The VLDB Journal manuscript No.
(will be inserted by the editor)

Third and Boyce-Codd Normal Form for Property Graphs

Foundations, Achievements, and Normalization

Philipp Skavantzos · Sebastian Link

Received: date / Accepted: date

Abstract Normalization minimizes sources of potential data inconsistency and costs of update maintenance incurred by data redundancy. For relational databases, different classes of dependencies cause data redundancy and have resulted in proposals such as Third, Boyce-Codd, Fourth and Fifth Normal Form. Features of more advanced data models make it challenging to extend achievements from the relational model to missing, non-atomic, or uncertain data. We initiate research on the normalization of graph data, starting with a class of functional dependencies tailored to property graphs. We show that this class captures important semantics of applications, constitutes a rich source of data redundancy, its implication problem can be decided in linear time, and facilitates the normalization of property graphs flexibly tailored to their labels and properties that are targeted by applications. We normalize property graphs into Boyce-Codd Normal Form without loss of data and dependencies whenever possible, but guarantee Third Normal Form in general. Experiments on real-world property graphs quantify and qualify various benefits of graph normalization: 1) removing redundant property values as sources of inconsistent data, 2) detecting inconsistency as violation of functional dependencies, 3) reducing overheads for updates by orders of magnitude, and 4) significant speed ups of queries.

Keywords Key; Functional dependency; Normal form; Normalization; Property graph; Query; Update

P. Skavantzos
School of Computer Science, The University of Auckland, Auckland, New Zealand
E-mail: philipp.skavantzos@auckland.ac.nz

S. Link
School of Computer Science, The University of Auckland, Auckland, New Zealand
E-mail: s.link@auckland.ac.nz

1 Introduction

Normalization minimizes sources of potential data inconsistency and costs of integrity maintenance incurred by updates of redundant data. Based on data dependencies that cause redundancy, classical normalization transforms schemata into normal forms where these dependencies can be enforced by keys only, or come close to it. For example, this is achieved by Boyce-Codd Normal Form (BCNF) for functional dependencies (FDs) [11,51], Fourth Normal Form for multivalued dependencies [13], Fifth Normal Form for join dependencies [42], Inclusion Dependency Normal Form for functional and inclusion dependencies [27], and Domain-Key Normal Form [14]. Third Normal Form (3NF) minimizes sources of data redundancy under the additional target of enforcing all FDs without joining relation schemata [7,25], and Bounded Cardinality Normal Form minimizes the level of data redundancy caused by FDs [30,28]. Some achievements carry forward to richer data formats, including SQL [23,24] and models with missing data [26,48,49], Nested [34,43], Object-Oriented [41], Temporal [22], Web [3,33], and Uncertain Databases [29].

Graph databases experience new popularity due to more mature technology in response to the demand of applications for finding relationships within large amounts of heterogeneous data, such as social network analysis, outlier and fraud detection. Graphs can represent data intuitively and efficiently. Both research and industry have brought forward sophisticated technologies with mature capabilities for processing graph data. Recently, classical classes of data dependencies, such as keys and FDs, have been extended to graph databases, and have been put to use for data cleaning and fraud detection tasks [15]. Indeed, Fan [15] remarks that graph

dependencies provide a rare opportunity to capture the semantics of application domains on graph databases, which are schema-less. Interestingly, however, the normalization of graph data has neither been mentioned in the literature nor has it been subject of investigation yet. This is surprising since it is a very natural question to ask what normalization of graph data may actually mean. Indeed, any mature data model needs to facilitate principles of data integrity. Since a strong use case of graph data is analytics, the quality of analysis depends fundamentally on the quality of graph data. This firmly underpins the need to understand sources of data inconsistency and other data quality issues. This includes the challenge of understanding opportunities for more efficient integrity maintenance and query processing, and database design principles within schema-less graph environments. In relational databases, constraints restrict instances of a given schema to those considered meaningful for the underlying application. Normalization restructures the schema and constraints such that data redundancy is minimized and constraint management made more efficient. When attempting to normalize property graphs, we do not have a schema and will therefore need to rely on the constraints exclusively. In particular, it means that a normalized set of constraints would not admit any graph with redundant data value occurrences, but without any restriction of such a graph's structure by any schema. This sounds intriguing and the flexibility of graph data may promote restructuring only that part of the graph required by an application. Our contributions towards the aim of initiating research on normalizing property graphs can be summarized as follows:

(1) We introduce uniqueness constraints and functional dependencies as declarative means to i) express completeness, integrity, and uniqueness requirements in the form of business rules that govern property graph data, and ii) form the source of redundant property values that drive goals for graph normalization.

(2) We show that our graph dependencies facilitate normalization as their implication problem can be captured axiomatically by finite Horn rules, and algorithmically by a linear-time decision algorithm.

(3) We normalize property graphs into lossless, dependency-preserving BCNF whenever possible, and guarantee 3NF in general. Normalization is tailored to nodes with labels and properties the target application requires. Unlike the relational case, our normalization can even be applied when FDs do not fully hold. Indeed, our normalization is still lossless and removes redundancy in the part conforming to the FDs.

(4) We demonstrate the benefits of property graph normalization experimentally. For some popular real-world property graphs we identify meaningful FDs and quantify how many redundant property values they cause. We provide examples of inconsistencies found as violations of meaningful FDs. We demonstrate that maintaining integrity and evaluating queries of different types improves by orders of magnitude on normalized property graphs. While not achieving the full scale of speed up that graph normalization accomplishes, indexing the left-hand side properties of FDs still gains significant efficiency in integrity maintenance and query processing without any change to the graphs.

Our results motivate a long line of future work on graph data normalization, including normal forms, the study of more expressive graph dependencies, relationships to conceptual and physical design of graph data, and the design for quality of graph data.

In what follows, we motivate our work with an application scenario in Section 2. Section 3 includes a concise review of relevant work. Section 4 contains a brief guide of concepts and notation. The semantics of property graphs and our class of graph-tailored constraints is given in Section 5. In Section 6, we establish axiomatic and algorithmic solutions for the implication problem of graph-tailored constraints. We then define Boyce-Codd and Third Normal Forms for property graphs in Section 7, before establishing their achievements in terms of eliminating data redundancy, sources of inconsistency and update inefficiency in Section 8. An algorithm for normalizing a given property graph into a lossless, dependency-preserving decomposition into 3NF is given in Section 9, which will be in Boyce-Codd Normal Form whenever possible. Experimental results that qualify and quantify the need and benefits of normalization are discussed in Section 10. We conclude and briefly comment on future work in Section 11. More details are available in our github repository[1].

Extension. Our current paper extends the conference version [39] in various directions. Firstly, it contains full proofs, which often provide constructions that reveal core insights for understanding the results but can also be used in practice. This includes the generation of property graphs that form counter-examples for implication problems or examples of data redundancy and update inefficiency. Secondly, we defined insertion inefficiencies and an Insertion Efficiency Normal Form that avoids insertion inefficiencies. We then show that Boyce-Codd Normal Form is equivalent to Insertion Efficiency Normal Form, adding another achievement that normalization of property graphs can accomplish. Thirdly, we then extended this new theoretical result to practice. In fact, some new experiments demonstrate

[1] `https://github.com/GraphDatabaseExperiments/normalization_experiments`

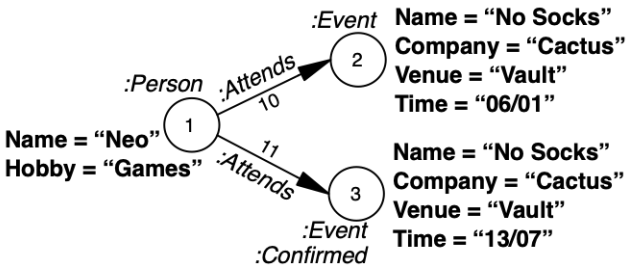


Fig. 1: Original Property Graph

the gain in efficiency when inserting new nodes in normalized property graphs. Fourthly, we also added a new type of queries that benefit from normalization. Here, redundant occurrences of the keyword `DISTINCT` can be eliminated by having `UNIQUE` indices present due to the unique constraints derived from normalization. Finally, we also added examples on the real-world property graph *Northwind* that illustrate meaningful FDs causing data inconsistency or redundancy, and how to use them for normalization.

2 Illustrative Example

In this section, we will use a minimal example to illustrate ideas and concepts that motivate our work.

The property graph G_0 in Figure 1 models an application where people attend events. Nodes and edges carry any number of labels, and may carry pairs of properties and values. In Figure 1 we have nodes labeled by *Person*, *Event*, and *Confirmed*. The latter label assures that *Event* nodes model events that have been confirmed. We also have edges from Person to (confirmed) Event nodes labeled by *Attends*, expressing that a person attends a (confirmed) event. Event nodes exhibit properties such as *N(ame)*, *C(ompany)*, *V(enue)* and *T(ime)*, expressing that a company (like "Cactus") is in charge of an event with a name (like "No Socks") held at a venue (like "Vault") and time (like "06/01").

Event data is subject to business rules expressed by uniqueness constraints (UCs) and FDs: Event nodes with properties C and T can be uniquely identified by the value combination on these two properties, $N \to C$ (events are managed by at most one company), $NT \to V$ (the time of events uniquely determines its venue), and $TV \to N$ (at any time any venue can host at most one event). Intuitively, FDs express that nodes with matching values on all properties of the left side have also matching values on all properties of the right side. However, characteristics of property graphs motivate additional features of graph dependencies. Firstly, properties may not exist on some nodes. Secondly, dependencies apply to different types of nodes. As a consequence, we want to provide data stewards with the ability to tailor uniqueness constraints and functional dependencies to i) completeness requirements for properties, and ii) the labels carried by nodes. For i), we take the principled approach that missing properties should not have an impact on the validity of constraints [48, 49]. Hence, graph-tailored UCs (gUC) and FDs (gFD) feature a property set P that restricts the set of vertices on which the constraint holds to those nodes for which all properties in P exist, and P contains at least all the properties that occur in the constraint. Moreover, a label set L is included in the specification of gUCs and gFDs that further restricts the set of vertices on which the constraint holds to those nodes which carry all the labels in L. We obtain the following constraints in our example: the gUC *Event*:CT:CT (σ_1), stipulating that all *Event* nodes that have properties C and T, are uniquely identified by the combination of values on these properties. The gFD *Event*:NC:$N \to C$ (σ_2) stipulates that *Event* nodes with properties N and C have matching values on C whenever they have matching values on N. Similarly, the gFDs *Event*:NTV:$NT \to V$ (σ_3) and *Event*:NTV:$TV \to N$ (σ_4) express that *Event* nodes with properties N, T, and V have matching values on V (N, respectively) whenever they have matching values on N and T (T and V, respectively). Finally, the gFD *Event*, *Confirmed* : $NCTV$: $NC \to T$ (σ_c) expresses that nodes with both labels *Event* and *Confirmed*, and with properties N, C, T, and V have matching values on T whenever they have matching values on N and C. The constraints exhibit non-trivial interactions. For instance, $\Sigma = \{\sigma_1, \ldots, \sigma_4\}$ implies σ_5 = *Event*:$NCTV$:TV, and every gUC L:P:X implies the gFD L:P:$X \to P$, but not vice versa. For instance, G_0 satisfies the gFD *Event*:NCV:$NC \to V$, but not the gUC *Event*:NCV:NC. In particular, no gUC can be expressed by any set of gFDs. As another example, G_0 satisfies σ_c, in particular, but not the gFD *Event*:$NCTV$:$NC \to T$. This illustrates the subtlety of reasoning with multiple labels. Similarly, if G_0' results from G_0 by adding label *Confirmed* to node 2 and removing property *Venue* from node 2, G_0' would still satisfy the gFD σ_c but not the gFD *Event*, *Confirmed*:NCT:$NC \to T$ resulting from σ_c by removing property V from $P = NCTV$.

We observe that every graph satisfying $\Sigma = \{\sigma_1, \ldots, \sigma_4, \sigma_c\}$, such as G_0, cannot exhibit any redundant property values on any nodes that carry both labels *Event*, *Confirmed* and all properties N, C, T, V. Interestingly, however, G_0 is one property graph that does exhibit a redundant data value occurrence on nodes that carry only label *Event* and all properties

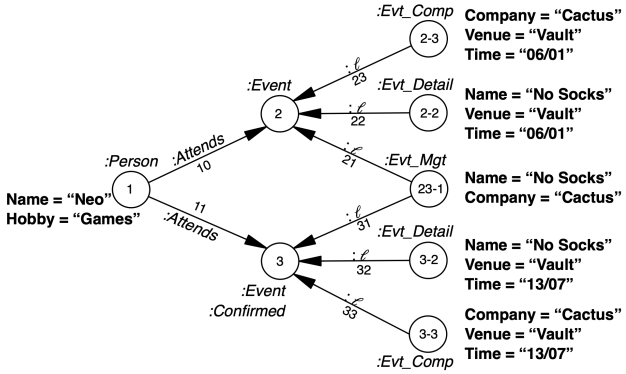


Fig. 2: Normalized Property Graph

$NCTV$. Indeed, each occurrence of company *Cactus* is redundant due to σ_2: If one occurrence of *Cactus* is changed to *any* different value, then σ_2 will be violated.

While G_0 is in BCNF for target label set $\{Event, Confirmed\}$ and property set $\{N, C, T, V\}$, and in 3NF for target label set $\{Event\}$ and property set $\{N, C, T, V\}$, it is not in BCNF for label set $\{Event\}$ and property set $\{N, C, T, V\}$. Figure 2 shows the normalized property graph G_n resulting from a non-obvious decomposition. Indeed, G_n is in BCNF tailored to the application requirements that decompose *Event* nodes exhibiting **all** properties in $NCTV$. The decomposition is lossless, as a simple join via ℓ-labeled edges would restore the original graph G_0. It is also dependency-preserving: gUC σ_1 adds gUC $\sigma_1' = Evt\_Comp{:}CVT{:}CT$, and gFDs $\sigma_2, \ldots \sigma_5$ result in gUCs $\sigma_2' = Evt\_Mgt{:}NC{:}N$, $\sigma_3' = Evt\_Detail{:}NTV{:}NT$, $\sigma_4' = Evt\_Detail{:}NTV{:}TV$, and even $\sigma_5' = Evt\_Comp{:}CVT{:}TV$. Note that σ_c is beyond scope of any decomposition not targeted at nodes with label *Confirmed*. The additional properties in the new constraints, such as V in σ_1', originate from the requirement for *Event* nodes to exhibit all properties $NCTV$. The new design eliminates all data redundancy caused by σ_2, for example in G_0. The fact that *Cactus* manages the event *No Socks* is only represented once (on the new node 23-1), avoiding data inconsistency, making update and query operations potentially more efficient. This is achieved by the new gUC σ_2', expressing that there cannot be two different *Evt_Mgt* nodes with properties N, C and matching values on N. Omitting the two *Evt_Comp*-nodes and their outgoing edges from G_n would still result in a lossless BCNF decomposition. However, the gUC σ_1 would be lost. In other words, the *Evt_Comp*-nodes ensure G_n is dependency-preserving.

Normalizing a property graph will minimize sources of potential inconsistency, bring forward more efficient updates of companies, and of aggregate queries that require the numbers of events a company manages. Intuitively, the benefits grow as the graph does. Hence, identifying opportunities and limits of graph databases in handling normalization has huge potential.

3 Previous Work

Normalization is a classical topic [31], but no framework exists for graph data yet. Normal forms characterize well-designed databases that only admit instances with no redundant data value caused by any dependency in the class considered. Fundamental are efficient solutions to the implication problem: BCNF and 3NF are founded on Armstrong's axioms ($\mathfrak{A}$) [4] and linear decision algorithms [6]. The latter drive decompositions into BCNF and 3NF [8].

Schema design for other data quality dimensions is in its infancy [5]. Completeness-tailored UCs and FDs were introduced for relations with missing values, and a normalization framework established that tailors relational design to completeness and integrity requirements [48,49]. Our work here extends this approach to property graphs with major differences: in contrast to [48,49] we cannot assume an underlying schema, we deal with graphs rather than relations, and we require an extension to accommodate labels.

Recently, much attention has been given to graph query languages, but several lines of work on integrity management have emerged, too. The comprehensive key proposal [2] sets out an expressive framework for specifying keys on nodes, edges, and properties. In particular, the gUC $\{L_1, \ldots, L_m\}{:}\{P_1, \ldots, P_n\}{:}\{U_1, \ldots, U_k\}$ can be specified as the exclusive PG-key below.

`FOR` $x{:}L_1 \cdots{:}L_m$ `WHERE` $x.P_1$ `IS NOT NULL` ...
`AND` $x.P_n$ `IS NOT NULL EXCLUSIVE` $x.U_1, \ldots, x.U_k$

While expressive and flexible, there are no technical results for PG-keys yet. Our class of gUCs was proposed in [38,40] to address the lack of constraints for data quality dimensions. They form a sub-class of PG-keys that enjoys good computational properties.

The work in [18,19,37] defines expressive graph dependencies, including FDs that compare values of properties or constants for all pairs of entities identified by a graph pattern. Their expressiveness is different from gFDs which allow multiple labels and require all properties in P to exist. While implication for FDs in [19] is *NP*-complete and they target entity resolution and fraud detection, implication for gFDs is decidable in linear time and they target normalization. Graph dependencies provide a rare opportunity to specify application semantics in graph databases [2,18,19,37,40].

