Extending Graph Rules with Oracles

XueLi Liu
Tianjin University, China
xueli@tju.edu.cn

Bowen Dong
Tianjin University, China
1020201107@tju.edu.cn

Nannan Wu
Tianjin University, China
nannan.wu@tju.edu.cn

Wenzhi Fu
University of Edinburgh, UK
wenzhi.fu@ed.ac.uk

Xin Wang
Tianjin University, China
wangx@tju.edu.cn

Wenjun Wang*
Tianjin University, China
wjwang@tju.edu.cn

ABSTRACT
This paper proposes a class of graph rules for deducing associations between entities, referred to as Graph Rules with Oracles and denoted by GROs. As opposed to previous graph rules, GROs support oracle functions to import (a) external knowledge, and (b) internal computations such as aggregate operators and machine learning predicates, and so on. Moreover, the semantics of GROs are defined in terms of pivoted dual simulation, in contrast to the subgraph isomorphism. We show how GROs can be used to predict links and catch anomalies, among other things. We formalize the association deduction problem with GROs in terms of the chase, and prove their Church-Rosser property. We show that both the deduction and incremental deduction problems with GROs are in PTIME, as opposed to the intractability of their counterparts with prior graph rules. We also provide sequential and parallel algorithms for association deduction and incremental deduction. Using real-life and synthetic graphs, we experimentally verify the effectiveness, scalability, and efficiency of the algorithms.

PVLDB Reference Format:
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/simplesunnylife/GROs.

1 INTRODUCTION
Rules for property graphs have gained attention in recent years, proving effective in various areas, including entity integrity, recommendation, query optimization, and consistency checks [6, 10, 13, 14, 19]. As opposed to relational databases, real-life graphs often do not come with a schema. To define rules for such graphs $G$, two essential components are needed. Firstly, a topological constraint $Q$ is utilized to identify associated entities within $G$, thereby specifying the scope of the rule application. Secondly, a dependency $X \rightarrow Y$ is established to capture the relationships between the attributes of the identified entities. By incorporating these components, researchers can define powerful rules that facilitate the comprehensive analysis and inference on complex graph structures.

A range of rules with varying expressive power have been developed by combining the topological constraint $Q$ with different syntax of dependency $X \rightarrow Y$. Graph Functional Dependencies (GFDs)[6] extend conditional functional dependencies (CFDs)[9] to graphs, supporting bindings of semantically related constants with constant literals represented as $x.A = c$, which associates a constant $c$ with the attribute $A$ of an entity $x$. After GFDs, several extensions have been studied. For instance, NGFDs [13] supporting linear arithmetic expressions and built-in comparison predicates, GEDs [14] supporting keys in graphs, and GARs [10] unifying ML classification predicates.

However, there are two issues with these graph rules. First, there are still some important semantic associations that are commonly found in applications but cannot be specified. For example, none of the prior rules is capable of expressing aggregation and regular expressions, which are crucial for tasks such as fraud detection, anomaly detection, and data validation. Second, the semantics of these rules is interpreted in terms of subgraph isomorphism or homomorphism. To apply these rules in a graph $G$, all subgraphs of $G$ that is isomorphic (homomorphism) to $Q$ must be computed, and then $X \rightarrow Y$ can be applied. This process is computationally intractable (cf. [26]), which hampers the applicability of these rules.

To overcome these limitations, there is a need to develop new approaches that can express more semantic associations within the rules. This will enhance their capabilities in existing applications and enable their application in crucial areas such as real-time risk detection and anomaly detection. Additionally, alternative techniques should be explored to address the computational challenges associated with the application of these rules, ensuring their practicality and scalability in real-world scenarios.

Example 1: Consider the following real-life examples.
(1) Collaboration recommendation. A practical rule employs an anomaly subgraph detection algorithm [4] to identify up-curve research teams and their core members, using external data sources that are not included in the collaboration recommender system. If researcher $x$ and $y$ are in the same field and $y$ is a core member of a up-curve team, then $y$ is recommended $x$ for collaboration. This rule can facilitate the formation of more productive research partnerships, which requires external data access and complex computations, and cannot be achieved using existing graph rules.

(2) Fraud detection. A money laundering detection rule states that if individual $x$ receives funds from others and transfers out a signif-

* corresponding author
This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
doi:XX.XX/XXX.XX
icant portion, such as 70\% of the received funds, it raises suspicion of potential involvement in money laundering activities. This rule enhances the ability to identify and prevent illicit financial activities. However, expressing this rule requires the use of aggregation operators, like sum, which are not supported by existing rules.

(3) **Art exhibitions recommendation.** An artwork exhibition gallery recommendation rule states that if an artist \( x \) creates a work \( y \), and gallery \( z \)'s reputation aligns with \( x \)'s influence, then \( y \) should be recommended for exhibition at \( z \). This rule enhances exhibition success and advances artists' careers. However, it depends on statistical methods from [25] to quantify reputation and influence, which existing rules do not support.

(4) **Event detection.** An anomaly event detection rule predicts that if more than 3 people post tweets in the same location \( z \) and discuss the same event \( x \), then \( x \) is occurring at location \( z \). This advancement improves real-time event detection for better crisis management and situational awareness. However, this rule cannot be expressed by existing rules since it requires aggregation operators.

(5) **Crisis prediction.** The event crisis prediction rule identifies a new event \( z \) as a potential trigger for a media crisis if it is similar to a previous crisis event \( z' \) and have over 10,000 followers. This rule enables PR agencies to proactively anticipate and manage potential crises. Although the similarity between events can be expressed using an ML model within GARs, aggregating over 10,000 followers cannot be done with existing graph rules. □

The examples raise several questions. Is it possible to have a class of graph rules to specify the above semantic associations that the previous rules cannot express? Can we reduce the complexity at the same time and make association deduction tractable? Can we parallelize association deduction to scale with large graphs?

**Contributions.** In response to practical need, this paper makes an effort to tackle these issues, all in the affirmative.

(1) **A new class of graph rules** (Section 3). We propose a class of rules, referred to as graph rules with oracle and denoted by GARs (Section 3). GARs also have the form of \( Q(X \rightarrow Y) \). However, unlike existing graph rules, GARs support oracle functions that import (a) external knowledge about the properties and features of nodes and edges; and (b) internal computations such as aggregate predicates including count, sum and avg. Like GARs [10], GARs can also embed machine learning (ML) models as predicates. Furthermore, the semantics of GARs is defined in terms of a revised notion of dual simulation [33], which is in polynomial time (PTIME) in contrast to the intractability (NP-Complete) of previous graph rules. These depart from GFDs, GEDs and GARs.

We show that GARs can be readily used for associations deduction [10], graph data cleaning [15], fraud detection [39] and annotation analysis [27], among other things. In this paper, we focus on the (incremental) deduction of associations with GARs.

(2) **Practical applications.** (Section 4) We formalize the association deduction problem by extending the chase [38] with a set \( \Sigma \) of GARs (Section 4). We show that the deduction has the Church-Rosser property, i.e., the chase converges at the same answer no matter what rules in \( \Sigma \) are used in what order the GARs are applied. Better yet, we prove that the association deduction problem is in PTIME, as opposed to the intractability of association deduction with GARs [10].

(3) **A parallel solution** (Section 5). To scale with large graphs, we parallelize SDeduce and develop a parallel association deduction algorithm PIncDeduce (Section 5). We show that PIncDeduce is parallelly scalable relative to SDeduce, i.e., it guarantees to reduce the parallel runtime when more processors are used. We propose a workload balance strategy to split skewed work units and dynamically balance workload, based on necessary affected area and bounded affected area, to balance computation and minimize communication across different processors.

(4) **Incremental deduction** (Section 6). We provide a parallel incremental algorithm PIncDeduce in response to graph update (Section 6). Real-life graphs are frequently updated, and it is costly to re-deduce associations starting from scratch when graphs change. In light of this, we develop the algorithm PIncDeduce to incrementally compute changes to associations, minimizing unnecessary recomputation. We show that the incremental deduction algorithm PIncDeduce is also parallel scalable.

(5) **Experimental study** (Section 7). Using real-life and synthetic graphs, we empirically verify the effectiveness, scalability, and efficiency of our (incremental) deduction methods. We find the following. (1) GARs are effective in association deduction by incorporating oracles and adopting revised simulation semantic. Our method has precision above 97\% on average, and its recall is above 72.5\%, which outperforms the SOTA method with GARs [10] by 23.6\%. (2) Our parallel deduction method is efficient and parallel scalable: on average, it is 5.0 times faster than PGAR on graphs of 6.2 million nodes and 33.4 million edges with 20 processors, and it is 4.2 times faster when the number of processors used is varied from 4 to 20. (3) Our incremental deduction algorithm PIncDeduce performs better than its batch counterpart even when updates \( \Delta G \) are up to 25\% of \( G \), 2.1 times faster when \( |\Delta G| = 10\%|G| \) on average.

2 **PRELIMINARIES**

We first review the basic notation. Assume that alphabets \( \Gamma, \Theta, \) and \( U \) denote labels, attributes, and constant values, respectively.

**Graphs.** We consider directed graphs \( G = (V, E, L, F_A) \), where (1) \( V \) is a finite set of nodes; (2) \( E \subseteq V \times V \) is a set of edges, in which (\( u, v' \)) denotes an edge from node \( v \) to \( v' \); (3) each node \( v \) in \( V \) (resp. edge \( e \) in \( E \)) carries label \( L(v) \) (resp. \( L(e) \)) in \( \Gamma \), and (4) for each node \( v \), \( F_A(v) \) is a tuple \( (A_1 = a_1, \ldots, A_N = a_N) \) such that \( A_i \neq A_j \) if \( i \neq j \), where \( a_i \) is a constant in \( U \), and \( A_i \) is an attribute of \( v \) drawn from \( \Theta \), written as \( v.A_i = a_i \), carrying the content of \( v \) such as keywords and blogs found in social networks.

