Discovering Graph Differential Dependencies

Yidi Zhang[1], Selasi Kwashie[2], Michael Bewong[3], Junwei Hu[1], Arash Mahboubi[3], Xi Guo[1(✉)], and Zaiwen Feng[1(✉)]

[1] College of Informatics, Huazhong Agricultural University, Wuhan, Hubei, China
{xguo,zaiwen.feng}@mail.hzau.edu.cn
[2] AI and Cyber Futures Institute, Charles Sturt University, Bathurst, NSW, Australia
[3] School of Computing, Mathematics and Engineering, Charles Sturt University, Wagga Wagga, NSW, Australia

Abstract. Graph differential dependencies (GDDs) are a novel class of integrity constraints in property graphs for capturing and expressing the semantics of *difference* in graph data. They are more expressive, and subsume other graph dependencies; and thus, are more useful for addressing many real-world graph data quality/management problems. In this paper, we study the *general discovery problem* for GDDs – the task of finding a non-redundant and succinct set of GDDs that hold in a given property graph. Indeed, we present characterisations of GDDs based on their semantics, extend existing data structures, and device pruning strategies to enable our proposed level-wise discovery algorithm, GDDMiner, returns a *minimal cover* of valid GDDs efficiently. Further, we perform experiments over three real-world graphs to demonstrate the feasibility, scalability, and effectiveness of our solution.

Keywords: Graph differential dependency · Graph dependencies · Data dependencies · Dependency discovery

1 Introduction

The study of *graph dependencies* is receiving increasing research attention in recent years due to the ubiquity of graph data and the usefulness of dependencies for addressing various graph data quality as well as graph data management problems (cf. [7,10,15,21]). *Graph differential dependencies* (GDDs) [15] are a novel class of graph dependencies that encode the semantics of difference among property values within topological patterns in a graph.

A graph differential dependency (GDD) σ over a graph G is a pair $\sigma : (Q[\bar{z}], \Phi_L(X) \rightarrow \Phi_R(Y))$ – where $Q[\bar{z}]$ is a topological pattern (*a.k.a.* graph pattern) in G serving as the "loose schema" (or scope) over which the differential dependency $\Phi_L(X) \rightarrow \Phi_R(Y)$ holds on the sets of attributes X, Y of $Q[\bar{z}]$. Intuitively, σ states that for any *homomorphic match*, h, of $Q[\bar{z}]$ in G, if h satisfies the closeness constraint(s) specified by $\Phi_L(X)$, then it must also satisfy the closeness constraint(s) in $\Phi_R(Y)$.

Z. Bao et al. (Eds.): ADC 2023, LNCS 14386, pp. 259–272, 2024.
https://doi.org/10.1007/978-3-031-47843-7_18

Motivated by the growing literature on the usefulness of graph dependencies for entity linking and resolution (ELR) across relations and graphs [8,15,18], consistency and fact checking in (social media) networks [11,16], synthetic knowledge graph generation [12], etc., this paper studies the problem of mining/discovering GDDs in graphs. That is, the problem of finding a complete, non-redundant, and concise set of GDDs that hold in a given property graph.

To the best of our knowledge, the only GDD mining solution in existence is presented in [15]. However, the GDD discovery algorithm of [15] mines a subclass of GDD with fixed consequent constraints for ELR. That is, the proposed algorithm in [15] is essentially a *reduction algorithm* not a *general discovery algorithm*, hence, incapable of finding the full class of GDDs.

Thus, in this work, we propose and study the general GDD discovery problem, and present an efficient and effective solution for the task. Specifically, the main contributions of this paper are summarised as follows. First, we introduce, formulate, and formalise a general GDD discovery problem based on GDD semantics (cf. Sect. 3). Secondly, we propose a simple, yet efficient and effective discovery algorithm for finding a succinct and non-redundant set of *any* GDD that holds in a given property graph G. Indeed, we device various data structures and pruning strategies to ensure a fast and effective traversal of the huge search space of candidate GDDs. Further, to aid effective use of the discovered rules, we introduce a semantically meaningful ranking measure and present a ranking of the discovered GDDs (see Sect. 4). Finally, we perform extensive experiments on real-world graphs to show the feasibility, scalability, and effectiveness of our proposals (cf. Sect. 5).

Related Works. The problem of mining data dependencies is a well-studied subject in the relational data context (cf. [1,19,23] for review of works on the topic). Of particular note, are the works [5,13,14,22] on distance-based dependency discovery from relational data. However, none of these approaches can be applied to mine GDDs as they are semantically different.

The works in [9,17,24] on various graph dependency mining are relatively closer to ours: [9] considers graph functional dependency discovery (GFD); [17,24] propose graph entity dependency (GED) and approximate GED mining techniques respectively. However, GFDs and GEDs can be considered as special cases of GDDs where all distance thresholds are set to zero. Therefore, just like the approach in [15], these solutions are only capable of finding a limited/subclass of GDDs. We seek a more general GDD discovery solution in this work.

2 Preliminaries

This section presents relevant definitions used in the paper, following [10,15]. We use $\mathcal{A}, \mathcal{L}, \mathcal{C}$ to denote sets of *attributes, labels,* and *constants* respectively.

2.1 Property Graph, Graph Pattern, and Matching

We consider directed **property graph**, $G = (V, E, \lambda, \rho)$, where: a) V is a finite set of nodes; b) E is a finite set of edges, given by $E \subseteq V \times \lambda \times V$; c) λ is a

function that assigns labels to node (resp. edge) from $\mathcal{L}$, e.g., $e = (u, l, v) \in E$ is an edge from node $u \in V$ to node $v \in V$, with label $\lambda(e) = l$; d) each node $v \in V$ has a label $\lambda(v)$ drawn from $\mathcal{L}$, and two key attributes id and eid, where id is the identity of the node, and eid is the identity of the real-world entity represented by the node (value often unknown/unavailable). Further, every node, v, has an associated list $\rho(v) = [(A_1, c_1), \cdots, (A_n, c_n)]$ of attribute-value pairs, where $c_i \in \mathcal{C}$ is a constant, $A_i \in \mathcal{A}$ is an attribute of v, written as $v.A_i = c_i$.

