as in Algorithm 1. In fact, a vertex follows exactly Algorithm 1 to evaluate its DAG relevance of a mach from vertices that materialize a child not included in the same *scc*. *childMatches* is initiated by giving to each materialized child an empty set of *SCCMatch*. The third superstep and beyond means that a child vertex has updated its status *matchSetRanks*. Therefore, the vertex needs to update its own status by calculating all potential *SCCMatch* that correspond to probable relevant $scc_p(u, v)$ as explained in Subsection 5.2. In Algorithm 2, this situation is expressed by the fact that a materialized child uID' in *childMatches* may have multiple *SCCMatch*. These sets will be used for a match uID to construct all possible associations by taking an *SCCMatch* from each needed materialized child uID'. Each association gives a new $SCCMatch$ realized by the union of *sccEdges* sets and a *dagRanks* map that takes the maximum DAG Rank for each match.

Optimization: To reduce the number of *SCCMatch* occurrences in *matchSetRanks* and *childMatches*, we propose an optimization function that removes those that can not contribute further to the computation of the vertex's or its parents' relevance. Indeed, $SCCMatch'$ can be omitted if it exists $SCCMatch$ such that:

- $SCCMatch'.sccEdges \subseteq SCCMatch.SCCEdges$, and
- $\forall(uID \rightarrow rank) \in sccMatch'.dagRanks$, $SCCMatch'.dagRanks(uID) \leq SCCMatch.dagRanks(uID)$.

These two conditions mean that all the edges from the relevant *scc* defined by $SCCMatch'$ are reached by $SCCMatch$ and all DAG ranks of matches from the relevant *scc* of $SCCMatch'$ are inferior or equal to the corresponding ranks in $SCCMatch$.

Data graph	Vertices	Edges	Labels
Amazon	403 394	3 387 388	100
WikiTalk	2 394 385	5 021 410	250
Flickr	2 302 925	33 140 017	500
LiveJournal	4 847 571	68 993 773	500

TABLE 4: Data graphs characteristics.

Class	Cyclic	DAG	Tree
1	(3,6)	(3,3)	(6,5)
2	(5,10)	(5,10)	(10,9)
3	(10,20)	(10,20)	(15,14)
4	(15,30)	(15,30)	(20,19)
5	(20,40)	(20,40)	(25,24)
6	(25,50)	(25,50)	(50,49)

TABLE 5: The ordered pair composed by the order $|V|$ and size $|E|$ of query graphs, respectively.

7.3. PDSTree and PDS: Partial dual simulation distributed algorithms

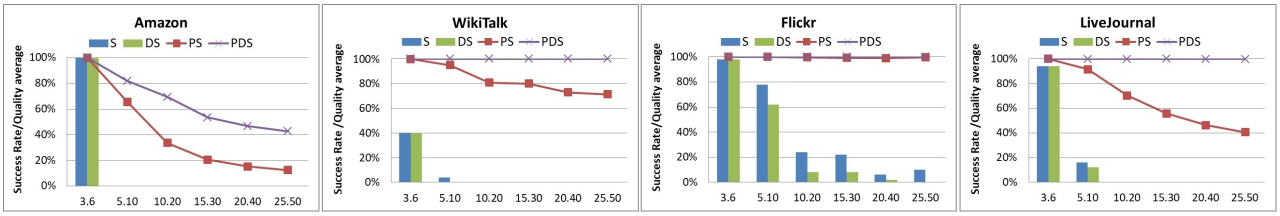
For partial dual simulation GPM the two algorithms,$PDSTree$ and PDS, are quite similar to those of partial simulation. However there are two key differences:

1. A data vertex communicate by sending and receiving messages with its child and parent vertices.
2. According to this first point and equations 5 and 7, the message exchanged between neighbors depends on the match vertex destination.

