Explaining Missing Data in Graphs: A Constraint-based Approach

Qi Song
Amazon.com
qison@amazon.com

Peng Lin
Washington State University
peng.lin@wsu.edu

Hanchao Ma, Yinghui Wu
Case Western Reserve University
{hxm382, yxw1650}@case.edu

***Abstract*—This paper introduces a *constraint-based approach* to clarify missing values in graphs. Our method capitalizes on a set Σ of graph data constraints. An explanation is a sequence of operational enforcement of Σ towards the recovery of interested yet missing data (*e.g.,* attribute values, edges). We show that constraint-based approach helps us to understand not only why a value is missing, but also how to recover the missing value. We study Σ*-explanation* problem, which is to compute the optimal explanations with guarantees on the informativeness and conciseness. We show the problem is in Δ_2^P for established graph data constraints such as graph keys and graph association rules. We develop an efficient bidirectional algorithm to compute optimal explanations, without enforcing Σ on the entire graph. We also show our algorithm can be easily extended to support graph refinement within limited time, and to explain missing answers. Using real-world graphs, we experimentally verify the effectiveness and efficiency of our algorithms.**

***Index Terms*—Graphs, Data Constraints, Data Provenance**

I. Introduction

Real-world graphs are incomplete [1]: attribute values of entities (nodes) and relations (edges) are often missing. Enhancing graphs from multiple data sources with entity matching and link prediction has been widely studied [1]. A desirable yet missing functionality is to clarify *why* certain expected data is missing in graph data, whether such data can be restored, and how. Such need is evident in knowledge fusion [2], user-centric data quality [3], and query suggestion [4], [5].

Missing data in a graph G can be captured by *data constraints for graphs* [6]–[11]. These data constraints identify node pairs (v, v') in G via *graph pattern matching* between a graph pattern P and a graph G, and either enforce node equivalence or assert a missing edge between v and v'. Consider the following established data constraints for graphs.

(1) *Key constraints* [8], [12]–[14] have a general form of $P \rightarrow (u.\mathsf{id} = u'.\mathsf{id})$, and state that "*A pair of nodes v and v' in a graph G should refer to the a real-world entity* (id)*, if they both match a graph pattern P via graph pattern matching*".

For example, graph keys [8] state that "any pair of nodes in a graph that match to a designated pattern node in a graph pattern P should refer to the same entity", where P is defined by variants of subgraph isomorphism [8], [11], [13], [14].

(2) Constraints that infer missing edges $P \rightarrow r(u, u')$ [7], [15]–[17] state that "*There is an edge between a pair of nodes v and v' in G with label r, if v and v' matches P via graph pattern matching*". Notable examples include AMIE [7] (where P are Horn clauses), conditional patterns [17], and graph association rules [15], [16] where P is specified by variants of subgraph isomorphism.

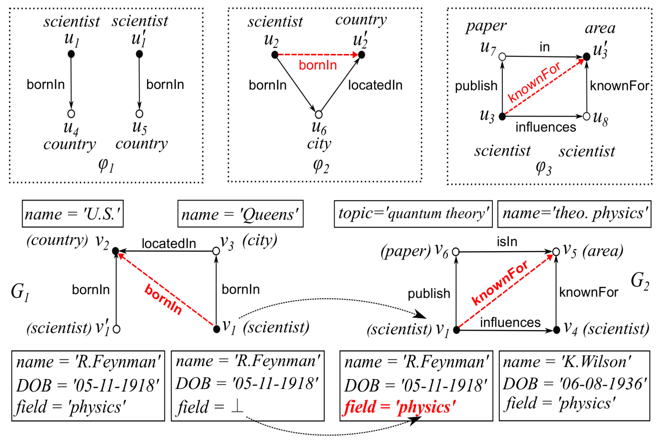


Fig. 1: Clarifying "*why the statement 'R.Feynman is known for physics' is missing?*" with graph data constraints.

Can these data constraints be used to explain "Why" specific data of interests is missing in a graph? Naturally, the enforcement of a data constraint infers missing data, hence provides useful provenance information for the latter [18], [19]. We illustrate this intuition with the following example.

Example 1: A fraction of a curated academic knowledge graph G is illustrated in Fig 1 (excluding dashed edges and colored attributes). Each node (*e.g.,* v_1) refers to an entity with a label (*e.g.,* a type scientist), and a tuple that maps attributes to their values (*e.g.,*name = "R.Feynman"). Each edge encodes a relation between two nodes, *e.g.,* "R.Feynman" was born in (bornIn) "U.S.". For the ease of discussion, we decompose G into components G_1 and G_2 with a shared node v_1.

There are two validated graph data constraints defined on G: a graph key φ_1 with graph pattern P_1, and a graph association rule φ_2 with pattern P_2 (Fig. 1), which states the following.

$\underline{\varphi_1}$ [8]: *"two **scientists** refer to the same person* (u_1.id = u_1'.id)*, if they have the same* name*, birth date, and was born in* (bornIn) *the same country"*(u_4.val = u_5.val)*.*

$\underline{\varphi_2}$ [15], [16]: *"a scientist* (u_2) *was born in a country* (u_2') *if the scientist was born in a city* (u_6) *which is located in* (locatedIn) *that country* (u_2')*."*

A missing edge $\mathsf{bornIn}(v_1, v_2)$ can be inferred by enforcing φ_2 in G_1. Indeed, v_1 and v_2 match the variables u_2 and u_2' in P_2 via subgraph isomorphism, but *violate* φ_2: there is no relation bornIn between them. This verifies the missing birth country 'US' of v_1 given his birth city 'Queens'. □

Nevertheless, simply enforcing all the constraints in a static "batch" may fail to explain why particular data is missing.

Example 2: Consider a third graph association rule φ_3 with pattern P_3 (Fig. 1) with the following semantic:

> φ_3: *"a scientist (u_3) is* ***known for*** *an area (u_3'), if the scientist published a paper (u_7) in the same area, and influenced a peer of his field (u_3.field = u_8.field) from that area."*

A user is expecting a statement "*the physicist R. Feynman* ***is known for*** *research in theo physics*" yet couldn't find it in the query result. A corresponding missing edge KnownFor(v_1 ('R. Feynman'), v_5('theo.physics')) could be inferred by enforcing φ_3. Nonetheless, φ_3 is already satisfied by G: it is unknown whether v_1 ('R. Feynman') has the same field as v_4 ('K. Wilson') due to the unknown 'field' (marked as '$\perp$'). Thus v_1 fails to match u_3 on "influence in physics". In fact, no violation or missing edges can be captured by φ_3 in G as is.

On the other hand, the missing edge KnownFor(v_1 ('R. Feynman'), v_5('theo.physics')) can be clarified by interleaving enforcement in a sequence that "evolves" G:

- Enforcing φ_2 explains an edge $\mathsf{bornIn}(v_1, v_2)$ is missing, and infers v_1's birth country given his birth city.
- Inserting the missing edge $\mathsf{bornIn}(v_1, v_2)$ enables the enforcement of φ_1, which clarifies why v_1 and v_1' *should* refer to the same scientist (given the same name, country and birthdate). It also infers a value 'physics' of v_1.field.
- The enriched node v_1 allows φ_3 to be enforced, as the 'influence' between physicists v_1 and v_4 are identified.

An explanation for "why" $\mathsf{knownFor}(v_1, v_5)$ is missing can be characterized by a sequence of "modifications" of G due to the enforcement of corresponding constraints $\{\varphi_2, \varphi_1, \varphi_3\}$. It also clarifies "how" to recover $\mathsf{knownFor}(v_1, v_5)$. This helps users to track and diagnose the refinement of graphs. □

Although desirable, clarifying *specific missing values* is nontrivial, due to the high computational complexity of enforcing the graph data constraints and the dynamic enforcement when G is large. These call for efficient solutions that are optimized to clarify specific missing values.

Contribution & Organization. We propose a *constraint-based approach* to clarify missing data in graphs, and study the problem complexity, measurements and feasible algorithms.

(1) Formal Characterization (Section III). Given a graph G and data constraints Σ of G, we introduce a notion of Σ-*explainability* for a missing value, determined by whether it can be derived via a sequence of enforcement of Σ ("an explanation"). As such, missing data is clarified with both operators ("*how*") and responsible constraints ("*Why*").

We show constraint-based approach ensures more *informative* and eventually a *unique* graph (Section III-B). This states the capacity of explanations: there is a unique, most informative graph up to graph homomorphic equivalence. These ensure the termination and correctness of our approach.

(2) Quality measures (Section III-C). We quantify the quality of constraint-based explanations with two measures: informativeness gain (amount of new values introduced to G), and conciseness. Based on the measures, We study Σ-*explanation* problem to compute concise and informative explanations for specified missing data of interests. We show that deciding Σ-explanability is complement to the validation problem of Σ, and is nontrivial (*e.g.,* Δ_2^P for graph keys). We also show that it is doable given an oracle that efficiently detects violations of Σ, and provide a relativized upper bound.

(3) Bidirectional Search (Section IV). We develop an algorithm BiExp for Σ-explanation. The algorithm uses bidirectional search to explore applicable actions (forward search) and virtual "enablers" (backward search) requested in order to derive missing data. We develop pruning techniques to reduce unnecessary exploration of enforcement, and showcase for graph keys and association rules.

(4) Extensions (Section V). We also extend BiExp for two applications. The first variant clarifies missing values in query results for subgraph queries. It incorporates query processing into a bidirectional search of BiExp and has the same worst-case time cost. The second, in support of online graph refinement, can progressively construct explanations for specific values of interests, and incurs a bounded cost on enforcing constraints (*e.g.,* time). It also guarantees a competitive ratio $6 \cdot ln|V| + ln(\frac{c_u}{c_l}) + 1$, where $[c_l, c_u]$ is the range of the costs for enforcing a single data constraint in Σ.

(5) Experimental study (Section VI). Using real-world graphs, we experimentally verify the efficiency and effectiveness of our constraint-based algorithms. (1) We find our algorithms effectively exploit data constraints to clarify the missing data of interests that may not be inferred by batch enforcement. (2) The bidirectional search and optimization techniques effectively reduce the cost. For example, BiExp improves its unoptimized counterparts by 2.1 times, and is 3.1 and 7.3 times faster than the counterparts using forward and backward search alone. (3) Our algorithms effectively clarify specific missing values with bounded enforcement cost and for subgraph queries, as verified by our case study.

Related work. We categorize the related work as follows.

Graph refinement. Link prediction has been well studied to infer missing links in graphs [1]. Notable methods include association analysis [7], [16], supervised link prediction [2], and embedding methods [1]. Entity resolution for graphs [20] aims to infer node equivalence by enforcing keys [8], [12], association rules [21] or online methods [22], [23].

In contrast, we exploit data constraints to clarify why specific values are missing, rather than completing the entire graph. Unlike prior work, we show that enforcing all the missing values is neither practical nor necessary to clarify specific missing values of interests, especially in large graphs.

Graph data repairing. Functional dependencies for graphs (GFDs) [9], [24], among other constraint-based methods [3] have been introduced to capture and remove inconsistencies in graphs. Chase has been extended for graphs [24], which captures a sequence of quotient graphs induced by equivalent classes of nodes upon the enforcement of GFDs.

Our work differs from these work. (1) Rather than removing data inconsistencies, we aim to clarify missing values of interests with graph data constraints. Our non-destructive approach enriches missing values to ensure more informative graph instances. (2) We also leverage Chase to characterize constraint-based explanations. Unlike the Chase process for repairing graph data [9], [24], our enforcement process preserves possible values for node attributes and edges (instead of choosing a preferred "correct" value) to avoid missing possible explanations for the missing values of interests.