A **graph pattern**, denoted by $Q[\bar{z}]$, is a directed graph $Q[\bar{z}] = (V_Q, E_Q, \lambda_Q)$, where: (a) V_Q and E_Q represent the set of pattern nodes and pattern edges respectively; (b) λ_Q is a label function that assigns a label to each node $v \in V_Q$ and each edge $e \in E_Q$; and (c) $\bar{z}$ is the list of all nodes, called (pattern) variables in V_Q. All labels are drawn from $\mathcal{L}$, including the wildcard "*" as a special label. Two labels $l, l' \in \mathcal{L}$ are said to *match*, denoted $l \asymp l'$ iff: (a) $l = l'$; or (b) either l or l' is "*".

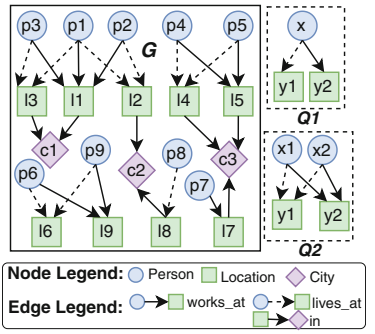

Fig. 1. A graph, G; and 4 graph patterns $Q_1 - Q_4$

Given a graph $G = (V, E, \lambda, \rho)$ and a graph pattern $Q[\bar{z}] = (V_Q, E_Q, \lambda_Q)$, a ***match*** of the graph pattern $Q[\bar{z}]$ in G is a homomorphism h from Q to G such that: (a) for each node $v \in V_Q$, there exists a node $h(v) \in G$ such that $\lambda_Q(v) \asymp \lambda(h(v))$; and (b) each edge $e = (u, l, v) \in E_Q$, there exists an edge $e' = (h(u), l', h(v))$ in G, such that $l \asymp l'$.

Given two graph patterns, $Q[\bar{z}] = (V_Q, E_Q, \lambda_Q)$ and $Q_1[\bar{z_1}] = (V_{Q_1}, E_{Q_1}, \lambda_{Q_1})$, we say $Q[\bar{z}]$ ***subsumes*** $Q_1[\bar{z_1}]$, denoted by $Q[\bar{z}] \sqsupseteq Q_1[\bar{z_1}]$, if $V_Q \supseteq V_{Q_1}$, $E_Q \supseteq E_{Q_1}$, and for each node $v_1 \in V_{Q_1}$ (resp. edge $e_1 \in E_{Q_1}$), we have $\lambda_{Q_1}(v_1) \asymp \lambda_Q(v), v \in V_Q$ (resp. $\lambda_{Q_1}(e_1) \asymp \lambda_Q(e), e \in E_Q$) and $\bar{z} \supseteq \bar{z_1}$.

We denote the set of all matches of $Q[\bar{z}]$ in G by $H(Q[\bar{z}], G)$. An example of a graph, graph patterns and their matches is illustrated in Example 1.

Example 1 (Graph, Graph Pattern and Matching). Figure 1 depicts a simple graph, G, (node properties are not shown); and two graph patterns. We illustrate the semantics of the graph patterns, and present their *matches* in G as follows:

- $Q_1[x, y_1, y_2]$ describes a `Person` node, x, with `lives_at` and `works_at` relationships with two `Location` nodes, y_1, y_2, respectively. The list of matches of pattern Q_1 in the example graph G is $H_1(x, y_1, y_2) = [\{p_3, l_3, l_1\}, \{p_1, l_3, l_1\}, \{p_1, l_2, l_1\}, \{p_2, l_2, l_1\}, \{p_4, l_4, l_5\}, \{p_5, l_4, l_5\}, \{p_6, l_6, l_9\}, \{p_9, l_6, l_9\}]$.
- $Q_2[x_1, x_2, y_1, y_2]$ specifies two `Person` nodes x_1, x_2, with the same `lives_at` and `works_at` relationships with two `Location` nodes y_1, y_2, respectively. Thus, the matches of Q_2 in G are $H_3(x_1, x_2, y_1, y_2) = [\{p_1, p_3, l_3, l_1\}, \{p_1, p_2, l_2, l_1\}, \{p_4, p_5, l_4, l_5\}, \{p_6, p_9, l_6, l_9\}]$.

Further, the following subsumption relation exist between the graph patterns in Fig. 1: $Q_2[x_1, x_2, y_1, y_2] \sqsupseteq Q_1[x, y_1, y_2]$. □

2.2 Graph Differential Dependency (GDD)

GDD Syntax. A *graph differential dependency* is a pair $\sigma : (Q[\bar{z}], \Phi_L(X) \rightarrow \Phi_R(Y))$, where: $Q[\bar{z}]$ is a graph pattern (*a.k.a.* the scope), and $\Phi_L(X) \rightarrow \Phi_R(Y)$ is called the dependency. $\Phi_L(X)$ and $\Phi_R(Y)$ are two (possibly empty) sets of distance constraints on the pattern variables $\bar{z}$. A distance constraint in $\Phi_L(X)$ and $\Phi_R(Y)$ on $\bar{z}$ is one of the following:

$$\delta_A(x.A, c) \leq t_A; \quad \delta_{A_1A_2}(x.A_1, x'.A_2) \leq t_{A_1A_2};$$
$$\delta_{\mathsf{eid}}(x.\mathsf{eid}, c_e) = 0; \quad \delta_{\mathsf{eid}}(x.\mathsf{eid}, x'.\mathsf{eid}) = 0;$$
$$\delta_{\equiv}(x.rela, c_r) = 0; \quad \delta_{\equiv}(x.rela, x'.rela) = 0;$$

where $x, x' \in \bar{z}$; A, A_1, A_2 are attributes in $\mathcal{A}$; c is a value of A; $\delta_{A_1A_2}(x.A_1, x'.A_2)$ or $\delta_{A_1}(x, x')$, if $A_1 = A_2$ is a user specified distance function over values of A_1, A_2; $t_{A_1A_2}$ is a threshold for $\delta_{A_1A_2}(\cdot, \cdot)$; $\delta_{\mathsf{eid}}(\cdot, \cdot)$ (resp. $\delta_{\equiv}(\cdot, \cdot)$) is a function on eid (resp. relations), and returns 0 or 1. $\delta_{\mathsf{eid}}(x.\mathsf{eid}, c_e) = 0$ if the eid value of x is c_e, $\delta_{\mathsf{eid}}(x.\mathsf{eid}, x'.\mathsf{eid}) = 0$ if both x and x' have the same eid value, $\delta_{\equiv}(x.rela, c_r) = 0$ if x has a relation named `rela` and ended with the node c_r, $\delta_{\equiv}(x.rela, x'.rela) = 0$ if both x and x' have the relation named rela and ended with the same node.