Table 1: Summary of Concepts and Notation To Be Introduced

Concept	*Notation*
Basic concepts for graphs and graph constraints:	
Property graph	$G = (V, Ed, \eta, \lambda, \nu)$
Label set and Property set	L and P
Property subsets	$X, Y \subseteq P$
graph-tailored FD (gFD)	$L : P : X \to Y$
graph-tailored UC (gUC)	$L : P : X$
gUC/gFD set	Σ
Translation into relational framework:	
L:P-FD set for Σ	$\Sigma_{L:P}$ (classical FDs originating from Σ)
Relation schema of P	$R_P := P \cup \{A_0\}$ with fresh attribute A_0
Decomposition of R_P	$\mathcal{D} \subseteq \{S \mid S \subseteq R_P\}$ where $\bigcup_{S \in \mathcal{D}} = R_P$
Projection of $\Sigma_{L:P}$ onto $S \subseteq R_P$	$\Sigma_{L:P}[S] = \{X \to Y \in \Sigma^+_{L:P} \mid XY \subseteq S\}$
Normal forms for property graphs and their achievements:	
Σ in L:P-BCNF/3NF/RFNF	$(R_P, \Sigma_{L:P})$ in BCNF/3NF/RFNF
Normalization of property graphs:	
L:P-decomposition of Σ wrt $\mathcal{D}$	$\Sigma^{\ell}_{L:P}[\mathcal{D}]$ with fresh edge label ℓ
L:P-projection of G onto $S \subseteq R_P$	$G^{\ell}_{L:P}[S]$ with fresh edge label ℓ
S-equivalence between nodes v, v'	$v \equiv_S v'$ (values of v and v' match on all properties in S)
L:P-decomposition of G wrt $\mathcal{D}$	$\bigcup_{S \in \mathcal{D}} \left(G^{\ell}_{L:P}[S] / \equiv_S\right)$

In contrast to normalization, [44] propose a schema design framework for graph data that is based on minimizing access to data that will likely co-occur in query results while keeping independent concepts separate. In contrast to normalization that is based on data dependencies, [44] requires a conceptual schema as input.

Schema information is beneficial for data management, including (property) graphs [32,1]. Schemata may interact non-trivially with dependencies, such as PG-keys or gFDs. This motivates further research, including the normalization of graph schemata.

Already Codd [10] suggests online and a-posteriori enforcement, where integrity is preserved either for every update or inconsistency reported casually, respectively. Full normalization with our framework supports online enforcement while casual normalization materializes a-posteriori enforcement. Both are balanced by tailoring normalization to target dependencies.

Our work is the first to address normalization for property graphs. The class of gFDs is new, and results on the combined implication for gUC/gFDs encompass simpler findings for gUCs alone. In our work, we transfer state-of-the-art normalization for FDs from relational to graph databases.

4 Guide for Concepts and Notation

Table 1 provides a brief outline which concepts and notation we will develop throughout. In Sec. 5 we will repeat concepts for property graphs, and introduce graph-tailored constraints called gFDs and gUCs. Our approach will enable us to translate graph constraints into classical FDs in Sec. 6.1, to take advantage of the existing theory. This will enable us to define BCNF and 3NF for property graphs in Sec. 6.2. In Sec. 6.3, we will transfer achievements for BCNF and 3NF from relational databases to property graphs, and establish a framework for normalizing property graphs in Sec. 6.4.

5 Graph-Tailored Constraints

We recall basics of property graphs, including gUCs [40]. We introduce gFDs and illustrate their use. The *property graph model* [9] use the following disjoint sets: $\mathcal{O}$ for a set of objects, $\mathcal{L}$ for a finite set of labels, $\mathcal{K}$ for a set of properties, and $\mathcal{N}$ for a set of values.

A *property graph* is a quintuple $G = (V, Ed, \eta, \lambda, \nu)$ where $V \subseteq \mathcal{O}$ is a finite set of objects, called *vertices*, $Ed \subseteq \mathcal{O}$ is a finite set of objects, called *edges*, $\eta : Ed \to V \times V$ is a function assigning to each edge an ordered pair of vertices, $\lambda : V \cup Ed \to \mathcal{P}(\mathcal{L})$ is a function assigning to each object a finite set of labels, and $\nu : (V \cup Ed) \times \mathcal{K} \to \mathcal{N}$ is a partial function assigning values for properties to objects, such that the set of domain values where ν is defined is finite. If $\nu(o, A)$ is defined, we write $\nu(o, A) = \downarrow$ and $\uparrow$ otherwise. Figures 1 and 2 show examples of property graphs.

Graph-tailored UCs (gUCs) [40] cover UCs used by Neo4j [21] as a special case. The subset $V_L \subseteq V$ of vertices that carry all labels of a given set $L \subseteq \mathcal{L}$ is defined by $V_L = \{v \in V \mid L \subseteq \lambda(v)\}$.

A *graph-tailored uniqueness constraint* (or *gUC*) over $\mathcal{L}$ and $\mathcal{K}$ is an expression L:P:X where $L \subseteq \mathcal{L}$ and $X \subseteq P \subseteq \mathcal{K}$. For a property graph $G = (V, Ed, \eta, \lambda, \nu)$ over $\mathcal{O}$, $\mathcal{L}$, $\mathcal{K}$, and $\mathcal{N}$ we say G *satisfies* the gUC L:P:X over $\mathcal{L}$ and $\mathcal{K}$, denoted by $\models_G L$:P:X, iff there are no

vertices $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$, for all $A \in P$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, and for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$.

Neo4j UCs are gUCs $L{:}P{:}X$ where $L = \{\ell\}$ and $P = X = \{p\}$, that is, $\{\ell\}{:}\{p\}{:}\{p\}$. Hence, we denoted them by $\ell{:}p$. Neo4j's composite indices are covered as the special case where $L = \{\ell\}$ and $P = X$, that is, $\{\ell\}{:}X{:}X$. Hence, we denote them by $\ell{:}X$.
We will now introduce *graph-tailored FDs*. Intuitively, they express that nodes carrying a given set of labels and values on a given set of properties, the combination of values on some of those properties uniquely determine the values on some other properties.

Definition 1 A *graph-tailored functional dependency* (or *gFD*) over $\mathcal{L}$ and $\mathcal{K}$ is an expression $L{:}P{:}X \rightarrow Y$ where $L \subseteq \mathcal{L}$ and $X, Y \subseteq P \subseteq \mathcal{K}$. For a property graph $G = (V, Ed, \eta, \lambda, \nu)$ over $\mathcal{O}$, $\mathcal{L}$, $\mathcal{K}$, $\mathcal{N}$, we say G *satisfies* the gFD $L{:}P{:}X \rightarrow Y$ over $\mathcal{L}$ and $\mathcal{K}$, denoted by $\models_G L{:}P{:}X \rightarrow Y$, iff there are no vertices $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$, for all $A \in P$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$ and for some $A \in Y$, $\nu(v_1, A) \neq \nu(v_2, A)$. □

The concept of gFDs provides users with the flexibility to layer rules for nodes with different sets of labels. While $L{:}P{:}X \rightarrow Y$ applies to all nodes when $L = \emptyset$, adding new labels to L allows the user to declare additional rules that only apply to nodes that carry all labels in L. Secondly, the property set P addresses completeness requirements of applications on the properties that nodes may have. Unless a node exhibits values on all properties in P, it does not need to comply with the FD $X \rightarrow Y$. Thirdly, X and Y are subsets of P. This choice is guided by the principle that missing properties should not affect the validity of a business rule. If completeness requirements are not available, we may simply use the gFD $L{:}XY{:}X \rightarrow Y$. Most gFDs that express meaningful rules will have this format, and they imply weaker gFDs $L{:}P{:}X \rightarrow Y$ where P contains XY. This has multiple benefits as illustrated later, such as tailoring normalization to different requirements, discovering gFDs and sources of inconsistent data.

Over schema R, the UC X is expressed by the FD $X \rightarrow R$. Indeed, relations are sets of records and no two different records can have values matching on all fields of R. This observation is significant for normalization which transforms the underlying schema until every FD that may cause data redundancy has been transformed into a key which cannot cause data redundancy.

This situation is different in property graphs that permit duplication. Indeed, no gUC $L{:}P{:}X$ can be expressed by any gFD since we can always have two different nodes with label set L and values matching on all properties in P. While this graph satisfies the gFD $L{:}P{:}X \rightarrow P$, it does not satisfy the gUC $L{:}P{:}X$. For example, graph G_0 in Figure 1 satisfies the gFD $\mathit{Event}{:}\mathit{NCV}{:}\mathit{NC} \rightarrow V$ but not the gUC $\mathit{Event}{:}\mathit{NCV}{:}\mathit{NC}$. As gFDs cause data redundancy, gUCs prohibit data redundancy, and gFDs cannot express gUCs, we need to study gUCs and gFDs together.

6 Foundations for Normalization

We will first establish axiomatic and algorithmic characterizations of the implication problem for gUCs and gFDs. These provide the necessary foundations for defining effective normal forms subsequently.

6.1 Reasoning

We formally define the implication problem for gUCs and gFDs over property graphs, illustrate it on examples, establish axiomatic and algorithmic solutions.

Let $\Sigma \cup \{\varphi\}$ denote a set of constraints over $\mathcal{L}$ and $\mathcal{K}$ from class $\mathcal{C}$. The *implication problem* for $\mathcal{C}$ is to decide, given set $\Sigma \cup \{\varphi\}$ of constraints from $\mathcal{C}$, whether Σ implies φ. In fact, Σ *implies* φ, denoted by $\Sigma \models \varphi$, if and only if every property graph G over $\mathcal{O}$, $\mathcal{L}$ and $\mathcal{K}$ that satisfies all constraints in Σ also satisfies φ.

Deciding whether φ is implied by Σ is fundamental for node integrity management on property graphs. If φ is implied by Σ, then φ is already specified implicitly by Σ. Otherwise, failure to specify φ explicitly may cause integrity faults that go undetected. Since FDs form a special case of gFDs, the implication problem for gFDs is hard for *PTIME* [6,12].

6.2 Axiomatic Characterizations

We will establish an axiomatization for the combined class $\mathcal{C}$ of gUCs and gFDs. The set $\Sigma^*_{\mathcal{C}} = \{\varphi \in \mathcal{C} \mid \Sigma \models \varphi\}$ denotes the *semantic closure of* Σ. We aim at computing $\Sigma^*_{\mathcal{C}}$ by applying *inference rules* of the form $\frac{\text{premise}}{\text{conclusion}}$. For a set $\mathfrak{R}$ of inference rules let $\Sigma \vdash_{\mathfrak{R}} \varphi$ denote the *inference* of φ from Σ by $\mathfrak{R}$. That is, there is some sequence $\sigma_1, \ldots, \sigma_n$ such that $\sigma_n = \varphi$ and every σ_i belongs to Σ or is the conclusion of applying an inference rule in $\mathfrak{R}$ to some premises in $\{\sigma_1, \ldots, \sigma_{i-1}\}$. Let $\Sigma^+_{\mathfrak{R}} = \{\varphi \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$ be the *syntactic closure* of Σ under inferences by $\mathfrak{R}$. $\mathfrak{R}$ is *sound* (*complete*) if for every set Σ of constraints from $\mathcal{C}$ we have $\Sigma^+_{\mathfrak{R}} \subseteq \Sigma^*_{\mathcal{C}}$ ($\Sigma^*_{\mathcal{C}} \subseteq \Sigma^+_{\mathfrak{R}}$). The (finite) set $\mathfrak{R}$ is a (finite) *axiomatization* if $\mathfrak{R}$ is both sound and complete.

Table 2: Axiomatization $\mathfrak{E} = \{\mathcal{R}, \mathcal{E}, \mathcal{T}, \mathcal{A}, \mathcal{W}, \mathcal{P}\}$ of gUC/FDs

$\dfrac{}{L{:}P{:}XY \to X}$ (reflexivity, $\mathcal{R}$)	$\dfrac{L{:}P{:}X \to Y}{L{:}P{:}X \to XY}$ (extension, $\mathcal{E}$)
$\dfrac{L{:}P{:}X}{LL'{:}PP'{:}XX'}$ (augmentation, $\mathcal{A}$)	$\dfrac{L{:}P{:}X \to Y \quad L'{:}P'{:}Y \to Z}{LL'{:}PP'{:}X \to Z}$ (transitivity, $\mathcal{T}$)
$\dfrac{L{:}P{:}X}{L{:}P{:}X \to P}$ (weakening, $\mathcal{W}$)	$\dfrac{L{:}P{:}X \to Y \quad L{:}P{:}XY}{L{:}P{:}X}$ (pullback, $\mathcal{P}$)

We assume the rules of $\mathfrak{E}$ in Table 2 contain well-formed gUCs and gFDs. As example, for the rule $\mathcal{A}$ with $L{:}P{:}X$ and $LL'{:}PP'{:}XX'$ we assume $X \subseteq P$ and $XX' \subseteq PP'$. $\mathcal{A}$ by itself is sound and complete for the implication of gUCs.

We will now show that $\mathfrak{E}$ from Table 2 forms indeed a sound and complete set of inference rules for the implication of gUCs and gFDs over property graphs. First, we establish soundness of the rules, meaning that any consecutive applications of the inference rules can never result in gUC or gFD not implied by the input set.

Lemma 1 *The inference rules in Table 2 are sound for the implication of gUCs and gFDs in property graphs.*

Proof Soundness of the augmentation rule $\mathcal{A}$ was already proven in [40]. We will now prove soundness of the remaining rules.

For soundness of the reflexivity axiom $\mathcal{R}$ we note that for every property graph G with two vertices that carry all labels in L and have values for all properties in P, values on properties in X match whenever values on all properties in $X \cup Y$ match.

For soundness of the extension rule $\mathcal{E}$ we use contra-position and assume there is a property graph G that violates the gFD $L : P : X \to XY$. We need to show that G also violates the gFD $L : P : X \to Y$. Since G violates $L : P : X \to XY$, there are vertices $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$ and for all $A \in P$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$, and for some $A \in XY$, $\nu(v_1, A) \neq \nu(v_2, A)$. Consequently, there is some $A \in Y$ such that $\nu(v_1, A) \neq \nu(v_2, A)$. We have just shown that G also violates the gFD $L : P : X \to Y$.

For soundness of the transitivity rule $\mathcal{T}$ we use contra-position and assume there is a property graph G that violates the gFD $LL' : PP' : X \to Z$. We need to show that G also violates the gFD $L : P : X \to Y$ or the gFD $L' : P' : Y \to Z$. Since G violates $LL' : PP' : X \to Z$, there are vertices $v_1, v_2 \in V_{LL'}$ such that $v_1 \neq v_2$ and for all $A \in PP'$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$, and for some $A \in Z$, $\nu(v_1, A) \neq \nu(v_2, A)$. If G violates the gFD $L : P : X \to Y$, then there is nothing more to show. Hence, we assume that G satisfies the gFD $L : P : X \to Y$. Consequently, we know that for all $A \in Y$, $\nu(v_1, A) = \nu(v_2, A)$. However, under this assumption we can see directly that G violates the gFD $L' : P' : Y \to Z$.

For soundness of the weakening rule $\mathcal{W}$ we use contra-position and assume there is a property graph G that violates the gFD $L : P : X \to Y$. We need to show that G also violates the gUC $L : P : X$. Since G violates $L : P : X \to Y$, there are vertices $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$ and for all $A \in P$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$, and for some $A \in Y$, $\nu(v_1, A) \neq \nu(v_2, A)$. Consequently, we have just shown that G also violates the gUC $L : P : X$.

For soundness of the pullback rule $\mathcal{P}$ we use contra-position and assume there is a property graph G that violates the gUC $L : P : X$. We need to show that G also violates the gFD $L : P : X \to Y$ or the gUC $L : P : XY$. Since G violates $L : P : X$, there are vertices $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$ and for all $A \in P$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, and for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$. If G violates the gUC $L : P : XY$, then there is nothing more to show. Hence, we assume that G satisfies the gUC $L : P : XY$. Consequently, we know for some $A \in Y$, $\nu(v_1, A) \neq \nu(v_2, A)$. However, under this assumption G violates the gFD $L : P : X \to Y$.

Lemma 2 *The following rules are sound*

$$\frac{L{:}P{:}X \to YZ}{L{:}P{:}X \to Y} \text{ (decomposition, } \mathcal{D}) \qquad \frac{L{:}P{:}X \to Y \quad L{:}P{:}X \to Z}{L{:}P{:}X \to YZ} \text{ (union, } \mathcal{U})$$

$$\frac{L{:}P{:}X \to Y}{LL'{:}PP'{:}X \to Y} \text{ (increase, } \mathcal{I})$$

for the implication of gFDs.

Proof The *decomposition*-rule $\mathcal{D}$ can be inferred from the rules in $\mathfrak{E}$ as follows.

$$\dfrac{L : P : X \to YZ \quad \overline{(\mathcal{R}) \quad L : P : YZ \to Y}}{(\mathcal{T}) \qquad L : P : X \to Y}.$$

For the *union*-rule $\mathcal{U}$, we first use the rules in $\mathfrak{E}$ to infer $L{:}P{:}XY \to YZ$ from $L{:}P{:}X \to Z$ as follows

$$\dfrac{\dfrac{\dfrac{\overline{(\mathcal{R}) \quad L{:}P{:}XY \to X} \quad L{:}P{:}X \to Z}{(\mathcal{T}) \qquad L{:}P{:}XY \to Z}}{(\mathcal{E}) \quad L{:}P{:}XY \to XYZ} \qquad \overline{(\mathcal{R}) \quad L{:}P{:}XYZ \to YZ}}{(\mathcal{T}) \qquad\qquad L{:}P{:}XY \to YZ}$$

and then infer $L{:}P{:}X \to YZ$ from $L{:}P{:}X \to Y$ and $L{:}P{:}XY \to YZ$ as follows.

$$\dfrac{\dfrac{L{:}P{:}X \to Y}{(\mathcal{E})\quad L{:}P{:}X \to XY} \quad \overline{L{:}P{:}XY \to YZ}}{(\mathcal{T})\qquad\qquad L{:}P{:}X \to YZ}$$

The *increase*-rule $\mathcal{I}$ can be inferred from the rules in $\mathfrak{E}$ as follows:

$$\dfrac{L : P : X \to Y \quad \overline{(\mathcal{R})\quad L' : P' : Y \to Y}}{(\mathcal{T})\qquad\qquad LL' : PP' : X \to Y}$$

This completes the proof.

Theorem 1 *The set $\mathfrak{E}$ forms a finite axiomatization for the implication of gUCs and gFDs.*

Proof The soundness of $\mathfrak{E}$ was established in Lemma 1.