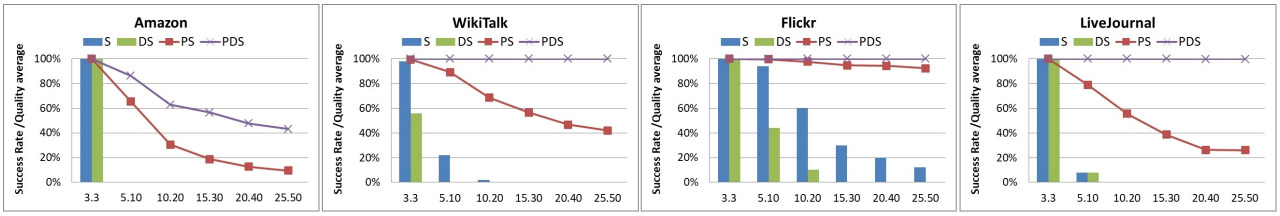
In partial simulation algorithms message transmitted to parents depends only on the match of the source data vertex. Whereas in partial dual simulation algorithms each time a vertex communicates with its neighbor that has a match uID', it has to calculate a message variant using all neighbors that not include a materialized child vertex uID'. For example, in the case of general pattern graph, the association calculation phase is applied to calculate the rank of a vertex and also has to be done to calculate the messages to be sent to each materialized neighbor vertex. Also, $childMatches$ data structure has to evolve to meet the requirement of that information received from neighbors, which dependent on the destined match of the neighbor data vertex: $ChildMatches : Map[uID \rightarrow Map[uID' \rightarrow Set\{SCCMatch\}]]$: to save the $SCCMatch$ data structures for each materialized neighbor match uID' sent to a match uID.

8. EXPERIMENTS

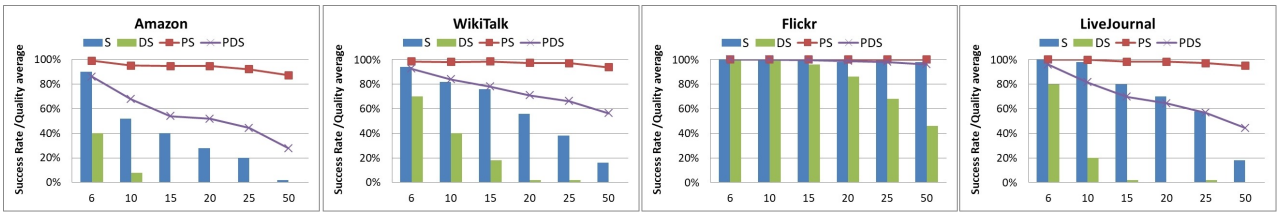
The goal of this experimental study is to evaluate our approaches and verify the effectiveness and the efficiency of the proposed distributed algorithms for ranked partial simulation and partial dual simulation, over various types of real data graphs with different number of labels, size of query graphs and topology.



(a) Cyclic patterns



(b) DAG patterns



(c) Tree patterns

FIGURE 3: Effectiveness of partial simulation approaches by data graph and pattern size.

8.1. Experimental setting

8.1.1. Environment

We realize all our experiments using GraphX [31], which is the Apache Spark [32] API for graph and graph parallel processing. It provides a set of operators to support graph computation, including a Pregel interface to implement vertex-centric like algorithms that we have used in our tests. All experiments are conducted on a Spark 2.3.1 cluster composed of 9 linux based nodes with 16 processor cores and 32Gb of memory in each one; one node plays the role of the master while the remaining nodes are considered as workers.

8.1.2. Datasets and query graphs

Four real datasets are used in the next experiments Amazon, WikiTalk, LiveJournal [33] and Flickr[34]. The characteristics of these data graphs are presented in Table 4. The Amazon data graph represents a network where vertices are products, and a directed edge from vertex A to B denotes that product A is frequently co-purchased with product B. LiveJournal and Flickr directed graphs represent the social networks of their users, linked by edges that indicates the relation of friendship between them. This friendship is a directed relation because both platforms allow people to declare which other members are their friends without requiring reciprocity. Finally, nodes in WikiTalk network represent Wikipedia users and a directed edge from node A to node B represents that user A at least once edited a talk page of user B. For all datasets, we generate synthetic labels, as shown in Table 4, randomly distributed on graph vertices.