We use two notions of subgraphs. A graph \( G' = (V', E', L', F'_A) \) is a subgraph of \( G = (V, E, L, F_A) \), denoted by \( G' \subseteq G \), if \( V' \subseteq V \), \( E' \subseteq E \), and for each node \( v \in V' \), \( L'(v) = L(v) \) and \( F'_A(v) = F_A(v) \); similarly, for each edge \( e \in E' \), \( L'(e) = L(e) \).

A subgraph \( G' \) is induced by a set \( V' \) of nodes if \( V' \subseteq V \) and \( E' \) consists of all edges in \( E \) whose endpoints are both in \( V' \).

**Patterns.** A graph pattern is defined as a DAG (Directed Acyclic Graph) \( Q[\tilde{x}] = (\tilde{V}_Q, \tilde{E}_Q, L_Q) \), where (1) \( \tilde{V}_Q \) (respectively, \( \tilde{E}_Q \)) is a finite set of pattern nodes(respectively, edges); (2) \( L_Q \) is a function with range in \( \Gamma \) that assigns a node label \( L_Q(u) \) and an edge label \( L_Q(e) \) for each \( u \in \tilde{V}_Q \) and \( e \in \tilde{E}_Q \); and (3) \( \tilde{x} \) is a list of distinct
variables to represent a subset of nodes in $V_Q$, referred to as the pivots of $Q$, for query interest. We limit the number of pivots to a maximum of 2, as we are primarily concerned with either the properties of a single entity or the associations between two entities. We allow wildcard "_" as a special label for nodes or edges in $Q$.

For each $v$ in $G$ or $Q$, we use $\text{pre}(v)$ to denote the incoming neighbors of $v$, and $\text{post}(v)$ as the outgoing neighbors of $v$.

For convenience, we do not differentiate between $Q$ and $Q[\bar{x}]$ when the focus is not on pivots.

**Remarks.** We define the graph pattern as a DAG since: (a) DAGs are well-suited for modeling many real-world systems. (b) DAGs strike a balance between expressive power and computational complexity. Unlike cyclic graphs, DAGs prevent the occurrence of long loops during graph traversal, which can cause challenges in applying parallel algorithms effectively.

**Example 2:** Five graph patterns are shown in Fig. 1. Here $Q_1$ depicts that a person $x_0$ and a team $z$ have interests in a research field, and another person $y$ is a member of team $z$. The pivot $\bar{x}$ of $Q_1$ is $x$, it is mapped to Bob and Phillip in $G_1$ in Fig. 1. Similarly, $Q_2$–$Q_4$ can be interpreted by referencing their counterparts in Fig. 1.

**Dual simulation** [34]. Graph $G$ matches pattern $Q$ via dual simulation, if there exists a binary relation $S \subseteq V_Q \times V$ such that (1) for each node $u \in V_Q$, there exists a node $v \in V$ such that $(u, v) \in S$, and (2) for each pair $(u, v) \in S$, (a) $L_Q(u) = L(v)$, (b) for each edge $(u, u') \in E_Q$, there exists an edge $(v, v')$ in $G$ with $(u_1, v_1) \in S$, and (c) for each edge $(u_2, u) \in E_Q$, there exists an edge $(v_2, v)$ in $G$ with $(u_2, v_2) \in S$; Here $L_Q(u) = L(v)$ holds if $L_Q(u)$ is an, i.e., wildcard matches any label to indicate generic entities.

A match $S_M$ of $Q$ in $G$ is maximum if for any match $S \subseteq G$, $S \subseteq S_M$. While there exist multiple matches in a graph $G$ for a pattern $Q$, there exists a unique maximum match $S_M$.

We also denote the match $S$ as consisting of $S(x)$ for all $x \in V_Q$. Intuitively, $V_Q$ is a list of entities to be identified by $Q$, and $S$ is such an instantiation in $G$.

Given $S_M$ of $Q$, its graph, denoted as $G_S$, is the subgraph $G'$ induced by nodes in $S$ but removing $(u', v')$ from $G'$ for $(u', v') \in S(u, v)$. We denote $(u, v)$ is not in $Q$.

**Pivoted match.** Given a graph $G$ and a pattern $Q[\bar{x}]$, a pivoted match of $Q[\bar{x}]$ at nodes $W \subseteq V$ in $G$ is a binary relation, denoted as $S_W$, such that (a) $S_W$ is a dual simulation match of $Q$, and (b) $|x| = |W|$, and for each $x \in \bar{x}$ and $w \in W$, $S_W(x) = w$.

We say a pivoted match $S_W$ is maximum if there exists no subset $S_W'$ such that $S_W'$ is a pivoted match of $Q$ and $S_W' \subseteq S_W$.

**Example 3:** For instance, consider pattern $Q_2$ and graph $G_2$ in Fig. 1. For pivot node $x_0$, there are two candidates Ann and Bob in $G_2$. The pivoted match $S_{Ann}$ of $Q_2$ at Ann is as follows. $S_{Ann}(x_0) = \{Ann\}$; $S_{Ann}(x_1) = \{Bob\}$; $S_{Ann}(y_1) = \{acc_1\}$; $S_{Ann}(y_2) = \{acc_2\}$; $S_{Ann}(y_3) = \{acc_1, acc_3\}$; $S_{Ann}(z_1) = \{transf_1\}$, $S_{Ann}(z_2) = \{transf_2, transf_3\}$. The pivoted match $S_{Bob}$ can be computed in a similar manner.

### 3 GRAPH RULES WITH ORACLES

We next define GROs. We start with oracles.

**Oracles.** In computational complexity theory, an oracle is an abstract machine developed for a decision or a function problem. Here we simply use oracles to import external knowledge and internal computations. More specifically, given a graph pattern $Q[\bar{x}]$, an oracle for $Q$ is a boolean function. It is either a unary function $f(x)$, or a binary function $f(x, y)$, where $x$ and $y$ are nodes in $Q$.

It’s important to note that our use of the term “function” here differs from the traditional mathematical definition. We define an oracle in a broader sense, allowing for various processes such as non-deterministic processes, human expertise, or pre-computed algorithms. This expanded definition provides flexibility and inclusiveness, enabling the incorporation of different sources of information or computation beyond what is typically considered within the realm of mathematical functions.

Intuitively, an oracle $f(x)$ may decide whether a node (entity) $x$ has a certain property, $f(x, y)$ may decide how nodes $x$ and $y$ are “associated”. It can be treated as a new attribute $f$ of node $x$ or an new event $(x, f)$. It may operate on data that is not in an input graph $G$ such as a knowledge base, i.e., it imports external knowledge beyond the input graph. It also supports internal computations such as (a) aggregate predicates involving count, sum and avg, and (b) ML predicates $M(x, y)$ and so on. These internal functions are applied to matches of a graph pattern $Q$ in a graph $G$, rather than to resort to external knowledge.

In the sequel, we refer to internal functions as **internal oracles**, and oracles that are computed externally as **external oracles**. To simplify the discussion, we use $f(x)$ and $f(x, y)$ to range over oracles, internal or external, when it is clear from the context.

**Predicates.** A predicate $p$ of $Q[\bar{x}]$ is one of the following: for $x, y \in Q$,

- unary oracle predicate $f(x)$;
- binary oracle predicate $f(x, y)$;
- constant attribute predicate $x.A = c$;
- variable attribute predicate $x.A = y.B$;

where $f(x)$, $f(x, y)$ are oracles, $c$ is a constant, and $A$ and $B$ are attributes of $x$ and $y$, respectively.
GROs. A graph rule with oracles (GRO) $\varphi$ is defined as

$$Q[x](X \rightarrow Y),$$

where $Q[x]$ is a graph pattern, and $X$ and $Y$ are (possibly empty) conjunctions of predicates of $Q[x]$. We refer to $Q[x]$ and $X \rightarrow Y$ as the pattern and dependency of $\varphi$, respectively.

GRO is a combination of topological constraint and semantic dependency. The pattern $Q$ identifies entities in a graph, and the dependency $X \rightarrow Y$ is applied to the entities.

**Remarks.** The following is a partial list of internal oracles. Assume $x, y \in Q[x]$.

(a) edge existent predicate $\iota(x, y)$, indicating the existence of an edge from $x$ to $y$ labeled $i \in \Gamma$.
(b) aggregate oracle $f(x)$, where the computation of $f(x)$ involves aggregation operators, such as count, avg, sum, min and max.
(c) a well trained ML model $f(x, y) = M(x, \tau, y)$ for link prediction to predict whether edge $(x, \tau, y)$ exists;
(d) similarity predicates $f(x, y) = \text{sim}(x, y)$, which returns true if $x$ and $y$ are similar.

To strike a balance of complexity and expressiveness for GROs, oracles are defined on the nodes of patterns (graphs). While their computation may involve the attributes of the nodes, they do not determine the associations between attribute values. For example, oracles cannot support polynomial arithmetic expressions related with node attributes.

We impose the restriction that internal oracles are computable in PTIME. But external oracles can be of any computational model, with a focus only on their results. This allows the sharing of knowledge and computed results across different applications. Through entity alignment, these oracles can be connected to our entities and treated as attributes or edges associated with aligned entities.

**Example 4:** One can use the GROs below to deduce associations described in Example 1, using patterns $Q_1$, $Q_2$ of Fig. 1.