We show the completeness of $\mathfrak{E}$ by contra-position. Let $\Sigma \cup \{\varphi\}$ denote a set of gUCs and gFDs over $\mathcal{L}$ and $\mathcal{K}$ such that $\varphi \notin \Sigma^+_{\mathfrak{E}}$. We will show that Σ does not imply φ by defining a property graph G that satisfies all gUCs and gFDs in Σ and violates φ. We distinguish two cases where φ denotes either the gUC $L{:}P{:}X$ or the gFD $L{:}P{:}X \to Y$, respectively. In either case, let

$$X^+_{\Sigma_{L:P}} = \{A \in P \mid \Sigma \vdash_{\mathfrak{E}} L{:}P{:}X \to A\}$$

denote the *property set closure* of X for $L{:}P$ and Σ. The soundness of the *union*-rule $\mathcal{U}$, established in Lemma 2, shows that $L{:}P{:}X \to X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$ holds.

We define the property graph $G = (V, Ed, \eta, \lambda, \nu)$ as follows: $V = \{v_1, v_2\}$, $Ed = \emptyset$. There is nothing to define for η, $\lambda(v_1) = L = \lambda(v_2)$, for all $A \in X^+_{\Sigma_{L:P}}$ we define $\nu(v_1, A) = 0 = \nu(v_2, A)$, and for all $A \in E - X^+_{\Sigma_{L:P}}$ we define $\nu(v_1, A) = 0$ and $\nu(v_2, A) = 1$.

Case 1. As $X \subseteq X^+_{\Sigma_{L:P}}$, it follows from the construction of G that G violates $L : P : X$. It therefore remains to show in this case that G satisfies every gUC and every gFD in Σ. Let σ denote such an element of Σ.

Case 1.a) Here, σ denotes the gUC $L'{:}P'{:}X' \in \Sigma$. Assume, to the contrary, that G violates σ. It follows from the construction of G that $L' \subseteq L$, $P' \subseteq P$ and $X' \subseteq X^+_{\Sigma_{L:P}}$ hold. Applying the *extension*-rule $\mathcal{E}$ to $L'{:}P'{:}X' \in \Sigma$ gives us $L{:}P{:}X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$. Since $L{:}P{:}X \to X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$ holds, we can apply the *pull-back*-rule $\mathcal{P}$ to infer $L{:}P{:}X \in \Sigma^+_{\mathfrak{E}}$. This is a contradiction to our assumption that $\varphi = L{:}P{:}X \notin \Sigma^+_{\mathfrak{E}}$. Consequently, G must not violate σ, and we conclude that G satisfies σ in this case.

Case 1.b) Here, σ denotes the gFD $L'{:}P'{:}X' \to Y' \in \Sigma$. Assume again, to the contrary, that G violates σ. It follows from the construction of G that $L' \subseteq L$, $P' \subseteq P$, $X' \subseteq X^+_{\Sigma_{L:P}}$, and $Y' \cap (P - X^+_{\Sigma_{L:P}}) \neq \emptyset$ all hold. From $L{:}P{:}X \to X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$ and $X' \subseteq X^+_{\Sigma_{L:P}}$ we infer $L{:}P{:}X \to X' \in \Sigma^+_{\mathfrak{E}}$ by applying the *decompose*-rule $\mathcal{D}$ from Lemma 2. Applying the *transitivity*-rule $\mathcal{T}$ to $L{:}P{:}X \to X' \in \Sigma^+_{\mathfrak{E}}$ and $L'{:}P'{:}X' \to Y' \in \Sigma$, we infer $LL'{:}PP'{:}X \to Y' \in \Sigma^+_{\mathfrak{E}}$. Since $L' \subseteq L$, $P' \subseteq P$, we actually have $L{:}P{:}X \to Y' \in \Sigma^+_{\mathfrak{E}}$. According to the definition of the property set closure, we must then have that $Y' \subseteq X^+_{\Sigma_{L:P}}$. This contradicts $Y' \cap (P - X^+_{\Sigma_{L:P}}) \neq \emptyset$. Consequently, G must not violate σ, and we conclude that G satisfies σ.

Case 2. If $Y \subseteq X^+_{\Sigma_{L:P}}$, then $L{:}P{:}X \to Y \in \Sigma^+_{\mathfrak{E}}$ contrary to our assumption. Hence, $Y \cap (P - X^+_{\Sigma_{L:P}}) \neq \emptyset$. Since $X \subseteq X^+_{\Sigma_{L:P}}$, $Y \cap (P - X^+_{\Sigma_{L:P}}) \neq \emptyset$, it follows from the construction of G that G violates $L{:}P{:}X \to Y$. Hence, we need to show that G satisfies every gUC and every gFD in Σ. Let σ denote such an element of Σ.

Case 2.a) Here, σ denotes the gUC $L'{:}P'{:}X' \in \Sigma$. Assume, to the contrary, that G violates σ. The construction of G entails that $L' \subseteq L$, $P' \subseteq P$ and $X' \subseteq X^+_{\Sigma_{L:P}}$ hold. Applying the *extension*-rule $\mathcal{E}$ to $L'{:}P'{:}X' \in \Sigma$ gives us $L{:}P{:}X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$. Since $L{:}P{:}X \to X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$ holds, we can apply the *pullback*-rule $\mathcal{P}$ to infer $L{:}P{:}X \in \Sigma^+_{\mathfrak{E}}$. From the *weakening*-rule $\mathcal{W}$ we can then infer $L{:}P{:}X \to P \in \Sigma^+_{\mathfrak{E}}$, and since $L{:}P{:}P \to Y \in \Sigma^+_{\mathfrak{E}}$, we can apply the *transitivity*-rule $\mathcal{T}$ to $L{:}P{:}X \to P \in \Sigma^+_{\mathfrak{E}}$ and $L{:}P{:}P \to Y \in \Sigma^+_{\mathfrak{E}}$ to infer the contradiction that $L{:}P{:}X \to Y \in \Sigma^+_{\mathfrak{E}}$. Hence, our assumption must have been wrong, and G satisfies σ in this case.

Case 2.b) Here, σ denotes the gFD $L'{:}P'{:}X' \to Y' \in \Sigma$. Assume again, to the contrary, that G violates σ. The construction of G entails that $L' \subseteq L$, $P' \subseteq P$, $X' \subseteq X^+_{\Sigma_{L:P}}$, and $Y' \cap (P - X^+_{\Sigma_{L:P}}) \neq \emptyset$ all hold. From $L{:}P{:}X \to X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$ and $X' \subseteq X^+_{\Sigma_{L:P}}$ we infer $L{:}P{:}X \to X' \in \Sigma^+_{\mathfrak{E}}$ by applying the *decomposition*-rule $\mathcal{D}$ from Lemma 2. Applying the *transitivity*-rule $\mathcal{T}$ to $L{:}P{:}X \to X' \in \Sigma^+_{\mathfrak{E}}$ and $L'{:}P'{:}X' \to Y' \in \Sigma$, we infer $LL'{:}PP'{:}X \to Y' \in \Sigma^+_{\mathfrak{E}}$. Since $L' \subseteq L$ and $P' \subseteq P$, we actually have $L{:}P{:}X \to Y' \in \Sigma^+_{\mathfrak{E}}$. According to the definition of the property set closure, we must then have that $Y' \subseteq X^+_{\Sigma_{L:P}}$. This contradicts $Y' \cap (P - X^+_{\Sigma_{L:P}}) \neq \emptyset$. Consequently, G must not violate σ, and we conclude that G satisfies σ. This completes the proof. □

We illustrate inferencing on our running example.

Example 1 Let Σ contain $\phi'_1 = \mathit{Event}{:}\mathit{CTV}{:}\mathit{CT} \to V$ and $\phi'_2 = \mathit{Event}{:}\mathit{NTV}{:}\mathit{VT} \to N$. Applying $(\mathcal{E})$ to ϕ'_1 gives us $\phi'_3 = \mathit{Event}{:}\mathit{CTV}{:}\mathit{CT} \to \mathit{CTV}$. $(\mathcal{R})$ gives us $\phi'_4 = \mathit{Event}{:}\mathit{CTV}{:}\mathit{CTV} \to \mathit{VT}$, and applying $(\mathcal{T})$ to ϕ'_4 and ϕ'_2 gives us $\phi'_5 = \mathit{Event}{:}\mathit{CTVN}{:}\mathit{CTV} \to N$. Finally, applying $(\mathcal{T})$ to ϕ'_3 and ϕ'_5 gives us $\varphi = \mathit{Event}{:}\mathit{CNTV}{:}\mathit{CT} \to N$. Hence, Σ implies φ. Note the subtlety in reasoning with the requirements for properties. As we will see below, Σ does not imply $\varphi' = \mathit{Event}{:}\mathit{CNT}{:}\mathit{CT} \to N$.

$\{\mathcal{R}, \mathcal{E}, \mathcal{T}\}$ forms an axiomatization for gFDs, a natural extension of the Armstrong axioms [4]. We will denote the latter by $\mathfrak{A}$.

6.3 Algorithmic Characterization

We use our axiomatization $\mathfrak{E}$ to establish an algorithm that decides implication efficiently.

For a set Σ of gUCs and gFDs, $L \subseteq \mathcal{L}$ and $P \subseteq \mathcal{K}$, we define the following set of FDs over the relation schema $R_P = P \cup \{A_0\}$:

$$\Sigma_{L:P} = \{X \to R_P \mid \exists L'{:}P'{:}X \in \Sigma \wedge L' \subseteq L \wedge P' \subseteq P\} \cup \{X \to Y \mid \exists L'{:}P'{:}X \to Y \in \Sigma \wedge L' \subseteq L \wedge P' \subseteq P\} .$$

Here, $A_0 \notin P$ is a fresh property not occurring elsewhere. A_0 is only required in R_P when there is no gUC $L'{:}P'{:}X \in \Sigma$ with $L' \subseteq L$ and $P' \subseteq P$. That is, if $L'{:}P'{:}X \in \Sigma$ with $L' \subseteq L$ and $P' \subseteq P$, then $R_P := P$ is sufficient. Next we reduce implication of gUCs and gFDs over property graphs to the implication of FDs over relation schemata.

Theorem 2 *For every set $\Sigma \cup \{L{:}P{:}X, L{:}P{:}X \to Y\}$ over $\mathcal{L}$ and $\mathcal{K}$ and $R_P = P \cup \{A_0\}$ with a fresh attribute $A_0 \notin P$, the following holds*

1. *$\Sigma \models L{:}P{:}X \to Y$ if and only if $\Sigma_{L:P} \models X \to Y$*
2. *$\Sigma \models L{:}P{:}X$ if and only if $\Sigma_{L:P} \models X \to R_P$*

Proof **1.** Suppose $\Sigma \not\models L{:}P{:}X \to Y$. Then there is a property graph $G = (V, Ed, \eta, \lambda, \nu)$ that satisfies all gUCs and gFDs in Σ but violates $L{:}P{:}X \to Y$. In particular, there are two vertices $v, v' \in V$ whose label sets include all labels in L, both vertices carry all properties in P, the values $\nu(v, A)$ and $\nu(v', A)$ are matching on all properties $A \in X$ but there is some property $B \in Y$ on which $\nu(v, B)$ and $\nu(v', B)$ are not matching. We define the following two-tuple relation $r := \{t_v, t_{v'}\}$ over $R_P = P \cup \{A_0\}$ as follows: $t_v(A) := \nu(v, A)$ for all $A \in P$ and $t_v(A_0) := 0$, and $t_{v'}(A) := \nu(v', A)$ for all $A \in P$ and $t_{v'}(A_0) := 1$. It follows that r violates $X \to Y$ since $XY \subseteq P$. It remains to show that r satisfies all $X' \to R_P$ and $X' \to Y' \in \Sigma_{L:P}$. For $X' \to R_P \in \Sigma_{L:P}$ it follows that $L'{:}P'{:}X' \in \Sigma$ for some $L' \subseteq L$ and $P' \subseteq P$. Since the property graph G satisfies $L'{:}P'{:}X'$ we must have $t_v(X') \neq t_{v'}(X')$. Consequently, r satisfies $X' \to R_P$. Since G satisfies $L'{:}P'{:}X' \to Y'$ the following holds: if $t_v(X') = t_{v'}(X')$, then $t_v(Y') = t_{v'}(Y')$. Consequently, r satisfies $X' \to Y'$.

Suppose now that $\Sigma_{L:P} \models X \to Y$ does not hold. Then there is a two-tuple relation $r = \{t, t'\}$ over R_P that satisfies $\Sigma_{L:P}$ and violates $X \to Y$. We define a property graph $G = (V, Ed, \eta, \lambda, \nu)$ where $V = \{v_t, v_{t'}\}$, $Ed = \emptyset$, $\lambda(v_t) = L = \lambda(v_{t'})$. For every $A \in P$ we define $\nu(v_t, A) := t(A)$ and $\nu(v_{t'}, A) := t'(A)$. Clearly, G violates $L{:}P{:}X \to Y$. It remains to show that G satisfies all gUCs $L'{:}P'{:}X'$ and all gFDs $L'{:}P'{:}X' \to Y'$ in Σ. In case $L' \subseteq L$ and $P' \subseteq P$, it follows that $X' \to R_P$ and $X' \to Y' \in \Sigma_{L:P}$. Hence, since r satisfies $X' \to R_P$ and $X' \to Y'$, G satisfies $L'{:}P'{:}X'$ and $L'{:}P'{:}X' \to Y'$. In particular, r contains two distinct tuples, so there is some attribute $B \in R_P$ such that $t(B) \neq t'(B)$. Consequently, $\nu(v_t, A) \neq \nu(v_{t'}, A)$ for some $A \in X'$. In the other case we have $L' \not\subseteq L$ or $P' \not\subseteq P$. In this case, by definition, any gUC $L'{:}P'{:}X'$ and any gFD $L'{:}P'{:}X' \to Y'$ would be satisfied since there are no nodes that apply to them. Hence, we have shown that $\Sigma \not\models L{:}P{:}X \to Y$.

2. Suppose $\Sigma \not\models L{:}P{:}X$. Then there is a property graph $G = (V, Ed, \eta, \lambda, \nu)$ that satisfies all gUCs and gFDs in Σ but violates $L{:}P{:}X$. In particular, there are two vertices $v, v' \in V$ whose label sets include all labels in L, both vertices v, v' carry all properties in P, and for all properties $A \in X$, the values $\nu(v, A)$ and $\nu(v', A)$ are matching. We define the following two-tuple relation $r := \{t_v, t_{v'}\}$ over $R_P = P \cup \{A_0\}$: $t_v(A) := \nu(v, A)$ for all $A \in P$ and $t_v(A_0) := 0$, and $t_{v'}(A) := \nu(v', A)$ for all $A \in P$ and $t_{v'}(A_0) := 1$. It follows that r violates $X \to R_P$ since $t_v(X) = t_{v'}(X)$ and $t_v(A_0) \neq t_{v'}(A_0)$. It remains to show that r satisfies all $X' \to R_P$ and $X' \to Y' \in \Sigma_{L:P}$. For $X' \to R_P \in \Sigma_{L:P}$ it follows that $L'{:}P'{:}X' \in \Sigma$ for some $L' \subseteq L$ and $P' \subseteq P$. Since the property graph G satisfies $L'{:}P'{:}X'$ it must be the case that $t_v(X') \neq t_{v'}(X')$. Consequently, r satisfies $X' \to R_P$. Since G satisfies $L'{:}P'{:}X' \to Y$ the following holds: if $t_v(X') = t_{v'}(X')$, then $t_v(Y') = t_{v'}(Y')$. Consequently, r satisfies $X' \to Y'$.

Suppose $\Sigma_{L:P} \models X \to R_P$ does not hold. Then there is a two-tuple relation $r = \{t, t'\}$ over R_P that satisfies $\Sigma_{L:P}$ and violates $X \to R_P$. We define a property graph $G = (V, Ed, \eta, \lambda, \nu)$ with $V = \{v_t, v_{t'}\}$, $Ed = \emptyset$, and $\lambda(v_t) = L = \lambda(v_{t'})$. For every $A \in P$ we define $\nu(v_t, A) := t(A)$ and $\nu(v_{t'}, A) := t'(A)$. Since r violates $X \to R_P$, $t(X) = t'(X)$. Hence, G violates $L : P : X$. It remains to show that G satisfies all gUCs $L'{:}P'{:}X'$ and all gFDs $L'{:}P'{:}X' \to Y'$ in Σ. In case $L' \subseteq L$ and $P' \subseteq P$, it follows that $X' \to R_P$ and $X' \to Y' \in \Sigma_{L:P}$. Hence, since r satisfies $X' \to R_P$ and $X' \to Y'$, G satisfies $L'{:}P'{:}X'$ and $L'{:}P'{:}X' \to Y'$. In particular, r contains two distinct tuples, so there is some attribute $B \in R_P$ such that $t(B) \neq t'(B)$ holds. Consequently, we must have $\nu(v_t, A) \neq \nu(v_{t'}, A)$ for some $A \in X'$. In the other case, $L' \not\subseteq L$ or $P' \not\subseteq P$. Then, by definition, every gUC $L'{:}P'{:}X'$ and every gFD $L'{:}P'{:}X' \to Y'$ would be satisfied since no nodes apply to them. Hence, we have shown that $\Sigma \not\models L{:}P{:}X$. □

Algorithm 1 Implication of gUCs and gFDs

Require: Set $\Sigma \cup \{\varphi\}$ of gUC/FDs,
$\varphi = L{:}P{:}X$ or $\varphi = L{:}P{:}X \to Y$
Ensure: *TRUE*, if $\Sigma \models \varphi$, and *FALSE*, otherwise
1: Compute $X^+_{\Sigma_{L:P}}$ by linear-time attribute set closure for FDs [6]
2: **if** $\varphi = L{:}P{:}X$ and $X^+_{\Sigma_{L:P}} = R_P$ **then**
3: **return** *TRUE*
4: **else if** $\varphi = L{:}P{:}X \to Y$ and $Y \subseteq X^+_{\Sigma_{L:P}}$ **then**
5: **return** *TRUE*
6: **else**
7: **return** *FALSE*

Theorem 2 motivates Algorithm 1, which computes $X^+_{\Sigma_{L:P}}$ of X for $\Sigma_{L:P}$ over R_P using the classical algorithm [6]. The decision branches in Algorithm 1 reflect Theorem 2. Hence, *PTIME*-completeness is inherited from the classical case [12,6].