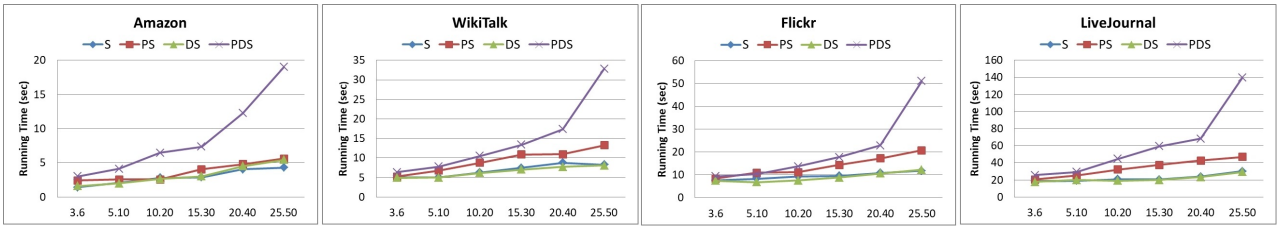
Regarding query graphs, we generate three sets of query graphs according to the various number of labels in data graphs (100, 250 and 500). For each set of query graphs, we generate three types of graphs, cyclic, DAG and tree with different sizes, depending on the number of vertices $|V_Q|$ and edges $|E_Q|$ as shown in Table 5. Every class contains 50 different graph randomly generated using jGrapht library [35] and all query vertices are randomly labeled.

8.1.3. Implementations

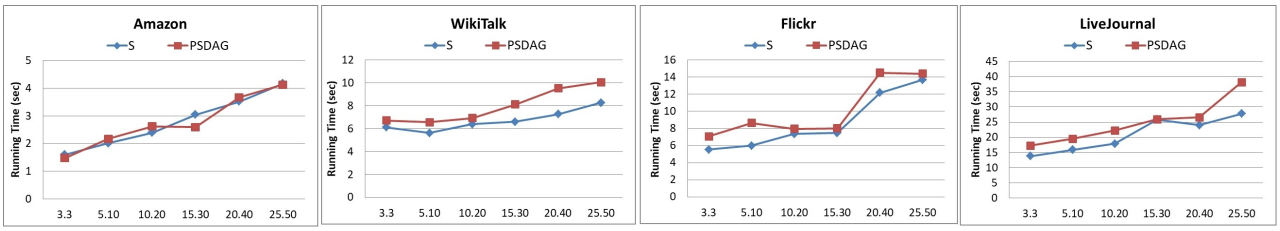
In addition to simulation S and dual simulation DS GPM algorithms proposed in [22], we implement partial simulation and partial dual simulation GPM algorithms for different cases of query graphs types, respectively: PS and $PSDAG$ for partial simulation GPM in the case of generic and DAG query graphs, PDS and $PDSTree$ for partial dual simulation GPM in general cases and the special case of tree patterns.

8.2. Experimental results

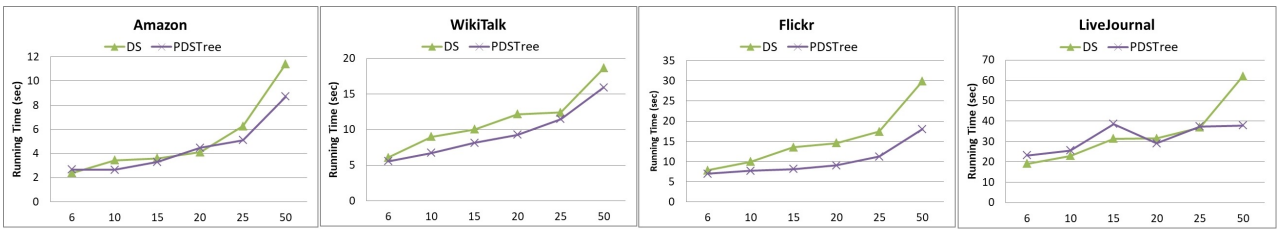
We next present our findings from different experiments.



(a) Cyclic patterns



(b) DAG patterns



(c) Tree patterns

FIGURE 4: Running time of different algorithms by data graph and pattern size.