Why provenance. Why-provenance has been studied for relation data [25], [26] and graphs [5], [27]. Data provenance (or Why-provenance) tracks the data that contributes to query answers [25]. Query provenance (or "Why-not" questions) identifies operators that lead to desired answers [5], [27]. Both approaches assume that the data of interests are already in the database, but cannot clarify missing data that is not in the data.

Constraints are used to explain certain answers in incomplete databases [28], [29], yet feasible algorithms are not discussed especially for graph data. Data provenance with functional dependencies [19], [30] apply static enforcement of data constraints *e.g.,* inclusion dependencies to derive non-operational explanations as possible values. These work do not consider operational enforcement of data constraints that involve graph patterns and their dynamic enforcement, which are needed to clarify specific missing values in graphs.

II. Graphs and Data Constraints

We start with a notion of graphs with missing values.

Graphs. A graph $G = (V, E, L, F, X)$ contains a finite set of nodes V, a set of edges $E \subseteq V \times V$, and a set of associated variables X to denote missing values.

(1) Each node $v \in V$ has a label $L(v)$ (*e.g.,* type) and an identifier id that refers to a real world entity (*e.g.,* URI). It carries a tuple $F(v)$ that is defined on a set of attributes $\mathcal{A}$. The value of an attribute $v.A$ in the tuple $F(v)$ is either a constant a from a finite set $\mathcal{C}$, or a distinct *node variable* x_{A_v} ='$\perp$' (marked null) from X that represents a missing value.

(2) Each edge $r(v, v') \in E$ carries an edge label $r \in \mathcal{C}$ (*e.g.,* a relation name). There is an *edge variable* $x^r_{vv'} \in X$ with a value'$\perp$' for each $r(v, v') \notin E$ ($x^r_{vv'}$ = 1 if $r(v, v')$ belongs to the missing fraction of G; $x^r_{vv'}$ = 0 otherwise).

We assume G has a complete set of nodes V. Each variable $x \in X$ from G denotes a missing *element*. An element is either an attribute-value pair $(v.A, a)$ that specifies a value a for attribute $v.A$, or an edge $r(v, v')$.

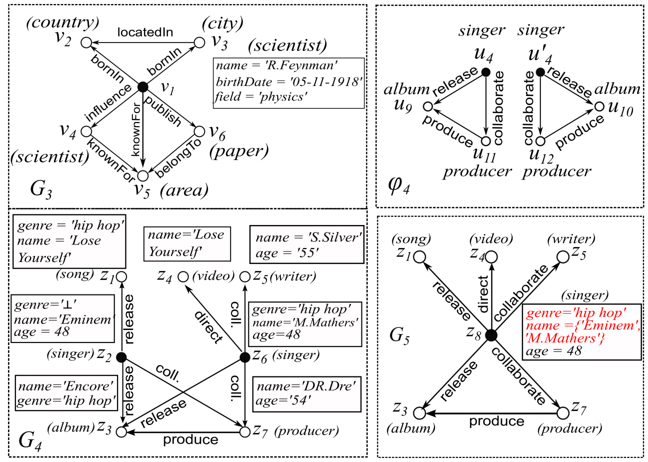


Fig. 2: Graphs and instances (a case of NE).

Valuation. Given a graph $G = (V, E, L, F, X)$, a *valuation* of G is a function defined on X such that there is at least one variable x='$\perp$ in X valuated to: (1) a set of constants $C_{A_v} \subseteq \mathcal{C}$, if x is a node variable, or (2) x=1, if x is an edge variable. An *instance* of G is a graph induced by a valuation. An instance can be a multigraph and may contain self-loop.

Example 3: An incomplete graph G is illustrated in Fig. 1 (excluding the edge $\mathsf{bornIn}(v_1, v_2)$ in G_1). Each node v_i has a distinct identifier id = i ($i \in [1,6]$). The set X marks three missing elements: $X = \{x_{field_{v_1}} =\perp, x^{bornIn}_{v_1 v_2} =\perp, x^{knownFor}_{v_1 v_5} =\perp\}$, which encode a missing attribute value, and two missing edges. A valuation that sets $x_{field_{v_1}}$ = 'physics' (resp. $x^{bornIn}_{v_1 v_2}$ = 1) induces an instance of G with new elements v_1.field = 'physics' (resp. $bornIn(v_1, v_2)$).

Similarly, Fig. 2 illustrates a graph G_4 of facts of entertainment, with a missing value genre at z_2. □

Remarks. We consider a weaker representation of incomplete graphs with marked nulls, similar to conditional tables [31]. A valuation can replace null with a set of possible values. This allows us to keep track of possible instances that may contribute to user-defined missing values (see Section III).

Graph data constraints. We next review data constraints for graphs. A data constraint φ defined on a graph G may

- enforce node equality (denoted as a NE constraint):

$$\mathcal{P} \rightarrow u_o.\mathsf{id} = u'_o.\mathsf{id}$$

- or assert a missing edge (denoted as an EG constraint):

$$\mathcal{P} \rightarrow \exists r(u_o, u'_o)$$

Here $\mathcal{P} = (P, \mathcal{L})$ (called a *pattern*) specifies a graph pattern $P(u_o, u'_o)$ and a set of literals $\mathcal{L}$ defined on attributes $\mathcal{A}$. (1) A *graph pattern* $P(u_o, u'_o)$ contains a set of nodes (variables) V_P and edges E_P. Each node $u \in V_P$ (resp. edge $e \in E_P$) has a label $L_P(u)$ (resp. $L_P(e)$). The nodes u_o and u'_o in V_P are two designated *entity nodes*. (2) Each literal $l \in \mathcal{L}$ is either a constant literal $u.A = a$ (a is a constant), or a variable literal $u.A = u'.A'$ ($u, u' \in V_P$, $A, A' \in \mathcal{A}$).

Matches. We characterize graph data constraints with *graph homomorphism* [32], which subsumes several common semantics [7]–[10], [13], [14]. Consider φ with pattern $\mathcal{P}$=$(P, \mathcal{L})$.

(1) A *matching* from graph pattern $P(u_o, u'_o)$ to G is a function h from V_P to V, such that (a) for each edge $(u, u') \in E_P$, $(h(u), h(u')) \in E$, $L_P(u) = L(h(u))$, $L_P(u') = L(h(u'))$, and (b) $L_P(h(u), h(u')) = L((u, u'))$. A pair of nodes (v, v') is a *match* of P induced by h, if $h(u_o) = v$ and $h(u'_o) = v'$.

(2) A pair of nodes (v, v') is a *match* of φ, if it is a match of $P(u_o, u'_o)$ induced by a matching h from $P(u_o, u'_o)$ to G, and h satisfies $\mathcal{L}$, *i.e.,* for each literal $l \in \mathcal{L}$, $h(u).A = a$ (resp. $h(u).A = h(u').A$) if $l = (u.A = a)$ (resp. $u.A = u'.A$).

Example 4: Fig. 1 illustrates the patterns of constraints φ_1, φ_2, φ_3 with conditions and matches on G specified as follows.

$\mathcal{P}$	Graph pattern P	Literals $\mathcal{L}$	Matches
$\mathcal{P}_1$	$P_1(u_1, u'_1)$	$\{u_1$.name $=u'_1$.name, u_1.DOB $=u'_1$.DOB $\}$	$\emptyset$
$\mathcal{P}_2$	$P_2(u_2, u'_2)$	$\emptyset$	(v_1, v_2)
$\mathcal{P}_3$	$P_3(u_3, u'_3)$	$\{u_3$.field $=u_8$.field $\}$	$\emptyset$
$\mathcal{P}_4$	$P_4(u_4, u'_4)$	$\{u_9$.name $= u_{10}$.name, u_9.genre$=u_{10}$.genre, u_{11}.name$=u_{12}$.name$\}$	(z_2, z_6)

Fig. 2 illustrates another pattern $\mathcal{P}_4 = (P_4(u_4, u'_4), \mathcal{L}_4)$, with the conditions and matches in G_4 also illustrated above. The matches $\mathcal{P}_4(G_4) = \{(z_2, z_6)\}$ is induced by a matching h, where $h(u_{11})$=$h(u_{12})$=z_7, and $h(u_9)$=$h(u_{10})$=z_3. $\square$

Semantics. A NE (resp. EG) φ with pattern $\mathcal{P}$=$(P(u_o, u'_o), \mathcal{L})$ states that "*for any match* (v, v') *of* (u_o, u'_o)*,* v *and* v' *are equivalent and should refer to a same entity*" (resp. "*has a missing edge* $r(v, v')$"). A match (v, v') of a NE (resp. EG) φ is a *violation* of φ if v.id $\neq v'$.id (resp. $r(v, v') \notin E$).

A graph G *satisfies* a NE (resp. EG) φ, denoted as $G \models \varphi$, if there exists no violation of φ in G. It satisfies a set of constraints Σ ($G \models \Sigma$), if for every $\varphi \in \Sigma$, $G \models \varphi$. In the rest of the paper, we consider Σ as a set of NEs and EGs.

Example 5: Fig. 1 illustrates three constraints: a NE φ_1: $\mathcal{P}_1 \rightarrow u_1.\mathsf{id} = u'_1.\mathsf{id}$, an EG φ_2: $\mathcal{P}_2 \rightarrow \exists\, \mathsf{bornIn}(u_2, u'_2)$, and an EG φ_3: $\mathcal{P}_3 \rightarrow \exists\, \mathsf{knownFor}(u_3, u'_3)$. As is, $G \models \{\varphi_1, \varphi_3\}$, and $G \not\models \varphi_2$, where (v_1, v_2) is a violation of φ_2.

We illustrate three more data constraints on G_4 in Fig. 2 and Fig. 3: NE φ_4: $\mathcal{P}_4 \rightarrow u_4.\mathsf{id} = u'_4.\mathsf{id}$, EG φ_5: $\mathcal{P}_5 \rightarrow \exists\, \mathsf{ost}(u_5, u'_5)$ and EG φ_6: $\mathcal{P}_6 \rightarrow \exists\, \mathsf{collaborate}(u_6, u'_6)$. $\square$

III. Explaining Missing Values with Constraints

A. Constraint Enforcement and Sequences

We next characterize the enforcement of data constraints following Chase [31], [32]. We start with a notion of operators.

Operators. An operator "modifies" graph G to remove a violation of a data constraint. We consider the following.

(1) A *merge* operator $\circ(v, v')$ for a NE φ replaces v and v' with a new node v'' (an "equivalent class") as follows. (a) $L(v'') = L(v) = L(v')$, v''.id $= \min(v.\mathsf{id}, v'.\mathsf{id})$. (b) Set tuple $F(v'')$ by setting $v''.A = v.A \cup v'.A$ for each attribute A in $F(v)$ or $F(v')$ ($v.A = \emptyset$, if A is not in $F(v)$ or $v.A = \perp$; similarly for $v'.A$). (c) Redirects the edges of v and v' to v''.

(2) An *insertion* operator $\oplus((v, v'), r)$ enforced by an EG φ over a violation (v, v') inserts a new edge $r(v, v')$.

Sequences. Given a graph G and constraints Σ, an *action* s is a triple $((v, v'), o(v, v'), \varphi)$, where (v, v') is a violation of φ in G, and $o(v, v')$ is an operator ($\circ$ or $\oplus$) that removes the violation (v, v') of φ from G. The *result* of s on G, denoted as G^s, refers to the graph obtained by applying $o(v, v')$ on G.

A *sequence* $\rho = \{s_1, \ldots, s_n\}$ is a nonempty sequence of actions from G with result G' (denoted as $G' = G^\rho$), where $G_i = G^{s_i}_{i-1}$ ($i \in [1, n]$, G_0=G, G_n=G'). Specifically, ρ can be a single "identity" action $\{\epsilon\}$, which yields G itself (G^ϵ=G).