(1) $\varphi_1 = Q_1[x_0](\text{Up_curve}(z) \land \text{Core}(y, z) \rightarrow \text{recom}(y, x_0))$, Here $\text{Up_curve}(z)$ and $\text{Core}(y,z)$ are external oracles, for deciding whether a team is on its up-curve and whether researcher $y$ is a core member in team $z$, respectively. The GRO says that if a person $x_0$ and a team $z$ are in the same research field $w$, team $z$ is on its up-curve, and if $y$ is a core member of $z$, then recommend $y$ to $x_0$ for collaboration.

(2) $\varphi_2 = Q_2(x_0)(\text{Circle}(x_0) \rightarrow \text{Mlauder}(x_0))$, where $\text{Circle}(x_0)$ is an internal oracle involves aggregate operator sum, computed by $\sum(\text{z1.amount}) \geq 0.7 \times \sum(\text{z2.amount}); \text{Mlauder}(x)$ is an internal oracle means person $x$ involves in money laundering. It states that if $x$ receives funds from others and transfers out more than 70%, then $x_0$ is involved in money laundering.

(3) $\varphi_3 = Q_3[x_0](\text{consistent}(x_0, z) \rightarrow (y, \text{exhibit}, z))$, where consistent $(x_0, z)$ is external oracle returns that whether the influence of $x_0$ and the reputation of $z$ are consistent. It states that if the influence of an artist $x_0$ is consistent enough with a library $z$’s reputation, then the artwork $y$ created by $x_0$ is recommended to library $z$ on exhibition.

(4) $\varphi_4 = Q_4[x_0](\text{Majority}(x_0) \rightarrow (x_0, \text{occur}, z_1))$. Here Majority$(x_0)$ is an oracle that computed by count$(x) \geq 3$, and val is an attribute of $z_1$. It says that if there are more than 3 people tweeting the same event $x_0$ in the same place $z_1$, then the event is predicted to happen in place $z_1$.

(5) $\varphi_5 = Q_5[z_1, z_2](\text{Crisis}(z_2) \land \text{follows}(z_1) \land \text{Similar}(z_1, z_2) \rightarrow \text{Crisis}(z_1))$. Here follows$(z_2)$ is computed by count$(x) \geq 10000$, and Similar$(z_1, z_2)$ means event $z_1$ and $z_2$ are similar. It states that if an event $z_1$ is similar with a crisis event $z_2$ and $z_1$ has been followed by a majority population, then $z_1$ is reported as a media crisis event.

**Semantics.** To interpret GRO $\varphi = Q[x](X \rightarrow Y)$, we use the following notations. Denote by $S_W$ a maximum pivoted match of $Q[x]$ in a graph $G$ at a vector $W$, and $p$ a predicate of $X$ and $Y$.

We say that $S_W$ satisfies a predicate $p$, denoted by $S_W \models p$, the following condition is satisfied:

(a) for each predicate $x.A = c$ in $X$, and for each $x' \in S_W'(x)$, $x'.A = c$;
(b) for each predicate $x.A = y.B$ in $X$, and for each $x' \in S_W'(x)$ ($(y' \in S_W'(y))$, there exists $y' \in S_W'(y)(x' \in S_W'(x))$ such that $x'.A = y'.B$ is true.
(c) for each predicate $f(x)$ in $X$, and for each $x' \in S_W'(x)$, $f(x')$ is true; and
d) for each predicate $f(x, y)$ in $Y$, and for each $x' \in S_W'(x)$ ($(y' \in S_W'(y))(x' \in S_W'(x))$ such that $f(x', y')$ is true.

For the set $X(Y)$ of predicates, we write $S_W \models X(Y)$ if match $S_W$ satisfies all the predicates in $X(Y)$.

If $X$ (resp. $Y$) is $\emptyset$ (i.e., true), then $S_W \models X$ (resp. $S_W \models Y$) is trivially true.

We write $S_W \models X \rightarrow Y$ if $S_W \models X$ implies $S_W \models Y$.

A graph $G$ satisfies GRO $\varphi$, denoted by $G \models \varphi$, if for all matches $S_W$ of $Q$ in $G$, $S_W \models X \rightarrow Y$. Otherwise we say $S_W \not\models X \rightarrow Y$. We say that $G$ satisfies a set $\Sigma$ of GROs, denoted by $G \models \Sigma$, if for all $\varphi \in \Sigma, G \models \varphi$, i.e., $G$ satisfies every GRO in $\Sigma$.

**Remarks.** In contrast to isomorphism matching based on function, pivoted match is based on sets. When interpreting GROs, it is necessary to consider the association between two sets.

If "existential" semantic was adopted, it would be too relaxed and result in a significant number of false positive associations when applying GROs. Conversely, directly applying the "all" semantic would be overly strict and result in a majority of false negative associations. To strike a balance, we utilize a variant of the "all" semantic: it expresses the "all" semantic for unary predicates in $X$ and $Y$, while for binary predicates, it implies an "existential" semantic.

**Example 5:** Consider graph $G_4$ of Fig. 1 and GRO $\varphi_4$ in Example 4. $G_4 \not\models \varphi_4$, since there exists pivoted match $S_{\text{Blizzard}}$ such that $S_{\text{Blizzard}} \models X_4$, but there exists no edge (Blizzard, occur, Beijing) in $G_4$. Hence $S_{\text{Blizzard}} \not\models X_2 \rightarrow Y_2$, i.e., $S_{\text{Blizzard}}$ witnesses $G_2 \not\models \varphi_2$. Similarly, $G_1 \not\models \varphi_i$ for other $i \in [1, 5]$.

### 4 DEDUCTION ASSOCIATIONS

One of the applications of GROs is deducing associations. GROs can deduce associations listed as follows: (a) associations between entities deduced by $\iota(x, y)$ and $f(x, y)$; (b) associations of attribute values deduced by $x.A = c$ and $x.A = y.B$; (c) properties of $x...
deduced by \( f(x) \). Unlike GARs, GROs can deduce associations between entities described by oracles and the properties of entities.

We first revise the chase [38] for GROs, and show that chasing with GROs has the Church-Rosser property. Based on these, we will study a practical sequential algorithm to deduce associations.

### 4.1 Chasing with GROs

Consider a graph \( G = (V, E, L, F_A) \), a set \( \Sigma \) of GROs, a set \( F \) of oracles defined on nodes \( x \) and \( y \), and a set \( F_p \) to store the result returned by oracles in \( F \). We study the chase of \( G \) by \( \Sigma \).

**Association relation.** We define the chase as a sequence of association relations \( R \), and \( R \) includes \( (a) \) equivalence relations \( Eq \), as described in [14], for each attribute \( x.A \) of \( x \), its equivalence class \( [x.A]_{Eq} \) is a set of attributes \( y.B \) and constants \( c \); if \( x.A = y.B \) and \( x.A = c \). (b) a set \( E' \) of edges, updated by literal \( \ell(x,y) \), and (c) a boolean set of oracles \( F \), updated by \( f(x) \) and \( f(x,y) \).

**Chasing.** We starts with \( R_0 \), an initial association relation initialized as empty. Then each chase step \( i \) extends \( R_{i-1} \) to get \( R_i \) by applying a GRO.

A chase step of \( G \) by \( \Sigma \) is defined as: \( R \Rightarrow (\varphi, S_W) \), where \( \varphi = Q(\ell(x), Y) \rightarrow Y \) is a GRO in \( \Sigma \), and \( S_W \) is the pivoted simulation of \( Q \) pivoted at \( W \) such that \( S_W \models X \). \( R' \) extends \( R \) by enforcing one literal \( l \in Y \) by using \( S_W \).

More specifically, based on \( l, R' \) is defined as follows.

1. If \( l = x.A = c \), then for each \( x' \in S_W(x) \), \( R' \) extends \( R \) by (a) including a new equivalence class \( [x'.A]_{Eq} \) if \( x'.A \) is not in \( Eq \) and (b) adding \( c \) to \( [x'.A]_{Eq} \).
2. If \( l = x.A = y.B \), then for each \( x' \in S_W(x) \) and \( y' \in S_W(y) \), \( R' \) extends \( R \) by adding (a) \([x'.A]_{Eq} \) if \( x'.A \) is not in \( Eq \) and (b) \([y'.B]_{Eq} \) to \([x'.A]_{Eq} \).
3. If \( l = f(x) \), then for each \( x' \in S_W(x) \), \( R' \) extends \( R \) by adding \( f(x') \) to \( F \).
4. If \( l = f(x, y) \), then for each \( x' \in S_W(x) \) and \( y' \in S_W(y) \), \( R' \) extends \( R \) by adding \( f(x', y') \) to \( F \).

**Consistency.** Conflicts may emerge when enforcing GROs. We say that \( R \) is inconsistent if when it enforces predicate \( p \), either (a) \( p \) is \( x.A = c \), but there exists \( x.A = d \) or (b) \( p \) is \( x.A = y.B \), but there exists \( x.A = c \) and \( y.B = d \), where \( c \) and \( d \) are constants and \( c \neq d \).

The step is valid if Eq is consistent, note that (a) edge literals do not incur inconsistencies as multiple edges can co-exist between a pair of nodes, and (b) \( F \) will not conflict since \( Y \) do not contain negative predicates.

**Chasing sequences.** A chasing sequence \( \rho \) of \( G \) by \( \Sigma \) is \( (R_0, \ldots, R_k) \) where for all \( i \in [0, k-1] \), there exist a GRO \( \varphi = Q(\ell(x), Y) \rightarrow Y \) in \( \Sigma \) and the pivoted match \( S_W \) of graph pattern \( Q \) pivoted at \( W \) in \( G \) such that \( R_i \Rightarrow R_{i+1} \) is a valid chase step.