8.2.1. Effectiveness

First, we define $\Delta(Q,G)$ called the *Partial simulation grade* of a query graph Q in a data graph G as the sum of the best partial simulation ranks of each query vertex that can be materialized in $M_P(Q,G)$:

$$\Delta(Q,G) = \sum_{u \in V_Q} \max_{(u,v) \in M_P(Q,G)} \Delta(u,v)$$

Then, the *Partial simulation quality* is defined as the ratio between $\Delta(Q,G)$ and the maximum partial simulation grade that can be achieved from the query graph Q in any possible data graph. This situation corresponds to the grade of the perfect match $\Delta(Q,Q)$. This ratio is less than or equal to 1 and is indicated by

$$\rho(Q,G) = \frac{\Delta(Q,G)}{\Delta(Q,Q)}$$

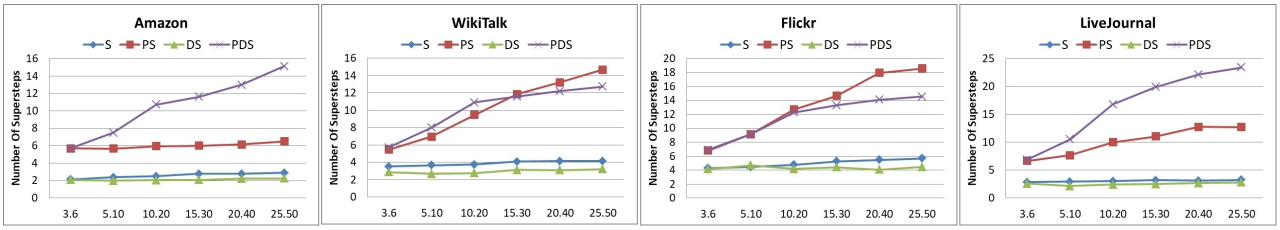
.

Second, we define the simulation GPM success rate for a query graph class as the ratio of matching query graphs to the total number of patterns in the class. Following that, we compare the mean of partial simulation quality values to the simulation GPM success rate for each class of query graphs to demonstrate the effectiveness of our proposed partial simulation GPM. By comparing the two, we can determine what advantages a partial simulation GPM has over a simulation GPM.

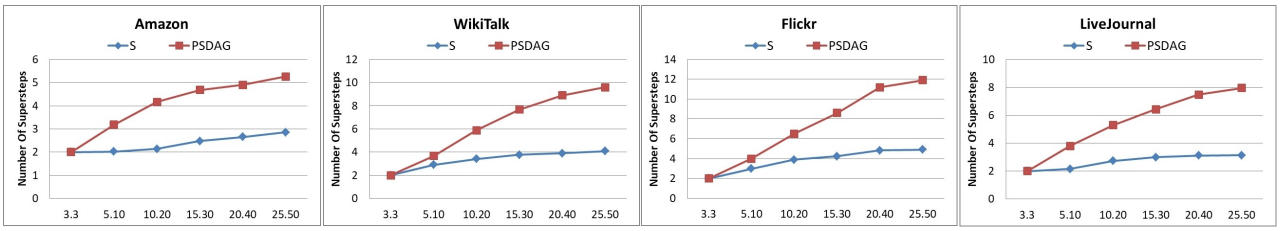
The same definitions are applicable respectively for partial dual simulation GPM.(the same definitions are applicable to partial dual simulation).

In Figure 3, we show histograms representing the success rate of simulation and dual simulation GPM, as well as the quality of partial simulation and partial dual simulation GPM, for each data graph and query graph class. These charts show that the number of simulation and dual simulation GPM matches reduces as pattern size grows. In the cases of Amazon, Livejournal, and WikiTalk data graphs with generic and DAG query graphs, the success rate drops to zero starting from the third pattern size class. There are more positive matches for tree pattern graphs, but just a minority for dual simulation and the largest pattern graphs. Also, Flickr surpasses other data graphs, particularly for tree pattern graphs. Nevertheless, for other types of patterns, the success rate for simulation GPM remains less than 20% and less than 10% for dual simulation GPM starting from pattern graphs in the third class.