Corollary 1 *Algorithm 1 decides the* PTIME*-complete implication problem for gUCs and gFDs in linear input time.* □

We illustrate the algorithm on our running example.

Example 2 For $\Sigma = \{\phi'_1, \phi'_2\}$ and φ' from Example 1, Algorithm 1 returns *FALSE* as $\Sigma_{Event:CNT} = \emptyset$ and $N \notin (CT)^+_{\Sigma_{Event:CNT}} = CT$. As $\Sigma_{Event:CNTV} = \{CT \to V, VT \to N\}$, Algorithm 1 returns *TUE* for φ since $N \in (CT)^+_{\Sigma_{Event:CNTV}} = CTVN$.

7 Normal Forms for Property Graphs

We define BCNF and 3NF for property graphs. We will first explain our approach, describe our proposals, and present results on their achievements.

7.1 Approach

Since property graphs have no schema, it is challenging to define normal forms for graph data. We overcome this challenge by using our graph-tailored constraints. Since applications target graph objects based on their labels and properties, we view these features as requirements: The application targets only nodes that exhibit a given set L of labels and a given set P of properties. With that approach, we then normalize that part of the graph which meets the targets. Hence, normalization becomes flexible and driven by application requirements.

7.2 Boyce-Codd Normal Form

Classical BCNF casts a syntactic definition that prevents any possible occurrence of redundant data values by stipulating that every FD, which could cause redundancy, is actually a key dependency (unable to ever cause any redundancy). We will now define BCNF for gUCs and gFDs, aimed at preventing redundant property values on graphs that satisfy the constraints.

Definition 2 ($L{:}P$-BCNF) Let Σ denote a set of gUCs and gFDs over $\mathcal{L}$ and $\mathcal{K}$. For sets $L \subseteq \mathcal{L}$ and $P \subseteq \mathcal{K}$, we say that Σ is in $L{:}P$-Boyce-Codd Normal Form ($L{:}P$-BCNF) if and only if for every gFD $L{:}P{:}X \to Y \in \Sigma^+_{\mathfrak{E}}$ it is true that $Y \subseteq X$ or $L{:}P{:}X \in \Sigma^+_{\mathfrak{E}}$. □

We illustrate the definition on our running example.

Example 3 Property graph G_0 from Figure 1 satisfies $\Sigma = \{\sigma_1, \ldots, \sigma_5\}$. Indeed, Σ is in *Event*:CT-BCNF, but neither in *Event*:NC-, *Event*:NCT-, *Event*:NTV-, nor *Event*:$NCTV$-BCNF. In contrast, property graph G_n from Figure 2 satisfies $\Sigma' = \{\sigma'_1, \ldots, \sigma'_5\}$ from Section 2, which is in *Evt_Mgt*:NC-BCNF, *Evt_Comp*:CVT-BCNF, and *Evt_Detail*:NVT-BCNF.

For any label set L and property set P, we can check whether Σ is in $L{:}P$-BCNF by checking if $(R_P, \Sigma_{L:P})$ is in BCNF. That is, our BCNF definition is tailored to label and property sets of graphs.

Theorem 3 *For every label set L and property set P, Σ is in $L{:}P$-BCNF ifif $(R_P, \Sigma_{L:P})$ is in BCNF.*

Proof By Theorem 2(1) we have $L{:}P{:}X \to Y \in \Sigma^+_{\mathfrak{E}}$ if and only if $X \to Y \in (\Sigma_{L:P})^+_{\mathfrak{A}}$, and by Theorem 2(2) we have $L{:}P{:}X \in \Sigma^+_{\mathfrak{E}}$ if and only if $X \to R_P \in (\Sigma_{L:P})^+_{\mathfrak{A}}$. Hence, Σ is in $L{:}P$-BCNF iff for every FD $X \to Y \in (\Sigma_{L:P})^+_{\mathfrak{A}}$ it is true that $Y \subseteq X$ or $X \to R_P \in (\Sigma_{L:P})^+_{\mathfrak{A}}$. This, however, means that $(R_P, \Sigma_{L:P})$ is in BCNF. □

Following Example 3, Σ is not in *Event*:NTV-BCNF as $R_P = NTVA_0$ is not in BCNF for $\Sigma_{Event:NTV} = \{VT \to N, NT \to V\}$, since $A_0 \in R_P$ and $VT \to R_P \notin \Sigma^+_{Event:NTV}$. Σ is not in *Event*:$NCTV$-BCNF as $R_P = NCTV$ is not in BCNF for $\Sigma_{Event:NCTV} = \{N \to C, CT \to NV, NT \to V, VT \to N\}$, since $N \to R_P \notin \Sigma^+_{Event:NCTV}$.

The condition for Σ to be in $L{:}P$-BCNF is independent of how Σ is represented. That is, for every gUC/FD set Θ where $\Sigma^+_{\mathfrak{E}} = \Theta^+_{\mathfrak{E}}$, Σ is in $L{:}P$-BCNF iff Θ is in $L{:}P$-BCNF. Indeed, Definition 2 checks all gFDs in $\Sigma^+_{\mathfrak{E}}$, which may be exponential in Σ. We can show it suffices to check Σ itself, so testing $L{:}P$-BCNF is efficient.

Theorem 4 *Σ is in $L{:}P$-BCNF iff for every gFD $L'{:}P'{:}X \to Y \in \Sigma$ where $L' \subseteq L$ and $P' \subseteq P$, $Y \subseteq X$ or $L{:}P{:}X \in \Sigma^+_{\mathfrak{E}}$.*

Proof By Theorem 3 it follows that Σ in $L{:}P$-BCNF if and only if for every FD $X \to Y \in (\Sigma_{L:P})^+_{\mathfrak{A}}$ it is true that $Y \subseteq X$ or $X \to R_P \in (\Sigma_{L:P})^+_{\mathfrak{A}}$. Since $(R_P, \Sigma_{L:P})$ is in BCNF iff for every FD $X \to Y \in \Sigma_{L:P}$ it is true that $Y \subseteq X$ or $X \to R_P \in (\Sigma_{L:P})^+_{\mathfrak{A}}$, and $X \to Y \in \Sigma_{L:P}$ iff there is some $L'{:}P'{:}X \to Y \in \Sigma$ such that $L' \subseteq L$ and $P' \subseteq P$, we conclude that Σ in $L{:}P$-BCNF iff for every gFD $L'{:}P'{:}X \to Y \in \Sigma$ where $L' \subseteq L$ and $P' \subseteq P$ it is true that $Y \subseteq X$ or $L{:}P{:}X \in \Sigma^+_{\mathfrak{C}}$. □

Theorem 4 allows us to check in time quadratic in $|\Sigma|$ whether Σ is in $L{:}P$-BCNF. We simply test if $X^+_{\Sigma_{L:P}} = R_P$ for every $L'{:}P'{:}X \to Y \in \Sigma$ where $L' \subseteq L$, $P' \subseteq P$ and $Y \not\subseteq X$. We compute $X^+_{\Sigma_{L:P}}$ in time linear in $|\Sigma_{L:P} \cup \{X\}|$ using the classical algorithm [6].

Corollary 2 *The condition whether Σ is in $L{:}P$-BCNF can be checked in time quadratic in $|\Sigma|$.* □

7.3 Third Normal Form

While a lossless BCNF decomposition is always achievable, some FDs may be lost. These require a join of schemata resulting from the decomposition before their validity can be tested. As this is expensive, dependency-preservation is another goal of normalization. Current state-of-the-art finds a lossless, dependency-preserving decomposition into 3NF, which is in BCNF whenever possible. We target this result for property graphs.

Towards defining 3NF, we say property $A \in P$ is *$L{:}P$-prime for Σ* iff there is some $L{:}P{:}X \in \Sigma^+_{\mathfrak{C}}$ such that $A \in X$, and for all proper subsets $Y \subset X$, $L{:}P{:}Y \notin \Sigma^+_{\mathfrak{C}}$. Hence, A is contained in some minimal key for $\Sigma_{L:P}$. If no key exists, there is no prime property.

Definition 3 Let Σ be a set of gUCs and gFDs over $\mathcal{L}$ and $\mathcal{K}$. For $L \subseteq \mathcal{L}$ and $P \subseteq \mathcal{K}$, Σ is in *$L{:}P$-Third Normal Form ($L{:}P$-3NF)* if and only if for every gFD $L{:}P{:}X \to Y \in \Sigma^+_{\mathfrak{C}}$ it is true that $Y \subseteq X$ or $L{:}P{:}X \in \Sigma^+_{\mathfrak{C}}$ or every property in $Y - X$ is $L{:}P$-prime. □

Example 3 showed that Σ is not in *Event*:$NCTV$-BCNF. Due to gUC σ_1, gUCs *Event*:$NCTV$:VT and *Event*:$NCTV$:NT are in $\Sigma^+_{\mathfrak{C}}$, which are *Event*:$NCTV$-minimal. As every property in $NCTV$ is *Event*:$NCTV$-prime, Σ is in *Event*:$NCTV$-3NF.

Like $L{:}P$-BCNF, $L{:}P$-3NF is grounded in classical 3NF but tailored to graph features.

Theorem 5 *For every label set L and property set P, Σ is in $L{:}P$-3NF if and only if $(R_P, \Sigma_{L:P})$ is in 3NF.*

Proof The proof is similar to the proof of Theorem 3 and uses the fact that a property $A \in P$ is $L{:}P$-prime for Σ if and only if A is prime for $\Sigma_{L:P}$. □

Given the set of $L{:}P$-prime properties for Σ, the quadratic time required to validate 3NF for $\Sigma_{L:P}$ extends to $L{:}P$-3NF for Σ.

Theorem 6 *Σ in $L{:}P$-3NF if and only if for every gFD $L'{:}P'{:}X \to Y \in \Sigma$ where $L' \subseteq L$ and $P' \subseteq P$ it is true that $Y \subseteq X$ or $L{:}P{:}X \in \Sigma^+_{\mathfrak{C}}$ or every property in $Y - X$ is $L{:}P$-prime.* □

Testing $L{:}P$-BCNF is efficient, but validating $L{:}P$-3NF is likely intractable as it is *NP*-complete to decide if a property is $L{:}P$-prime, already when $L = \emptyset$ and $P = R$ [6]. It is *coNP*-complete to decide if for Σ, $\Sigma_{L:P:S}$ is in $L{:}P$-BCNF where $S \subseteq P$ and $\Sigma_{L:P:S} = \{L'{:}P'{:}X \to Y \in \Sigma^+_{\mathfrak{C}} \mid L' \subseteq L \wedge P' \subseteq P \wedge XY \subseteq S \subseteq P\}$.

Theorem 7 *Deciding for Σ, if $\Sigma_{L:P:S}$ is in $L{:}P$-BCNF, is* coNP*-complete. Deciding whether Σ is in $L{:}P$-3NF is* NP*-complete.* □

8 Achievements of Normal Forms

We justify our normal forms by the absence of redundancy, sources of inconsistency and update inefficiency.

8.1 Eliminate Redundancy & Sources of Inconsistency

First, we aim at minimizing sources of property values that may occur redundantly in graphs that satisfy the given gUCs and gFDs.

Let v denote a node of property graph G that carries all labels in L and all properties in P. Let $A \in P$. An *$L{:}P$-replacement of v on A* is any property graph G' that results from G by changing value $\nu(v, A)$ to some different value. The occurrence $\nu(v, A)$ is *$L{:}P$-redundant for Σ* if and only if for **every** $L{:}P$-replacement G' of v on A, the graph G' violates some constraint $L{:}P{:}X$ or $L{:}P{:}X \to Y$ in $\Sigma^+_{\mathfrak{C}}$.

Definition 4 Σ is in *$L{:}P$-Redundancy Free Normal Form (RFNF)* iff there is no property graph G that satisfies Σ, no node $v \in V_{L:P}$ in G, and no property $A \in P$ such that $\nu(v, A)$ is $L{:}P$-redundant for Σ.

In graph G_0 of Figure 1, each occurrence $\nu(2, \text{Company})$ and $\nu(3, \text{Company})$ of "Cactus" is *Event*:$NCVT$-redundant. For instance, if G'_0 results from G_0 by replacing $\nu(2, \text{Company})$ by a value different from "Cactus", G'_0 will violate gFD *Event*:$NCVT$:$N \to C \in \Sigma^+_{\mathfrak{C}}$. Hence, Σ is not in *Event*:$NCVT$-RFNF. In contrast, the occurrence of $\nu(23 - 1, \text{Company}) =$ "Cactus" in graph G_n of Figure 2 is not *Evt_Mgt*:NC-redundant. Indeed, Σ' is in *Evt_Mgt*:NC-RFNF.

Theorem 8 *For all sets Σ of gUC/FDs, for all label sets L and property sets P, we have Σ is in L:P-RFNF iff $(R_P, \Sigma_{L:P})$ is in RFNF.*

Proof We show first: if Σ is not in L:P-RFNF, then $(R_P, \Sigma_{L:P})$ is not in RFNF. By hypothesis, there is some property graph G that satisfies Σ, some node $v \in V_{L:P}$, and some property $A \in P$ such that $\nu(v, A)$ is L:P-redundant for Σ. Hence, for every L:P-replacement G' of v on A, G' violates some gUC L:P:X or gFD L:P:$X \to Y$ in $\Sigma^+_{\mathfrak{C}}$ such that $L' \subseteq L$ and $P' \subseteq P$. Consequently, there is some node $v' \in V_{L:P}$ such that $v \neq v'$, $\nu(v, XA) = \nu(v', XA)$ and $A \in Y - X$ for some non-trivial gFD L':P':$X' \to Y' \in \Sigma$ with $L' \subseteq L$ and $P' \subseteq P$. In particular, $X' \to Y' \in \Sigma_{L:P}$. Let $r := \{t_v, t_{v'}\}$ be a relation over R_P such that for all $B \in R_P$, $t_v(B) = t_{v'}(B)$ iff $B \in (X')^+_{\Sigma_{L:P}}$. Note that $R_P - (X')^+_{\Sigma_{L:P}}$ is non-empty as otherwise $\Sigma_{L:P}$ would imply $X' \to R_P$, meaning L:P:X' would be implied by Σ, and the latter would entail that $v = v'$ since $\nu(v, X') = \nu(v', X')$. Since $R_P - (X')^+_{\Sigma_{L:P}}$ is non-empty, $t_v \neq t_{v'}$. We show further that r satisfies $\Sigma_{L:P}$. Suppose $t_v(X) = t_{v'}(X)$ for some FD $X \to Y \in \Sigma_{L:P}$. Then $X \subseteq (X')^+_{\Sigma_{L:P}}$, and $X' \to X$ is implied by $\Sigma_{L:P}$. Hence, $X' \to Y$ is implied by $\Sigma_{L:P}$, too. We conclude $Y \subseteq (X')^+_{\Sigma_{L:P}}$ and $t_v(Y) = t_{v'}(Y)$. Hence, $t_v(A)$ in r is redundant for $\Sigma_{L:P}$ since for every replacement $\bar{t}_v$ of t_v on A, $\bar{r} := (r - \{t_v\}) \cup \{\bar{t}_v\}$ violates the FD $X' \to Y'$ in $\Sigma_{L:P}$. Hence, $(R_P, \Sigma_{L:P})$ is not in RFNF.

We now show: if $(R_P, \Sigma_{L:P})$ is not in RFNF, then Σ is not in L:P-RFNF for Σ. By hypothesis, there is some relation r over R_P that satisfies $\Sigma_{L:P}$, some $t \in r$, and $A \in R_P$ such that $t(A)$ is redundant for $\Sigma_{L:P}$. That is, for every replacement $\bar{t}$ of t on A, $\bar{r} := (r - \{t\}) \cup \{\bar{t}\}$ violates some constraint in $\Sigma_{L:P}$. Hence, there is some $t' \in r$ such that $t \neq t'$, $t(X'A) = t'(X'A)$ and $A \in Y' - X'$ for some FD $X' \to Y' \in \Sigma_{L:P}$. In particular, $Y' \subset R_P$ as otherwise $t = t'$. Hence, there is some L':P':$X' \to Y' \in \Sigma$ such that $L' \subseteq L$ and $P' \subseteq P$. In particular, $A \in Y' \subseteq P' \subseteq P$. Let $G^{t,t'}$ be defined as follows: $V = \{v_t, v_{t'}\}$ with $\mu(v_t) \neq \mu(v_{t'})$, $\lambda(v_t) = L = \lambda_{v_{t'}}$, $Ed = \emptyset$, and $\nu(v_t, A) := t(A)$ and $\nu(v_{t'}, A) := t'(A)$ for all $A \in P$. It follows that $G^{t,t'}$ satisfies every gUC L'':P'':X'' and every gFD L'':P'':$X'' \to Y''$ in Σ. Indeed, if $L'' \not\subseteq L$ or $P'' \not\subseteq P$, then $G_{L'':P''} = \emptyset$ and the gUCs and gFDs above are satisfied. Otherwise $L'' \subseteq L$ and $P'' \subseteq P$, and then $G_{L'':P''} = G$ satisfies X'' and $X'' \to Y''$ because $\{t, t'\}$ satisfies $X'' \to R_P$ and $X'' \to Y''$ in $\Sigma_{L:P}$. Hence, there are a property graph $G^{t,t'}$ that satisfies Σ, a node $v_t \in V_{L:P}$ in $G^{t,t'}$, and a property $A \in P$ such that $\nu(v_t, A)$ is L:P-redundant for Σ. Hence, Σ is not in L:P-RFNF. □

In illustrating Theorem 8, the following relation r

Name	*Company*	*Venue*	*Time*
No Socks	Cactus	Vault	06/01
No Socks	Cactus	Vault	13/07

corresponds to node set $V_{Event:NCTV}$ of graph G_0 in Figure 1. It satisfies $\Sigma_{Event:NCTV} = \{N \to C, NT \to V, TV \to N, CT \to NV\}$, and each occurrence of "Cactus" is redundant. This example is representative that BCNF captures RFNF. Indeed, Theorem 8 lifts the result from relational databases to property graphs.