The user-specified distance function $\delta_{A_1A_2}(x.A_1, x'.A_2)$ is dependent on the types of A_1 and A_2. It can be an arithmetic operation of interval values, an edit distance of string values or the distance of two categorical values in a taxonomy, etc. The functions handle the wildcard value "*" for any domain by returning the 0 distance. We call $\Phi_L(X)$ and $\Phi_R(Y)$ the LHS and the RHS functions of the dependency respectively.

GDD Semantics. Given a GDD, $\sigma = (Q[\bar{z}], \Phi_L(X) \rightarrow \Phi_R(Y))$, and a match h of $Q[\bar{z}]$ in a graph G: h satisfies $\Phi_L(X)$, denoted by $h \vDash \Phi_L(X)$, if h satisfies every distance constraint in $\Phi_L(X)$; and $h \vDash \Phi_R(Y)$ is defined in the same way.

Let $H(Q[\bar{z}], G)$ be all the matches of $Q[\bar{z}]$ in G. We write $H(Q[\bar{z}], G) \vDash \sigma$, iff: for every $h \in H(Q[\bar{z}], G)$, if $h \vDash \Phi_L(X)$, then $h \vDash \Phi_R(Y)$ also. That is, G satisfies σ, denoted by $G \vDash \sigma$, if $H(Q[\bar{z}], G) \vDash \sigma$.

Let $sat(\Phi(X))$ denote the set of all matches in $H(Q[\bar{z}], G)$ that satisfy a constraint $\Phi(X)$ over $Q[\bar{z}]$. Then, Lemma 1 holds by the semantics of GDDs.

Lemma 1 (GDD satisfaction). *Given the matches $H(Q[\bar{z}], G)$ of $Q[\bar{z}]$ in G, $G \vDash \sigma = (Q[\bar{z}], \Phi_L(X) \rightarrow \Phi_R(Y))$ if and only if: $sat(\Phi_L(X)) \subseteq sat(\Phi_R(Y))$.* □

We illustrate the semantics of GDDs in Example 2 below.

Example 2 (GDD Semantics). Consider the GDD $\sigma_1 : (Q_2[x_1, x_2, y_1, y_2], \Phi_L(X) \rightarrow \Phi_R(Y))$, where $\Phi_L(X) = \{\delta_{\mathsf{name}}(x_1, x_2) \leq 1, \delta_{\mathsf{dob}}(x_1, x_2) \leq 2\}$ and $\Phi_R(Y) = \{\delta_{\mathsf{eid}}(x_1, x_2) = 0\}$. σ_1 states that for any two `Person` nodes x_1, x_2 that share the same living and working locations y_1, y_2 in G, if the distances between the `name` and `dob` attributes of x_1, x_2 as measured by the respective distance functions are within 1 and 2 respectively, then the nodes x_1, x_2 refer to the same real-world entity (i.e., $\delta_{\mathsf{eid}}(x_1, x_2) = 0$). Intuitively, σ_1 asserts that for any match of Q_2 in G, if the `Person` nodes have similar `name` and `dob` values, then they refer to the same person in the real-world. Thus, σ_1 is an example of entity linking rule. □

3 Problem Formulation

Given a property graph, G, the set of valid dependencies in G are often large, which affects their utility in downstream tasks. In this work, we are interested in finding a succinct set of GDDs in G. Thus, in the following, we present relevant definitions on our GDDs of interest, and present the formal problem definition.

Representative Graph Patterns. Recall, a GDD is a pair: $Q[\bar{z}]$, $\Phi_L(X) \rightarrow \Phi_R(Y)$. The graph pattern $Q[\bar{z}]$ of a GDD is the scope (i.e., "loose schema") over which the dependencies hold. Therefore, it is important to find "representative" graph patterns for the dependencies. We consider a graph pattern to be ***representative*** if it is *frequent* and *non-redundant*, as defined below.

Frequent Graph Pattern. Let the set $I = \{i_1, \cdots, i_m\}$ be the set of isomorphisms of a graph pattern $Q[\bar{z}]$ in a graph G, where $\bar{z} = x_1, \cdots, x_n$; and let $D(x_i) = \{i_1(x_i), \cdots, i_m(x_i)\}$ be the set of distinct nodes in G that map onto x_i via the isomorphisms. The *minimum image based support* [4] (MNI)[1] of $Q[\bar{z}]$ in G is defined as:

$$mni(Q[\bar{z}], G) = \min\{|D(x_i)|, \forall\ x_i \in z\}.$$

Thus, given a minimum MNI support, τ, we say a graph pattern $Q[\bar{z}]$ is ***frequent*** in G iff: $mni(Q[\bar{z}], G) \geq \tau$.

Example 3 (Frequent Graph Pattern). Consider the graph pattern $Q_2[x_1, x_2, y_1, y_2]$ and the graph G in Fig. 1. The isomorphisms of Q_2 in G are shown in Fig. 2 a). Specifically, $D(x_1) = \{p3, p2, p4, p6\}$, $D(x_2) = \{p1, p5, p9\}$, $D(y_1) = \{l3, l2, l8, l9\}$ and $D(y_2) = \{l1, l5, l9\}$. Thus, $mni(Q_2, G) = \min\{|D(x_1)|, |D(x_2)|, |D(y_1)|, |D(y_2)|\} = \min\{4, 3, 4, 3\}$. $\therefore$ $mni(Q_2, G) = 3$; and frequent if $\tau \leq 3$. □

Non-redundant Graph Patterns. Although the frequency of graph patterns is useful for eliminating non-persistent patterns, it is not sufficient. This is because a frequent graph patterns set often has redundancies. For example, in Fig. 2 b) is the set of frequent graph patterns in the example graph G (of Fig. 1) for $\tau = 3$. In this example, some of the redundancies include: $q0 \supset q1 \supset q4$; $q6 \supset q7 \supset q8$; etc. Hence, we are interested in frequent *and* non-redundant graph patterns.