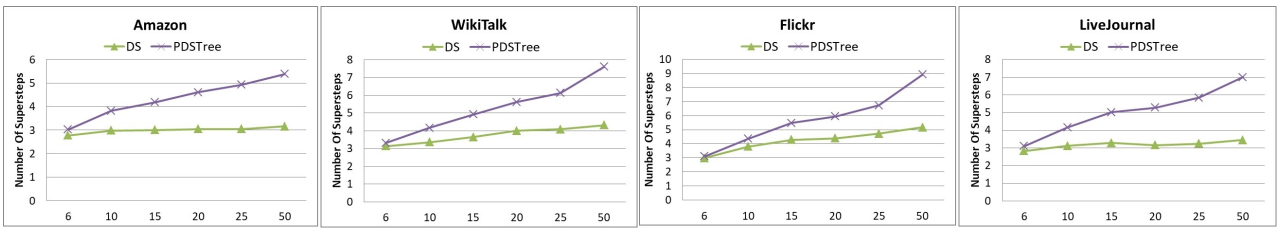
On the other hand, partial methods for pattern matching delivered acceptable quality when there were few or no matches at all using S and DS algorithms. For instance, S returns no matches for generic patterns in the Live Journal data graph since the third class, whereas PS produces matches with valuable quality reaching 58% of preserved edges from query graphs with 30 edges and 40% from patterns with 50 edges.



(a) Cyclic patterns



(b) DAG patterns



(c) Tree patterns

FIGURE 5: Number of supersteps of different algorithms by data graph and pattern size.

Additionally, if a match is found using graph simulation or dual simulation, partial simulation GPM approaches always produce a match with the greatest possible score of 100%. Besides that, *PDS* beats *PS* in generic and DAG patterns due to the existence of cycles that allow *PDS* to match larger cycles by locating a potential neighbor vertex, whether child or parent.

From the above, it is clear how partial simulation approaches are effective since they enable users to acquire partial results in the absence of a complete simulation match. This is extremely useful for exploring large datasets or determining the best result for a query graph.

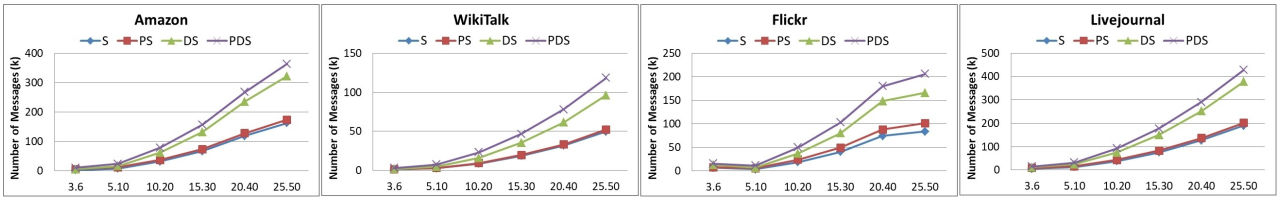
8.2.2. Efficiency

We evaluate the efficiency of the proposed algorithms in terms of execution time, number of supersteps and number of exchanged messages, according to each query graph type (cyclic, DAG or tree) and size in different data graphs (Amazon, Wikitalk, Flickr and Livejournal).

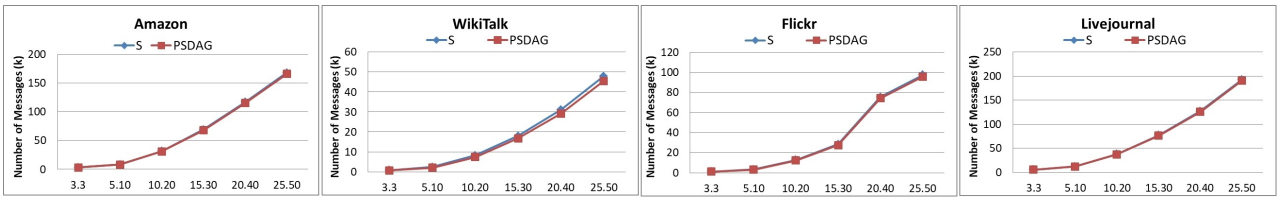
a) Running Time. We clearly notice in Figure 4 that the execution time for all algorithms in all provided data graphs increases as the size of the data graphs or query graphs grows. Because, as expected, the number of candidate vertices to a match and the number of relationships (child or parent) to verify increases.