Corollary 3 *For all sets Σ of gUC/FDs, for all label sets L and property sets P, we have Σ is in L:P-RFNF iff Σ is in L:P-BCNF.*

Proof Theorem 3 has shown that Σ is in L:P-BCNF iff $(R_P, \Sigma_{L:P})$ is in BCNF. However, we know that classical BCNF captures classical BCNF, that is, $(R_P, \Sigma_{L:P})$ is in BCNF iff $(R_P, \Sigma_{L:P})$ is in RFNF. However, the latter holds iff Σ is in L:P-RFNF by Theorem 8. □

Classical 3NF exhibits the fewest sources of data redundancy among all dependency-preserving decompositions [25]. Due to Theorem 5, these results carry over to L:P-3NF, pending our definitions below.

8.2 Efficient Updates

Traditionally, database normalization also aims at handling updates efficiently. In contrast to gFDs that typically require a whole scan of the underlying property graph that is often considered too much of an overhead, gUCs can be enforced in time logarithmic in the input based on unique indices. The idea is therefore to view update operations, such as node inserts, deletions or property value replacements, as efficient when it suffices to validate gUCs as opposed to validating the whole set of given constraints. As node deletions cannot result in violations of gUCs or gFDs, it suffices to consider node insertions. Indeed, replacing property values in a node is the two-step process of (i) deleting the node and (ii) inserting the node matching the previous one except for the values that were replaced. We will now introduce a semantic normal form to formalize these ideas. For a given set Σ of gUCs and gFDs let

$$\begin{aligned}\Sigma_u = \{&L{:}P{:}X \mid L{:}P{:}X \in \Sigma^* \wedge \neg\exists L'{:}P'{:}X' \in \Sigma^* \\ &(L' \subseteq L \wedge P' \subseteq P \wedge X' \subseteq X \wedge \\ &(L' \subset L \vee P' \subset P \vee X' \subset X))\}\end{aligned}$$

denote the set of minimal gUCs implied by Σ. Indeed, any gUC L:P:X in Σ_u is minimal, since none of the labels in L, properties in P nor X can be removed without the resulting gUC becoming non-implied by Σ. For every label set L and property set P, let $\Sigma_u^{L:P} = \{L'{:}P' :$

$X \in \Sigma_u \mid L' \subseteq L \wedge P' \subseteq P\}$ denote the minimal gUCs implied by Σ that are application-relevant for the given label requirement L and property requirement P. In what follows $G \cup \{v\}$ denotes the property graph obtained from G by inserting the node v together with its set of labels, properties, and values.

Definition 5 Σ has an *L:P-insertion inefficiency* iff there is some property graph G that satisfies Σ and some node v such that $G \cup \{v\}$ satisfies $\Sigma_u^{L:P}$ but violates some gFD $L{:}P{:}X \to Y$ in Σ^*. Σ is in *L:P-Insertion Efficiency Normal Form (IENF)* iff Σ has no L:P-insertion inefficiency.

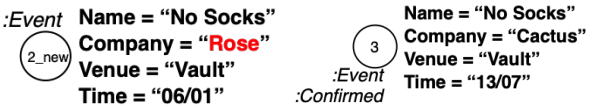
We observe that insertion inefficiencies and value redundancies go hand-in-hand.

Lemma 3 *For all sets Σ of gUC/FDs, for all label sets L and property sets P, we have Σ is in L:P-RFNF iff Σ is in L:P-IENF.*

Proof We show first: if Σ is not in L:P-RFNF, then Σ is not in L:P-IENF. By hypothesis, there is some property graph G that satisfies Σ, some node $v \in V_{L:P}$, and some property $A \in P$ such that $\nu(v, A)$ is L:P-redundant for Σ. Hence, for every L:P-replacement G' of v on A, G' violates some gUC $L{:}P{:}X$ or gFD $L{:}P{:}X \to Y$ in $\Sigma^+_{\mathfrak{C}}$. As changing $\nu(v, A)$ to any value not present in G can never violate any gUC, for every L:P-replacement G' of v on A, G' violates some gFD $L{:}P{:}X \to Y$ in $\Sigma^+_{\mathfrak{C}}$. Hence, there is some node $v' \in V_{L:P}$ such that $v \neq v'$, $\nu(v, XA) = \nu(v', XA)$ and $A \in Y - X$ for some non-trivial gFD $L'{:}P'{:}X' \to Y' \in \Sigma$ with $L' \subseteq L$ and $P' \subseteq P$. In particular, $X' \to Y' \in \Sigma_{L:P}$.

Hence, there is a property graph G' that is an L:P-replacement of v on A such that G' satisfies Σ_u but violates some gFD $L{:}P{:}X \to Y$ in $\Sigma^+_{\mathfrak{C}}$. Let G'_v denote the property graph that only contains the replacement node v_{new} of v together with all its labels, properties, and values from G', but with no other nodes or edges. Consequently, G'_v satisfies Σ. Consider the property graph $G'_v \cup \{v'\}$, where v' has the properties as stated above. It follows that $G'_v \cup \{v'\}$ satisfies $\Sigma_u^{L:P}$ but violates some gFD $L{:}P{:}X \to Y \in \Sigma^*$. Hence, Σ is not in L:P-IENF.

We show now: If Σ is not in L:P-IENF, then Σ is not in L:P-RFNF. By hypothesis, there is some property graph G that satisfies Σ and some node v_0 such that $G \cup \{v_0\}$ satisfies $\Sigma_u^{L:P}$ but violates some gFD $L{:}P{:}X \to Y$ in Σ^*. Hence, there must be some node $v \in V_{L:P}$ of G such that $v \neq v_0$ and some property $A \in Y - X$ such that $\nu(v, X) = \nu(v_0, X)$ but $\nu(v, A) \neq \nu(v_0, A)$. Let u be a new node with label set L and property set P such that for every property $A \in X^+_{\Sigma_{L:P}}$, $\nu(u, A) := \nu(v, A)$, and for all $B \in P - X^+_{\Sigma_{L:P}}$, $\nu(u, B)$ is some distinct value that does not occur elsewhere in G. It then follows that the value occurrence $\nu(v, A)$ is L:P-redundant in $G \cup \{u\}$. In other words, Σ is not in L:P-RFNF. □



(a) $G'_v = \{2\_\text{new}\}$ (b) $G'_v \cup \{3\} = \{2\_\text{new}, 3\}$

Fig. 3: Property graph $\{2\_\text{new}, 3\}$ satisfies $\Sigma_u = \{\sigma_1\}$ but not $\sigma_2 \in \Sigma$

Figure 12 illustrate that our running example with gUC/gFD set $\Sigma = \{\sigma_1, \ldots, \sigma_4\}$ is not in *Event*:*NCTV*-IENF. While the property graph $G'_v = \{2\_\text{new}\}$ satisfies Σ, $G'_v \cup \{3\}$ satisfies $\Sigma_u = \{\sigma_1\}$ with gUC $\sigma_1 = \mathit{Event} : \mathit{CT} : \mathit{CT}$ but not gFD $\sigma_2 = \mathit{Event}{:}\mathit{NC}{:}N \to C$.

Hence, L:P-BCNF is equivalent to L:P-IENF.

Corollary 4 *For all sets Σ of gUC/FDs, for all label sets L and property sets P, we have Σ is in L:P-IENF iff Σ is in L:P-BCNF.*

Proof Σ is in L:P-RFNF iff Σ is in IENF by Lemma 3, so Corollary 4 follows directly from Corollary 3. □

9 Normalizing Property Graphs

We will show how to restructure, without loss of information and guided by target sets L of node labels and P of properties, a set Σ and a property graph G that satisfies Σ such that the restructered constraint set is satisfied by the restructured graph and is in $L : P$-3NF, and $L : P$-BCNF if possible. We will first describe the method informally, illustrate it on our running example, and then provide the technical definitions.

9.1 Method

Intuitively, the normalization process is as follows.

1) Given L, P, G and Σ, for each node $v \in V_{L:P}$ and each element S of a decomposition for P (a set $\mathcal{D}$ of subsets for R_P), we introduce nodes v_S with fresh label ℓ_S and directed edges (v_S, v) with fresh label ℓ, and transfer the properties in S from v to v_S. For each S, these operations generate the projection $G^{\ell}_{L:P}[S]$ of $G_{L:P}$ onto S, with the restriction $G_{L:P}$ of G onto $V_{L:P}$.

2) We then materialize the "redundancy elimination" by identifying new nodes v_S and v'_S whenever they exhibit matching values on all properties in S. Technically, this is achieved by a congruence relation $\equiv_S$, and forming the quotient graph $G^{\ell}_{L:P}[S]/\equiv_S$.

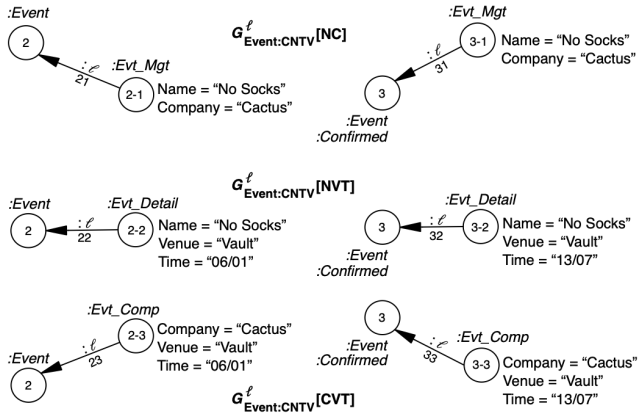


Fig. 4: Projections of G_0 onto $\mathcal{D} = \{NC, NVT, CVT\}$

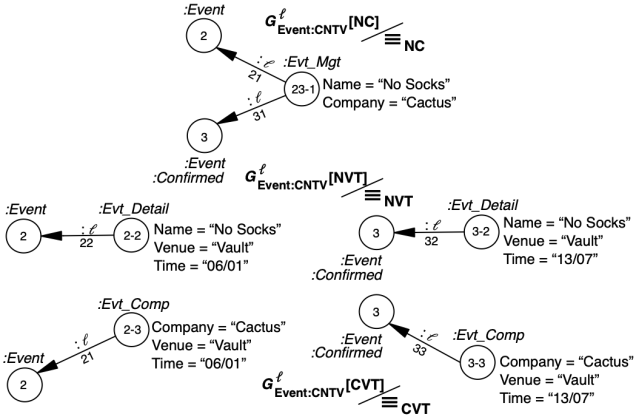


Fig. 5: Quotient Graphs $G^{\ell}_{Event:CNTV}[S]/\equiv_S$ of G_0

3) We then take the union of quotient graphs over all elements S of the decomposition $\mathcal{D}$ and the original graph G. The resulting property graph $G^{\ell}_{L:P}[\mathcal{D}]$ is an L:P-decomposition of G onto $\mathcal{D}$.

4) Similarly, the L:P-decomposition $\Sigma^{\ell}_{L:P}[\mathcal{D}]$ of Σ onto $\mathcal{D}$ is obtained by adding gUCs ℓ_S:S:X for each $X \to R_P \in \Sigma_{L:P}[S]$ and adding gFDs ℓ_S:S:$X \to Y$ for each $X \to Y \in \Sigma_{L:P}[S]$ for $Y \neq R_P$.

This construction can easily be inverted by collapsing all edges (v_S, v) labeled ℓ and transferring back the property/value pairs from v_S to the node v they originated from. The original dependencies imply new ones on the new nodes, transforming gFDs into gUCs whenever possible, which is why property value redundancy is removed as far as possible. Due to labels, we can simply add the new ones, and remove them when the decomposition is inverted.

Consider again Example 3. The set Σ' is a lossless, dependency-preserving *Event*:$CNTV$-decomposition of Σ into BCNF. The property graph G_n in Figure 2 is the Event:$NCTV$-decomposition of G_0 in Figure 1, based on the decomposition $\mathcal{D} = \{NC, NVT, CVT\}$ of R_{NCTV}. Indeed, Figure 4 shows the three projections $G^{\ell}_{Event:CNTV}[S]$ of $G_{Event:CNTV}$ onto $S \in \mathcal{D}$ from step 1) of the process above, including new nodes 2-1 ($=2_{NC}$), 2-2($=2_{NVT}$), 2-3($=2_{CVT}$), 3-1($=3_{NC}$), 3-2($=3_{NVT}$) and 3-3($=3_{CVT}$), with node labels ℓ_{NC} = *:Evt_Mgt*, ℓ_{NVT} = *:Evt_Details* and ℓ_{CVT} = *:Evt_Comp*, and directed edges 21=(2-1,2), 22=(2-2,2), 23=(2-3,2), 31=(3-1,3), 32=(3-2,3), and 33=(3-3,3) with edge label ℓ.

Step 2) of the process is illustrated in Figure 5 where the quotient graphs of the projections are shown. Here, the only vertices identified are 2-1 and 3-1 based on their value equality on NC. Step 3) results in G_n (Figure 2) by taking the union of quotient graphs from Figure 5 and the original graph. Finally, step 4) results in the constraint set $\Sigma \cup \Sigma'$ where $\Sigma' = \Sigma^{\ell}_{Event:NCTV}[\mathcal{D}] = \{\sigma'_1, \ldots, \sigma'_5\}$.

9.2 Formal Definitions

For a property graph G, $L \subseteq \mathcal{L}$, and $P \subseteq \mathcal{K}$, we define $G_{L:P}$ to denote the restriction of G to the vertex set $V_{L:P}$. For a property set $S \subseteq P$, and a label $\ell \in \mathcal{L}$ that does not occur in G, we define the L:P-projection $G^{\ell}_{L:P}[S]$ of $G_{L:P}$ onto S by

- $V_{L:P}[S] := V_{L:P} \cup \bigcup_{v \in V_{L:P}} \{v_S\}$
- $Ed_{L:P}[S] := \bigcup_{v \in V_{L:P}} \{(v_S, v)\}$
- $\lambda_{L:P}[S] := \begin{cases} v \mapsto \lambda(v) & \text{, if } v \in V_{L:P} \\ v_S \mapsto \ell_S & \text{, if } v_S \in V_{L:P}[S] \\ (v_S, v) \mapsto \ell & \text{, if } (v_S, v) \in Ed_{L:P}[S] \end{cases}$
- $\nu_{L:P}[S] := \begin{cases} (v_S, A) \mapsto \nu(v, A) & \text{, if } A \in S \wedge \lambda(v_S, v) = \ell \\ (v_S, A) \mapsto \uparrow & \text{, if } A \notin S \wedge \lambda(v_S, v) = \ell \\ (v, A) \mapsto \nu(v, A) & \text{, if } A \notin S \wedge v \in V_{L:P} \\ (v, A) \mapsto \uparrow & \text{, if } A \in S \wedge v \in V_{L:P} \end{cases}$

For example, Figure 4 shows the projections of G_0 onto $S \in \mathcal{D} = \{NC, NVT, CVT\}$ with identifiers of new nodes v_S (edges (v_S, v)) marked within node circles (alongside the edges, respectively), and node labels ℓ_S carry have real names such as ℓ_{NC} = *:Evt_Mgt*.

For a property set $S \subseteq \mathcal{K}$ and two nodes v, v' of a property graph, we define $v \equiv_S v'$ if and only if for all $A \in S$, $\nu(v, A) = \nu(v', A)$. That is, the two nodes are equivalent on the property set S if and only if they have matching values on all the properties in S. Of course, $\equiv_S$ defines an equivalence relation between the nodes of a property graph G, so we may define the quotient graph $G/\equiv_S$. For example, the quotient graphs of G_0 onto $S \in \mathcal{D} = \{NC, NVT, CVT\}$ are shown in Figure 5, where nodes 2-1 and 3-1 are equivalent on NC.

For two property graphs G and G' over $\mathcal{O}$, $\mathcal{L}$, and $\mathcal{K}$ we define the union $G \cup G'$ as the property graph obtained as $V_G \cup V_{G'}$, $Ed_G \cup Ed_{G'}$, $\lambda_G \cup \lambda_{G'}$, $\mu_G \cup$

$\mu_{G'}$ but where $\nu_G \cup \nu_{G'}$ is defined by $\nu(v, A) \uparrow$ for any property $A \in \mathcal{K}$ whenever $\nu_G(v, A)$ and $\nu_{G'}(v, A)$ have non-matching values (eg. only one of them is defined). For example, property graph G_n from Figure 2 is the union of quotient graphs from Figure 5 and G_0.

For a property graph G and label $\ell \in \mathcal{L}$ we define $\overset{\ell}{\bowtie} G$ as follows:

- $V := V_G - \{v' \in V_G \mid \exists (v', v) \in Ed_G, \lambda(v', v) = \ell\}$
- $Ed := Ed\,|_V$, $\lambda := \lambda_G\,|_V$, $\mu := \mu_G\,|_V$, and
- $\nu := \begin{cases} (v, A) \mapsto \nu_G(v, A) & \text{, if } v \in V \wedge \nu_G(v, A) \downarrow \\ (v, A) \mapsto \nu_G(v', A) & \text{, if } (v', v) \in Ed_G \wedge \\ & \quad \lambda_G(v', v) = \ell \wedge \nu_G(v', A) \downarrow \end{cases}$

As example, $G_0 = \overset{\ell}{\bowtie} G_n$ with G_0 from Figure 1 and G_n from Figure 2.

In relational databases, a *decomposition* of attribute set R is a set $\mathcal{D}$ of subsets of R such that $\bigcup_{S \in \mathcal{D}} S = R$, for example $\mathcal{D} = \{NC, NVT, CVT\}$ of $CNTV$. For an FD set Σ over R, and subset $S \subseteq R$, $\Sigma[S] = \{X \to Y \in \Sigma^+_{\mathfrak{A}} \mid XY \subseteq S\}$ is the projection of Σ onto S. As example, for $\Sigma = \Sigma_{Event:NCTV} = \{N \to C, NT \to V, TV \to N, CT \to NV\}$ we have $\Sigma[NC] = \{N \to C\}$, $\Sigma[NTV] = \{TV \to N, NT \to V\}$ and $\Sigma[CTV] = \{CT \to V, VT \to C\}$.