The sequence is terminal if there exist no GROs \( \varphi \in \Sigma \) and match \( S_W \) of pattern \( Q \) of \( \varphi \) in \( G \) pivoted at \( W \), and association relation \( R_{k+1} \) such that \( R_k \Rightarrow R_{k+1} \) is a valid step. More specifically, it terminates in one of the following two cases.

1. \( R_k \) cannot be expanded and \( R_k \) is consistent (\( i \in [0, k] \)); if so, the chasing sequence is valid and its result is \( R_k \).
2. At some step \( i \), \( R_i \) is inconsistent; if so, the chasing sequence is invalid, and the result is undefined.

**Example 6:** Consider the graph \( G_4 \) shown in Fig. 1. Assume \( R_0 = \{ \text{Blizzard} \} \). \( Eq \) is given in Example 4. From \( R_0 = \{ \text{Blizzard} \} \), \( Eq \) is given in Example 4. From \( R_0 \Rightarrow \{ \text{Blizzard} \} \), \( R_1 \), match \( \text{Blizzard} \) is given in Example 5.

1. If \( Eq \) is consistent, note that (a) edge literals do not incur inconsistencies as multiple edges can co-exist between a pair of nodes, and (b) \( F \) will not conflict since \( Y \) do not contain negative predicates.

### 4.2 The Church Rosser Property

A major concern is whether the chase always terminates with the same result. Following [2], we say that chasing with GROs is Church-Rosser if for all graphs \( G \) and all sets \( \Sigma \) of GROs, all chasing sequences of \( G \) by \( \Sigma \) are terminal and converge to the same result, regardless of what GROs in \( \Sigma \) are applied.

**Theorem 1:** Chasing with GROs is Church-Rosser.

**Proof sketch:** The proof consists of two steps.

1. The size of chase relation \( R_i \) is bounded by \( |G| \times \Sigma \). It is because there exists at most \( |G| \times \Sigma \) equivalence classes in \( Eq\), edges in \( E \), and oracles in \( F \). It is easily verify that GROs in \( \Sigma \) check at most \( |\Sigma| \) oracles for each node and edges; and between each pair of nodes in \( G \), the labels of new edges are constrained by GROs in \( \Sigma \), and then at most \( |\Sigma| \) edges can be deduced.

2. All chasing sequences terminate at the same result. If there exist two terminal sequences having different results, then one of them is not terminal, a contradiction. Since \( G \) is not changed during the chase, the oracles introduced by \( \Sigma \) will not affect the Church-Rosser property of the chase.

**Example 6:** Consider the graph \( G_4 \) shown in Fig. 1. Assume \( R_0 \Rightarrow \{ \text{Blizzard} \} \). \( Eq \) is given in Example 4. From \( R_0 = \{ \text{Blizzard} \} \), \( Eq \) is given in Example 4. From \( R_0 \Rightarrow \{ \text{Blizzard} \} \), \( R_1 \), match \( \text{Blizzard} \) is given in Example 5.

### 5 PARALLEL DEDUCTION

Real-life graph data is often large, and even a polynomial time algorithm is unacceptable, especially for online analysis. This motivates us to develop a parallel algorithm with parallel scalability. We first review the notion (Section 5.1) and give a sequential algorithm (Section 5.2). We then develop a parallel algorithm with scalability property (Section 5.3).

To simplify the discussion, without loss of generality, we focus on GROs defined with connected graph patterns \( Q \) with one pivot. The algorithms can be readily extended to process GROs with disconnected patterns and patterns with two pivots.

#### 5.1 Parallel scalability Revisited

To characterize the effectiveness of the parallel algorithm for association deduction, we revisit the notion of parallel scalability that was introduced in [32] and has been widely used in practice. Consider a sequential algorithm \( A \) for association deduction, with cost \( t(G, |\Sigma|) \) measured in the sizes of graph \( G \) and \( \Sigma \) of GROs.

A parallel algorithm \( A \) for association deduction is said to be parallel scalable relative to yardstick \( A \) if its parallel running time
by using \( p \) processors can be expressed as follows:

\[
T(|G|, |\Sigma|, p) = O\left(\frac{t(|G|, |\Sigma|)}{p}\right),
\]

where \( p \ll |G| \), i.e., the number of processors is much smaller than real-life graphs \( G \), as commonly found in the real world.

Intuitively, parallel scalability guarantees speedup of \( \mathcal{A}_p \) relative sequential algorithms \( \mathcal{A} \). Such algorithm \( \mathcal{A}_p \) “linearly” reduces the running time of \( \mathcal{A} \) when \( p \) increases. Hence, \( \mathcal{A}_p \) is able to scale with large \( G \) by adding processors when needed.

Similarly, we say that a parallel algorithm \( \mathcal{A}_p \) for incremental association deduction is parallel scalable relative to a sequential incremental \( \mathcal{A} \) if its cost with \( p \) processors can be expressed as:

\[
T(|G|, |\Delta G|, |\Sigma|, p) = O\left(\frac{t(|G|, |\Delta G|, |\Sigma|)}{p}\right),
\]

where \( t(|G|, |\Delta G|, |\Sigma|) \) is the worst-case running time of \( \mathcal{A} \).

### 5.2 A sequential algorithm

Now we design a practical sequential algorithm for the chase, denoted as SDEDuc, to deduce new associations.

**Overview.** Algorithm SDEDuc takes a graph \( G = (V, E, L, F) \), a set \( F_0 \) of external oracle values, and a set \( M_0 \) of polynomial algorithms for oracles as input and deduces a set \( \mathcal{A} \) of associations. It works as follows. (a) It starts with \( \text{Assoc}(\Sigma, G) = \emptyset \). (b) For each \( v \in \Sigma \), it computes the candidates for the pivot nodes \( \mathcal{S}(v) \), which consist of all nodes in \( G \) that have the same label with \( v \). (c) For each \( w \in \mathcal{S}(v) \), it enumerates all pivoted matches \( \mathcal{S}_w \) of \( G \) at \( w \). (d) For each \( w \), such that \( \mathcal{S}_w \models X \), it adds associations by forward chasing step with \((v, \mathcal{S}_w)\) to Assoc.

The core procedure of SDEDuc is to enumerate the pivoted matches \( \mathcal{S}_w \) such that \( \mathcal{S}_w \models X \). However, there is no applicable algorithm in place. In this section, we provide a match enumeration algorithm, denoted as SMatch.

The skeleton of SMatch is the dual simulation procedure [33], denoted as DualSim. Given a pattern \( Q \) and a graph \( G \), let’s review the procedure DualSim.

**Procedure** DualSim \((Q, G)\). For each node \( u \) in \( Q \), DualSim first initialize a set \( S(u) \) contains all nodes of \( G \) that have the same label as \( u \). Then update \( S \) by the definition of dual simulation, a node \( v \) is removed from \( S(v) \) unless (a) if there is a parent node \( u' \) of \( u \), then there exists a parent node \( v' \in S(u') \); and (b) if there is a child node \( u' \) of \( u \), then there exists a child node \( v' \in S(u') \). This process is repeated until there are no more changes.

Then SMatch enumerates \( \mathcal{S}_w \) such that \( \mathcal{S}_w \models X \) as follows.

**Step 1: compute pivoted simulation match \( \mathcal{S}_w \).** We find that the pivoted simulation for a DAG \( Q \) has locality property. That is, the pivoted match \( \mathcal{S}_w \) is within a subgraph surrounding \( w \). We define such subgraphs as balls.

**Balls.** For a node \( w \) in a graph \( G \) and a positive integer \( r \), the ball with center \( v \) and radius \( r \) is a subgraph of \( G \), denoted by \( G[\{v, r\}] \), such that (1) for all nodes \( v' \in G[\{v, r\}] \), the shortest distance \( \text{dist}(w, v') \leq r \), and (2) it is induced by the nodes of \( G[\{v, r\}] \).

Unless stated otherwise, the term "shortest distance" refers to calculations conducted on the graph assuming it is undirected.

**Lemma 2:** Given a pattern \( Q[x] \), a graph \( G \) and a pivot match \( w \), let \( d_Q \) be the longest shortest distance form \( x \) to other nodes \( v \in Q \), the pivoted match \( \mathcal{S}_w \) is within the ball \( G[w, d_Q] \). □

**Proof:** This follows from the definition of pivoted simulation and DAG \( Q \). Since \( Q \) has no cycle, for any nodes \( u' \in S(u) \), \( \text{dist}(u', w) \leq \text{dist}(u, x) \leq d_Q \). □

According to Lemma 2, for each \( w \in S(x) \), SMatch first compute the ball \( G[w, d_Q] \). Then it invoke DualSim \((Q, G[w, d_Q])\) by fixing \( \mathcal{S}_w(x) = \{w\} \) to get \( \mathcal{S}_w \).

**Step 2: compute \( \mathcal{S}_w \) such that \( \mathcal{S}_w \models X \).** According to the satisfaction definition, we first handle the unary predicates and binary predicates separately. (a) For each unary predicate \( x.A = e(f(x)) \) in \( X \), and for each \( u' \in \mathcal{S}_w(u) \), if \( u.A \neq e(f(x)) \), then remove the nodes \( u' \) from \( \mathcal{S}_w(u) \). (b) For each binary predicate \( x.A = y.B(f(x, y)) \) in \( X \), we create a bipartite graph \( G \) as follows. For each \( u \in \mathcal{S}_w(x) \), we create a node \( u \) in the left (right) part of \( G \), if \( u.A = v.B(f(x, y)) \), then add an edge from \( u \) to \( v \) in \( G \). At the end, we remove isolate nodes in \( G \) from \( \mathcal{S}_w \). Then we update \( \mathcal{S}_w \) using the update process in DualSim.