Given a minimum MNI support threshold τ, we say a graph pattern $Q[\bar{z}]$ in G is ***non-redundant*** iff: a) it is frequent, i.e., $mni(Q[\bar{z}], G) \geq \tau$; b) there exists no other pattern $Q_1[\bar{z_1}]$ where $Q_1[\bar{z_1}] \sqsupseteq Q[\bar{z}]$ in G such that $mni(Q[\bar{z}], G) = mni(Q_1[\bar{z_1}], G)$.

Example 4 (Non-redundant Graph Patterns). Given $\tau = 3$, all graph patterns in Fig. 2 b) are frequent. Graph patterns within the blue (i.e., $q10, q11$) and red (i.e., $q12$) boundaries have supports of 4 and 5 respectively, and every other pattern has a support of 3. Thus, it is easily verifiable that the non-redundant patterns are those with bold dotted line boundaries: $q0, q2, q6, q10, q11$, and $q12$.

[1] we adopt this graph pattern counting metric due to its anti-monotone property.

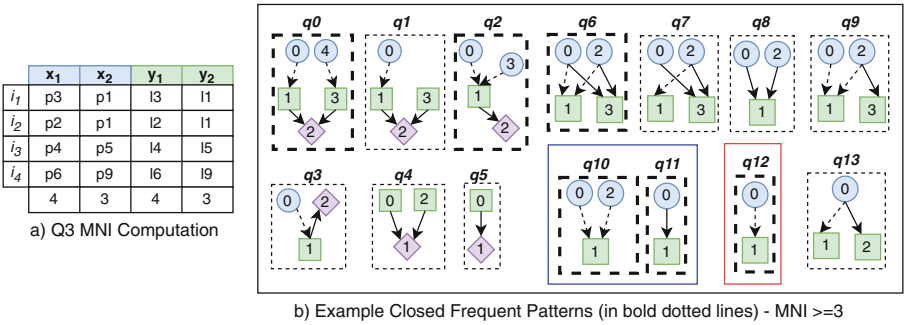

Fig. 2. Example of Representative Graph Patterns

We remark that the notion of non-redundant graph patterns is equivalent to that of *closed patterns* [3] in the frequent itemset mining literature.

Irreducible Dependencies. Without loss of generality, we consider GDDs with singleton RHSs, i.e., $\Phi_R(Y)$ has one and only one distance constraint.

The space of candidate GDDs is huge and complex as any number of distance constraints can be defined over attribute sets. Even more challenging, these constraints can interact. Let $\Phi(X_1), \Phi(X_2)$ be two sets of distance constraints over a given graph pattern $Q[\bar{z}]$.

The set $\Phi(X_1)$ is said to ***subjugate*** $\Phi(X_2)$, denoted by $\Phi(X_1) \succeq \Phi(X_2)$, iff.: (i) for every distance constraint $\delta_A(x.A, c) \leq t_A^{(1)}$, or $\delta_{A_1A_2}(x.A_1, x'.A_2) \leq t_{A_1A_2}^{(1)}$ in $\Phi(X_1)$, there exists $\delta_A(x.A, c) \leq t_A^{(2)}$, or $\delta_{A_1A_2}(x.A_1, x'.A_2) \leq t_{A_1A_2}^{(2)}$ in $\Phi(X_2)$, such that $t_A^{(1)} \geq t_A^{(2)}$, or $t_{A_1A_2}^{(1)} \geq t_{A_1A_2}^{(2)}$, respectively; and (ii) for every constraint $\delta_{\equiv/\mathsf{eid}}(\cdot,\cdot) = 0$ in $\Phi(X_1)$ there exists $\delta_{\equiv/\mathsf{eid}}(\cdot,\cdot) = 0$ in $\Phi(X_2)$ as well.

Intuitively, if $\Phi(X_1) \succeq \Phi(X_2)$, then for any match h of $Q[\bar{z}]$ in G that satisfies $\Phi(X_2)$, i.e., $h \vDash \Phi(X_2)$, it always satisfies $\Phi(X_1)$ too (i.e., $h \vDash \Phi(X_1)$).

Property 1 (Trivial GDD). For any two sets, $\Phi(X_1), \Phi(X_2)$, of constraints over $Q[\bar{z}]$, if $\Phi(X_1) \succeq \Phi(X_2)$, then GDD $(Q[\bar{z}], \Phi(X_2) \rightarrow \Phi(X_1))$ holds in G. □

Consider a set Σ of GDDs and a single GDD σ. We say Σ *implies* σ, denoted by $\Sigma \models \sigma$ if and only if $\Sigma \vDash G$ implies $\sigma \vDash G$. We want Σ without any implied rules. It, thus, suffices to ensure any GDDs in Σ is *irreducible* as defined below.

Property 2 (Irreducible GDD). A GDD $\sigma : (Q[\bar{z}], \Phi_L(X) \rightarrow \Phi_R(Y)) \in \Sigma$, is said to be *irreducible* if and only if: a) any attribute $A \in \Phi_R(Y)$ does not exist in $\Phi_L(X)$; b) there does not exists another GDD $\sigma_1 : (Q_1[\bar{z_1}], \Phi_l(X_1) \rightarrow \Phi_r(Y_1)) \in \Sigma$ s.t.: (i) $Q_1[\bar{z_1}] \sqsupseteq Q[\bar{z}]$, (ii) $\Phi_l(X_1) \succeq \Phi_L(X)$, and (iii) $\Phi_R(Y) \succeq \Phi_r(Y_1)$. □

We say a set Σ of GDDs is *minimal* iff: for any $\sigma \in \Sigma$, we have $\Sigma \not\equiv \Sigma \backslash \sigma$. That is, Σ contains no redundant GDD. Furthermore, a set Σ_c is a cover of Σ if: $\Sigma_c \subseteq \Sigma$ and $\Sigma_c \equiv \Sigma$.

Problem Definition. This paper studies the following GDD discovery problem.

Definition 1 (Discovery of GDDs). *Given a property graph, G, and a user-specified MNI threshold, τ: find a minimal cover set Σ_c of all irreducible GDDs that hold over τ-frequent and non-redundant graph patterns in G.* □

4 The Proposed GDD Mining Approach

In this section, we introduce the proposed GDD mining solution, GDDMiner. It consists of three major tasks, viz: a) finding and matching representative graph patterns (in Sect. 4.1), b) mining distance dependencies over the matches of the graph patterns (in Sect. 4.2), and c) pruning and ranking the mined rules to produce a ranked minimal cover set (in Sect. 4.3). A sketch of the pseudo-code for the proposed three-phase solution is presented in Algorithm 1.