Definition 6 For gUC/gFD set Σ, label set L, property set P, and decomposition $\mathcal{D}$ of R_P, we define the *L:P-projection $\Sigma^\ell_{L:P}[\mathcal{D}]$ of Σ onto $\mathcal{D}$* by $\Sigma \cup \{\ell_S{:}S{:}X \mid X \to R_P \in \Sigma_{L:P}[S] \text{ for } S \in \mathcal{D}\} \cup \{\ell_S{:}S{:}X \to Y \mid X \to Y \in \Sigma_{L:P}[S] \wedge Y \neq R_P \wedge S \in \mathcal{D}\}$. We say the *$L$:$P$-decomposition $\Sigma^\ell_{L:P}[\mathcal{D}]$ of Σ is in BCNF (3NF)* iff for all $S \in \mathcal{D}$, $\Sigma^\ell_{L:P}[S]$ is in $\ell_S{:}S$-BCNF (3NF). The L:P-decomposition $\Sigma^\ell_{L:P}[\mathcal{D}]$ of Σ is *dependency-preserving* iff $\Sigma_{L:P}$ and $\bigcup_{S \in \mathcal{D}} \left(\Sigma^\ell_{L:P}[S]\right)_{\ell_S:S}$ are covers of one another. The *L:P-decomposition $G^\ell_{L:P}[\mathcal{D}]$ of a property graph G onto $\mathcal{D}$* is defined by $G^\ell_{L:P}[\mathcal{D}] := G \cup \bigcup_{S \in \mathcal{D}} G^\ell_{L:P}[S] / \equiv_S$. The L:P-decomposition $\Sigma^\ell_{L:P}[\mathcal{D}]$ of Σ is *lossless* iff for every property graph G that satisfies Σ, the L:P-decomposition $G^\ell_{L:P}[\mathcal{D}]$ of G onto $\mathcal{D}$ satisfies $G_{L:P} = \overset{\ell}{\bowtie} G^\ell_{L:P}[\mathcal{D}]$. □

As example, for $\Sigma = \{\sigma_1, \ldots, \sigma_4\}$, $L = Event$, $P = CNTV$, and BCNF-decomposition $\mathcal{D} = \{NC, NVT, CVT\}$ of P, $\Sigma^\ell_{L:P}[\mathcal{D}] = \Sigma \cup \Sigma'$ where $\Sigma' = \{\sigma'_1, \ldots, \sigma'_5\}$. Indeed, $\Sigma^\ell_{L:P}[\mathcal{D}]$ is in BCNF since it is in *Evt_Mgt*:NC-BCNF, *Evt_Comp*:CVT-BCNF, and *Evt_Detail*:NVT-BCNF, see Example 3. The decomposition is also dependency-preserving since $\Sigma_{Event:NCTV}$ and the union of $\Sigma_{Event:NCTV}[NC]$, $\Sigma_{Event:NCTV}[NTV]$ and $\Sigma_{Event:NCTV}[CTV]$ cover one another.

Algorithm 2 NormaG

Require: Property graph G that satisfies gUC/FD set Σ; label set $L \cup \{\ell\}$; property set P
Ensure: Property graph $G^\ell_{L:P}[\mathcal{D}]$ that satisfies $\Sigma^\ell_{L:P}[\mathcal{D}]$, which is a lossless, dependency-preserving L:P-decomposition of Σ into 3NF (which is in BCNF whenever possible)
1: Compute atomic closure $\overline{\Sigma}_a$ of $\Sigma_{L:P}$ on R_P [35];
2: $\Sigma_a \leftarrow \overline{\Sigma}_a$
3: **for all** $X \to A \in \Sigma_a$ **do**
4: **for all** $Y \to B \in \overline{\Sigma}_a (YB \subseteq XA \wedge XA \not\subseteq Y^+)$ **do**
5: **if** $\Sigma_a - \{X \to A\} \models X \to A$ **then**
6: $\Sigma_a \leftarrow \Sigma_a - \{X \to A\}$ {Eliminate critical schemata}
7: $\mathcal{D} \leftarrow \emptyset$
8: **for all** $X \to A \in \Sigma_a$ **do**
9: **if** $\Sigma_a - \{X \to A\} \models X \to A$ **then**
10: $\Sigma_a \leftarrow \Sigma_a - \{X \to A\}$ {Eliminate redundant schemata}
11: **else**
12: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(XA, \overline{\Sigma}_a[XA])\}$
13: Remove all $(S, \overline{\Sigma}_a[S]) \in \mathcal{D}$ if $\exists (S', \overline{\Sigma}_a[S']) \in \mathcal{D} (S \subseteq S')$
14: **if** there is no $(R', \Sigma') \in \mathcal{D}$ where $R' \to R_P \in \Sigma^+_{L:P}$ **then**
15: Choose a minimal key K for R_P with respect to $\Sigma_{L:P}$
16: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(K, \overline{\Sigma}_a[K])\}$
17: **return** $\left(G^\ell_{L:P}[\mathcal{D}], \Sigma^\ell_{L:P}[\mathcal{D}]\right)$

9.3 Decomposition Algorithm

Our decomposition is always lossless, but only when a gFD is converted into a gUC, all redundancy caused by the gFD is eliminated. Indeed, normalizing a property graph will eliminate redundancy on those equivalence classes where the underlying gFD holds.

Algorithm 2 normalizes a property graph G and gUC/FD set Σ tailored to label set L and property set P. Our techniques make it possible for lines (1-16) to apply state-of-the-art normalization from relational databases that achieves a lossless, dependency-preserving 3NF decomposition $\mathcal{D}$ into BCNF whenever possible. $\mathcal{D}$ is then converted into the output $\left(G^\ell_{L:P}[\mathcal{D}], \Sigma^\ell_{L:P}[\mathcal{D}]\right)$ in line (17).

Theorem 9 *On input $((G, \Sigma), L \cup \{\ell\}, P)$ such that G satisfies Σ, Algorithm 2 returns the property graph $G^\ell_{L:P}[\mathcal{D}]$ that satisfies $\Sigma^\ell_{L:P}[\mathcal{D}]$, which is a lossless, dependency-preserving L:P-decomposition of Σ into 3NF that is in BCNF whenever possible.*

Proof Lines 1-16 ensure that a lossless, dependency-preserving decomposition $\mathcal{D}$ of $(R_P, \Sigma[L{:}P])$ into 3NF is returned that is in BCNF whenever possible. $\Sigma^\ell_{L:P}[\mathcal{D}]$ is always lossless, and $\bigcup_{S \in \mathcal{D}} \left(\Sigma^\ell_{L:P}[S]\right)_{\ell_S:S}$ is a cover of $\Sigma_{L:P}$ by definition of $\Sigma^\ell_{L:P}[S]$. Finally, for every $S \in \mathcal{D}$, it is true that $\Sigma^\ell_{L:P}[S]$ is in $\ell_S{:}S$-3NF (or -BCNF, respectively) if and only if $(S, \Sigma_{L:P}[S])$ is in 3NF (BCNF). Hence, the result follows by construction. □

Given G_0 from Figure 1, $L = Event$ and $P = CNTV$, Algorithm 2 returns G_n from Figure 2 and gUC/gFD set $\Sigma \cup \Sigma'$ from Section 2.

Table 3: Details of graph datasets from the experiments

Graph data	L	$\|V_L\|$	$\|P\|$	$\%V_{L:P}$	#gFDs	AvgRed	#gUCs
Northwind	Order	830	14	35.54%	555	25.13	49
Offshore	Entity	814,345	18	26.14%	1414	47,919.13	102

10 Experiments

Our experiments will showcase the extent of both opportunities and benefits of normalizing graph data. This will be done quantitatively and qualitatively using popular real-world property graphs, but also synthetic graph data for scalability tests. The research questions we aim to answer by our experiments are:

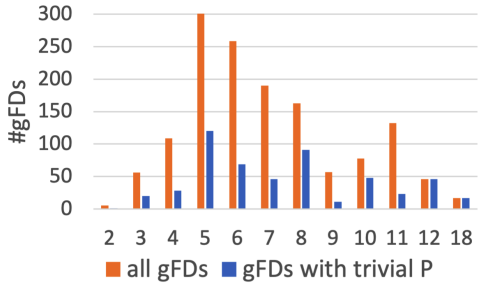
Q1) What gFDs do property graphs exhibit?
Q2) What gFDs cause much data redundancy?
Q3) How much inconsistency can gFDs avoid?
Q4) What does graph normalization actually look like?
Q5) How much better is integrity management after normalization?
Q6) How much faster are queries after normalization?
Q7) How do the benefits of normalization scale?

Q1)-Q3) will illustrate why normalization is necessary. Q4) will showcase normalization on a real-world graph, and Q5-Q7) will underline benefits of normalization at the operational level.

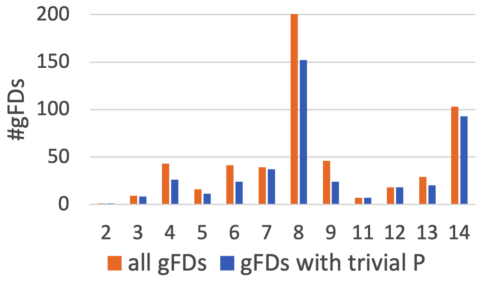
10.1 Data Sets and Measures

Details of experiments are on our Github repository `https://github.com/GraphDatabaseExperiments/normalization_experiments`. We analyzed graphs *Northwind* (`https://github.com/neo4j-graph-examples/northwind`) with 1,035 nodes, 3,139 edges, and sales data; and *Offshore* (`https://github.com/ICIJ/offshoreleaks-data-packages`) with 2,016,524 nodes, 3,336,971 edges, and global company data. Table 3 shows the node labels L we target, the number $|V_L|$ of nodes with label L, the number $|P|$ of properties for those nodes, the percentage $\%V_{L:P}$ of nodes with these properties, the numbers #gFDs and #gUCs in a minimal cover of constraints that hold on the data sets, and the average number of redundant property values caused by gFDs. Note the high number on *Offshore*.

We used *Neo4j* and its query language *Cypher* as currently most popular graph database (`https://db-engines.com/en/ranking/graph+dbms`), its support of unique constraints, indexes, and the measure of *database hits*, an abstract unit of the storage engine related to requests for operations on nodes or edges. For comparison with a cloud-based provider, we also used



(a) Offshore



(b) Northwind

Fig. 6: #gFDs by Property Size $|P|$

Amazon Neptune, which has no support of indexes or database hits. We also measured run times. We used Python 3.9.13. Experiments were conducted on a 64-bit operating system with an Intel Core i7 Processor with 16GB RAM. Details of experiments are available in the Artifact URL.

10.2 What gFDs do graphs exhibit?

We mined gFDs with fixed target labels *Entity* (Offshore) and *Order* (Northwind). Figure 6 classifies the gFDs $L : P : X \to Y$ by the size $|P|$ of their property set P. If $P = XY$, we call P trivial.

The mined gFDs include interesting examples. On *Offshore*[2], for instance, we have the gFDs

- *Entity* : *address*, *country_codes*, *countries*: *countries*, *address* → *country_codes*
- *Entity* : *service_provider*, *country_codes*, *countries*: *countries* → *country_codes*.

In particular, for every mined L:P:$X \to Y$, removing any property from P or X will result in a gFD that is violated by the dataset. Hence, the gFD *Entity*:*country_codes*, *countries*:*countries* → *country_codes* is not satisfied by the dataset.

[2] Properties described here: `https://guides.neo4j.com/sandbox/icij-paradise-papers/datashape.html`

Table 4: gFDs on Offshore Ranked by Redundancy Caused

$P\backslash XY$	X	Y	$\#red$	$\#inc$
incorporation_date	*jurisd_desc*, *lastEditTimestamp*, *sourceID*	*jurisdiction*	788,408	20,000
incorporation_date	*jurisd_desc*, *sourceID*, *valid_until*	*jurisdiction*	788,365	175,871
	incorporation_date, *jurisd_desc*, *valid_until*	*service_provider*	754,283	1,047
ibcRUC	*jurisd_desc*, *valid_until*	*service_provider*	555,353	20,000
ibcRUC	*jurisd_desc*, *lastEditTimestamp*	*service_provider*	555,353	175,888
ibcRUC	*jurisdiction*, *lastEditTimestamp*, *valid_until*	*sourceID*	555,338	20,000
country_codes	*jurisd_desc*, *lastEditTimestamp*, *sourceID*	*jurisdiction*	504,944	20,000
countries	*jurisd_desc*, *lastEditTimestamp*, *sourceID*	*jurisdiction*	504,944	20,000
country_codes	*jurisd_desc*, *sourceID*, *valid_until*	*jurisdiction*	504,902	113,055
countries	*jurisd_desc*, *sourceID*, *valid_until*	*jurisdiction*	504,902	113,055
	country_codes, *sourceID*	*countries*	504,424	83,647
	countries, *sourceID*	*country_codes*	504,424	83,647
	country_codes, *valid_until*	*countries*	504,418	83,647
	countries, *valid_until*	*country_codes*	504,418	83,647
	countries, *jurisd_desc*	*country_codes*	504,227	83,653
	countries, *jurisdiction*	*country_codes*	504,151	83,647

Table 5: gFDs on Northwind Ranked by Redundancy Caused

$P\backslash XY$	X	Y	$\#red$	$\#inc$
	shipCity, *shipCountry*	*shipRegion*	830	33
	shipCity, *shipAddress*	*shipCountry*, *customerID*, *shipRegion*	829	31
	shipCountry, *shipAddress*	*shipCity*, *customerID*, *shipRegion*	829	31
	shipAddress	*shipPostalCode*	829	31
	customerID	*shipCity*, *shipPostalCode*, *shipCountry*, *shipAddress*, *shipRegion*	829	31
	shipName	*shipCity*, *shipPostalCode*, *shipCountry*, *shipAddress*, *customerID*, *shipRegion*	828	31
	shipCity, *shipPostalCode*	*shipCountry*, *shipRegion*	717	31
	shipPostalCode, *shipCountry*	*shipCity*, *shipRegion*	717	31
shipPostalCode	*shipAddress*	*shipCity*, *shipCountry*, *customerID*, *shipRegion*	717	31
	shipCity, *shipAddress*, *employeeID*	*shipName*	586	6
	shipCountry, *shipAddress*, *employeeID*	*shipName*	586	6
	customerID, *employeeID*	*shipName*	586	6
shipPostalCode	*shipAddress*, *employeeID*	*shipName*	514	6

Further interesting examples can be found in the *Northwind*[3] dataset where we consider $P = \{$*customerID (cID)*, *shipCity (sci)*, *shipPostalCode (sp)*, *shipCountry (sco)*, *shipAddress (sa)*, *shipRegion (sr)*$\}$.

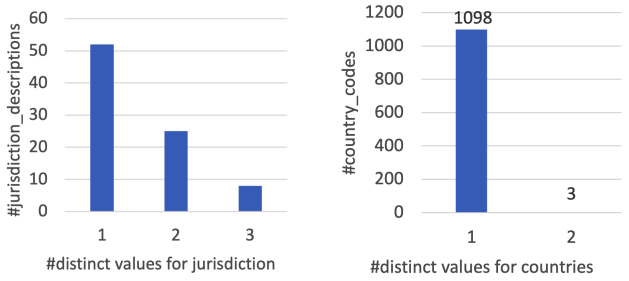
10.3 What gFDs cause much data redundancy?

Interesting for normalization are gFDs that cause many occurrences of redundant property values. Ultimately, human experts decide which constraints express meaningful business rules. However, ranking gFDs by the number of redundant property value occurrences they cause can provide helpful guidance for such decisions. For *Offshore*, Table 4 shows gFDs with Label *Entity* that cause the most number of redundant value occurrences ($\#red$). These numbers are huge, and ought to be targeted by normalization. While the following gFDs $L{:}P{:}X \to Y$ may appear to be meaningful:

1. *Entity* : *jurisd_desc*, *jurisdiction*: *jurisd_desc* $\to$ *jurisdiction*
2. *Entity* : *country_codes*, *countries*: *country_codes* $\to$ *countries*

neither of them actually holds. Nevertheless, adding few properties to P or X results in various gFDs that do

[3] Properties described here: `https://neo4j.com/graphgists/northwind-recommendation-engine/`



(a) *jurisd_desc* $\to$ *jurisdiction* (b) *country_codes* $\to$ *countries*

Fig. 7: Consistency Profiles for gFDs in Offshore

hold and exhibit many redundant property values. This makes us wonder if gFDs (1) or (2) are only violated due to data inconsistencies that are a result of data redundancy and the fact these gFDs are not enforced.

Looking at the gFDs that are satisfied in the Northwind dataset and cause high levels of redundancy in Table 5 the gFD

3. *Order* : *cID*, *sci*, *sp*, *sco*, *sa*, *sr* : *cID* $\to$ *sci*, *sp*, *sco*, *sa*, *sr*

appears to be a prime candidate for normalization. This gFD seems to represent a meaningful business rule as the property *customerID* provides information about shipping details.

10.4 How much inconsistency can gFDs avoid?

We have seen various gFDs that cause many redundant value occurrences. If these gFDs represent actual business rules, they form a primary target for graph normalization. We will now illustrate how to inform decisions whether gFDs are meaningful and violations constitute inconsistencies. We will discuss a negative and positive case, further strengthening the use of graph constraints and normalization to avoid data redundancy and sources of inconsistency.