**Deducing associations.** After obtaining \( \mathcal{S}_w \) such that \( \mathcal{S}_w \models X \), SDEDuc deduces associations using \( Y \). That is, for each \( x.A = e(f(x)) \) in \( Y \), for each \( u \in \mathcal{S}_w(x) \), add \( u.A = e(f(u)) \) to \( \text{Assoc}(\Sigma, G) \). For each \( x.A = y.B(f(x, y)) \) in \( Y \), for each \( u \in \mathcal{S}_w(x) \) and \( v \in \mathcal{S}_w(y) \), add \( u.A = v.B(f(u, v)) \) to \( \text{Assoc}(\Sigma, G) \).

**Optimization strategy.** (a) We build a candidate space structure [37], denoted as CS, to maintain the edges between candidates \( S(v) \) and those in \( S(v') \) if \( (v, v') \) is in \( Q \). Is a graph structure and keeps the candidates’ edge information to avoid checking. Furthermore, CS is continually updated during the computation of \( \mathcal{S}_w \).

CS is sound for enumerating pivoted simulation matches, i.e., enumerating \( \mathcal{S}_w \) on CS is equivalent to that on \( G \). It is easily proved by contradiction.

(b) SMatch also use a partial match extraction technique introduced in [10] to filter \( \mathcal{S}_w \) vs. \( \mathcal{S}_w \).

**Correctness.** We show that SDEDuc correctly computes \( \text{Assoc}(\Sigma, G) \) to chase \( G \) by \( \Sigma \). First, the result returned by SDEDuc is in \( \text{Assoc}(\Sigma, G) \). This follows from the definition of the chase and the correctness of SMatch, since no associations are deduced from SDEDuc until partial matches become complete and \( X \rightarrow Y \) is not satisfied. On the contrary, all associations in \( \text{Assoc}(\Sigma, G) \) are computed by SDEDuc, since SDEDuc inspects all candidate matches that can contribute to the deduction of new associations.

**Theorem 3:** The association deduction of GROs is in PTIME. □

**Proof:** We prove it by showing that algorithm SDEDuc runs in polynomial time. For each GRO \( \varphi = Q[x](X \rightarrow Y) \) in \( \Sigma \), the set of candidates is computed in \( O(|V|^2|E_G|) \). SMatch takes \( O(|V||E_G|(|E|+|E_Q|))+T \), where \( T \) is the dependency checking time, in PTIME. And deducing associations takes \( O(|V||E_G|(|E|+|E_Q|)) \). Thus, algorithm SDEDuc is in \( O(|E||V|^3|E_G|)+2(|E||V|+|V||E_Q||E|+|E_Q|)+T \). It is in \( O(|E||V|^3+T) \), a polynomial time. □

**Analysis.** We conduct a comparison between the deduction of GROs and other graph rules, such as GEDs [14], GARS [10], GFDs [6], and NGFDs [13], with the form of \( Q(X \rightarrow Y) \).

The complexity of deduction for these rules lies in computing the matches of \( Q \) and checking \( X \rightarrow Y \) on those matches. In our case,
since we restrict our internal oracles to be in $P$, checking $X \rightarrow Y$ in GROs is also in $P$, similar to other graph rules.

However, the deduction complexity of those comparison rules is known to be NP-Complete. This intractability arises from the match semantic they adopt, which is either isomorphism or homomorphism. In contrast, the simulation-based semantics adopted by GROs make their deduction tractable. In fact, even when the pattern contains cycles, GROs deduction remains in $P$.

5.3 A parallel scalable algorithm

Overview. Algorithm PIncDeduce works with $p$ processors $S_1, \ldots, S_p$ on a graph $G$ that is partitioned via edge-cut [3] or vertex-cut [31]. The processor $S_i$ is taken as the coordinator. It first computes the candidate sets $S$ by DualSim and constructs the candidate space $CS$ for each GRO $\varphi$ in $\Sigma$ in parallel. Then for each $\varphi = \Sigma(x \rightarrow y)$ in $\Sigma$, it constructs a set of work units $(\varphi, w)$ for $w \in S(x)$. Finally, each processor handles its workload and deduces associations in parallel, like in SDec.

However, there are two challenges. (1) The candidate space of a work unit may reside in different fragments. (2) The workloads of some processors may be skewed, since (a) the workloads in each processor may be unbalanced; and (b) some work units may take much longer e.g., when accessing a large candidate space.

To cope with these challenges, previous graph rules with isomorphism semantics adopt their locality property, reducing the computation on a (large) graph to small areas surrounding $AFF$, and then transfer AFF of the border nodes and the skew work unit to other processors. e.g., $G_d(v)$ for a pivot $v$ in $Q$ where $d_2$ is the maximum diameter $d_Q$ for all patterns $Q$ that appear in $\Sigma$ [13].

The locality property also applies to the computation of pivoted matches for GROs, as the patterns $Q$ in GROs are acyclic. However, in dense graphs, the size of the set $AFF$, represented by $G_d(v)$, can be significantly large. Nonetheless, the candidate set (CS) for GROs is relatively smaller as it only includes the neighboring nodes of pivots that are considered potential matches. Furthermore, the size of the CS gradually decreases during the match enumeration process.

To improve the efficiency of matching GROs, we propose two techniques. First, we introduce a “necessary affected area” for border nodes, which enables us to determine their status by assembling the necessary information from other processors at once. Secondly, we adopt a “bounded affected area” to balance the workload. In cases of uneven distribution, heavy processors can send work units within the bounds of the affected area.

(a) Necessary affected area. According to lemma 4, the necessary affected area for $Q$, denoted as $NFA(Q)$, is defined as the connected parts of $CS$, that include the border nodes. Then $NFA(\Sigma) = U_{\varphi \in \Sigma, Q \in \varphi} NFA(Q)$.

Lemma 4: For a given pattern $Q$ and a fragment $F_1$, if a node $x \in F_1$ is in a final pivoted match, then it must belong to the candidate space $CS$, of $Q$ in $F_1$. Here, $CS$ is the updated $CS$ structure by setting the matching status of all border nodes as true.

Proof: The lemma can be proved by contradiction. Suppose that there exists a node $v \in F_1$ that is not in $CS_2$, but is in a pivoted match $S_w$. Let us assume that $v \in S_w(u)$ for $u \in Q$. If $v$ is a border node in $F_1$, since its status $S(u)$ is true for each $u$ in $Q$ such that $L_Q(u) = L(v)$. Therefore, it must be in $CS_2$, which contradicts the assumption. On the other hand, if $v$ is not a border node in $F_1$, then since $v$ is in $S_w(u)$, it must be in $CS_2$, again contradicting the assumption. Thus, the lemma is satisfied.

(b) Bounded affected area. Since $Q$ is acyclic, we define the bounded affected area as the subgraph $CS_{(\varphi, w)}$ induced by $w$ of $CS$. Its size is not larger than $G_d_4(w)$. We balance the workload by distributing work units $(\varphi, w)$ with $CS_{(\varphi, w)}$ to other processors. To further reduce communication cost, we can compute partial matches of $CS$ in the local worker and then distribute work units $(\varphi, w)$ with updated $CS_{(\varphi, w)}$.

Workload balancing. PIncDeduce adopts the workload balancing mechanism in [10]. We define the skewness of $S_i$ as cost($W_i$)

$$\text{cost}(W_i) = \frac{cost(W_i)}{\sum_i \text{cost}(W_i)}$$

Here cost($W_i$) = $\sum_{(\varphi, w) \in W_i} |CS_{(\varphi, w)}|$.

It checks the skewness of the processors in a time interval intvl. If the skewness of $S_i$ exceeds a threshold $\eta$ (2.5 in experiments), it evenly distributes the work units and their union of candidate space in $W_i$ to those $S_j$’s that have skewness below $\eta’$ (0.5 in experiments). The distribution method is the same as described in [6].

Algorithm. Putting these together, we present algorithm PIncDeduce in Fig. 2. It first constructs the workload $W$ (lines 2). Then it balances the workload and makes all work units evenly distributed among each worker (lines 5-6). Next, PIncDeduce invoke Expand to deduce associations Assoc($F_i, \Sigma$) in parallel for each $i \in [1, p]$ (line 7). It periodically balances workload (line 9), until all processors complete their work (line 11). Finally, PIncDeduce collects local associations Assoc($F_i, \Sigma$) from all processors and assembles the partial matches as $\Delta$Assoc. The union of all Assoc($F_i, \Sigma$) and $\Delta$Assoc is Assoc($G_1, \Sigma$) (line 10) and is returned (line 11).

At each processor $P_i$, procedure Expand processes each work unit to enumerate the pivoted matches and deduce associations (lines 2-3). During the enumeration process, workloads are balanced based on cost estimation, as described above. The local associations deduction Assoc($F_i, \Sigma$) and workload $W_i$ are updated accordingly. It returns Assoc($F_i, \Sigma$) to $S_i$ when no work units remain in $W_i$, i.e., when $P_i$ finishes its workload.

Example 8: Recall graph $G$ along with its partitions, and GROs $\Sigma$ from Example 7. PIncDeduc first computes the candidate sets $\Sigma$ for each in each fragment $F_i$ for $i \in [1, 2]$. Then workloads $W_i$ in $F_i$ is created for $i \in [1, 2]$, where $W_1 = \{w_1 = (\varphi_2, \text{Bob})\}, W_2 = \{w_2 = (\varphi_2, \text{Ann})\}$. The size of the workload of $W_1$ is 7, while the size of the workload of $W_2$ is 8. PIncDeduce now begins to process border nodes to get $NFA_2(\varphi_2)$ in $F_1$, and $NFA_2(\varphi_2)$ in $F_2$ (See Example 7). Then $NFA_2(\varphi_2)$ is send to $F_2$ and $NFA_2(\varphi_2)$ is send to $F_1$. [7]
Algorithm: PDeduce

Input: A fragmented graph $G$ across $p$ processors $S_1, \cdots, S_p$, a set $\Sigma$ of GROs.