First, when the pattern is cyclic, we can see in Figure 4 that *PS* scored values equal to or greater than *S* by a relatively tiny percentage between 1% and 60%. The gap between *PDS* and *DS* algorithms is much more substantial, reaching 5 times. This result can not be explained only by the computational complexity of the partial GPM algorithms, but by the low success rate of simulation and dual simulation GPM as shown in the effectiveness section. Moreover, *S* and *DS* algorithms terminate earlier, because a data vertex is flagged as unmatched as soon as any condition is not verified, although in partial versions, a vertex continues computations as long as it has at least one matched child or parent vertex. Additionally, matching cycles by partial simulation and partial dual simulation GPM may be longer than those in simulation and dual simulation GPM resulting in more significant running time, as we will check in the coming section about the number of supersteps.

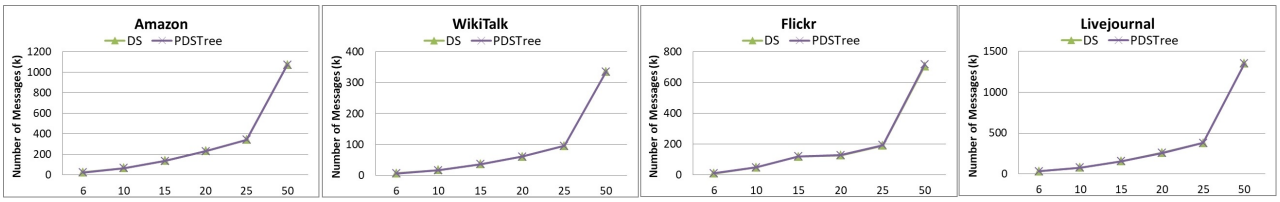
From charts dedicated to DAG patterns in Figure 4, *PSDAG* was able to achieve approximately the same running duration as *S* implementation, independently from pattern size or data graph. Similarly, in the third line of Figure 4 we can observe that in most cases, *PDSTree* outperforms *DS* implementation. These results demonstrate the efficiency of our *PSDAG* and *PDSTree* implementations and provide an efficient alternative to *S* and *PS* algorithms provided in [22], in addition to the ability to catch and evaluate incomplete



(a) Cyclic patterns



(b) DAG patterns



(c) Tree patterns

FIGURE 6: Number of exchanged messages of different algorithms by data graph and pattern size.

matches.

b) Supersteps. As shown in the first series of graphs (a) in Figure 5 we can directly distinguish that *PS* and *PDS* need more supersteps to terminate. This result is directly related to the low success rate of *S* and *DS* and to the existence of cycles in patterns as explained previously. In fact, because we use random query graphs that in most cases have no match (as shown in the effectiveness section), the number of supersteps for *S* and *DS* is nearly constant, indicating that the verification process declares all data vertices as unmatched after nearly the same number of supersteps. Besides that, partial simulation approaches accentuate the phenomena of longer matched cycles in simulation GPM, since they have more relaxed constraints and, as a result, a greater chance of generating longer matching cycles. The major drawback is the direct impact on running time scores but, on the other hand, our methods succeed in returning a partial result when the old approaches returns none.

When *PSDAG* and *PDSTree* are applied to patterns without cycles, as seen in (b) and (c) series from Figure 5, the number of supersteps is significantly reduced and follow a linear growth as the pattern graph size increases. These results confirm the efficiency of partial simulations implementations in the case of acyclic pattern graphs because of the reduced number of supersteps, linearly linked to the size of the query and data graphs.

c) Messages. As predicted, the number of messages exchanged is proportional to the query graph's size. More importantly, all charts in Figure 6 demonstrate the efficiency of our proposed algorithms for partial simulation approaches because, despite the increased number of supersteps (2 to 4 times), it generates only 60% more messages for cyclic patterns and practically the same number of messages when *PSDAG* and *PDSTree* are applied.

9. CONCLUSION

In this paper, we proposed partial simulation and partial dual simulation as new inexact GPM approaches capable of catching meaningful results when graph simulation and dual simulation GPM can not find matches. These new graph analysis techniques give the opportunity to catch vertices from a data graph that match query vertices based on graph simulation or dual simulation GPM applied to a pattern graph with possible missing nodes and/or edges. Furthermore, we introduced relevance functions that assign a quality value to each matching vertex as a criterion for ranking outputs according to their ability to preserve pattern graph topology. These new GPM techniques are critical for a wide variety of application fields, especially considering everyone knows that big data suffers from uncertainty issues.