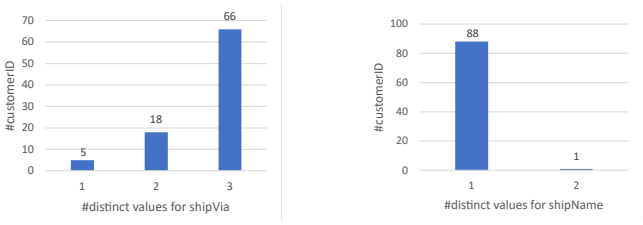
Table 4 lists the potential level of inconsistency ($\#inc$) associated with a gFD $Entity{:}P{:}X \rightarrow Y$ on *Offshore*. Hence, if the gFD is not enforced, there may be up to $\#inc$ nodes in $V_{Entity:P}$ that have matching values on all properties in X but have each different values on properties in Y. For each of the gFDs, $\#inc$ represents the worst-case scenario of not enforcing the constraint.

Let us examine gFD (1) which is violated due to *Entity*-nodes with matching values for property *jurisd_desc* and different values for property *jurisdiction*. For instance, nodes with *jurisd_desc* 'Bahamas' have either *jurisdiction* 'BAH', 'BHS' or 'BA'. Similarly, there are multiple *jurisdiction*s associated with the same *jurisd_desc*-value in 32 other cases. This consistency profile is illustrated in Figure 7(a), where we list the number of *jurisdiction_descriptions* that have n distinct *jurisdictions* associated with them, for $n = 1, 2, 3$. It is plausible that multiple jurisdictions can be associated with the same jurisdiction description. Hence, gFD (1) may not be meaningful.

In contrast, gFD (2) exhibits a different consistency profile, as shown in Figure 7(b). There are only three different *country_codes* that have two distinct *countries* linked to them, while the 1098 other codes are linked to unique countries. Indeed, for *country_code* 'COK' there are 464 nodes with value "Cook Islands" and 1389 nodes with value "COK" for property *countries*. The only other inconsistencies are linked to values "GBR;VGB" and "VGB;COK" for *country_codes*.

We will now look at gFD (3) from the *Northwind* dataset. Given the semantics of this gFD we ask why the property *customerID* determines the majority of shipping information, yet not the properties *shippinDate*, *shipVia* and *shipName*. It seems logical that *customerID* does not determine *shippinDate* as different orders by the same customer can be shipped on different dates. We further analyse the other two attributes by providing their consistency profiles in Figure 8.

The profile for the violated graph functional dependency $\{Order\} : \{customerID, shipVia\} : \{customerID\} \rightarrow \{shipVia\}$ can be seen in Figure 8 (a). Here, the majority of values for *customerID* are associated with multiple values for *shipVia*. Further analysis shows that the values for *shipVia* can be either 1, 2 or 3. This profile suggests the gFD does not represent a business rule.



(a) $customerID \rightarrow shipVia$ (b) $customerID \rightarrow shipName$

Fig. 8: Consistency profiles for gFDs in Northwind

In contrast, the profile for the graph functional dependency $\{Order\} : \{customerID, shipName\} : \{customerID\} \rightarrow \{shipName\}$, as depicted in Figure 8 (b), shows that this gFD might be meaningful. It is only violated due to data inconsistency as every but one value for *customerID* is linked to a single value for *shipName*. Further investigation shows that the only *customerID* that is associated with two different values for *shipName* carries multiple times the value "Alfred's Futterkiste" and once "Alfreds Futterkiste". This shows data inconsistency and suggests this gFD represents a business rule.

Hence, mined gFDs and their ranking provide useful heuristics to identify meaningful gFDs and data inconsistency in the form of their violations. Meaningful gFDs and consistent graph data form input desirable for normalization.

10.5 What does graph normalization look like?

While our running example is sufficiently small to illustrate our concepts and ideas, we will now examine three applications of Algorithm 2 to the property graph *Offshore*. All three applications target nodes with label *Entity (E)* but different property sets:

- P_1 ={*jurisd_desc(jd)*,*countries(c)*, *service_provider (sp)*, *country_codes(cc)*},
- P_2 = {*jd*,*valid_until (v)*,*c*, *sourceID (s)*, *cc*} and
- $P_3 = P_1 \cup P_2$.

As set Σ we use gFDs $E{:}P{:}X \rightarrow Y$ (we write $P = P \backslash XY$) as follows:

$$E{:}sp{:}c \rightarrow cc;\ E{:}\emptyset{:}c, jd \rightarrow cc;\ E{:}sp{:}cc \rightarrow c;\ E{:}\emptyset{:}c, s \rightarrow cc;$$
$$E{:}\emptyset{:}c, v \rightarrow cc;\ E{:}\emptyset{:}cc, s \rightarrow c;\ E{:}\emptyset{:}cc, v \rightarrow c;$$
$$E{:}\emptyset{:}sp \rightarrow s, v;\ E{:}sp{:}s \rightarrow v;\ E{:}sp{:}v \rightarrow s.$$

Table 6: Summary of Normalizing *Offshore*

P	$\lvert P_i\rvert$	#gFDs	#FDs	#red	#dbhits	time (ms)	$\lvert\mathcal{D}_i\rvert$
P_1	4	3	2	684,608	8,930,544	5,566	2
P_2	5	5	5	1,008,998	28,845,388	22,697	4
P_3	6	10	5	1,369,802	39,474,122	17,284	5

For $R_1 = R_{P_1} = \{jd, c, sp, cc, a_1\}$ and $\Sigma_1 = \Sigma_{E:P_1} = \{c \to cc, cc \to c\}$ we get the BCNF decomposition $\mathcal{D}_1$ of (R_1, Σ_1) into

- $R_1^1 = \{c, cc\}$ with $\Sigma_1^1 = \{c \to cc; cc \to c\}$, and
- $R_1^2 = \{jd, sp, c, a_1\}$ with $\Sigma_1^2 = \emptyset$.

Hence, we obtain the gUCs $\ell_{R_1^1}{:}R_1^1{:}\{c\}$ and $\ell_{R_1^1}{:}R_1^1{:}\{cc\}$, and $\Sigma^{\ell}_{E:P_1}[\mathcal{D}_1]$ is in BCNF. The only properties with $E{:}P_1$-redundant values are countries and country_codes. These have been eliminated by the decomposition without losing dependencies.

For $R_2 = R_{P_2} = \{jd, v, c, s, cc, a_2\}$ and $\Sigma_2 = \Sigma_{E:P_2} = \{c, jd \to cc; c, s \to cc; c, v \to cc; cc, s \to c; cc, v \to c\}$ we obtain the BCNF decomposition $\mathcal{D}_2$ of (R_2, Σ_2):

- $R_2^1 = \{c, cc, v\}$ with $\Sigma_2^1 = \{c, v \to cc; cc, v \to c\}$
- $R_2^2 = \{c, cc, s\}$ with $\Sigma_2^2 = \{c, s \to cc; cc, s \to c\}$
- $R_2^3 = \{c, cc, jd\}$ with $\Sigma_2^3 = \{c, jd \to cc\}$
- $R_2^4 = \{jd, s, v, c, a_2\}$ with $\Sigma_1^4 = \emptyset$.

Hence, we get the gUCs $\ell_{R_2^1}{:}R_2^1{:}\{c, v\}$; $\ell_{R_2^1}{:}R_2^1{:}\{cc, v\}$; $\ell_{R_2^2}{:}R_2^2{:}\{c, s\}$; $\ell_{R_2^1}{:}R_2^1{:}\{cc, s\}$; $\ell_{R_2^3}{:}R_2^3{:}\{c, jd\}$ and $\Sigma^{\ell}_{E:P_2}[\mathcal{D}_2]$ is in BCNF. While $L{:}P_2$-redundant values still occur on countries and country_codes only, there are more sources (left-hand sides of FDs) for them compared to P_1. Correspondingly, our decomposition contains more schemata to eliminate the redundancies and preserve more FDs.

For $R_3 = R_{P_3} = \{jd, sp, v, c, s, cc, a_3\}$ and $\Sigma_3 = \Sigma_{E:P_3} = \{c \to cc; cc, \to c; sp \to s, v; s \to v; v \to s\}$ we obtain the BCNF decomposition $\mathcal{D}_3$ of (R_3, Σ_3):

- $R_3^1 = \{c, cc\}$ with $\Sigma_3^1 = \{c \to cc; cc \to c\}$
- $R_3^2 = \{sp, s\}$ with $\Sigma_3^2 = \{sp \to s\}$
- $R_3^3 = \{sp, v\}$ with $\Sigma_3^3 = \{sp \to v\}$
- $R_3^4 = \{s, v\}$ with $\Sigma_3^4 = \{s \to v; v \to s\}$
- $R_3^5 = \{jd, sp, c, a_3\}$ with $\Sigma_3^4 = \emptyset$.

Hence, we obtain the gUCs $\ell_{R_3^1}{:}R_3^1{:}\{c\}$; $\ell_{R_3^1}{:}R_3^1{:}\{cc\}$; $\ell_{R_3^2}{:}R_3^2{:}\{sp\}$; $\ell_{R_3^3}{:}R_3^3{:}\{sp\}$; $\ell_{R_3^4}{:}R_3^4{:}\{s\}$; $\ell_{R_3^4}{:}R_3^4{:}\{v\}$ and $\Sigma^{\ell}_{E:P_3}[\mathcal{D}_3]$ is in BCNF. Given $P_1 \cup P_2$, the additional sources for $E{:}P_2$-redundancy in countries and country_codes become obsolete again, but new schemata are required to eliminate new $E{:}P_1 \cup P_2$-redundant values on sourceID and valid_until, and preserve all FDs.

We then used Cypher to compute, for $i = 1, 2, 3$, the *Entity*:P_i-decomposition $G^{\ell}_{Entity:P_i}[\mathcal{D}_i]$ of graph G (*Offshore*). The results are summarized in Table 6. For

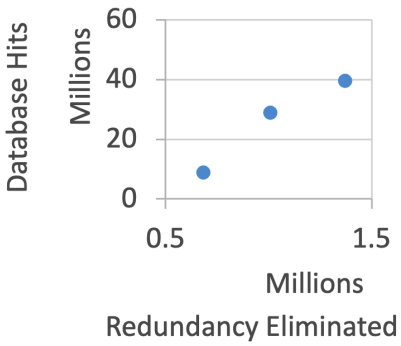
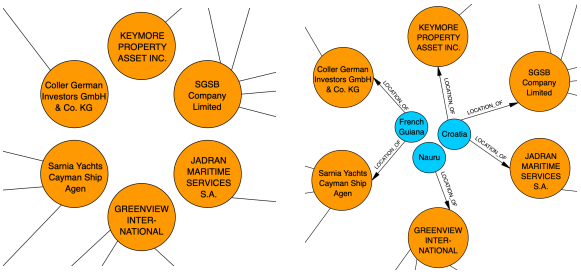


Fig. 9: Good Riddance



(a) Original Snippet (b) Normalized Snippet

Fig. 10: Illustration how redundant property values in an *Offshore* snippet are eliminated by normalization

each property set P_i we show its size $\lvert P_i\rvert$, the number #gFDs of gFDs *Entity*:$P' {:} X \to Y \in \Sigma$ such that $P' \subseteq P_i$, the number #FDs in a cover of $\Sigma_{Entity:P_i}$, the number #red of distinct redundant value occurrences in G caused by the gFDs, the number #dbhits of database hits for computing $G^{\ell}_{Entity:P_i}[\mathcal{D}_i]$, the time to compute $G^{\ell}_{Entity:P_i}[\mathcal{D}_i]$, and the size $\lvert\mathcal{D}_i\rvert$ of decomposition $\mathcal{D}_i$. Figure 9 illustrates that the graph normalization query uses its access and time effectively to eliminate redundant property value occurrences. Finally, Figure 10 shows a glimpse into the effect of normalizing *Offshore* into $L{:}P_1$-BCNF. The figure illustrates how redundant values on the property *countries* are eliminated on some nodes. In fact, the number of outgoing edges indicate for each new node (in blue) how many redundant occurrences of the values on node *country* have been eliminated.

We will now examine the effects of normalization for property graph *Northwind*. Here we choose label *Order (O)* and investigate properties *orderID (oID)*, *orderDate (oD)*, *customerID (cID)*, *shipCity (sci)*, *shipName (sn)*, *shipPostalCode (sp)*, *shipCountry (sco)*, *shipAddress (sa)* and *shipRegion (sr)*, that is, we have the property set $P = \{oID, oD, cID, sci, sn, sp, sco, sa, sr\}$ which represents our completeness requirements on nodes labelled *Order*.

The *Northwind* graph seems to be a translation from the relational model where property *orderID* constitutes a key on table *Order*, and property *customerID* serves as a key on table *Customer* that is repeated as foreign key on the table *Order*. More specifically, on nodes labelled *Order* we have the gUC $O : P : oID$. The gFD $O{:}P{:}cID \rightarrow sci, sn, sp, sco, sa, sr$ causes high levels of data redundancy as discussed earlier.

The normalization approach decomposes $R_P = P$ into $R^1 = \{cID, sci, sn, sp, sco, sa, sr\}$ and $R^2 = \{oID, oD, cID\}$ with constraint sets $\Sigma^1 = \{\ell_{R^1} : R^1 : \{cID\}\}$ and $\Sigma^2 = \{\ell_{R^2} : R^2 : \{oID\}\}$. In the original property graph *Northwind* redundant values occurred on 829 out of 830 *Order* nodes. These occurrences are eliminated by normalizing the graph, which took 845 ms and required 40,652 database hits in Neo4j.

10.6 How does integrity management improve?

We will quantify experimentally how updates of properties and node insertions can be processed more efficiently following normalization.

10.6.1 How do updates improve?

Graph normalization targets the elimination of redundant property values to avoid sources of inconsistency and minimize update overheads. Here, we will compare update performance between the original and normalized graphs for the following gFDs $L : P : X \rightarrow Y$:

- For *Offshore*: $L = \{E\}$, $P = \{sp, s, v\}$, $X = \{sp\}$, $Y = \{s, v\}$
- For *Northwind*: $L = \{O(rder)\}$, $P = \{customerID(cI), shipCity(sC), shipName(sN), shipPostalCode(sP), shipCountry(sCo), shipAddress(sA), shipRegion(sR)\}$, $X = \{cI\}$, $Y = \{sC, sN, sP, sCo, sA, sR\}$.

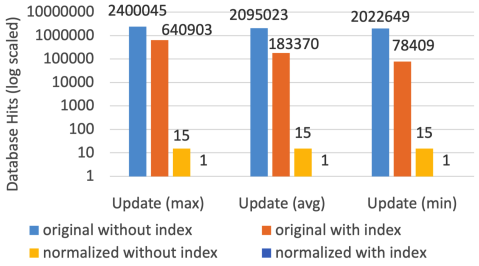
We updated the original *Offshore* graph as follows:

`MATCH` $(e : Entity)$ `WHERE EXISTS` $(e.service\_provider)$ `AND EXISTS` $(e.sourceID)$ `AND EXISTS` $(e.valid\_until)$ `AND` $e.service\_provider = \text{'Appleby'}$
`SET` $e.valid\_until = \text{'Appleby data is current through 2015'}$

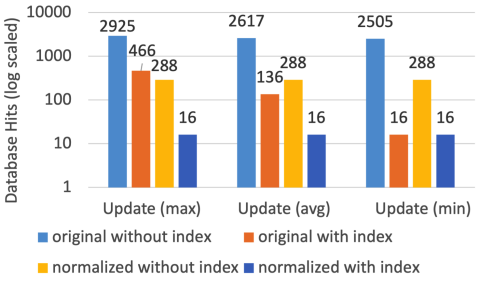
and also updated the original *Northwind* graph by:

`MATCH` $(o : Order)$ `WHERE EXISTS` $(o.customerID)$ `AND EXISTS` $(o.shipCity)$ `AND EXISTS` $(o.shipName)$ `AND EXISTS` $(o.shipPostalCode)$ `AND EXISTS` $(o.shipCountry)$ `AND EXISTS` $(o.shipAddress)$ `AND EXISTS` $(o.shipRegion)$ `AND` $o.customerID = \text{'CENTC'}$
`SET` $o.shipCountry = \text{'Estados Unidos Mexicanos'}$.

The queries were run using values for *sevice_provider* and *customerID* with the min, avg, and max number of redundant occurrences. We then performed these updates on the graphs normalized by the gFDs above, compared the number of database hits, and the runtime. We also performed the operations using an index



(a) Offshore - db hits



(b) Northwind - db hits

	redundancy		
data (index)	max	avg	min
orig, no	806 (36905)	699 (10114)	650 (4244)
orig, yes	404	100	38
norm, no	0.2 (1,001)		
norm, yes	0.2		

(c) Offshore - times in ms on Neo4j (Neptune)

	redundancy		
data (index)	max	avg	min
orig, no	1.5 (135)	0.9 (122)	0.9 (118)
orig, yes	0.4	0.3	0.2
norm, no	0.4 (98)		
norm, yes	0.2		

(d) Northwind - times in ms on Neo4j (Neptune)

Fig. 11: Update Comparison: Original vs. Normalized

for V_L on the property X. The different results can be seen in Figure 11. Normalization for *Offshore* took 6,475 ms (103,995 ms) in Neo4j (Neptune), and 494 ms (3769 ms) for *Northwind*.

Neptune queries are cloud-based, so cannot be compared to Neo4j. Important is the runtime difference between original and normalized graphs. Due to high redundancy in *Offshore*, normalization improves update performance by multiple orders of magnitude. On *Northwind*, with less redundancy, update performance still improves by an order of magnitude. The benefits already apply for normalization with a single FD. While indices result in further optimization for database hits and runtime, these are marginal compared to normalization. Normalized graphs outperform the original graph when indexed, which is similar for queries as shown next.