Output: The set $\text{Assoc}(\Sigma, G)$ of associations.

1. $\text{Assoc} = \emptyset$.
2. create and estimate workload $W_i$ for each worker in parallel;
3. compute NFA($\Sigma_F$) of border nodes in parallel;
4. send NFA($\Sigma_F$) to other workers;
5. create a workload plan $\text{plan}$ to balance workload;
6. send the $\text{plan}$ to all processors
7. invoke Expand to compute new associations in parallel;
8. repeat
9. periodically balance workload at interval intvl.
10. until all $S_i$ return $\text{Assoc}(F_i, \Sigma)$
11. get new associations $\text{Assoc}$ by assembling partial matches;
12. return $\text{Assoc} \leftarrow \bigcup_{j \in [1, p]} \text{Assoc}(F_i, \Sigma) \cup \text{Assoc}$

Procedure: Expand executed at each worker $S_i$ in parallel */

Input: fragment $F_i$, Message $M_i$, a set of work units $W_i$

Output: $\text{Assoc}(F_i, \Sigma)$

1. while $W_i$ is not empty do
2. fetch a work unit $(\varphi, w)$ from $W_i$
3. computes $S_w$ and deduce associations with $(\varphi, S_w)$;
4. if workload is skewed do
5. balance $W_i$ to other workers.
6. send $\text{Assoc}(F_i, \Sigma)$ to $S_e$

Figure 2: Algorithm PDeduce

Each $F_i$ then can correctly compute partial matches. The partial matches are assembled in coordinator $S_e$ as a complete match and associations $\text{Mlauder}(x)$ for $x \in \{\text{Bob, Ann}\}$ are deduced. □

Correctness. Although PDeduce computes associations simultaneously on multiple processors, the correctness of this parallel association deduction method is warranted. (1) The result returned by PDeduce is in $\text{Assoc}(G, \Sigma)$, which is ensured by SDEDuce since PDeduce just parallelizes SDEDuce. (2) all associations in $\text{Assoc}(G, \Sigma)$ are computed by PDeduce, including deduced matches involving multiple fragments and those can be locally computed in a single fragment.

Theorem 5: PDeduce is parallel scalable to SDEDuce. □

Proof: We show that with $p$ processors, PDeduce runs in $O(\Sigma(|V| + T)/p)$ time with $p$ processors. Obviously, identifying the candidates for $\Sigma$ takes $O(\Sigma(|V|)/p)$ time, compute the necessary affected area for border nodes takes less than $O(\Sigma(|V|)/p)$, send such area takes $O(|G|/p)$, receive such area from other processors take $O(|G|/p)$, locally compute associations take $O(|G|/p)$, send them to coordinator $S_e$ takes $O(|G|/p)$, workload balancing takes $O(|G|/p)$. Hence PDeduce takes at most $O(\Sigma(|V| + T)/p)$ time. □

6 INCREMENTAL DEDUCTION

Real-life graphs frequently change, and association deduction is costly over large-scale graphs. These highlight the need for incremental association deduction. Next, we develop a parallel algorithm for incremental deduction, denoted as PIncDeduce.

Challenges. The internal oracles in GROs make the impacts of the edges inserted and deleted in $\text{Assoc}(\Sigma, G)$ more complicated than that of GARs. For instance, oracles including aggregate operators.

Algorithm: PIncDeduce

Input: A fragmented graph $G$ across $p$ processors $S_1, \cdots, S_p$, a set $\Sigma$ of GROs, and a batch update $\Delta G$.

Output: The set $\Delta \text{Assoc}^+$ and $\Delta \text{Assoc}^-$ of associations.

1. $P = \emptyset$; "update triggers "$	riangleright$
2. for each unit insertion (resp. deletion) update of $e = (a, a')$ in $\Delta G$ and $e_p = (a, a')$ in Q of GRO $\varphi = Q(x_0)(X \rightarrow Y)$
   in $\Sigma$ having $\text{LD}_G(u) = L(v)$ and $\text{LD}_G(u') = L(v')$ do
3. add $(\varphi, e_p, +)$ (resp. $(\varphi, e_p, -)$) to $P$.
4. send $P$ to each processor $p_i$ and construct workload $W_i$ in parallel;
5. process the work units in parallel;
6. invoke DelAssoc to compute $\Delta \text{Assoc}^+$ and $\Delta \text{Assoc}^-$ in parallel;
7. $\Delta \text{Assoc}^+ = \bigcup_{i \in [1, p]} \Delta \text{Assoc}_i^+$; $\Delta \text{Assoc}^- = \bigcup_{i \in [1, p]} \Delta \text{Assoc}_i^-$;
8. return $\Delta \text{Assoc}^+$ and $\Delta \text{Assoc}^-;

Figure 3: Algorithm PIncDeduce

Both the inserted edges and the deleted edges could trigger the generation of new associations and remove some associations. These results in the association status, i.e., invalid or valid, may be changed more than one time in the incremental computation.

Auxiliary structures. To address the above problem, PIncDeduce maintains an association dependency graph $G_d$ to store the affected relation for each deduced association. We say that an association $\alpha$ is affected by $\alpha'$, if removal $\alpha'$ makes (a) the pivoted match that can deduce $\alpha$ no longer holds, or (b) the precondition is not satisfied.

Then $G_d$ is constructed as follows. (a) $G_d$ is initialized by all deduced associations and a special node $v_0$ with empty. (b) For any two associations $\alpha$ and $\alpha'$ in $G_d$, if the association $\alpha$ is affected by $\alpha'$ through $(\varphi, w)$, then there exists an edge $(\alpha', \alpha)$ with the label $(\varphi, w)$ in $G_d$. (c) If there is no $\alpha'$ that affects $\alpha$, and $\alpha$ is deduced by $(\varphi, w)$, then the edge $(\varphi, w, \alpha)$ with the label $(\varphi, w)$ is added to $G_d$. $G_d$ can be readily obtained when running PDeduce and its size is polynomial in $|G|$ and $|\Sigma|$ (the proof of Theorem 1).

Note that in PIncDeduce, an association $\alpha$ can be deduced by the following three cases. (a) is deduced by a completed match $S_w$ of GRO $\varphi$; (b) has been deduced by $S_w$ but can also be deduced by another completed match $S_{w'}$ of GRO $\varphi$; and (c) $\alpha$ has been deduced but can also be deduced by a partial match $S_{w''}$ of GRO $\varphi$. For cases (a) and (b), the affection relationship can easily be constructed in $G_d$. However, for case (c), the affection relationship must be verified by completing $S_{w''}$. To improve efficiency, we avoid computing affection relations for case (c) and instead store the partial match $(\varphi, w, \alpha)$ in a list $L$ within $G_d$. This list will be processed for association removal checking when necessary.

Algorithm. As shown in Fig. 3, PIncDeduce takes as input $\Sigma, \Delta G$ and, moreover, the chase graph $G_c$ and the auxiliary structure $G_d$ that are cached after the batch execution of PDeduce and are distributed across workers. Denote by $\Delta \text{Assoc}^+$ and $\Delta \text{Assoc}^-$ the inserted and deleted edges in $\Delta G$, respectively. PIncDeduce computes the changes deduced $\Delta D(G, \Delta G, \Sigma)$ to the old associations deduced.

After adjusting $G_c$ with update $\Delta G$ (line 2), PIncDeduce first constructs the workload $W$ as follows. For each edge $e = (a, a')$ in $\Delta G$, it computes the update triggers for $e$, i.e., the set of GROs in $\Sigma$ that may deduce associations by changes in $e$ (lines 2-3). Then for each update trigger $(\varphi, e_p, +)$ or $(\varphi, e_p, -)$, it computes a set of work units $(\varphi, w)$ that is in $S(x_0)$ and is connected with $e$ in $G$ according to a partial match of a shortest path from $e_p$ to
Remove delay strategy: PIncDeduce first processes $\varphi$ that can deduce new associations, since new associations may affect the removal of old associations. But the opposite cannot happen. Hence PIncDeduce adopts a remove delay strategy to ensure that all new associations are deduced prior to all invalid old associations. More specifically, when an invalid association $a$ is deduced by $\varphi$ and its match $S_a$, the information $(\varphi, a, w)$ is recorded in a set $C_{del}$ to perform removal checks in the DelAssoc procedure.

Invalid associations. After processing all the work units, procedure DelAssoc began to refine $C_{del}$ to catch invalid associations. For each $(\alpha, \varphi, w) \in C_{del}$, it first removes in-going edges of $\alpha$ with label $(\varphi, S_w)$, and then check the in-degree of $\alpha$. If there exist no precedence nodes for $\alpha$, we further check its list $L$. (a) If $\alpha \in L$, then $\alpha$ can be safely removed. Else (b) $(\varphi', S_{w'})$ in $L$ need to be expanded and completed to decide whether it can deduce $\alpha$. This process continues until all elements $L$ are processed.

If $\alpha$ is removed, the associations $\alpha'$ affected by $\alpha$ may be removed successively. It checks $\alpha'$ just as checking $\alpha$ does. The process continues until there exists no association that can be removed.