Example 6: Given graph data constraints $\Sigma = \{\varphi_1, \varphi_2, \varphi_3\}$, and a sequence of actions $\rho = \{s_1, s_2, s_3\}$ where $s_1 = ((v_1, v_2), \oplus((v_1, v_2), \mathsf{bornIn}), \varphi_2)$, s_2=$((v_1, v'_1), \circ(v_1, v'_1), \varphi_1)$, and $s_3 = ((v_1, v_5), \oplus((v_1, v_5), \mathsf{knownFor}), \varphi_3)$, we have $G_3 = G^\rho$ (as illustrated in Fig. 2). Note that $G_3 \models \Sigma$.

For $\Sigma = \{\varphi_4, \varphi_5, \varphi_6\}$ (Fig. 2, 3), $G_5 = G_4^{s_1}$ (resp. $G_6 = G_5^{s_2}$) with $s_1 = ((z_2, z_6), \circ(z_2, z_6), \varphi_4)$ (resp. $s_2 = ((z_1, z_9), \oplus((z_1, z_9), ost), \varphi_5)$). The node z_8 is obtained from $\circ(z_2, z_6)$, which takes their common attribute age ='48', resolves possible values for name as a set $C_{name_{z_8}}$ = {'Eminem', 'M.Mathers'}, and retains 'genre' from z_6. One can verify that $G_5 \models \varphi_4$, $G_5 \not\models \varphi_5$, and $G_6 \models \{\varphi_4, \varphi_5\}$. $\square$

The result below verifies that sequences preserve the information of G. In other words, they are "non-destructive".

Lemma 1: *Given graph G and constraints* Σ*, for any sequence* ρ*, there is a graph homomorphism* h^ρ *from* G *to* G^ρ*, and for any element* g=$(v.A, a)$ *(resp.* $r(v, v')$*) in* G*,* $h^\rho(g) = (h^\rho(v).A, a)$ *(resp.* $r(h^\rho(v), h^\rho(v'))$*) is an element in* G^ρ*.* $\square$

Proof sketch: Let $\rho = \{s_1, \ldots, s_n\}$. We construct h^ρ as a composition function $h^{s_1} \ldots h^{s_n}$. Each h^{s_i} $(i \in [1, n], G_0 = G, G_n = G^\rho)$ is a function from the nodes of G_i to those in G_{i+1}, where $h^{s_i}(v) = h^{s_i}(v') = v''$, if $s_i = \circ(v, v')$ and yields v'' in G_{i+1}, or $h^{s_i}(v) = v$ otherwise. We can verify that each h^{s_i} $(i \in [1, n])$ (resp. h^ρ) is a graph homomorphism from G_i to G_{i+1} (resp. G to G^ρ) that preserves the elements of G. $\square$

Given Lemma 1, we say an element $g = (v.A, a)$ (resp. $r(v, v')$) *occurs* in the instance G^ρ of G, simply denoted as $g \in G^\rho$, if $h^\rho(v).A$=a (resp. $r(h^\rho(v), h^\rho(v')$ is an edge) in G^ρ.

Remarks. Unlike Chase for data repairing (cf. [32]), the merge operator resolves attribute values with union. It can also be implemented as resolution functions [33] that consistently choose a best value from a set based on trust [34] or reliability [2]. Our techniques readily extend to these specifications.

Properties. We justify the sequences by showing that they ensure the *informativeness* of the results towards a *unique* result. We start with a notion of *informativeness ordering*.

Informativeness. Given G and Σ, we denote the set of all the results of a sequence from G (including G) as G_Σ. Given two instances G' and G'' in $\mathcal{G}_\Sigma$, we say G' is *not more informative* then G'', denoted as $G' \preceq G''$, if $G''_\Sigma \subseteq G'_\Sigma$.

Intuitively, G'' is "more informative" if it has *less* results G''_Σ. That is, the more possible instances of G' that can be derived by a sequence that enforces the data constraints from Σ, the less informative G' is. We have the following result.

Theorem 2: *Given a graph G and constraints Σ, for any instance $G' \in \mathcal{G}_\Sigma$ and any sequence ρ from G', $G' \preceq G'^\rho$.* □

We prove Theorem 2 by showing the following result.

Lemma 3: *The relation $\preceq$ is a partial order over $\mathcal{G}_\Sigma$.* □

Intuitively, a sequence results valuated instances that become more "certain" on missing values. We provide the detailed proof of Lemma 3 and Theorem 2 in [35].

Uniqueness. Following Chase and its Church-Rosser property [31], [32], we next present a *uniqueness* guarantee, which states that any sequence will terminate at a "unique" result. We say a sequence ρ *terminates* at an instance $G' {\in} \mathcal{G}_\Sigma$ if $G'^s{=}G'$ for any possible action s. We show the following result.

Theorem 4: *Given G and Σ, (1) any sequence from G terminates with at most $|V| + |V|^2$ actions, and (2) for any two sequences ρ and ρ', the results G^ρ and $G^{\rho'}$ are homomorphically equivalent,* i.e., *there exists a graph homomorphism from G^ρ to $G^{\rho'}$, and vice versa.* □

Proof sketch: Theorem 4 (1) can be verified by observing that any action reduces either one node or insert one s edge from a finite node set V of G. We show Theorem 4(2) by contradiction. Assume G^ρ to $G^{\rho'}$ are not homomorphically equivalent, then there exists at least one violation in *w.l.o.g.* G^ρ and an action s such that $G^{\rho s} \neq G^\rho$, which contradicts that ρ terminates at G^ρ. We present the details in [35]. □

B. Explanations for Missing Values

We next characterize the explanations for specific element.

Σ-explainable. Given graph G and constraints Σ, a missing element g not in G is *Σ-explainable*, if there is a sequence ρ such that $g \in G^\rho$. We say ρ is an *explanation* of g.

Example 7: Continue with Example 6. The two sequences ρ_1 = $\{s_1, s_2, s_3\}$ over G (Fig. 1) and ρ_2 = $\{s_1, s_2\}$ over G_4 in Example 6 are explanations for missing elements $\mathsf{KnownFor}$ (v_1, v_5) in G and $\mathsf{ost}(z_1, z_9)$ in G_4, respectively. Note that ρ_1 terminates at G_3 given $\Sigma{=}\{\varphi_1, \varphi_2, \varphi_3\}$. □

Given Theorem 2 and Theorem 4, an explanation ρ of a missing element g is well-defined with guarantees on informative instances, ensures the occurrence of g, and eventually terminates at unique result up to graph homomorphism.

C. Measures for Explanations

There can be multiple explanations for a missing element. We next introduce measurements for "good" explanations.

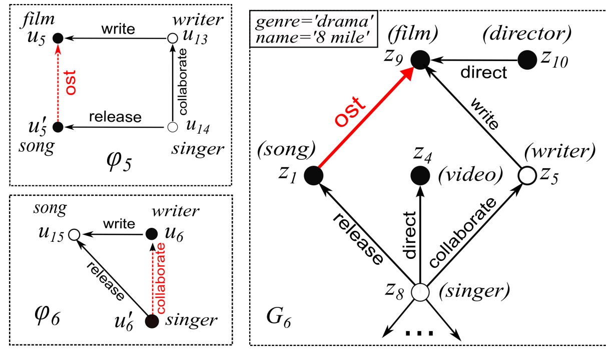


Fig. 3: Instances by enforcing NEs and EGs

Informativeness. Given an explanation ρ of element g that starts from G, the *cumulative informativeness gain*, denoted as $\mathsf{cg}(\rho, G)$, is defined as

$$\mathsf{cg}(\rho, G) = \sum_{s \in \rho} \mathsf{supp}(s, G) \cdot \mathsf{cg}(s, G)$$

where $\mathsf{supp}(s, G)$ for s that enforces φ (with $\mathcal{P}$ and graph pattern P) is computed as the fraction of the matches that satisfy φ to the total matches $P(G)$. $\mathsf{cg}(s, G)$ is the informativeness gain of an action $s{\in}\rho$, and is separately defined as follows:

$$\mathsf{cg}(s, G) = \begin{cases} \frac{1}{|r(L(v), L(v'))|}, & \text{if } s = \oplus((v, v'), r) \\ \mathsf{diff}(v, v'), & \text{if } s = \circ(v, v') \end{cases}$$

(1) For edge insertion $\oplus$, $r(L(v), L(v')) = \{(v_1, v_2) | (v_1, v_2) \in E, h(v_1) = h(v), h(v_2) = h(v')\}$. The measure $\mathsf{cg}(s, G)$ ($\mathsf{cg}(s, G) \in (0, 1]$) quantifies the additional gain under partial closed world assumption [2], [36]: the more edges similar to $r(v, v')$ are known, the less the gain is. $\mathsf{cg}(s, G) = 1$ if $r(L(v), L(v'))$=$\emptyset$ (by inserting "a first of its kind") [36].
(2) For node merge $\circ$, $\mathsf{cg}(s, G)$ favors to merge two equivalent nodes with tuples $F(v)$ and $F(v')$ that are more different. This can be quantified as the symmetric difference $\mathsf{diff}(v, v')$ of their attributes [37], which indicates more variables can be valuated, or more possible values can be identified.

Conciseness. We follow the principle of minimality for explanations. A *minimal* explanation for an element g is a sequence ρ such that $g \in G^\rho$, and $g \notin G^{\rho'}$ for any subsequence ρ' of ρ. We prefer minimal explanations within a bounded length. Note that $|\rho| \leq |V|^2 + |V|$ for any sequence ρ (Theorem 4).

Example 8: Assume the nodes z_2 and z_6 in Example 6 have another common attribute $\mathsf{occupation}$ (not shown) then $\mathsf{cg}(s_1, G_4) = \frac{2}{4}$. We can also verify that $\mathsf{cg}(s_2, G_4) = 1$. Given that $\mathsf{supp}(s_1, G_4) = 1$ and $\mathsf{supp}(s_2, G_4) = 1$, we have $\mathsf{cg}(\rho_2, G_4) = 1.5$. □

Properties of measures. We show the following properties of our measures. (1) $\mathsf{cg}(\rho) = 0$ if $\rho = \emptyset$ (Consistency:"no gain if not an explanation"). (2) $\mathsf{cg}(\rho) \geq \mathsf{cg}(\rho')$ if $G^\rho \preceq G^{\rho'}$ (Informativeness: "more informative result, more gain"). (3) $\mathsf{cg}(\rho\{s\}) \geq \mathsf{cg}(\rho\{s'\})$, if $s.\varphi \models s'.\varphi'$ (Generality: "prefer ρ that enforces constraints which logically imply those enforced by others"). These properties justify our measures.

We present the detailed proofs of these properties in [35].

Problem Statement. Based on these measures, we study the following optimization problem, denoted as Σ-*explanation*.

- **Input**: Graph G, a missing element $g \notin G$ (can be a *wildcard* '_'), graph data constraints Σ, and a bound b;
- **Output**: a minimal explanation ρ for g s.t.

$$\rho = \arg\max_{|\rho' \leq b|} \mathsf{cg}(\rho', G)$$

The targeted element g can be a wildcard ("don't care"). This is useful when no preference or ground truth is specified, and one wants to find what can be inferred (see Section V-A).

Complexity. We relativise the hardness of Σ-explanation with the *validation* problem, which is to decide whether $G \models \Sigma$ (Section II). We make a case for Σ as a set of graph keys and graph association rules, and show the following result.

Theorem 5: *Σ-explanation is in Δ_2^P for Σ defined as a set of graph keys and graph association rules.* □

Proof sketch: We prove Theorem 5 as follows.