Algorithm 1: GDDMiner

```
   Input: MNI thres. τ; dist. const. Δ (optional)
   Output: Minimal cover Σ_c of GDDs
   Data: Property graph, G
1  Σ := ∅, Σ_c : ∅
   /* find representative graph patterns */
2  Q ← rGraMi(G, τ)
   /* mine GDDs over rep. graph patterns */
3  for Q_i[z̄_i] ∈ Q do
4      H(Q_i[z̄_i], G) ← hMatches(Q_i[z̄_i], G)
5      Σ_i ← redGDDs(H, Δ)
6      Σ ← Σ ∪ Σ_i
   /* prune implications                 */
7  Σ_c ← cover(Σ)
8  return Σ_c
```

4.1 Graph Pattern Mining and Matching

Mining representative graph patterns. Frequent subgraph mining (FSM) in a given graph G is a non-trivial, and well-studied problem. In this work, we extend the well-known *constraint satisfaction problem* (CSP) based FSM solution, GraMi [6] for the task of finding *frequent* and *non-redundant* graph patterns. For brevity and space constraints, we refer interested readers to [6] for the full details of the FSM solution.

Note that GraMi returns frequent graph patterns and *not* frequent non-redundant graph patterns. Thus, to achieve our goal, we extend the underlining data structure in GraMi, the *subgraph generation tree*, to enable the return of the frequency count associated with each frequent pattern. This adaptation allows us to effectively prune redundant frequent graph patterns based on the subsumption relation of graph patterns (see Sect. 2.1), with the same frequency as illustrated in Example 3 and 4. Given a minimum MNI threshold, τ, our extended FSM algorithm, rGraMi, returns the set $\mathcal{Q}$ of all representative graph patterns (cf. line 2 of Algorithm 1).

	x_1: person			y_1: location					x_2: person			y_2: location				
	id	*Name (A1)*	*DOB (A2)*	*id*	*Type (A3)*	*Street Name (A4)*	*State (A5)*	*Zipcode (A6)*	*id*	*Name (A1)*	*DOB (A2)*	*id*	*Type (A3)*	*Street Name (A4)*	*State (A5)*	*Zipcode (A6)*
h1	p1	William Johnson	1970.7.31	l3	Res	46 Adrian Ave	SA	5120	p3	Bill Johnson	7/31/70	l1	Comm	17 Tea Tree Gully	SA	5053
h2	p1	William Johnson	1970.7.31	l2	Res	28 Main Rd	NSW	2758	p2	Mary Smith	Jun. 1990	l1	Comm	17 Tea Tree Gully	SA	5053
h3	p4	Paul H. Colbert		l4	Com	93 High St	NSW	2000	p5	Colber, P. H.	25/12/89	l5	Com	93 High Street	NSW	2000
h4	p6	Tom Engel	1 Jun. 1956	l6		99 Mawson blvd	WA	6019	p9	Engel, Thomas	06/01/65	l9	Com	199 Main Road	WA	6000

Fig. 3. An example pseudo-table of matches of Q_2 in G

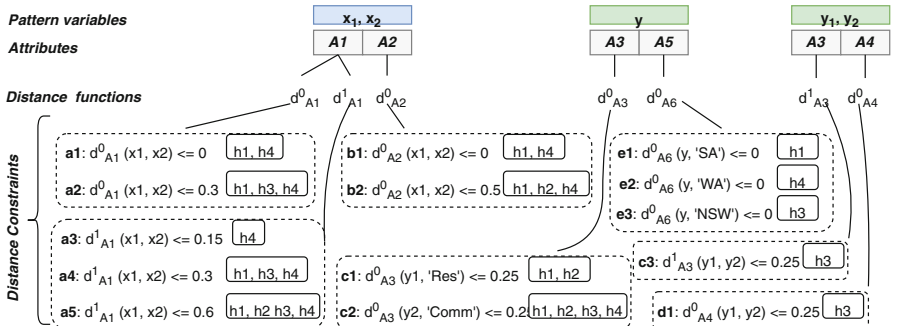

Fig. 4. Examples of distance constraints over matches of Q_2

Finding Homomorphic Matches of Graph Patterns. Given a representative graph pattern $Q[\bar{z}] \in \mathcal{Q}$, we find the set $H(Q[\bar{z}], G)$ of homomorphic matches of $Q[\bar{z}]$ in G using the efficient worst-case optimal join (WCOJ) based algorithm in [20] (i.e., line 4 of Algorithm 1). We generate a pseudo-relational table with the properties/attributes of the matches of the graph pattern. For example, Fig. 3 is an example of such a table over matches of the pattern Q_2 in G from our running example in Fig. 1.

4.2 Finding Valid GDDs over Graph Patterns

Pre-processing: Distance Functions and Thresholds. Given the set $H(Q[\bar{z}], G)$ of matches of $Q[\bar{z}]$ in G, various distance functions can be defined (with corresponding distance thresholds) over the attributes of the pattern variables based on domain knowledge. The domain knowledge may include which distance function(s) to define over specific pattern variable(s) as well as what thresholds of the distance functions are semantically meaningful. For instance, given the matches of Q_2 in Fig. 3, examples of possible distance functions and constraints are shown in Fig. 4. In exception of the `name` attribute, $A1$, each attribute has one normalised distance function with multiple constraints in Fig. 4. In particular, d^0_{A1} and d^1_{A1} measure phonetic and Sørensen-Dice distances of two `name` sets.

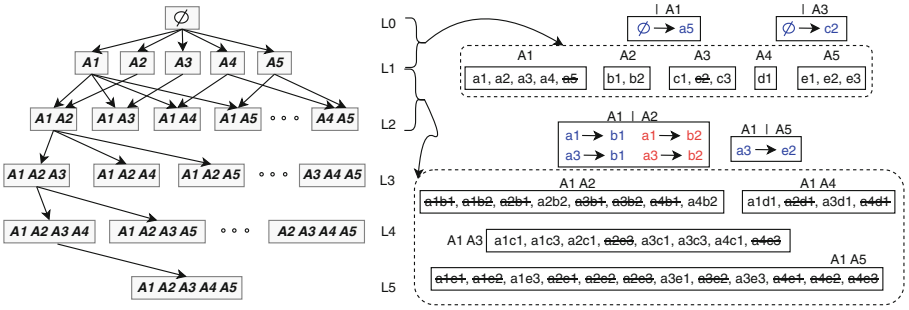

Fig. 5. Example of attribute lattice (left); and exemplar traversal & pruning of search space of candidate GDDs (right)

In this paper, we assume such domain knowledge exists, as an optional input Δ to Algorithm 1. However, in its absence, one can employ arbitrary distance functions with equidistant thresholds.