Example 9: Recall graph $G$ and GROs $\Sigma$ from Example 8. Consider $\Delta G$ that inserts $e_1 = (Alice, owns, acc_1)$ and deletes $e_2 = (Ann, owns, acc_2)$. PIncDeduce first computes the triggers $P = \{(\varphi_1, e_1), (\varphi_2, e_1), (\varphi_1, e_2), (\varphi_2, e_2)\}$, where $e_1 = (x_0, y_2)$ and $e_2 = (x_1, y_1)$. $G_d$ is constructed as $(\emptyset, (\varphi_1, Bob), Mlauder(Bob))$, and $(\emptyset, (\varphi_2, Ann), Mlauder(Ann))$ with $L = \emptyset$. PIncDeduce then creates the workload $W_1$ in $F_1$ and $W_2$ in $F_2$ according to $P$, where $W_1 = \{(w_1 = (\varphi_1, Bob), w_2 = (\varphi_2, Ann), w_3 = (\varphi_2, Alice))\}$, and $W_2 = \{(w_1 = (\varphi_2, Bob), w_2 = (\varphi_2, Ann), w_3 = (\varphi_2, Alice))\}$. Then $W_1$ and $W_2$ will be computed in a way similar to that of Example 8. During the process, $G_d$ is updated by adding $(\emptyset, (\varphi_2, Alice), Mlauder(Alice))$, and $C_{del} = \{(Mlauder(Ann), \varphi_2, Ann))\}$. Then it catches invalid associations by $C_{del}$ and $G_d$, and Mlauder(Ann) is removed from $G_d$. Finally, $\Delta_{Assoc^+} = \{Mlauder(Alice)\}$ and $\Delta_{Assoc^-} = \{Mlauder(Ann)\}$.

We have the following about algorithm PIncDeduce.

Proposition 6: The associations in $\text{Assoc}(G \oplus \Delta G, \Sigma) \setminus \text{Assoc}(G, \Sigma)$ and $\text{Assoc}(G, \Sigma) \setminus \text{Assoc}(G \oplus \Delta G, \Sigma)$ are computed without any unnecessary invalid attempts in algorithm PIncDeduce.

Proof: It is ensured by (a) new associations deduced in PIncDeduce only have one chance to become invalid, and (b) once an association is confirmed invalid, it cannot become valid anymore.

Theorem 7: PIncDeduce is parallel scalable relative to its sequential algorithm.

Proof: We show that with $p$ processors. Its sequential algorithm first computes update triggers just like PIncDeduce does. Then it constructs work units and computes them in $G$. It takes $O(|\Sigma||\Delta G||V|^3)$ time. Now we prove that PIncDeduce runs in $O(|\Sigma||\Delta G||V|^3/p)$ time for $p \leq |V|^3 + T/(|\Sigma||\Delta G|)$. First, computing update triggers and work units takes $O(|\Sigma||\Delta G|)$ time; then the work units computing in parallel takes $O(|\Sigma||\Delta G|(|V|^3 + T/p))$. When $p \leq |V|^3 + T/(|\Sigma||\Delta G|)$, taken together, the cost of PIncDeduce is $O(|\Sigma||\Delta G||V|^3/p)$. This verifies the relative parallel scalability of PIncDeduce.

7 EXPERIMENTAL STUDY

Using real-life and synthetic graphs, we experimentally evaluated the accuracy, efficiency and scalability of our (incremental) association deduction algorithms. We also conducted a case study to demonstrate the effectiveness of GROs with real-life data.

Experimental setting. We used four real-life graphs as summarized in Table 1. We also generated synthetic graphs with size up to 300 million vertices and a billion edges, with $L$ drawn from an alphabet of 30 labels, and $F_2$ assigning 5 attributes with values from an active domain of 1000 values, to test scalability.

Oracles. We utilize both external and internal oracles to enhance our analysis. External oracles include (1) up-curve researcher detection [4] for DBLP, which identifies researchers who are currently producing high-quality work, and (2) movie scores for IMDB from douban, a Chinese social media platform that enables users to rate and review movies, books, and other forms of media.

Internal oracles consist of (1) avg and count operators for papers and authors in DBLP, (2) avg and count operators for movies in IMDB, (3) rank operator for people in DBpedia and YAGO2, and (4) ML models that follow GARS [10] for all datasets, including two well-trained ML classifiers: SimpleE [30] and ComplEx [42].

GRO generator. For each graph, we generated GROs as follows. (a) We added all external oracles $f(x)$ for the nodes $x$ and took $f(x)$ as an attribute value of $x$; for the external oracle $f(x, y)$, we added a new edge between $x$ and $y$ with the label $f$. (b) We added all missing links predicted by the ML classifier between the nodes in each graph. (c) We designed an internal oracle set $F$ and polynomial algorithms to compute them. (d) We then used an extension of the discovery algorithm for GFDs [12] to discover GROs. This algorithm interleaves vertical spawning to extend the patterns $Q$ and horizontal spawning to find dependencies $X \rightarrow Y$.

We made several modifications for GROs. First, we used pivoted
match semantics instead of isomorphism semantics. We assume that only nodes of specified types can be used as pivots, and once a pivot is designated, it will not be changed during the expansion process. Second, when discovering dependencies \((X \rightarrow Y)\), we consider not only constant and variable attribute predicates, but also oracles that have been designed. The oracles are computed by its designed algorithm and then used as boolean predicates in the expansion procedure. Third, for each discovered GRO \(\varphi = Q(x)(X \rightarrow Y)\), we replaced an edge \(e = (x, r, y)\) with the predicate ML model \(M(x, y, t)\) if its match was obtained primarily by models ML \((f(x, y))\). If there exists a predicate \(x.A = \text{true}\) and \(A\) imported by an external oracle \(f\) in \(F\), we replace \(x.A = \text{true}\) with the predicate \(f(x)\).

We discovered and manually selected 200 GROs for the real-life datasets and the synthetic graph. These GROs have at most 7 nodes in their patterns and the average number of predicates is 4.

We also discovered and then selected 200 GARs following [10] from DBpedia, YAGO2, DBLP, IMDB, and the synthetic graph, respectively. To make it fair, these rules have a similar number of pattern nodes and literals with selected GROs. \(\Delta G\). We generated random updates \(\Delta G\) for real-life and synthetic graphs, controlled by the size \(|\Delta G|\) and the ratio \(\tau\) of edge deletions to insertions. We set \(\tau\) to 1 by default, and i.e., the sizes of the graphs remain stable after updates.

Baseline. Apart from implementing SDeduc, PDeduce (Section 5) and PlncDeduce (Section 6) in C++, we also compared with the following baselines. (1) A variant PDeduce\(_N\) of PDeduce, without workload balancing; and a variant PlncDeduce\(_N\) of PlncDeduce without removal delay strategy. (2) The sequential deducing association method for GARs [10], denoted as SGAR, its parallel version, denoted as PGAR, and its incremental version, denoted as PlncGAR. We did not compare our approach with other association deduction algorithms, as GARs have been shown to possess greater expressive power than existing graph rules and have defeated them in [10].

Accuracy. The accuracy, to evaluate the quality of associations deduced, is evaluated over a noisy version of each real-life graph. Following [5, 10, 21], we treated the original graphs as “correct” and introduced noises by randomly removing 3% edges and 3% attributes of each data set. We measured the accuracy by precision, recall, and F-score, which are defined as (1) the ratio of removed associations deduced to all associations deduced by the methods, (2) the ratio of associations correctly deduced to all associations removed, and (3) \(2 \cdot (\text{precision} \cdot \text{recall})/(\text{precision} + \text{recall})\), respectively.

The experiments were conducted on GRAPE [22], deployed on an HPC cluster of up to 10 machines connected by 10Gbps links. For each machine, we used 2 processors powered by Intel Xeon 2.2GHz and 64G memory. Each experiment was run 5 times. The average is reported here.

Experimental results. Next, we report our findings.

Exp-1: Accuracy. We first tested the accuracy of GROs. Table 2 reports the accuracy of GROs and GARs over four real-life graphs. It shows that both GROs and GARs have very high precision, up to 99%. However, on average, GROs beats GARs on recall and F-score by 16.4% and 13.4%, respectively. These results can be attributed to the following reasons: (a) Besides ML models, GROs includes more computation models, which contribute to its high recall in predictions. (b) GROs employ simulation-based semantics, which provide a more relaxed interpretation of patterns while still preserving their topological structures.

Exp-2: Efficiency. We next evaluated the efficiency of sequential algorithm SDeduc, and parallel algorithms PDeduce and IncDeduce. The number \(|\Sigma|\) of rules, the average number of nodes \(|\Sigma_G|\) of the patterns in \(\Sigma\), the size \(|\Delta G|\) of updates for incremental deduction, and the number \(n\) of processors for parallel algorithms were fixed as 120, 5.9, 10\%|\(G|\) and 12, respectively, unless otherwise stated.

Exp-2-1: Sequential cost. Figure 4(r) reports the cost of sequential SDeduc and SGAR. Algorithm SDeduc performs reasonably well: it takes up to 6.37 hours on DBpedia. SDeduc beats SGAR by 4.36, 4.91, 4.01, 4.63 times on DBpedia, YAGO2, DBLP, and IMDB, respectively.

Exp-2-2: Parallel cost. Figures 4(a)-(4(h) reports the parallel cost of PDeduce and PlncDeduce by varying \(|\Sigma|\) and \(|\Sigma_G|\). Based on the results, we can summarize the following findings.

(1) Parallel deducution. (a) On average, PDeduce is 5.4, 6.9,4.7, 2.3 times faster than PGAR on DBpedia, YAGO2, DBLP, and IMDB, respectively. This verifies that the simulation-based semantic significantly enhances the efficiency of deducing associations. (b) On average, PDeduce is on average 2.7, 2.5,2.0, 2.0 times faster than PDeduce\(_N\), validating that the optimization strategies really work.

(2) Incremental deduction. On average, PlncDeduce is 4.1, 2.3, 4.3, 1.7 times faster than PGAR on the DBpedia, YAGO2, DBLP, and IMDB, respectively. It is also on average 2.0, 1.9,2.7, 1.7 times faster than PlncDeduce\(_N\). These results also highlight the advantages of using simulation-based semantics and optimization strategies.