(1) Consider the following special case of Σ-explanation: given G, Σ and g='_', it is to decide whether there exists an explanation ρ with size 1 (a single action). We show that this problem is a complement of the validation of Σ over G: there exists an explanation if and only if $G \not\models \Sigma$.

(2) We next show that Σ-explanation is equivalent to deciding whether there is a path with length bounded by b from G to an instance with g, in the lattice $(G_\Sigma, \preceq)$ induced by the partial order $\preceq$ (Lemma 3). We provide a PTIME algorithm to solve Σ-explanation that invokes an oracle for the validation of Σ.

As the validation problem of graph keys and association rules is coNP-complete [38], Theorem 5 follows. □

We present the detailed proof in [35].

IV. Computing Optimal Explanations

We next introduce algorithms for Σ-explanation. We assume an "oracle" (denoted as DetVio) is available to detect the violations of Σ in G. It can be efficiently implemented by *e.g.,* incremental and parallel pattern matching [38].

A naive solution. Following Theorem 5, one may simply invokes DetVio to construct a lattice $\mathcal{E} = (G_\Sigma, \preceq)$, and computes the explanation as a shortest path (by reversing gains to distances). Nevertheless, this can be expensive due to the excessive number of instances and violation detection.

We next develop an efficient algorithm, denoted as BiExp, to compute explanations without constructing $\mathcal{E}$.

A. Bi-directional Exploration

The algorithm BiExp initializes and explores a partially observed lattice $\mathcal{E} = (G_\Sigma, \preceq)$ (not known a-priori) with a *bi-directional* exploration strategy as follows.

- *Forward search*: starts from G and invokes DetVio to explore admissible actions (called *forward frontier*).
- *Backward search* simultaneously starts with a "virtual" instance G_g that contains g, and "reverse engineers" Σ enforcement to explore a set of *enabling* actions (maintained in its *backward frontier*) that may result G_g.

Algorithm BiExp

Input: Graph G, element g, data constraints Σ, size bound b.
Output: a minimal explanation ρ for g.

initializes queue $\mathcal{Q}_f$:=$\{s_r\}$; $\mathcal{Q}_b$:=$\{s_g\}$; tree T_f=$\{s_r\}$; T_b=$\{s_g\}$
while $\mathcal{Q}_f \neq \emptyset$ and $\mathcal{Q}_b \neq \emptyset$ and $h(T_f) + h(T_b) \leq b$ **do**
  if $\mathcal{Q}_f \neq \emptyset$ **then** s_f := $\mathcal{Q}_f$.Dequeue(); /**Forward search**/
    if s_f=s_g **or** $s_f \in \mathcal{Q}_b$ **then**
      return ρ := ConstrExp(T_f, T_b, s_f);
    for each $s'_f \in$ Forward(T_f, s_f) **do**
      $\mathcal{Q}_f$.Enqueue(s'_f);
  if $\mathcal{Q}_b \neq \emptyset$ **then** s_b := $\mathcal{Q}_b$.Dequeue(); /*Backward search*/
    if s_b=s_r **or** $s_b \in \mathcal{Q}_f$ **then**
      return ρ:=ConstrExp(T_f, T_b, s_b);
    for each $s'_b \in$ Backward(T_b, s_b) **do**
      $\mathcal{Q}_b$.Enqueue(s'_b);
return $\emptyset$;

Fig. 4: Algorithm BiExp

The algorithm reconstructs explanations upon frontier intersection over explored fraction of $\mathcal{E}$.

The bidirectional search reduce unnecessary exploration by refining forward frontier given current backward frontier, and vice versa. As the unknown G_Σ is monotonically decreasing (ensured by Theorem 2), BiExp ensures better explanations by making the graph G more "certain" in the exploration.

We start with the auxiliary structures of BiExp.

Auxiliary structures. Algorithm BiExp maintains the following. (1) A set of Boolean variables (called *element variable*), where a variable $x(g, G)$ is 1 if $g \in G$, and 0 otherwise. The values can be easily maintained by tracking the valuation of G (the values of the variables X). (2) A *forward tree* T_f and a *backward tree* T_b, specified as follows.

Forward tree. The forward tree T_f has a root s_r. Each node in T_f is an action $s = ((v, v'), o, \varphi)$ ($s_r = (\emptyset, \epsilon, \emptyset)$). There is an instance G_s associated with node s, where $G_s = G^\rho$, and ρ is the sequence (path) from s_r to s in T_f. The *forward frontier* o_f of T_f is a set of leaves (initialized as s_r) that are selected to be applied to generate new instances.

Backward tree. The backward tree T_b maintains a set of *enabling* actions that are required (depending on forward frontier). T_b is initialized as a root node s_g as follows.

- If $g = r(v, v')$, $s_g = ((v, v'), \oplus((v, v'), r), \varphi))$, where EG $\varphi \in \Sigma$ asserts a missing edge $r(u_o, u'_o)$ such that v and u_o (resp. v' and u'_o) have the same label.
- If $g = (v.A, a)$, $s_g = (\emptyset, \epsilon, \emptyset))$.

Sequences are constructed in a backward direction in T_b. The *backward frontier* o_b of T_b contains a set of (backward) leaves, which are selected to explore preceding actions that *may* lead to g. For each node $s = ((v, v'), o(v, v'), \varphi)$ in o_b, the following are dynamically maintained:

- "virtual" instance G_s associated to s,
- a set of *enablers*, where each enabler s' of s is an action that makes (v, v') a violation of φ in $G_{s'}$, and
- a Boolean condition s.con on the element variables.

Procedure Backward
Input: backward tree T_b, node s.
Output: backward frontier o_b.
1. initializes enabler set $S_e := \{s\}$;
2. induce graph $G(s) :=$ InduceSG(G_s, d);
3. **while** $s.con \neq$ False **and** $S_e \neq \emptyset$ **do**
   /* generate enablers for action s */
4. $\quad S_e :=$ GenAction$(G(s), s)$;
5. $\quad$ **for each** $s_e \in S_e$ and $\varphi \in \Sigma$**do**
6. $\quad\quad$ **if** $(s_e.v, s_e.v')$ may violate φ **then**
7. $\quad\quad\quad S := S \cup \{((s_e.v, s_e.v'), s_e.o, \varphi)\}$;
8. $\quad o_b = o_b \cup$ PruneBwd(S); update T_b and s.con;
9. **return** o_b;

Fig. 5: Procedure Backward

The element variables are shared by T_f and T_b. The condition s.con, incrementally evaluated by the shared element variables, tracks whether the required elements occurs to enable the application of s towards interested element g. (s.con = true means s becomes applicable; see "Optimization").

Example 9: Consider the computation of the explanation for element $g = x^{ost}_{z_1,z_9}$ in Figure 3. Following Example 6, s_1 is an enabler of s_2 as (z_1, z_9) becomes a a violation of φ_5, encoded as an edge (s_1, s_2) in backward search T_b. The enabling condition for s_2 is initialized as $s_2.con = x(collab(z_2, z_5), G_4)$. This reduces the problem to computing explanations for collab(z_2, z_5). □

Algorithm. Our algorithm BiExp (as illustrated in Fig. 4) initializes the forward tree T_f (resp. backward tree T_b) as a single root node s_r (resp. s_g) (line 1). It also maintains two queues, $\mathcal{Q}_f$ and $\mathcal{Q}_b$ to store the forward frontier o_f and backward frontier o_b, respectively (line 1).

BiExp next performs a bidirectional Breadth-First search (BFS) (up to size bound b; line 2), by invoking a procedure Forward (resp. Backward) to grow T_f (resp. T_b backwardly) and refine o_f (resp. o_b), until a common action $s \in o_f \cap o_b$ is identified (lines 4-5, 9-10). This indicates the action s is both verified by forward search and meanwhile "needed" as an enabler in backward search to include g, resulting a sequence ρ from s_r to s_g passing s with a result that includes g.

Upon the frontier intersection, BiExp invokes a procedure ConstrExp (not shown) to construct the optimal explanation (lines 5, 10). There may be multiple actions in $o_f \cap o_b$. For each $s \in o_f \cap o_b$, an explanation is constructed as the sequence from s_r to s_g passing s. The best one that maximizes accumulated gain cg is then returned ($\emptyset$ if $o_f \cap o_b$=$\emptyset$).

Procedure Forward. Given the current forward frontier o_f, procedure Forward identifies a set of applicable actions. For each action node $s = ((v, v'), o, \varphi) \in o_f$ with instance G_s, Forward invokes procedure DetVio to compute a set of violations Vio$(G_s, \Sigma) = \bigcup_{\varphi \in \Sigma}$ Vio(G_s, φ), where Vio(G_s, φ) refers to the violations of φ. It also invokes a procedure PruneFwd, to refine o_f according to desired actions in o_b (see "Optimization"). The new actions are added as children of s in T_f.

Procedure Backward. Similarly as Forward but more involved, Backward (illustrated in Fig. 5) determines a set of enablers

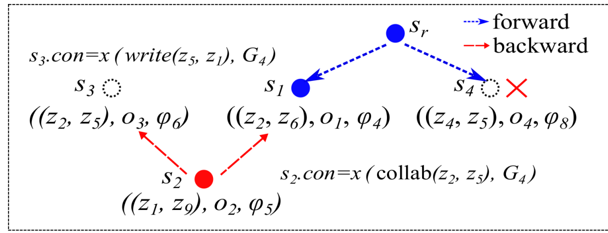


Fig. 6: Bidirectional construction for element $g = x^{ost}_{z_1,z_9}$

for the current o_b, and refines o_b with enablers.

Computing enablers. For each action $s = ((v, v'), o, \varphi) \in o_b$, Backward computes a set of enablers as follows.

(1) If (v, v') is not already a match of φ with pattern $\mathcal{P}$=$(P, \mathcal{L})$) in G, Backward first computes a set of *enabling elements* S_e for s. A missing element g' is an enabling element for s, if adding it to G_s make (v, v') a match of φ in G_s. Specifically, Backward performs the following.

- Induce a subgraph $G(s)$ of G_s (procedure InduceSG, line 2) with the d-hop neighbors $N_d(s)$ of v and v' ($N_d(s)$ = $N_d(v) \cup N_d(v')$), where d is the diameter of the graph pattern $P(u_o, u'_o)$ of $\mathcal{P}$. Given the data locality of graph homomorphism, it suffices to consider $G(s)$.
- Construct a set of missing edges $r(v_1, v_2)$ and missing attribute values $(v_1.A, a)$ from $G(s)$, where $r(v_1, v_2)$ and $v_1.A$ are needed to form a graph homomorphism from $P(u_o, u'_o)$ to $G(s)$ that also satisfy $\mathcal{L}$.

(2) For each enabling element g' and a corresponding $\varphi \in \Sigma$, an enabler is constructed if enforcing φ may introduce g' (Procedure GenAction, lines 4-7). The enablers are further refined by procedure PruneBwd (see "Optimization") and are added to backward frontier o_b (line 8). .

Example 10: Continue with Example 9, Backward first induces a subgraph (with $d = 2$) from z_1 and z_9. It then generates two enablers (illustrated in Figure 6): s_1 which merges node z_2 and z_6, and s_3 which adds an edge between z_2 and z_5. BiExp continues from forward search and finds $s_1 \in o_f \cap o_b$. This generates a best explanation $\rho = \{s_1, s_2\}$ for the missing element $g = x^{ost}_{z_1,z_9}$ (Figure 3). □

B. Optimization

BiExp uses two strategies below to further reduce the cost.

Early termination with conditions. BiExp maintains a condition s.con = $\bigvee \bigwedge(x)$ for each $s \in o_b$. Each clause $\bigwedge(x)$ involves element variables for a set of elements in a same matching h to to make $s.(v, v')$ a match of $s.\varphi$ in G_s.