Search Space Construction and Traversal. Here, we discuss the search space of candidate GDDs. In principle, the space of possible GDDs is exponential to the number of distance constraints. Thus, we present data structures and strategies to enable effective and efficient search for valid GDDs.

We extend the attribute lattice [2] to model the GDD search space. The lattice is a level-wise directed graph with an empty-set attribute as the `root` node (at Level-0, $L0$). The first level, $L1$, consists of children nodes of $L0$ made up of all attributes of the pattern variables. Each attribute node $A_i \in L1$ is associated with a set of distance constraints defined over it. Nodes at subsequent levels are formed by combining two nodes from the previous level as follows. Let $n_{i\cdot j}, n_{i\cdot k}$ be the j-th and k-th nodes at level i respectively. $n_{i\cdot j}$ and $n_{i\cdot k}$ can be combined to form the node $n_{i+1\cdot l}$ at level $i+1$ if and only if the size of the set union of the attribute sets $|n_{i\cdot j} \cup n_{i\cdot k}| = i+1$. Further, $n_{i+1\cdot l}$ is assigned the set union of distance constraints from $n_{i\cdot j}$ and $n_{i\cdot k}$, $\Phi(X)$, if the associated satisfaction set, $sat(\Phi(X))$, computed as the set intersection of the two parent satisfaction sets is non-empty. Then, edges directed from $n_{i\cdot j}$ and $n_{i\cdot k}$ to $n_{i+1\cdot l}$. For example, Fig. 5 shows the attribute lattice (left) and associated distance constraint sets for levels $L1, L2$ (right) for our running example with 5 attributes $A1 - A5$. Note that, we only show all edges in levels $L0 - L2$ of the lattice for clarity.

We adopt a top-down left-right search strategy to traverse the lattice space.

Candidate GDD Generation and Validation. The construction of the lattice and generation of candidate GDDs is performed simultaneously in a level-wise fashion. Indeed, candidate GDDs are explored between two successive levels $Li, Li+1$ from top to down the lattice. All distance constraints at level Li form candidate LHS of the rule, whiles $Li+1$ constraints are potential RHSs.

More specifically, for every directed edge between the node $n_{i\cdot j} \in Li$ and $n_{i+1\cdot l} \in Li+1$ we form and test candidate GDDs as follows. Let $\Phi_L(X), \Phi_R(Y)$

be sets of distance constraints in $n_{i\cdot j}$ and $n_{i+1\cdot l}$ respectively. A candidate GDD is of the form: $\Phi_L(X) \rightarrow \Phi_R(A)$, where $A = Y \backslash X$, and $\Phi_R(A)$ a singleton distance constraint over A (this satisfies condition *a)* of Property 2).

For efficiency of search, and in accordance with conditions $b) - (ii)$ and $b) - (iii)$ of Property 2, we consider LHS constraints of $n_{i\cdot j}$ in decreasing order of thresholds, and conversely, the RHS constraints in the increasing order. A candidate GDD $\Phi_L(X) \rightarrow \Phi_R(A)$ is validated via Lemma 1, i.e., iff: $sat(\Phi_L(X)) \subseteq sat(\Phi_R(Y))$. Further, we use the following pruning rule: if $\Phi_L(X) \rightarrow \Phi_R(Y)$, then $\Phi_R(Y)$ is excluded from candidate generation at the next level.

Example 5 (Finding valid GDDs). Consider the the set of distance constraints over attributes $A1 - A5$ of the pattern variables in Q_2 as shown in Fig. 4. The corresponding lattice for the search space of candidate GDDs is presented in Fig. 5. To find valid GDDs, we first evaluate candidates between levels $L0/L1$. Specifically, the only possible LHS constraint at $L0$ is $\Phi_L(X) = \{\emptyset\}$; and every distance constraint (on Y) at $L1$ is as a candidate RHS $\Phi_R(A) = \Phi_R(Y \backslash X)$. That is, the set of candidate GDDs on edge $\emptyset \rightarrow A1$ is $\{\emptyset \rightarrow a1, \emptyset \rightarrow a2, \cdots, \emptyset \rightarrow a5\}$, whereas for edge $\emptyset \rightarrow A2$ is $\{\emptyset \rightarrow b1, \emptyset \rightarrow b2\}$. And, similarly for other edges between $L0/L1$. Note that $sat(\emptyset) = \{h1, h2, h3, h4\}$, i.e., every match satisfies an empty-set constraint by definition. To validate the candidate GDDs on each edge between $L0/L1$, we test if $sat(\emptyset) \subseteq sat(w)$, where w is RHS constraint. Thus, the only valid rules are: $\emptyset \rightarrow a5$ and $\emptyset \rightarrow c2$ as shown in blue in Fig. 5. Consequently, constraints $a5, c2$ can be eliminated from forming nodes in level $L2$. Distance constraints of attribute sets at $L2$ are shown alongside the valid GDDs, and pruned constraints. We show all valid irreducible GDDs in blue, and valid but implied GDDs in red. □

4.3 Finding Cover and Ranking GDDs

Pruning Implied GDDs. So far, the set Σ of GDDs produced from the foregoing are irreducible *w.r.t.* a single pattern $Q[\bar{z}]$. That is, all conditions except the graph pattern interaction condition b)-(i) in Property 2 is satisfied by all GDDs in Σ. Therefore, we ensure general irreducibility of rules in Σ via the $\mathsf{cover}(\Sigma)$ function (in line 7 of Algorithm 1).