(3) PlncDeduce outperforms PDeduce on average 1.5, 4.5, 2.4, 2.8 times on the DBpedia, YAGO2, DBLP, and IMDB, respectively, demonstrating the necessity of an incremental algorithm.

Exp-2-3: Parameters. We now evaluate the effect of the parameters for parallel algorithms.

Varying \(|\Sigma|\). Varying \(|\Sigma|\) from 40 to 200, Figures 4(a)-(4(h) show that (a) the more rules are used, the longer all parallel algorithms take. (b) PDeduce and IncDeduce are feasible with real-life GROs, e.g., they take 20.45 s and 11.40 s over DBpedia when \(|\Sigma| = 200\) as opposed to 40.93 s by PDeduce\(_N\) and 21.02 s by PlncDeduce\(_N\). The results on YAGO2, DBLP, and IMDB are consistent.

Varying \(|\Sigma_G|\). We varied \(|\Sigma_G|\) from 3 to 7. As shown in Figures 4(e)-(4(f), (a) all algorithms take longer on larger \(|\Sigma_G|\), as expected. (b) PDeduce and IncDeduce are feasible with real-life GROs, e.g., they take 17.7s and 4.2s over DBpedia when \(|\Sigma_G| = 5\), as opposed to 304.5s by PDeduce\(_N\) and 33.9s by PlncDeduce\(_N\). The results on YAGO2, DBLP, and IMDB are consistent.

Varying \(|\Delta G|\). Varying \(|\Delta G|\) from 5% up to 25% of \(||G|\|\), Figures 4(i)-(4(k) report the results on four datasets. (1) PlncDeduce is 1.6 to 1.1 (resp. 4.7 to 4.3, 2.5 to 1.7, and 3.0 to 1.2) times faster than PDeduce over the DBpedia (resp. YAGO2, DBLP, and IMDB) when \(|\Delta G|\) varies from 5% to 25%. (2) PlncDeduce beats PDeduce even when \(|\Delta G|\) is up to 25% of \(||G|\|\). This justifies the need for the incremental deduction. (3) All incremental methods take longer for larger \(|\Delta G|\), while batch methods are indifferent to \(|\Delta G|\).

Exp-3: Scalability. In the same default setting as for Exp-2, we
next evaluated the scalability of the deduction approaches.

Varying $n$. We varied the number $n$ of processors from 4 to 20. Figures 4(m) to 4(p) show the results. On the DBLP, DBpedia, YAGO2, and IMDB datasets.

1. **Parallel deduction.** (a) PReduce scales well: the improvement is 4.2, 3.5, 4.5, and 4.1 times, respectively, when $n$ varies from 4 to 20. (b) PReduce works well on real-life graphs: it takes only 2091 s, 835 s, 1064 s, respectively, to process 10% updates using 20 processors.

2. **Incremental deduction.** (a) PIncDeduce scales well: the improvement is 3.3, 2.8, 3.5, and 3.3 times, respectively, when $n$ varies from 4 to 20. (b) PIncDeduce works well on real-life graphs: it takes only 2091 s, 835 s, 1064 s, respectively, to process 10% updates using 20 processors.

**Impact of $G$**. Varying the scale factor from 0.2 to 1.0, we tested (incremental) association deduction on synthetic graphs. Fig. 4(q) shows: (a) all algorithms take longer over larger $G$. (b) PReduce and PIncDeduce are feasible on large graphs, taking 131.42 s, 72.11 s using 120 GROs on graphs with 1.7 million nodes and 5 million edges. In contrast, PGAR and PIncGAR ran over 1857 s and 622 s.

**Exp-4: Case study.** We take two examples during a program about Heritage Security Risk Analysis using big data technology, shown...
in Figure 4(t) and Figure 4(s).

**Institution recommendation.** To form program teams, we have a GRO \( g'_a = Q'[x, y](\text{Norisk}(x) \land \text{count}(z) \geq 2 \rightarrow (x, \text{recom}, y)) \).

It recommends an institution \( x \) for collaboration with program \( y \) if \( x \) poses no risk and has participated in related programs \( y' \), where programs are related if they share at least two keywords, and Norisk is an oracle derived from external information.

Figure 4(s) shows the rule identified that the “Jingyi” recommended by our knowledge graph, was at risk of being sold by their parent company through a news announcement in the Announcement of Listed Companies. This early detection allowed us to adjust the team and keep the program on track.

**Risk detection.** We devised the rule GRO \( \phi = Q(z)(\text{IsLoss}(z) \rightarrow \text{TheftRisk}(z))) \) to detect heritage item theft risks. It states that if a listed-for-sale heritage item is reported as lost, it is considered at risk of theft. Figure 4(s) shows the rule found the “Heaven and Earth Flow” wood carving being sold on “7788” and reported as stolen on the website of “Stolen Cultural Relics Information Publishing Platform of China”. We reported our findings to the Public Security Bureau, and the heritage was successfully reclaimed.

**Summary.** We find the following. (1) GROs are highly effective in association deduction. On average, GROs have precision above 97%. And it achieves a recall rate of 72.5%, surpassing the performance of the SOTA method with GARs [10] by 23.6%. (2) Algorithms PDeduce scale well with the number \( n \) of processors and large graphs: their runtime is improved by 4.1 times on average when \( n \) varies from 4 to 20, and it beats PGAR by 5.2 times on graphs with 1.3 billion nodes and edges. (3) Incremental PncDeduce beats batch PDeduce by 2.1 times on average when \( |AG| \) is 10%\(|G|\) and works better even when \( |AG| \) is up to 25%\(|G|\). (4) The optimization strategies are effective: it improves the performance of PDeduce and PncDeduce by 2.0 and 1.83 times on average, respectively.

**8 RELATED WORK**

We categorize related work as follows.

**Graph dependencies and association rules.** Dependencies and association rules are being investigated to catch semantic inconsistencies and extract relations from graphs, respectively. Various graph dependencies [13–15, 21], association rules [8, 10, 19, 20] and repairing rules [5] have been studied for property graphs. These graph dependencies and association rules are often defined in the form of \( Q(x)(X \rightarrow Y) \), where \( Q \) is a topological pattern and \( X \rightarrow Y \) is an attribute dependency (possibly empty). They are differentiated from each other with the expressive power and complexity, as indicated by the syntax of pattern \( Q \) and dependency \( X \rightarrow Y \).

The novelty of this work consists of the following.

1. This research incorporates both external and internal oracles into graph rules. External oracles leverage knowledge from applications or knowledge experts, while internal oracles enable computations on the match of \( Q \). Many practical operations, such as aggregates, regular expression matching, fall within the \( \text{PTIME} \) complexity class, making them suitable for efficient online oracles.

2. Unlike traditional isomorphism and homomorphism semantics, this work adopts simulation-based semantics for graph rules. Simulation-based semantics relaxes constraints, reducing pattern matching complexity from \( \text{NP} \)-complete to \( \text{PTIME} \), enhancing the applicability of graph rules in practical scenarios.

**Relaxed pattern matching.** Graph simulation, a more flexible matching semantics with low polynomial complexity, was proposed in [36]. Three representative extensions include dual simulation [33], strong simulation [33], and bounded simulation [17]. The first algorithm for graph simulation is proposed in [28], and a refined version improves it in quadratic time [11]. Incremental algorithms have been studied for simulation [16, 17]. There have also been parallel algorithms for graph simulation [18, 23, 24, 35], strong simulation [33, 43] and dual simulation [24, 40].

These techniques are based on (1) distribute graph processing models, e.g., [18] integrates partial evaluation and message passing, [16, 23] parallelizes and incrementalizes the sequential algorithm [28] under GRAPE, a graph-centric graph programming model; [24] follows Pregel for simulation, and [40] computes dual simulation on RDF data in GraphX; (2) data locality enhancement, e.g., strongly connected components defined in [35] for simulation matches to be within an \( \text{AFF} \) area identified in [17], a ball of maximum diameter of patterns [33] for strong simulation, and a ball of maximal diameter \( d_Q \) of pattern trees [43] for strong simulation.

This work differs from the prior work in the following.

1. Our sequential algorithm utilizes a smaller ball size compared to the one used in strong simulation [33] to implement the pivoted matching algorithm in an efficient manner.

2. Our parallel algorithm introduces concepts such as \( \text{necessary affected area} \) for border nodes and \( \text{bounded necessary affected area} \) to reduce communication costs.

3. Our incremental algorithm incorporates an \( \text{association dependency graph} \) and a \( \text{removal delay strategy} \) to maintain the affection relationship for each association. This helps to minimize the number of times we need to recheck invalid associations that are affected by an edge \( e \) but remain valid due to other chasing sequences.

**9 CONCLUSION**

The novelty of the work consists of the following: (1) a class of graph rules GROs for deducing associations that support oracles to incorporate external knowledge and are defined in terms of a revised notion of dual simulation; the rules depart from previous rules in their ability to import external computations and its \( \text{PTIME} \) complexity; (2) a sequential association deduction algorithm with GROs in \( \text{PTIME} \), and a parallel deduction algorithm with the parallel scalability; and (3) a parallel incremental deduction algorithm with the parallel scalability. Our experimental study has verified that GROs are promising in real-life applications.

Future work includes the following three topics: (a) exploring applications of GROs e.g., risk analysis, anomaly detection, public opinion analysis, among others; (b) collecting oracles and ML models for GROs; and (c) developing algorithms for discovering GROs.

**ACKNOWLEDGMENTS**

This work was supported by National Key R&D Program of China (No. 2021YFC2600501), and National Natural Science Foundation of China (61972275).