The distributed implementations succeeded in scoring acceptable times in case of cyclic patterns and equaled or outperforms the performance of simulation and dual simulation implementations when the pattern graphs are respectively a DAG or a tree. In the same

time, it is worth remembering that partial simulation GPM techniques consistently succeed in returning results while others declare unmatched patterns in early phases. The performance results in the case of cyclic patterns are caused by the much longer matched cycles than the originals. This characteristic of simulation GPM is well-known and several propositions were made to reduce this unwanted discrepancy. The next step would be to extend our proposed model to the pattern matching by strong simulation [11] that adds locality to dual simulation and allows to limit the size of the matched cycle if there is one in the pattern. This work would help to reduce the execution time in case of generic pattern graphs, while preserving their topology. Subgraph mining and dynamic graphs are other applications that can benefit from the proposed inexact pattern matching, which are important in many modern applications.

10. DATA AVAILABILITY STATEMENT

The implementation and experiments data used to support the findings of this paper are available from the corresponding author upon request.

ACKNOWLEDGEMENTS

The experiments presented in this work were carried out using the High Performance Computing Platform IBNBADIS provided by the Research Center on Scientifc and Technical Information - CERIST (Algeria).

REFERENCES

[1] Meta reports second quarter 2022 results. `https://investor.fb.com/investor-news/press-release-details/2022/Meta-Reports-Second-Quarter-2022-Results/default.aspx`. Accessed on 2022-08-12.

[2] Gallagher, B. (2006) Matching structure and semantics: A survey on graph-based pattern matching. *AAAI Fall Symposium: Capturing and Using Patterns for Evidence Detection*, Washington, DC, 13–15 October, pp. 45–53. AAAI Press.

[3] Ullmann, J. R. (1976) An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, **23**, 31–42.

[4] Boldi, P., Bonchi, F., Castillo, C., and Vigna, S. (2009) From "Dango" to "Japanese Cakes": Query Reformulation Models and Patterns. *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, Milan, Italy, 15-18 September, pp. 183–190. IEEE.

[5] Vasilyeva, E., Thiele, M., Bornhövd, C., and Lehner, W. (2016) Answering "Why Empty?" and "Why So Many?" queries in graph databases. *Journal of Computer and System Sciences*, **82**, 3–22.

[6] Stotz, A., Nagi, R., and Sudit, M. (2009) Incremental Graph Matching for Situation Awareness. *12th International Conference on Information Fusion Seattle, WA, USA,*, Seattle, WA, USA, July, pp. 452–459. IEEE.

[7] Gross, G., Nagi, R., and Sambhoos, K. (2014) A fuzzy graph matching approach in intelligence analysis and maintenance of continuous situational awareness. *Information Fusion*, **18**, 43–61. Publisher: Elsevier.

[8] Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004) Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, **18**, 265–298. Publisher: World Scientific.

[9] Riesen, K., Jiang, X., and Bunke, H. (2010) Exact and Inexact Graph Matching: Methodology and Applications. In Aggarwal, C. C. and Wang, H. (eds.), *Managing and Mining Graph Data*, pp. 217–247. Springer US, Boston, MA.

[10] Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., and Wu, Y. (2010) Graph pattern matching: from intractable to polynomial time. *Proceedings of the VLDB Endowment*, **3**, 264–275.

[11] Ma, S., Cao, Y., Fan, W., Huai, J., and Wo, T. (2011) Capturing topology in graph pattern matching. *Proceedings of the VLDB Endowment*, **5**, 310–321.

[12] Fard, A., Abdolrashidi, A., Ramaswamy, L., and Miller, J. (2012) Towards Efficient Query Processing on Massive Time-Evolving Graphs. *Proceedings of the 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, Pittsburgh, United States, October, pp. 567–574. IEEE.

[13] Brynielsson, J., Horndahl, A., Johansson, F., Kaati, L., Mårtenson, C., and Svenson, P. (2013) Harvesting and analysis of weak signals for detecting lone wolf terrorists. *Security Informatics*, **2**, 1–15.

[14] Choudhary, P. and Singh, U. (2015) Article: A survey on social network analysis for counter-terrorism. *International Journal of Computer Applications*, **112**, 24–29.