Note that Σ is a multi-set $\{\Sigma_1, \cdots, \Sigma_k\}$ of GDDs, where each $\Sigma_i \in \Sigma$ corresponds to a set of GDDs over a particular scope $Q_i[\bar{z}_i]$. Thus, we generate a minimal cover Σ_c of Σ by eliminating all implied GDDs due to pattern interactions in Σ through two simple steps. First, for any $\Sigma_i \in \Sigma$, the full set Σ_i is added to Σ_c if and only if no other Σ_j exists in Σ such that $Q_j[\bar{z}_j] \sqsupseteq Q_i[\bar{z}_i]$. Secondly, for any pair $\Sigma_i, \Sigma_j \in \Sigma$ such that $Q_i[\bar{z}_i] \sqsupseteq Q_j[\bar{z}_j]$: any GDD $\sigma' \in \Sigma_j$ is eliminated from Σ_j if there exists $\sigma \in \Sigma_i$ such that $LHS(\sigma) \succeq LHS(\sigma')$ and $RHS(\sigma') \succeq RHS(\sigma)$. This routine is repeated for all pairs Σ_i, Σ_j until no further changes occur, and the added to Σ_c.

Ranking GDDs. In practice, the minimal cover set of dependencies, albeit without any redundancies, is often still large. This makes using of the rules in downstream applications challenging. Thus, a measure to assess the quality of the rules is useful for the selection, and adoption of the discovered rules. We present an intuitive, easy to compute, and semantically meaningful interestingness measure for GDDs, inspired by the *lift* measure (from associations rule mining literature).

Recall, the lift of rule $R = A \rightarrow B$, is given as $lift(R) = \frac{conf(A \rightarrow B)}{conf(\emptyset \rightarrow B)}$, where $conf(x \rightarrow y)$ is the confidence (a.k.a conditional probability), given as $conf(x \rightarrow y) = \frac{P(x,y)}{P(x)}$. Simply, the lift of a rule R is quotient of the posterior and the prior confidence of the rule. A lift value near 1 indicates independence. A larger than 1 lift value indicates A has a positive effect on occurrence of B, and conversely, a less than 1 value show A has an adverse effect on B occurrence.

We define the *interestingness*, $\mathsf{intr}(\sigma)$, of a GDD σ as:

$$\mathsf{intr}(\sigma) = \frac{conf(Q[\bar{z}], \Phi_L(X) \rightarrow \Phi_R(Y))}{conf(Q[\bar{z}], \emptyset \rightarrow \Phi_R(Y))} = \frac{|H(Q[\bar{z}], G)|}{|sat(\Phi_R(Y))|}. \quad (1)$$

Note that, $conf(Q[\bar{z}], \Phi_L(X) \rightarrow \Phi_R(Y)) = 1$ for all valid GDDs, and $conf(Q[\bar{z}], \emptyset \rightarrow \Phi_R(Y)) = P(Y)$. Further, the quantity $|H(Q[\bar{z}, G])|$ is fixed for any $\sigma \in \Sigma$, thus, the only determinant of interestingness of any given GDD is the size of its satisfaction set $sat(\Phi_R(Y))$ – which relies on the strictness of the threshold(s) in $\Phi_R(Y)$. Essentially, GDDs with strict RHSs, hence, smaller $|sat(\Phi_R(Y))|$ are more interesting – consistent with GDD semantics.

5 Experiments

We present an empirical evaluation of our proposed GDD mining solution, GDDMiner, using three real-world graph data sets as described in Table 1. We defined several distance functions over node attributes of the graphs; and used up to three distance constraints per attribute with equi-distant thresholds. All implementations are conducted in Java, and the experiments run on a 2.20 GHz Intel Xeon processor computer with 128 GB memory running Linux OS.

Table 1. Summary of data sets

Graph	#Nodes (N)	#Edges (E)	#N Types	#E Types
DBLP	300K	800K	3	3
IMDB	300K	650K	5	8
YAGO4	380K	800K	7764	83

Feasibility and Scalability of GDDMiner. Here, we demonstrate the feasibility and scalability of GDDMiner as follows. We set the MNI threshold, $\tau = 500$, to ensure the discovered scopes of the dependencies are persistent topological patterns in the data sets. For each dataset, we vary the number of attributes and the proportion of the matches of the pattern. The performance of GDDMiner over each dataset is presented in Fig. 6 a)–c). In general, the discovery time increases with number of attributes and the proportion of matches of the pattern (from 50% to 100%) in the graph.

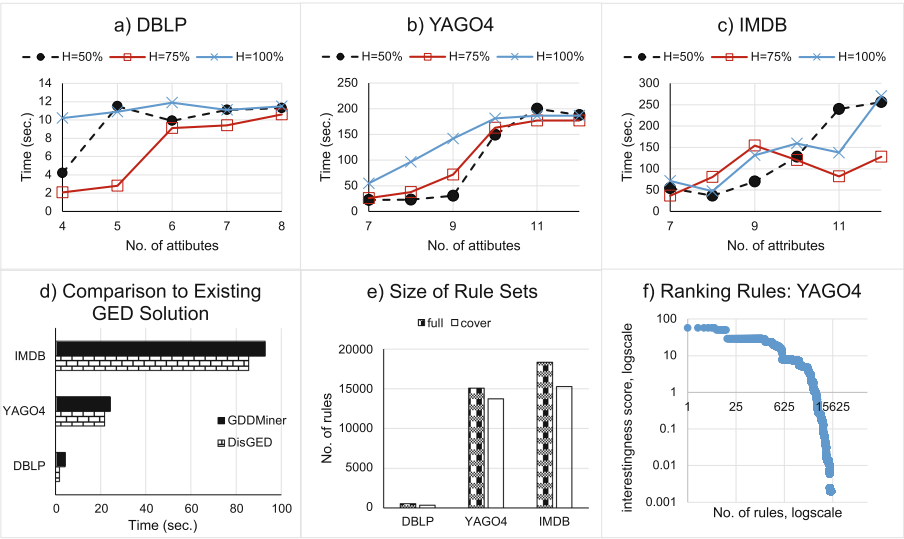

Fig. 6. Evaluation of GDDMiner on real-world graphs

Comparison to Existing Graph Dependency Miner. In this experiment, we set all distance thresholds to 0, to assess the performance of mining GEDs with our solution, compared to the GED mining technique in [17]. Figure 6 d) shows a close and comparable performance of the two algorithms on finding GEDs (i.e., 0-constraint GDDs) in all datasets. The DisGEDs algorithm [17] perform marginally better than ours, and a detailed analysis shows the advantage lies in the graph partitioning strategy used in [17] during the graph pattern mining stage. It is noteworthy that whereas our solution can be co-opted to find GEDs, DisGEDs [17] cannot be used to mine GDDs.