BiExp incrementally update s.con with partial evaluation [39]. It dynamically induces residual conditions upon the assignment of an element variable x (true or false), without waiting for all the element variables to be evaluated. This enables early pruning of branches (when all disjunct clause becomes false, *i.e.,* s.con=false), or early detection of frontier intersection (when a disjunct clause is true, *i.e.,* s.con=true).

Bidirectional pruning. The forward and backward search interactively refines each other by "pre-matching" the enablers and possible actions without verification.

Closures. We introduce a preorder $\vdash$ on Σ. Given Σ and two data constraints $\varphi, \varphi' \in \Sigma$, φ triggers φ', denoted as $\varphi \vdash \varphi'$, if one can find small model (G, G_1, G_2) such that $G_1, G_2 \in G_\Sigma$, $G_1 = G^s$, $G_2 = G^{s,s'}$, and s enforces φ, s' enforces φ'.

BiExp estimates a *forward closure* $\varphi^{+\downarrow}$ and a *backward closure* $\varphi^{+\uparrow}$ of a data constraint $\varphi \in \Sigma$ as follows. We observe $\varphi \not\vdash \varphi'$ if (a) a NE φ enforces equivalence on nodes with label $L(u_o) = L(u'_o)$, and no nodes in the pattern of φ' has the same label; (b) an EG φ enforces edge $r(v, v')$ and no edge in the pattern of φ' has matching node and edge label. This can be checked in PTIME. Denote all such data constraints φ' (resp. φ) for φ (resp. φ') as $\varphi{\downarrow}^-$ (resp. $\varphi'{\uparrow}^-$).

(1) $\varphi^{+\downarrow}$ is define as: (a) $\varphi \in \varphi^{+\downarrow}$, and (b) $\varphi' \in \varphi^{+\downarrow}$ if there exists a data constraint $\varphi'' \in \varphi^{+\downarrow}$ such that $\varphi' \notin \varphi''{\downarrow}^-$.

(2) $\varphi^{+\uparrow}$ is defined as: (a) $\varphi \in \varphi^{+\uparrow}$, and (b) $\varphi' \in \varphi^{+\uparrow}$ if there exists a data constraint $\varphi'' \in \varphi^{+\uparrow}$ such that $\varphi' \notin \varphi''{\uparrow}^-$.

The forward closures of a set $\Sigma^{+\downarrow}$ is defined as $\bigcup_{\varphi \in \Sigma} \varphi^{+\downarrow}$. $\Sigma^{+\uparrow}$ is defined similarly. Denote the constraints involved in the enablers and actions in o_f and o_b as Σ_o and Σ_b, respectively. PruneBwd and PruneFwd refine o_f and o_b as follows.

Lemma 6: *For any action s that enforces $\varphi \in \Sigma$, and any explanation ρ of g such that $G^\rho \in G_\Sigma$, $s \notin \rho$ if (1) $s \in o_b$, and $\varphi^{\uparrow+} \cap \Sigma_o^{+\downarrow} = \emptyset$; or (2) $s \in o_f$, and $\varphi^{\downarrow+} \cap \Sigma_b^{+\uparrow} = \emptyset$.* $\square$

Lemma 6 allows the forward and backward search to iteratively refine each other (see detailed proof in [35]).

Example 11: Following Example 10, s_3 requires a single enabler to introduce $\mathsf{collab}(z_2, z_5)$. As $\{\varphi_6\}^{\uparrow+} = \emptyset$ (no constraint in Σ can trigger φ_6), PruneBwd removes s_3 from o_b. $\square$

These optimization is quite effective. For a real graph with 4.5 million edges, the pruning reduces the time cost by 52% without losing the quality of explanations (see Section VI).

Analysis of BiExp. We observe the following.

Time complexity. BiExp simulates a bidirectional breadth first search of the lattice $(G_\Sigma, \preceq)$, with at most $\min(|V|^2|\Sigma|, b)$ forward or backward spawning. Each spawning invokes DetVio with a time T. Each spawning triggers at most $|V|^2$ violations, thus generate up to $|V|^2|\Sigma|$ actions. The bidirectional search is thus in $O(T \cdot (|V|^2|\Sigma|)^{\frac{b}{2}})$, given that $b \ll |V|$.

Correctness. BiExp correctly terminates with a minimal explanation ρ ($\emptyset$ if g is not Σ-explainable). It computes ρ with the maximized information gain as a shortest path from s_r to s_g in the lattice $(G_\Sigma, \preceq)$ (Theorem 5). .

V. Extensions

In this section, we extend BiExp to support cost-effective graph refinement, and to clarify missing answers.

A. Budgeted Graph Refinement

Refining graphs by inferring new elements with Σ can still be expensive when no targeted element is provided (g='_') and for large G (*e.g.,* social networks). It is often desirable to obtain some result first, and "gradually" improve the results towards overall solution [23], [40]. Such need can be addressed by solving the following budgeted Σ-explanation problem.

Algorithm ApxExp
Input: an incomplete graph G, data constraints Σ, resource bound B.
Output: a sequence ρ.

1. Queue $\mathcal{Q}$:=$\emptyset$; Sequence ρ:=$\emptyset$; Instance $G' = G$;
2. initialize action s_r:= $(\emptyset, \epsilon, \emptyset)$;
3. List $\mathcal{S}$:= NextBatch (s_r, G', Σ); $\mathcal{Q}$.Enqueue($\mathcal{S}$);
4. **while** $\mathcal{Q} \neq \emptyset$ and $c(\rho) \leq B$ **do**
5.   action s := $\mathcal{Q}$.Dequeue();
6.   **if** $\frac{\mathsf{cg}(s,G')}{c(s)} \geq \Psi(\frac{c(\rho)}{B})$ **then**
7. $\rho := \rho \cup \{s\}$; $G' := G'^s$;
8. $\mathcal{S}$:= NextBatch(s, G', Σ); $\mathcal{Q}$.Enqueue($\mathcal{S}$);
9. **return** ρ;

Fig. 7: Algorithm ApxExp

- **Input**: Graph G, a wildcard element, data constraints Σ, and a cost bound B;
- **Output**: a sequence $\rho^* = \arg\max_{|c(\rho) \leq B|} \mathsf{cg}(\rho, G)$.

where the cost $c(\rho) = \sum_{s \in \rho} c(s)$. Moreover, each action s has a cost $c(s)$ that is not known a priori (*e.g.,* time, memory cost).

The hardness of Σ-explanation remains intact for the above problem. As the costs of an action can only be determined upon observed, we aim to incrementally maintain a sequence ρ^* that has the largest $\mathsf{cg}(\rho)$ among the sequences over all the currently observed actions. We show the following result.

Theorem 7: *There is a one-pass algorithm for budgeted Σ-explanation with wildcard, with a competitive ratio $6 \cdot ln|V| + ln(\frac{c_u}{c_l}) + 1$ at any time, where c_u and c_l is the maximum and minimum cost of an action, respectively.* $\square$

That is, the algorithm maintains a sequence ρ at any time, such that $\mathsf{cg}(\rho^*) \leq (6 \cdot ln(|V|) + ln(\frac{c_u}{c_l}) + 1)\ \mathsf{cg}(\rho)$. As a proof of Theorem 7, we outline such an algorithm.

Algorithm. The one-pass algorithm ApxExp is shown in Figure 7. It uses a queue $\mathcal{Q}$ to store candidate actions, and maintains an instance to be updated (initialized as G). Its main driver progressively extends a sequence ρ as follows. It first invokes a procedure NextBatch to populate a batch of promising actions $\mathcal{S}$ (line 3). For each action $s \in \mathcal{S}$, it verifies whether its gain-to-cost ratio $\frac{\mathsf{cg}(s,G')}{c(s)}$ is no less than threshold $\Psi(\frac{c(\rho)}{B})$, and extends ρ with s if so (lines 6-7). The threshold function $\Psi(\cdot)$ for input z is defined as

$$\begin{cases} \frac{1}{|V|^4 \cdot c_u} & \text{if } z \leq \frac{1}{1+6\cdot ln(|V|)+ln(\frac{c_u}{c_l})} \\ (\frac{|V|^6 \cdot c_u \cdot e}{c_l})^z (\frac{1}{e|V|^4 \cdot c_u}) & \text{otherwise} \end{cases}$$

where $\Psi(0)$ provides a lower bound of the gain for any single action. It then invokes NextBatch to spawn a new batch of actions given selected s to be verified (line 8).

The process repeats until resource bound B is reached, or a relative complete instance is reached (line 4). ApxExp then returns ρ. It also provides a refined instance G^ρ of G and explanations of elements of interests upon request.

Procedure NextBatch**.** The procedure NextBatch performs forward search as Forward in BiExp, and refines the actions by pruning those that can no longer introduce gain beyond the threshold $\Psi(\cdot)$. It also sorts $\mathcal{S}$ to early terminate its sequential processing in ApxExp against a non-decreasing threshold. We present the details of NextBatch in [35].

Analysis. We present the following analysis for ApxExp.

Competitive ratio. Algorithm ApxExp approximately maintains the optimal sequence by solving a budgeted secretary problem [41], [42], which is to choose a set of items from a sequence to maximize the total value under a fixed budget. Let the lower and upper bound of the gain-to-cost ratio of an action to be L and U respectively. The selection of actions against gain-to-cost threshold ensures a competitive ratio [41], [42] as $ln(U/L) + 1 = 6 \cdot ln(|V|) + ln(\frac{c_u}{c_l}) + 1$.

Theorem 7 follows from the above analysis. When shorter inferences are preferred ($c(\rho) = |\rho|$), we show that ApxExp achieves better approximation of $6 \cdot ln(|V|) + 1$ (see [35]).

Time cost. NextBatch takes $O(T|\Sigma||V|^2)$ time (T is the cost of DetVio) to expand an action in forward search. As at most $\frac{B}{c_l}$ actions are inspected, the total cost is in $O(\frac{B}{c_l} \cdot |\Sigma||V|^2 T)$.

B. Explaining Missing Answers

Our second extension has a practical premise that it is useful to provide users with operators to change G such that a query Q returns desired yet missing answer (cf. [26]).

Queries. A query Q maps an instance G to an answer $Q(G)$. The answer $Q(G)$ is a set of elements (attribute value pairs or edges) that only contains constant values. An element g is a missing answer if $g \notin Q(G)$. We make case for subgraph queries Q, which returns a set of nodes induced by a homomorphism from Q to G, *e.g.,* SPARQL.

We study the following variant of Σ-explanation.

- **Input**: instance G, a query Q and query answer $Q(G)$, an element $g \notin Q(G)$; constraints Σ, and a size bound b;
- **Output**: a minimal sequence ρ s.t. $g \in Q(G^\rho)$ and $|\rho| \leq b$.

We next outline a variant of BiExp, denoted as BiExpQ to compute explanations for a missing answer $g \notin Q(G)$.

Algorithm. The algorithm BiExpQ uses the bidirectional search as in BiExp. The differences are as follows.

(1) Besides the instance graph G_s at each node s, BiExpQ also tracks the query answer $Q(G_s)$, in both forward and backward search. $Q(G_s)$ can be incrementally maintained [43].

(2) Given an action s in o_b, BiExpQ extends Backward to generate refined enabling elements. These elements not only make s applicable, but are also the missing elements in the potential matches of Q in order to make g a part of the answer.

(3) Unlike BiExp, when an element s is identified in $o_b \cap o_f$, it extracts ρ that contains s, and verifies whether $g \in Q(G^\rho)$.

Analysis. The correctness of BiExpQ follows from the variant that any returned sequence ρ, if not $\emptyset$, is an explanation of g, and the verification further ensures $g \in Q(G^\rho)$. For time cost, BiExpQ performs one additional step for query processing for each action, and remains to be in $O(T \cdot (|V|^2|\Sigma|)^{\frac{b}{2}})$ time.