[15] Schuurman, B., Bakker, E., Gill, P., and Bouhana, N. (2018) Lone Actor Terrorist Attack Planning and Preparation: A Data-Driven Analysis,. *Journal of Forensic Sciences*, **63**, 1191–1200.

[16] Svenson, P., Svensson, P., and Tullberg, H. (2006) Social network analysis and information fusion for anti-terrorism. *Proceedings of the conference on civil and military readiness*, Enköping, Sweden, 16-18 May. Försvarets Materielverk.

[17] Krebs, V. E. (2002) Mapping networks of terrorist cells. *Connections*, **24**, 43–52.

[18] Sparrow, M. K. (1991) The application of network analysis to criminal intelligence: An assessment of the prospects. *Social networks*, **13**, 251–274.

[19] Coffman, T., Greenblatt, S., and Marcus, S. (2004) Graph-based technologies for intelligence analysis. *Communications of the ACM*, **47**, 45.

[20] Cordella, L. P., Foggia, P., Sansone, C., and Vento, M. (2004) A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, **26**, 1367–1372.

[21] Henzinger, M. R., Henzinger, T. A., and Kopke, P. W. (1995) Computing simulations on finite and infinite graphs. *Proceedings of IEEE 36th Annual Foundations*

of Computer Science, USA, 23-25 October, pp. 453–462. IEEE.

[22] Fard, A., Nisar, M. U., Ramaswamy, L., Miller, J. A., and Saltz, M. (2013) A distributed vertex-centric approach for pattern matching in massive graphs. *2013 IEEE International Conference on Big Data*, Silicon Valley, CA, USA, October, pp. 403–411. IEEE.

[23] Bouhenni, S., Yahiaoui, S., Nouali-Taboudjemat, N., and Kheddouci, H. (2021) A survey on distributed graph pattern matching in massive graphs. *ACM Computing Surveys (CSUR)*, **54**, 1–35.

[24] Fan, W., Li, J., Ma, S., Tang, N., and Wu, Y. (2011) Adding regular expressions to graph reachability and pattern queries. *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, Hannover, Germany, 11–16 April, pp. 39–50. IEEE.

[25] Gao, J., Liu, P., Kang, X., Zhang, L., and Wang, J. (2016) PRS: Parallel relaxation simulation for massive graphs. *The Computer Journal*, **59**, 848–860.

[26] Habi, A., Effantin, B., and Kheddouci, H. (2019) Diversified top-k search with relaxed graph simulation. *Social Network Analysis and Mining*, **9**, 1–15.

[27] Fan, W., Wang, X., and Wu, Y. (2013) Diversified top-k graph pattern matching. *Proceedings of the VLDB Endowment*, **6**, 1510–1521.

[28] Chen, L., Gupta, A., and Kurul, M. E. (2005) Stack-based algorithms for pattern matching on dags. *Proceedings of the 31st International Conference on Very Large Data Bases*, Trondheim, Norway, pp. 493 – 504. VLDB Endowment.

[29] Chang, L., Lin, X., Zhang, W., Yu, J. X., Zhang, Y., and Qin, L. (2015) Optimal enumeration: efficient top-k tree matching. *Proceedings of the VLDB Endowment*, **8**, 533–544.

[30] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010) Pregel: A system for large-scale graph processing. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, Indianapolis, USA, June, pp. 135–146. ACM.

[31] Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., and Stoica, I. (2014) GraphX: Graph processing in a distributed dataflow framework. *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, Broomfield, CO, October, pp. 599–613. USENIX Association.

[32] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., Stoica, I., and others (2010) Spark: Cluster computing with working sets. *HotCloud*, **10**, 95.

[33] [Dataset] Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford Large Network Dataset Collection. `http://snap.stanford.edu/data`. Accessed on 2022-08-12.

[34] [Dataset] Mislove, A., Koppula, H. S., Gummadi, K. P., Druschel, P., and Bhattacharjee, B. (2008) Growth of the Flickr Social Network. *Proceedings of the 1st ACM SIGCOMM Workshop on Social Networks (WOSN'08)*, Seattle, WA, August, pp. 25–30. ACM.

[35] Michail, D., Kinable, J., Naveh, B., and Sichi, J. V. (2020) JGraphT—A Java Library for Graph Data Structures and Algorithms. *ACM Trans. Math. Softw.*, **46**.