Discovered Rules. Figure 6 e) is a plot of the sizes of the irreducible and the minimal cover sets (i.e., Σ vs. Σ_c) found in the three graphs. It is clear that whilst further pruning of Σ to Σ_c (cf. Sect. 4.3) reduces that final set size, it is still quite large (especially for YAGO4 and IMDB). This shows the need for ranking the results for ease of use. Thus, we show the effectiveness of our interestingness measure (i.e., Eq. 1) for ranking the discovered rules in Fig. 6 f). Due to space constraints, we show only the case for the YAGO4 data as the same pattern is observed for all datasets. The scatter plot shows the ranking helps identify the most interesting GDDs easily. For example, the top 625 GDDs have at least an interestingness value of 10.

6 Conclusion

This paper introduces and studies the *general discovery problem* of GDDs. The proposed discovery algorithm, GDDMiner, leverages various properties of GDDs

to search and prune the huge candidate space effectively. Empirical evaluations of GDDMiner on real-world graphs show its feasibility and scalability.

In the future, we shall study the simultaneous mining and matching of representative graph patterns, and automatic fine tuning of the thresholds of distance constraints, to boost the performance of GDDMiner in large graphs.

Acknowledgement. This research project was supported in part by the following grant schemes: the Innovation fund of Chinese Marine Defense Technology Innovation Center under Grant JJ-2021-722-04; the Fundamental Research Funds for the Chinese Central Universities under Grant 2662023XXPY004; the Inner Mongolia Key Scientific and Technological Project under Grant 2021SZD0099.

References

1. Abedjan, Z., Golab, L., Naumann, F.: Profiling relational data: a survey. VLDB J. **24**(4), 557–581 (2015). https://doi.org/10.1007/s00778-015-0389-y
2. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proceedings of 20th International Conference on Very Large Data Bases, VLDB, vol. 1215, pp. 487–499. Santiago, Chile (1994)
3. Bastide, Y., Pasquier, N., Taouil, R., Stumme, G., Lakhal, L.: Mining minimal non-redundant association rules using frequent closed itemsets. In: Lloyd, J., et al. (eds.) CL 2000. LNCS (LNAI), vol. 1861, pp. 972–986. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44957-4_65
4. Bringmann, B., Nijssen, S.: What is frequent in a single graph? In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 858–863. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68125-0_84
5. Caruccio, L., Deufemia, V., Polese, G.: Mining relaxed functional dependencies from data. Data Min. Knowl. Disc. **34**(2), 443–477 (2020)
6. Elseidy, M., Abdelhamid, E., Skiadopoulos, S., Kalnis, P.: Grami: frequent subgraph and pattern mining in a single large graph. Proc. VLDB Endowment **7**(7), 517–528 (2014)
7. Fan, W.: Big graphs: challenges and opportunities. Proc. VLDB Endowment **15**(12), 3782–3797 (2022)
8. Fan, W., Geng, L., Jin, R., Lu, P., Tugay, R., Yu, W.: Linking entities across relations and graphs. In: 2022 IEEE 38th International Conference on Data Engineering (ICDE), pp. 634–647. IEEE (2022)
9. Fan, W., Hu, C., Liu, X., Lu, P.: Discovering graph functional dependencies. ACM Trans. Database Syst. (TODS) **45**(3), 1–42 (2020)
10. Fan, W., Lu, P.: Dependencies for graphs. ACM Trans. Database Syst. (TODS) **44**(2), 1–40 (2019)
11. Fan, W., Lu, P., Tian, C., Zhou, J.: Deducing certain fixes to graphs. Proc. VLDB Endowment **12**(7), 752–765 (2019)
12. Feng, Z., et al.: A schema-driven synthetic knowledge graph generation approach with extended graph differential dependencies (gdd x s). IEEE Access **9**, 5609–5639 (2020)
13. Kwashie, S., Liu, J., Li, J., Ye, F.: Mining differential dependencies: a subspace clustering approach. In: Wang, H., Sharaf, M.A. (eds.) ADC 2014. LNCS, vol. 8506, pp. 50–61. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08608-8_5

14. Kwashie, S., Liu, J., Li, J., Ye, F.: Efficient discovery of differential dependencies through association rules mining. In: Sharaf, M.A., Cheema, M.A., Qi, J. (eds.) ADC 2015. LNCS, vol. 9093, pp. 3–15. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19548-3_1
15. Kwashie, S., Liu, L., Liu, J., Stumptner, M., Li, J., Yang, L.: Certus: an effective entity resolution approach with graph differential dependencies (GDDs). Proc. VLDB Endowment **12**(6), 653–666 (2019)
16. Lin, P., Song, Q., Wu, Y.: Fact checking in knowledge graphs with ontological subgraph patterns. Data Sci. Eng. **3**(4), 341–358 (2018)
17. Liu, D., et al.: An efficient approach for discovering graph entity dependencies (GEDs). arXiv preprint arXiv:2301.06264 (2023)
18. Liu, J., Kwashie, S., Li, J., Liu, L., Bewong, M.: Linking graph entities with multiplicity and provenance. In: 2nd International Workshop on EntitY REtrieval: EYRE 2019, pp. 1–7. Association for Computing Machinery (ACM) (2019)
19. Liu, J., Li, J., Liu, C., Chen, Y.: Discover dependencies from data-a review. IEEE Trans. Knowl. Data Eng. **24**(2), 251–264 (2010)
20. Mhedhbi, A., Salihoglu, S.: Optimizing subgraph queries by combining binary and worst-case optimal joins. Proc. VLDB Endowment, vol. 12. no. 11 (2019)
21. Song, Q., Lin, P., Ma, H., Wu, Y.: Explaining missing data in graphs: a constraint-based approach. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE), pp. 1476–1487. IEEE (2021)
22. Song, S., Chen, L.: Differential dependencies: reasoning and discovery. ACM Trans. Database Syst. (TODS) **36**(3), 1–41 (2011)
23. Song, S., Gao, F., Huang, R., Wang, C.: Data dependencies extended for variety and veracity: a family tree. IEEE Trans. Knowl. Data Eng. **34**(10), 4717–4736 (2020)
24. Zhou, G., et al.: FASTAGEDS: fast approximate graph entity dependency discovery. arXiv preprint arXiv:2304.02323 (2023)