VI. Experiment

Using real-world graphs and query benchmark, we evaluate (1) the effectiveness and efficiency of BiExp and BiExpQ on clarifying missing elements; (2) the effectiveness of ApxExp for budgeted inference, and (3) case analysis for applications.

Experimental setting. We used the following setting.

Datasets. We use three real graphs. Each graph G contains nodes that are curated from two real knowledge bases, with a set of "ground truth" node pairs Γ. Each pair in Γ is either equivalent or bears a missing edge.

(1) DBYa[1]: [12] with $592K$ nodes, $4.5M$ edges, and $50K$ equivalent pairs (covering 10 types of entities) with aligned attributes curated from knowledge bases DBPedia and YAGO.

(2) DBIM[2]: [44] contains $33K$ nodes, $200K$ edges and $33.4K$ entities covering 10 types, totaling $9.5K$ equivalent pairs across DBPedia and IMDb (a movie knowledge base).

(3) OAG[3]: an open academic graph which unifies Microsoft Academic Graph and Aminer. We sample graphs that contain papers with selected topics (*e.g.,* "database", "machine learning") and related information (authors, citation). The graph contains $2.5M$ nodes, $5.2M$ edges and $106K$ equivalent pairs.

(4) SynDBYa: To evaluate the scalability of algorithms, we also generate synthetic graphs SynDBYa with ground truth, initialized by DBYa and its ground truth. We enhance ground truth Γ obtained from DBYa by duplicating the equivalent node pairs and missing edges, and sample missing elements from the enhanced ground truth Γ. We produced synthetic graphs with size up to $2M$ nodes and $25M$ edges.

We induce missing elements of interests from the corresponding ground-truth pairs in Γ, which refer to "one-sided" values that occur in only one source (thus missing in another).

Constraint generation. We calibrated the discovery of data constraints from Γ of all the graphs, to ensure they have high precision and recall over ground truth.

(1) We choose graph keys as NEs and use the algorithm in [11] to discover keys that cover ground-truth in Γ. We set support and confidence threshold as 0.8 and 0.9, respectively, and extracted 250, 20 and 12 graph keys, which cover $40K$ (80%), $7.6K$ (80%), and $90K$ (85%) equivalent node pairs in DBYa, DBIM, and OAG, respectively.

(2) We choose graph association rules [15], [16] as EGs, and detect EGs with confidence threshold 0.8 and a smaller support threshold 0.1 to cover missing edges with multiple types in Γ. We extracted in total 750, 25 and 10 EGs for DBYa, DBIM and OAG, respectively.

Query generation. We generate queries based on DBpedia SPARQL Benchmark [45] for DBYa. We choose queries to have some answers in the ground truth but not seen in the incomplete graphs, and sample missing elements for tests.

[1]https://github.com/lgalarra/vickey
[2]https://www.csd.uoc.gr/~vefthym/minoanER/datasets.html
[3]https://www.openacademic.ai/oag/

Algorithms. We implemented BiExp (Section IV), ApxExp (Section V-A), BiExpQ (Section V-B), and the following.

(1) We compare BiExp with (a) BiExp_N, a variant of BiExp without pruning techniques (PruneFwd and PruneBwd); (b) BiExp_Fwd and BiExp_Bwd, variants of BiExp_N that performs only forward and backward search, respectively; (c) BiExp_NE and BiExp_EG, which access Σ that contains only NEs and EGs respectively. Similarly, we compare BiExpQ with BiExpQ_N, BiExpQ_NE and BiExpQ_EG.

(2) We compare the approximation algorithm ApxExp with (a) ApxExp_N, a variant without forward pruning PruneFwd; (b) ApxExp_NE (resp. ApxExp_EG) which access Σ that only contains NEs (resp. EGs).

(3) Batch enforcement: $\mathsf{NE} + \mathsf{EG}$, which stacks the batch enforcement of NEs followed by EGs, and a reversed counterpart $\mathsf{EG} + \mathsf{NE}$ that enforce EGs first. These methods simulate entity matching and link prediction over the entire G.

We adopt VF2 [46] to detect violations (DetVio), and incremental pattern matching [43] to maintain query answers in BiExpQ. It takes on average 8 seconds to detect all the violations for graph patterns with 7 nodes and edges.

Metrics. We report the following for the effectiveness. (1) For BiExp (resp. BiExpQ), we report *coverage*, the ratio of the number of missing elements (resp. missing answers) explained by BiExp to the total N requests (resp. queries). (2) For ApxExp, we define the *normalized informativeness gain* as $\frac{\mathsf{cg}(\rho)}{\mathsf{cg}(\rho^*)}$ to measure the closeness between computed explanations and optimal counterpart (obtained by enumeration). (3) To evaluate the applications in graph refinement, we also report standard precision $\frac{|\mathcal{U}\cap\Gamma|}{|\mathcal{U}|}$ and recall $\frac{|\mathcal{U}\cap\Gamma|}{|\Gamma|}$, where $\mathcal{U}$ refers to the elements inferred by the algorithms.

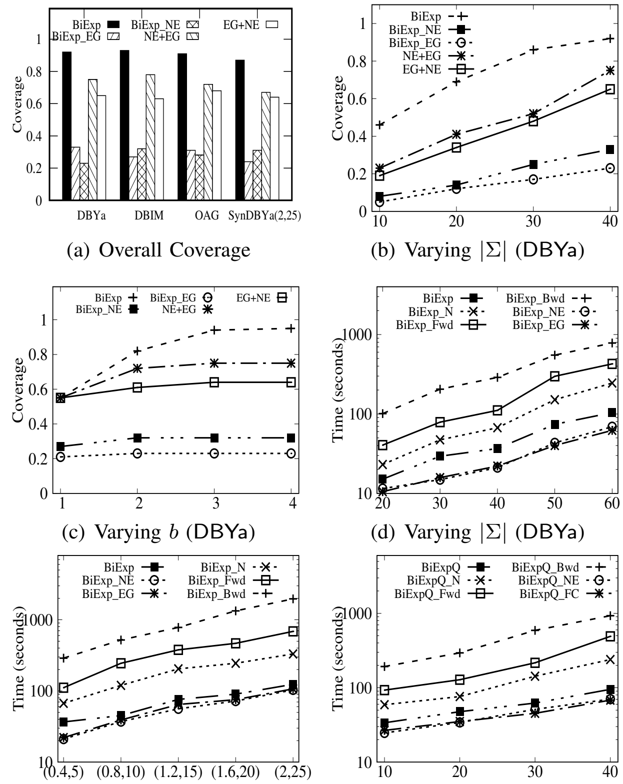
By default, we set N=20, a size bound b = 4 for BiExp, a budget B=800 as the total number of actions for ApxExp, and Σ contains 20 NEs and 20 EGs per test.

Environment. All the algorithms are implemented in Java. We ran all our experiments on a Linux machine powered by an Intel 2.4 GHz CPU with 128 GB of memory. We ran each experiment 10 times and reported the averaged results.

Experimental results. We next report our findings.

Exp-1: Coverage. We sampled missing elements to be clarified from ground truth and report the coverage of requests in Figure 8(a). BiExp_N, BiExp_Fwd and BiExp_Bwd (omitted) produce the same results as BiExp.

For all cases, BiExp covers on average 91% of the 20 requests with explanations that contain at most 4 actions. BiExp outperforms BiExp_EG (resp. BiExp_NE) by 62% (resp. 65%). We found that 34% of the explanations generated by BiExp have at least 2 actions. These actions can not be derived by a batch enforcement of NEs or EGs. BiExp further improves the coverage by 17% (resp. 27%) compared with $\mathsf{NE} + \mathsf{EG}$ (resp. $\mathsf{EG} + \mathsf{NE}$). These results verify that our methods can effectively clarify targeted missing elements without entity matching or link prediction over the entire graph.



(a) Overall Coverage (b) Varying $|\Sigma|$ (DBYa)

(c) Varying b (DBYa) (d) Varying $|\Sigma|$ (DBYa)

(e) Varying $|G|$(SynDBYa) (f) Varying N (DBYa)

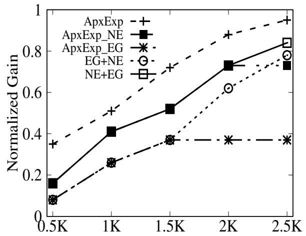
Fig. 8: Performance of BiExp and BiExpQ

The impact of $|\Sigma|$. Varying $|\Sigma|$ from 10 to 40, we report the coverage of the algorithms in Figure 8(b). The coverage of all algorithms increases as more data constraints are available. This leads to richer semantics of the incomplete graphs, thus is more likely to recover missing elements. On the other hand, the coverage of BiExp increases from 46% to 92% as $|\Sigma|$ varies from 10 to 40, while BiExp_NE and BiExp_EG explain up to 33% and 23% of the missing elements. This shows that BiExp can effectively exploit more types of data constraints and their interactions to explain missing elements.

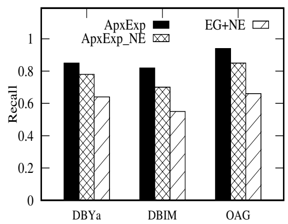
The impact of size bound b. Using the same setting with 20 requests, we varied b from 1 to 4. Figure 8(c) shows that all algorithms cover more requests over larger b. BiExp "degrades" to EG +NE and NE +EG when b=1, but effectively exploits useful actions to cover more requests for larger b. For N requests with queries and missing elements, the coverage of BiExpQ is close to that of BiExp (thus not shown).

Exp-2: Efficiency. Using the same setting in Exp-1, we report the efficiency of our algorithms.

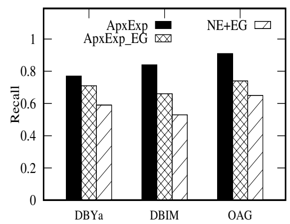
Varying $|\Sigma|$. Fixing N = 20 and varying $|\Sigma|$ from 20 to 60, we report the performance of the algorithms over DBYa in Figure 8(d). (1) All the algorithms take more time to explore more actions and data constraints. BiExp takes up to 1.9 seconds to explore 40 data constraints per request. (2) BiExp outperforms all the variants. It outperforms BiExp_N, BiExp_Fwd and BiExp_Bwd by 2.1, 3.1 and 7.3 times on average, due to the pruning strategy in bidirectional search. (3) BiExp_N improves the efficiency of BiExp_Bwd and BiExp_Fwd by 1.7 and



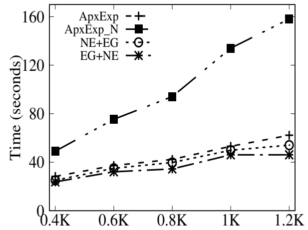
(a) Varying B (DBYa)



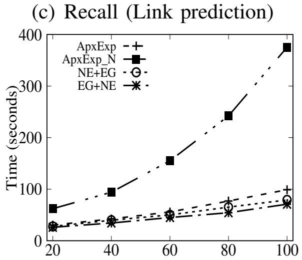
(b) Recall (Entity matching)



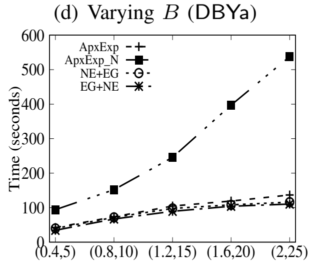
(c) Recall (Link prediction)



(d) Varying B (DBYa)



(e) Varying $|\Sigma|$ (DBYa)



(f) Varying $|G|$(SynDBYa)

Fig. 9: Performance of ApxExp

4 times with the bidirectional strategy alone. BiExp_Bwd takes the most time, as it explores an excessive number of enablers not refined by the forward search. (4) BiExp_EG and BiExp_NE take less time compared with BiExp, as less actions are triggered with only NE or EG.

Scalability. Using larger synthetic graphs SynDBYa and varying $|G|$ (denoted by $(|V|, |E|)$) from $(0.4M, 5M)$ to $(2M, 25M)$, we report the scalability of the algorithms in Figure 8(e) (in log scale). While all the methods scale well, BiExp is the least sensitive among all variants. It takes less than 120 seconds over graphs of size $(2M, 25M)$ to process 20 requests, and take on average 5.9 per request. The pruning strategy improves the performance better over larger G. For example, BiExp outperforms BiExp_N, BiExp_Fwd, BiExp_Bwd by 2.7, 5.5 and 13 times over a graph with size $(2M, 25M)$.

Varying N (BiExpQ). Varying N from 10 to 40, we report the time cost of BiExpQ and its variants in Figure 8(f). As other methods are not designed for explaining missing answers, we report their time for clarifying the missing element. While all algorithms scale well over N, BiExpQ takes up to 90 seconds to explain all 40 missing values in the answer (less than 2.3 seconds per query). On average, it outperforms BiExpQ_N, BiExpQ_Fwd and BiExpQ_Bwd by 2, 1.7 and 3.8 times due to pruning. This verifies the advantage of bidirectional search.

Exp-3: Effectiveness of ApxExp**.** To evaluate ApxExp, We set budget B as the total number of actions allowed to be explored (same as the number of calls for oracle; Section V-A). ApxExp_N (omitted) generates the same result.

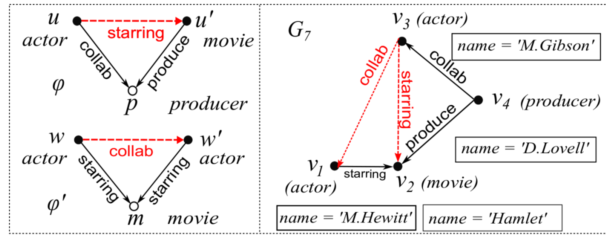


Fig. 10: Clarifying inaccurate elements: case study.

The impact of resource bound B. As shown in Figure 9(a) over DBYa, all the algorithms achieve higher informativeness gain when more actions are allowed to be explored. We observe the following. (1) ApxExp achieves the highest gain among all the methods. It converges earlier at the highest gain compared with other methods. This verifies that ApxExp can make "better" decisions early by prioritizing the actions. (2) ApxExp_NE (resp. ApxExp_EG) terminates at maximum gain after B =$2K$ (resp. $B < 1.5K$) actions, as it reaches relative completeness instances using NEs (resp. EGs) alone. (3) EG + NE continues to improve the gain and outperforms ApxExp_EG when $B > 1.5K$. We found more node merges are still applicable due to new edges inserted by enforcing EGs. Similarly, NE + EG outperforms ApxExp_NE when $B > 2K$.

Accuracy. Fixing $B = 2K$, we report the recall (precision not shown) of ApxExp_EG using NEs (entity matching) and EGs (link prediction) over corresponding ground truth in Figure 9(b) and 9(c), respectively. For entity matching, we omit ApxExp_EG and NE + EG, since EGs do not trigger merging (NE + EG is equivalent to ApxExp_NE). Similarly, we omit ApxExp_NE and EG + NE.

We observe the following. (1) ApxExp improves the recall and also precision (not shown) for both entity matching and link prediction as more missing elements are inferred by interacting NEs and EGs. (2) NE + EG (resp. EG + NE) can capture some new violations, thus improves precision of ApxExp_NE (resp. ApxExp_EG). However, EG + NE (resp. NE + EG) consumes majority of resource on link prediction (resp. entity matching) alone, and has the lowest recall.

Exp-4: Efficiency of ApxExp**.** Using the same setting as in Exp-3, we evaluate the impact of several factors.

The impact of budget B. Figure 9(d) reports the the time cost of the algorithms up to the exploration of B actions.

(1) All the algorithms take longer time with larger B. ApxExp is quite feasible in budgeted scenario. It incurs a delay time on average 0.25 (resp. 0.19) second per batch, and 42 (resp. 22) seconds to explore 800 actions over DBYa (resp. DBIM).

(2) The online prioritization and forward pruning is quite effective. ApxExp outperforms ApxExp_N by 2.36 times on average. We observed that the pruning effectively reduces 76% of graph homomorphism (isomorphism) verifications.

(3) ApxExp performs more verifications than NE + EG and EG + NE but incurs comparable time cost. EG + NE terminates when $b > 1K$ and exhausts applicable actions. In contrast, ApxExp is able to exploit more and useful actions.

Varying $|\Sigma|$. Figure 9(e) verifies that all algorithms scale well over $|\Sigma|$. Specifically, ApxExp takes 98.9 seconds with 100 constraints and is 3.8 times faster than ApxExp_N.

Scalability. Figure 9(f) shows ApxExp scales well with $|G|$, and outperforms ApxExp_N by 3.9 times when $|G|$ is varied to $(2M, 25M)$. Algorithms ApxExp scales well with both $|\Sigma|$ and $|G|$, with performance comparable with $\mathsf{NE} + \mathsf{EG}$.

Exp-5: Case Study. We show that BiExp can be naturally used to clarify the occurrence of erroneous elements that are inferred by the constraints. Fig. 10 illustrates a fragment of DBIM (G_7) with two discovered graph association rules:

(φ):*"an* ***actor*** *(u)* ***stars*** *in a* ***movie*** *(u') if a* ***producer*** *(p) he collaborates also produces the same movie"*

(φ'): *"an* ***actor*** *(w)* ***collaborates*** *with an* ***actor*** *(w') if they both starred a movie (m)."*

The insertion of an edge collab (v_1, v_3) is annotated as "inaccurate" (*'M.Gibson' and 'M.Hewitt' collaborated in a movie*). BiExp generates an explanation, which states that an edge insertion $\oplus((v_3, v_2), \mathsf{starring})$ (by enforcing φ) leads to the inaccurate element collab (v_1, v_3) (by enforcing φ'). A closer inspection suggests that φ can be an "overkill", given the exception of (v_3, v_2) ("an actor may not always be starring a movie produced by a producer he collaborated with").

VII. Conclusion

We have introduced a constraint-based approach to clarify missing elements with established graph data constraints. We have formulated the measurements and explanation problems. We have developed bidirectional and one-pass algorithm with quality guarantees. Our experimental study has verified that our constraint-based methods can effectively and efficiently clarify targeted missing values in large graphs and under resource budgets. One future topic is to study parallel algorithms for constraint-based methods. Another topic is to investigate constraint-based methods to clarify erroneous data in graphs.

References

[1] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic web*, vol. 8, no. 3, pp. 489–508, 2017.

[2] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang, "From data fusion to knowledge fusion," *PVLDB*, 2014.

[3] A. Arioua and A. Bonifati, "User-guided repairing of inconsistent knowledge bases," 2018.

[4] X. Wang, L. Haas, and A. Meliou, "Explaining data integration," *Data Engineering Bulletin*, 2018.

[5] M. H. Namaki, Q. Song, Y. Wu, and S. Yang, "Answering why-questions by exemplars in attributed graphs," in *SIGMOD*, 2019.

[6] G. Lausen, M. Meier, and M. Schmidt, "Sparqling constraints for rdf," in *EDBT*, 2008.

[7] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek, "Amie: association rule mining under incomplete evidence in ontological knowledge bases," in *WWW*, 2013.

[8] W. Fan, Z. Fan, C. Tian, and X. L. Dong, "Keys for graphs," *PVLDB*, vol. 8, no. 12, 2015.

[9] W. Fan, Y. Wu, and J. Xu, "Functional dependencies for graphs," in *SIGMOD*, 2016.

[10] L. Caruccio, V. Deufemia, and G. Polese, "Relaxed functional dependencies—a survey of approaches," *TKDE*, vol. 28, no. 1, 2016.

[11] A. Morteza and C. Fei, "Keyminer: Discovering keys for graphs," in *VLDB workshop*, 2018.

[12] D. Symeonidou, L. Galárraga, N. Pernelle, F. Saïs, and F. Suchanek, "Vickey: mining conditional keys on knowledge bases," in *ISWC*, 2017.

[13] H. Ma, M. Alipourlangouri, Y. Wu, F. Chiang, and J. Pi, "Ontology-based entity matching in attributed graphs," *PVLDB*, 2019.

[14] S. Kwashie, L. Liu, J. Liu, M. Stumptner, J. Li, and L. Yang, "Certus: an effective entity resolution approach with graph differential dependencies (gdds)," *PVLDB*, vol. 12, no. 6, pp. 653–666, 2019.

[15] W. Fan, X. Wang, Y. Wu, and J. Xu, "Association rules with graph patterns," *PVLDB*, 2015.

[16] P. Lin, Q. Song, J. Shen, and Y. Wu, "Discovering graph patterns for fact checking in knowledge graphs," in *DASFAA*, 2018.

[17] G. Fan, W. Fan, Y. Li, P. Lu, C. Tian, and J. Zhou, "Extending graph patterns with conditions," in *SIGMOD*, 2020.

[18] J. Cheney, A. Ahmed, and U. A. Acar, "Provenance as dependency analysis," in *DBPL*, 2007, pp. 138–152.

[19] J. Xu, W. Zhang, A. Alawini, and V. Tannen, "Provenance analysis for missing answers and integrity repairs." *IEEE Data Eng. Bull.*, vol. 41, no. 1, pp. 39–50, 2018.

[20] I. Bhattacharya and L. Getoor, "Entity resolution in graphs," *Mining graph data*, 2006.

[21] M. Pershina, M. Yakout, and K. Chakrabarti, "Holistic entity matching across knowledge graphs," in *Big Data*, 2015.

[22] W. E. Moustafa, H. Miao, A. Deshpande, and L. Getoor, "Grdb: a system for declarative and interactive analysis of noisy information networks," in *SIGMOD*, 2013.

[23] D. Firmani, B. Saha, and D. Srivastava, "Online entity resolution using an oracle," *PVLDB*, 2016.

[24] W. Fan, P. Lu, C. Tian, and J. Zhou, "Deducing certain fixes to graphs," *PVLDB*, 2019.

[25] P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and where: A characterization of data provenance," in *ICDT*, 2001.

[26] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu, "Why so? or why no? functional causality for explaining query answers," *arXiv preprint arXiv:0912.5340*, 2009.

[27] Q. Song, M. H. Namaki, and Y. Wu, "Answering why-questions for subgraph queries in multi-attributed graphs," in *ICDE*, 2019.

[28] G. Amendola and L. Libkin, "Explainable certain answers." in *IJCAI*, 2018.

[29] L. Libkin, "Certain answers meet zero-one laws," in *PODS*, 2018.

[30] B. Glavic, S. Köhler, S. Riddle, and B. Ludäscher, "Towards constraint-based explanations for answers and non-answers," in *TAPP*, 2015.

[31] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of databases: the logical level*, 1995.

[32] W. Fan and P. Lu, "Dependencies for graphs," in *PODS*, 2017.

[33] F. Naumann and M. Häussler, "Declarative data merging with conflict resolution," 2002.

[34] W. Gatterbauer and D. Suciu, "Data conflict resolution using trust mappings," in *SIGMOD*, 2010.

[35] Full version. *https://songqi1990.github.io/Files/paper/icde2021full.pdf.*

[36] L. Galárraga, S. Razniewski, A. Amarilli, and F. M. Suchanek, "Predicting completeness in knowledge bases," in *WSDM*, 2017.

[37] S. Guo, X. L. Dong, D. Srivastava, and R. Zajac, "Record linkage with uniqueness constraints and erroneous values," *PVLDB*, 2010.

[38] W. Fan, "Dependencies for graphs: Challenges and opportunities," *JDIQ*, vol. 11, no. 2, pp. 1–12, 2019.

[39] N. D. Jones, "An introduction to partial evaluation," *ACM Comput. Surv.*, vol. 28, no. 3, p. 480–503, 1996.

[40] S. E. Whang, D. Marmaros, and H. Garcia-Molina, "Pay-as-you-go entity resolution," *TKDE*, vol. 25, no. 5, pp. 1111–1124, 2012.

[41] M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg, "A knapsack secretary problem with applications," in *Approximation, randomization, and combinatorial optimization*, 2007, pp. 16–28.

[42] Y. Zhou, D. Chakrabarty, and R. Lukose, "Budget constrained bidding in keyword auctions and online knapsack problems," in *WINE*, 2008.

[43] W. Fan, X. Wang, and Y. Wu, "Incremental graph pattern matching," *TODS*, vol. 38, no. 3, pp. 1–47, 2013.

[44] V. Efthymiou, K. Stefanidis, and V. Christophides, "Benchmarking blocking algorithms for web entities," *IEEE Trans. on Big Data*, 2016.

[45] M. Morsey, J. Lehmann, S. Auer, and A.-C. N. Ngomo, "Dbpedia sparql benchmark–performance assessment with real queries on real data," in *ISWC*, 2011.

[46] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE TPAMI*, vol. 26, no. 10, pp. 1367–1372, 2004.

VIII. Appendix

Proof of Lemma 3. Given any G' and G'' ($G' \neq G''$) in G_Σ, we prove that the relation $\preceq$ is a partial order by showing the following. (1) $G' \preceq G'$ (G'=G'^{ϵ}). (2) If $G' \preceq G''$, then $G'' \not\preceq G'$. For any G''=G'^{ρ}, there is a graph homomorphism h^ρ from G' to G'' (Lemma 1) such that G'' either has one merged node or a new edge, which cannot find counterparts in G', as a result of applying at least one operator in ρ ($\rho \neq \emptyset$). Thus $G'' \in G'_\Sigma$, and $G' \notin G''_\Sigma$. (3) If $G' \preceq G''$ and $G'' \preceq G'''$, then $G' \preceq G'''$. Let G''=G'^{ρ} and G'''=$G''^{\rho'}$, then $G''' = G'^{\rho\rho'}$ via a sequence that concatenates ρ and ρ'.

Given Lemma 3, it is trivial that based on the monotonicity of partial ordered sets, Theorem 2 holds.

Proof of Theorem 4. Note that an action either merges two nodes or adds a new edge, thus given a graph G with $|V|$ nodes, a sequence can apply at most $|V|$ (resp. $|V|^2$) merge (resp. insertion) operators to merge all nodes in the graph into one node (resp. insert an edge between each pair of nodes). We next prove Theorem 4(2) by contradiction. Assume there are two relatively complete instances G' and G'' in G_Σ that are not homomorphically equivalent. Assume there is no graph homomorphism from G' to G''. Then there exists a pair of nodes (v, v') in G', such that $(h(v), h(v'))$ is not in G''. Then either v or v' has no counterpart in G'' or $(h(v), h(v'))$ is not an edge of G''. There exists at least a match (v_o, v'_o) in G that are merged in G' due to enforced equivalence or inserted as a missing edge, but not processed in G''. This contradicts that G'' is relatively complete, *i.e.,* there exists an action s such that $G''^s \neq G''$.

Proof of properties of measures. (1) It is trivial that $\mathsf{cg}(\rho) = 0$ if ρ is empty. (2) if $G^\rho \preceq G^{\rho'}$, then ρ contains at least one more operator compared with ρ', thus $\mathsf{cg}(\rho) \geq \mathsf{cg}(\rho')$. (3) We prove that for any φ, $\varphi' \in \Sigma$ such that $\varphi \models \varphi'$, $\mathsf{cg}(\rho\{s\}) \geq \mathsf{cg}(\rho\{s'\})$ for action s (resp. s') that enforces φ (resp. φ'). Given two data constraints φ and φ' in Σ. $\varphi \models \varphi'$ if for all graphs G such that $G \models \varphi$, $G \models \varphi'$. That is, if (v, v') violates φ', then (v, v') violates φ, but not vice versa. Thus φ' has no more support than φ and introduces no more violations. Hence $G^{\rho\{s'\}} \preceq G^{\rho\{s\}}$, and $\mathsf{cg}(\rho\{s\}) \geq \mathsf{cg}(\rho\{s'\})$ for any s and s' that enforce φ and φ', respectively.

Proof of Theorem 5. We start with Σ-explainability, a special case of Σ-explanation: Given G, Σ and a "wildcard" element, it is to decide whether there exists an explanation ρ with size 1 (a single action). We first show the following result.

Lemma 8: *Σ-validation is a complement of Σ-explainability.* □

Proof: We show $G \not\models \varphi$ if and only if there exists an explanation ρ with size 1. For if, assume ρ explains an element g with a single action s. Then there is a pair (v, v') that violates at least one data constraint $\varphi \in \Sigma$. If g is an attribute-value pair $(v.A, a)$ (resp. edge $r(v, v')$), then s enforces an NE (resp. EG) $\varphi \in \Sigma$ over (v, v'), where $v'.A$ has a possible value a in G and variable $X_{A_v} = \perp$ (resp. $r(v, v')$ is a missing edge and $X^r_{vv'} = \perp$). Hence $G \not\models \Sigma$. The only if condition follows from the definition of explanation, as if $G \not\models \varphi$, there exist an action s that either merges two nodes or adds one edge for NE and EG respectively. Then ρ=$\{s\}$ is the explanation. □

Procedure NextBatch
Input: an action s, instance G', constraints Σ.
Output: a ranked list of actions $\mathcal{S}$.
1. list $\mathcal{S}$:= $\mathsf{Forward}(s, G', \Sigma)$;
2. **for each** action $s \in \mathcal{S}$ **do**
3. **if** $\frac{\mathsf{cg}(s,G)}{c(s)} < \Psi(\frac{\mathsf{cg}(\rho,G)}{B})$ **then** $\mathcal{S} := \mathcal{S} \setminus \{s\}$;
4. sort $\mathcal{S}$ in descending order of $\mathsf{cg}(s, G)$;
5. **return** $\mathcal{S}$;

Fig. 11: Procedure NextBatch

We show an upper bound for Σ-explanation given an oracle for Σ-explanability problem that is in class $\mathcal{X}$.

Lemma 9: *Σ-explanation is in $P^{\mathcal{X}}$ if Σ-explanability is in $\mathcal{X}$.* □

Proof: We reduce Σ-explanation with an oracle for Σ-reachability to the reachability problem over the lattice ($\mathcal{G}_\Sigma, \preceq$) induced by informativeness ordering $\preceq$. It is in PTIME to construct the lattice (by verifying $|V|^2|\Sigma|$ possible violations). We show that there exists a sequence ρ from G to any relative complete instances $G' \in g_\Sigma$ (decidable in PTIME) if and only if ρ explains g. If ρ explains g, then ρ itself is path from G to the instance g in the lattice. For only if, it is to decide if g is reachable from the root node in the lattice to the instance of g with length bound b. Thus it is equivalent to find the shortest path from the root node to the instance of g (in PTIME) and verify if the path has length no larger than b. □

This provides matching upper bound given the complexity of validation. As Σ-validation is shown to be coNP-complete for graph keys and association rules as graph data dependencies [38], Σ-explainability is NP-complete, hence the upper bound Δ_2^P follows.

Proof of Lemma 6 . (1) Given an action s, if s is in an explanation ρ then it must be an enabler of its succeed action towards g since there exists an edge in the backward tree that connects s and the succeed action. Given that $\varphi^{\uparrow+} \cap \Sigma_o^{+\downarrow} = \emptyset$, which means none of the constraints can generate s as an enabler, $s \notin \rho$. (2) Similarly for forward search, if $\varphi^{\downarrow+} \cap \Sigma_o^{+\uparrow} = \emptyset$, then applying s can not trigger any enablers that lead to g. Thus we have $s \notin \rho$.

Procedure NextBatch**.** The procedure NextBatch (illustrated in Figure 11) performs forward search as Forward in BiExp, to generate a set of actions $\mathcal{S}$ (line 1) and refines the actions by pruning those that can no longer introduce gain beyond the threshold $\Psi(\cdot)$. sorts $\mathcal{S}$ (line 2-3). It also sorts $\mathcal{S}$ to early terminate its sequential processing in ApxExp against a non-decreasing threshold.

A running example of ApxExp**.** Let $B = 2$, $|V_4| = 10$, $c(s)$ be a unit cost, and $\Sigma = \{\varphi_4, \varphi_5, \varphi_7\}$ defined on G_4, where φ_7 (not shown) states that *"a director* collaborates *with a* *writer if they write the same movie"*. φ_7 triggers an action $s_4((z_5, z_{10}), o_4, \varphi_7)$ where $o_4 = \oplus((z_5, z_{10}), collab)$. Assume G_4 contains 20 collaborate edges, then $\mathsf{cg}(s_4) = \frac{1}{20}$. (1) ApxExp invokes NextBatch to initialize $\mathcal{Q}$ with an action $\{s_1\}$, and applies s_1 to enrich G_4 to the instance G_5. NextBatch then identifies two actions s_4 and s_2. (2) ApxExp then selects s_2 instead of s_4, as $\frac{\mathsf{cg}(s_4,G_4)}{c(s_4)}$=0.05<$\Psi(\frac{1}{2})$=0.061. With cost 2, it achieves at an instance G_4^ρ with a total gain $\mathsf{cg}(\rho, G)$=1.5.

Proof of competitive ratio. The *weighted secretary problem* [41], [42] is to decide whether to add a new item (upon its arrival from an unknown sequence with a value and a cost) to a set, such that the total value is maximized with a bounded total cost. The algorithm ApxExp solves a constraint online budgeted secretary problem with a matching competitive ratio [42] by the following reduction.

(1) The "stream" of ApxExp steps with completeness gain and cost is correctly produced in an online manner. At runtime, NextBatch enforces an order of the underlying set of input items as the input of the secretary problem: any step s' generated by NextBatch is seen only after s is applied.

(2) Each step carries an operator with corresponding gain and cost. Any subset of items preserves the order they are seen in the stream. The cost and total gain $\mathsf{cg}(\cdot)$ remains the same due to order independence as informativeness gain $\mathsf{cg}(s, G)$ is calculated based on the input graph G.

The triggered steps are a fraction of all possible BiExp sequences. An offline process can enumerate all actions generated by NextBatch and compute a relative optimal result with maximum gain. For edge insertion $\oplus$, the informativeness gain is within $[\frac{1}{|V|}, 1]$. Since we define

$$\mathsf{diff}(v, v') = \frac{1}{2}(\mathsf{diffN}(v, v') + \mathsf{diffF}(v, v'))$$

where $\mathsf{diffN}(v, v') \in [\frac{1}{|V|}, |V|^2]$ is the difference of the neighborhood of v and v' and $\mathsf{diffF}(v, v') \in [0, |V|^2]$ quantifies the difference between the tuples $F(v)$ and $F(v')$. Given that $\mathsf{supp} \in [\frac{1}{|V|^2}, 1]$, we have the the lower and upper bound of the gain-to-cost ratio of each operator as $L = \frac{1}{|V|^4 c_u}$ and $U = \frac{|V|^2}{c_l}$. If we take unit cost, *i.e.,* $c_u = c_l = 1$, then the thresholding ensures a competitive ratio of $ln(U/L) + 1$ = $6 \cdot ln(|V|) + 1$ by solving an online budgeted secretary problem.