10.6.2 How does insertion of nodes improve?

Graph normalization benefits the processing of node insertions by reducing integrity maintenance. Consider again the datasets *Offshore* and *Northwind* with respect to the gFDs discussed before. We performed node insertion in *Offshore* using

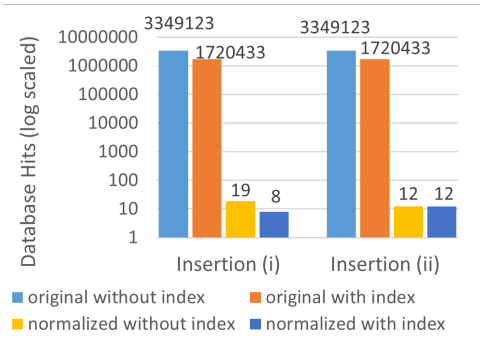
```
MERGE (e : Entity{name: 'new',
service_provider: 'Mossack Fonseca', sourceID: 'Panama Papers',
valid_until: 'The Panama Papers data is current through 2015'})
WITH e AS newnode
CALL{
MATCH (e : Entity) WHERE EXISTS (e.service_provider) AND
EXISTS (e.sourceID) AND EXISTS (e.valid_until)
WITH (e.service_provider) AS provider,
COUNT(DISTINCT(e.sourceID + e.valid_until)) AS amount
WHERE amount > 1 RETURN provider, amount
} DELETE newnode
```

which inserts a new *Entity* node, but deletes it whenever the insertion results in a violation of the corresponding gFD. Similarly, we performed

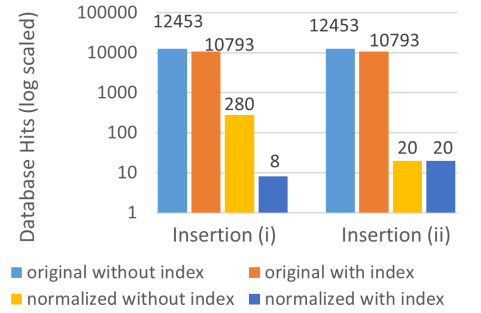
```
MERGE (o : Order{orderID: 99999, customerID: 'SAVEA',
shipCity: 'Boise', shipName: 'Save-a-lot Markets',
shipPostalCode: 83720, shipCountry: 'USA',
shipAddress: '187 Suffolk Ln.', shipRegion: 'ID'}
WITH o AS newnode
CALL{
MATCH (o : Order) WHERE EXISTS (o.customerID) AND
EXISTS (o.shipCity) AND EXISTS (o.shipName) AND
EXISTS (o.shipPostalCode) AND EXISTS (o.shipCountry) AND
EXISTS (o.shipAddress) AND EXISTS (o.shipRegion)
WITH (o.customerID) AS ids,
COUNT(DISTINCT(o.shipCity + o.shipName + o.shipPostalCode
+o.shipCountry + o.shipAddress + o.shipRegion)) AS amount
WHERE amount > 1 RETURN ids, amount
} DELETE newnode
```

on the *Northwind* dataset. In addition, we also performed the corresponding insertion operations on the normalized graphs. As the gFD from the original graph has been transformed into a gUC on the normalized graph, the latter motivates use of a `unique` index. Our results are presented in Figure 12. We see significant performance gains on both datasets for Neo4j and Amazon Neptune following normalization.

We have also analyzed the insertion of new nodes in the original graph using already existing values on properties *service_provider*, *sourceID* and *valid_until* in *Offshore* and *customerID*, *shipCity*, *shipName*, *shipPostalCode*, *shipCountry*, *shipAddress* and *shipRegion* in *Northwind*, denoted by scenario (i), and the insertion of new nodes using fresh values on these properties, denoted by scenario (ii). In the original graph there is no difference between these two scenarios. However, performance gains are already achieved when indexing the left-hand side of gFDs. For normalized graphs we see significant performance gains in both scenarios, and indexing the properties of resulting gUCs show further



(a) Offshore - db hits



(b) Northwind - db hits

	insertion scenario	
data (index)	(i)	(ii)
orig, no	984 (12128)	926 (12251)
orig, yes	581	607
norm, no	1.33 (1959)	0.8 (102)
norm, yes	0.94	0.6

(c) Offshore - times in ms on Neo4j (Neptune)

	insertion scenario	
data (index)	(i)	(ii)
orig, no	7.1 (416)	6.9 (372)
orig, yes	5.7	6.3
norm, no	0.6 (57)	0.5 (88)
norm, yes	0.5	0.4

(d) Northwind - times in ms on Neo4j (Neptune)

Fig. 12: Node insertion: Original vs. Normalized

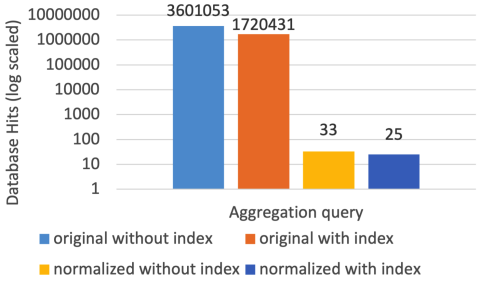
improvements, due to quicker matching of nodes for existing values in scenario (i), while the index in scenario (ii) cannot speed up the insertion as new nodes must not have values matching those on existing nodes.

10.7 How do queries improve?

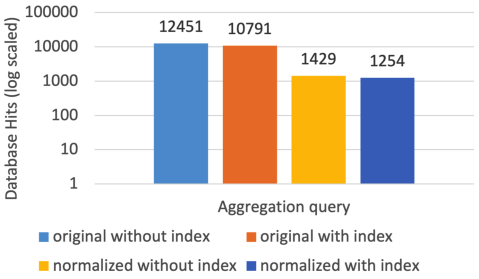
Here we investigate aggregate and distinct queries.

10.7.1 How do aggregate queries improve?

We illustrate the benefit of speeding up aggregate queries, using the parameters for node set $V_{L:P}$ from the previous section.



(a) Offshore - db hits



(b) Northwind - db hits

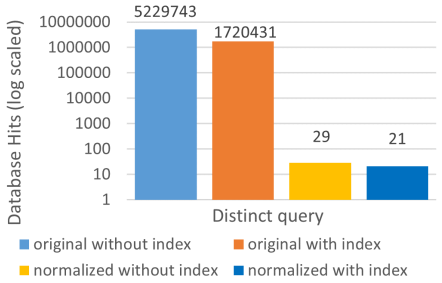
data/index	without index	with index
original	679 (9253)	414
normalized	0.3 (7022)	0.2

(c) Offshore - times in ms on Neo4j (Neptune)

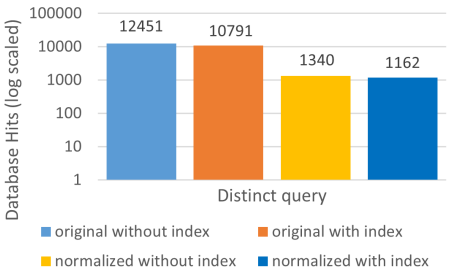
data/index	without index	with index
original	3 (128)	2.9
normalized	0.5 (113)	0.5

(d) Northwind - times in ms on Neo4j (Neptune)

Fig. 13: Aggregate Queries: Original vs. Normalized



(a) Offshore - db hits



(b) Northwind - db hits

data/index	without index	with index
original	932 (5647)	288
normalized	0.4 (52)	0.2

(c) Offshore - times in ms on Neo4j (Neptune)

data/index	without index	with index
original	3.1 (147)	1.9
normalized	1.4 (53)	1.0

(d) Northwind - times in ms on Neo4j (Neptune)

Fig. 14: `Distinct` Queries: Original vs. Normalized

As typical aggregate queries, we access information on the numbers of orders associated with a given *customerID* in *Northwind*, and on the numbers of entities for a given *service_provider* in *Offshore*:

```
MATCH (e : Entity) WHERE EXISTS (e.service_provider) AND
EXISTS (e.sourceID) AND EXISTS (e.valid_until)
WITH (e.service_provider) AS provider, COUNT(*) AS amount
RETURN min(amount), max(amount), avg(amount),
```

and

```
MATCH (o : Order) WHERE EXISTS (o.customerID) AND
EXISTS (o.shipCity) AND EXISTS (o.shipName) AND
EXISTS (o.shipPostalCode) AND EXISTS (o.shipCountry) AND
EXISTS (o.shipAddress) AND EXISTS (o.shipRegion)
WITH o.customerID AS orders, COUNT (*) AS amount
RETURN min(amount), max(amount), avg(amount) .
```

We compare their performance to corresponding queries on the normalized graph. Results are shown in Figure 13, including those after introducing an index for V_L on the property X. Normalization improves query performance by several orders of magnitude on *Offshore*, and one order of magnitude on *Northwind*. The index does improve performance, but has not as big an impact as for updates.

10.7.2 How do DISTINCT queries improve?

Graph normalization can also speed up queries using *DISTINCT* which remove duplicates from the output. For this we retrieved the number of unique *service_provider* in *Offshore* using the query

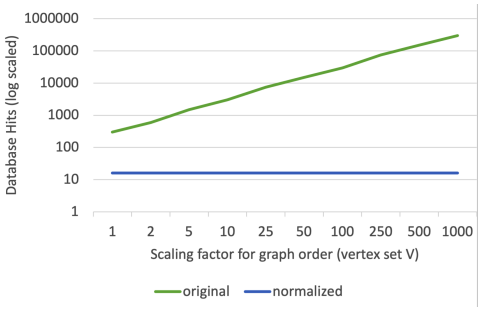
```
MATCH (e : Entity) WHERE EXISTS (e.service_provider) AND
EXISTS (e.sourceID) AND EXISTS (e.valid_until)
RETURN DISTINCT (e.service_provider),
```

and the amount of *customerID* in *Northwind* using

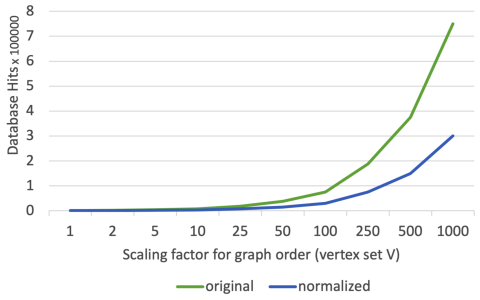
```
MATCH (o : Order) WHERE EXISTS (o.customerID) AND
EXISTS (o.shipCity) AND EXISTS (o.shipName) AND
EXISTS (o.shipPostalCode) AND EXISTS (o.shipCountry) AND
EXISTS (o.shipAddress) AND EXISTS (o.shipRegion)
RETURN DISTINCT (o.customerID)
```

on the original graphs. The use of `DISTINCT` is important to avoid counting duplicates.

The efficiency of these queries were compared to corresponding queries on the normalized *Offshore* graph where the property *service_provider* is unique on *Provider* nodes. Similarly, *customerID* is unique on



(a) Updates



(b) Aggregate Queries

Fig. 15: Scaling Comparison: Original vs. Normalized

Customer nodes in *Northwind*. This results in faster execution as fewer values need to be counted and the use of `DISTINCT` becomes redundant.

Figure 14 shows the results of these experiments. An index on the original graph already leads to performance gains, but these are outperformed by graph normalization. Indeed, the gUC resulting from normalization introduces a `unique` index that makes additional filtering for distinct values redundant.

10.8 How do the benefits of normalization scale?

We will report how graph size impacts on update and aggregate query performance, but also on the validation of our constraints and their features. We utilized synthetic datasets as follows.

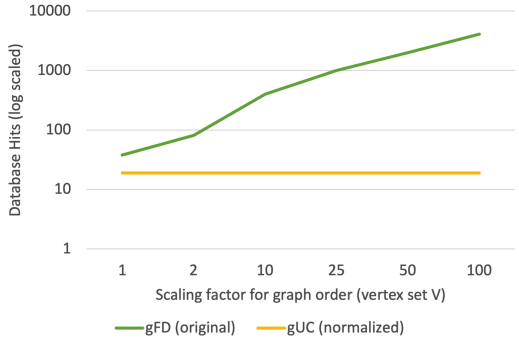
We created graphs that consist of a node labelled *Company* with edges to *Employee*-nodes that have properties *name*, *department* and *manager*, with additional properties for some experiments. Our underlying business rule says that every department has at most one manager, resulting in the gFD $\varphi = \{Employee\} : \{department, manager\} : \{department\} \rightarrow \{manager\}$. For each experiment, we perform a query on the same baseline graph and scale this graph by factor k to have k times as many *Employee*-nodes while keeping the number of departments fixed.

Figure 15 compares the performance of (a) updating manager names, and (b) querying the minimum, average and maximum number of employees per department, both between the original and normalized graph (with respect to gFD φ), respectively. In particular, Figure 15(a) conveys the main message that normalization scales update performance perfectly. Indeed, access to the normalized graph using gUCs remains constant while access to the original graph using gFDs keeps on growing. For aggregate queries the performance improvement is also very noticeable.

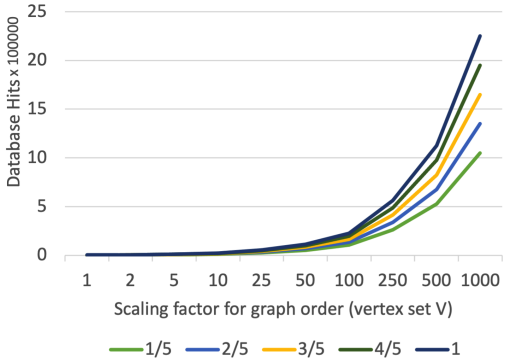
Figure 16(a) underlines the perfect scalability of validating gUCs resulting from gFD φ on the normalized graph in contrast to growing access necessary for validating φ on the original graph. From (b) it can be seen how validation performance scales with the ratio of nodes that have all properties in P. From (c) we observe how validation performance scales in the size of the underlying property set P. Indeed, Pi contains $i+1$ properties. Finally, (d) shows how validation of φ scales in the node selectivity of labels in φ. Indeed, the database hits required are directly proportional to the number of nodes with the given label set present.

11 Conclusion and Future Work

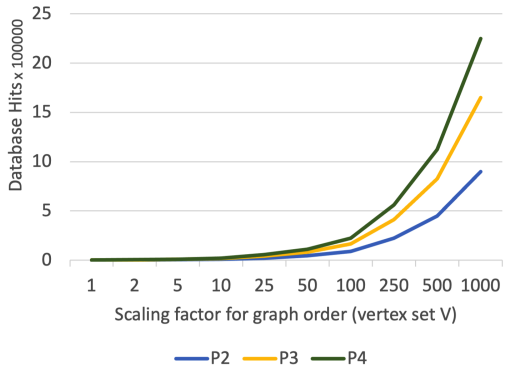
Our research is the first to address the challenging area of normalizing property graphs. Challenges include the unavailability of a schema, the desire to customize normalization of property graphs to flexible requirements of applications, the robustness of normalization under different interpretations of missing properties, the abilities to express and eliminate many redundant property values, and to transfer achievements of BCNF and 3NF from relational databases to property graphs. Indeed, we have turned these challenges into an opportunity by enabling our class of graph-tailored functional dependencies to express application-specific requirements for node labels and properties; plus specifying their semantics to be robust under different interpretations of missing property values. Having created this opportunity, we have then transferred comprehensive achievements from relational databases to property graphs, including BCNF, 3NF, and the State-of-the-Art algorithm that returns a lossless, dependency-preserving BCNF decomposition whenever possible. Our experiments with real-world graph data illustrate how our constraints capture many redundant property value occurrences and potential inconsistency, and how our algorithms transform graphs to eliminate/minimize them. Our experiments have further demonstrated the efficacy of property graph normalization. Indeed, the reduction of overheads for update maintenance and the speed up of



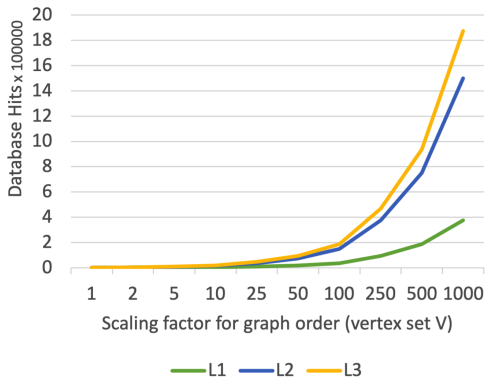
(a) gFDs vs gUCs



(b) Ratio of P-complete Nodes



(c) Size of P



(d) Node-selectivity of L

Fig. 16: Validation at Scale

aggregate queries by orders of magnitude, and the effort required to normalize the property graph are all proportional to the amount of redundancy removed.

In future work, we will address other classes of constraints and normal forms. We will also initiate research on *conditional normalization*, employing conditional versions of constraints [16] to graph normalization. Finally, we will address data-driven normalization by combining dependency discovery [17,20,36,47,45,46,50] with graph normalization.

References

1. R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, A. Green, J. Hidders, B. Li, L. Libkin, V. Marsault, W. Martens, F. Murlak, S. Plantikow, O. Savkovic, M. Schmidt, J. Sequeda, S. Staworko, D. Tomaszuk, H. Voigt, D. Vrgoc, M. Wu, and D. Zivkovic. Pg-schema: Schemas for property graphs. *Proc. ACM Manag. Data*, 1(2):198:1–198:25, 2023.
2. R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, K. W. Hare, J. Hidders, V. E. Lee, B. Li, L. Libkin, W. Martens, F. Murlak, J. Perryman, O. Savkovic, M. Schmidt, J. F. Sequeda, S. Staworko, and D. Tomaszuk. PG-keys: Keys for property graphs. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 2423–2436, 2021.
3. M. Arenas. Normalization theory for XML. *SIGMOD Record*, 35(4):57–64, 2006.
4. W. W. Armstrong. Dependency structures of data base relationships. In *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974*, pages 580–583, 1974.
5. C. Batini and A. Maurino. Design for data quality. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems, Second Edition*. 2018.
6. C. Beeri and P. A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Trans. Database Syst.*, 4(1):30–59, 1979.
7. P. A. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Trans. Database Syst.*, 1(4):277–298, 1976.
8. J. Biskup, U. Dayal, and P. A. Bernstein. Synthesizing independent database schemas. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, USA, May 30 - June 1*, pages 143–151, 1979.
9. A. Bonifati, G. H. L. Fletcher, H. Voigt, and N. Yakovets. *Querying Graphs*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018.
10. E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.
11. E. F. Codd. Further normalization of the database relational model. In *Courant Computer Science Symposia 6: Data Base Systems*, pages 33–64, 1972.
12. W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.*, 1(3):267–284, 1984.
13. R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst.*, 2(3):262–278, 1977.
14. R. Fagin. A normal form for relational databases that is based on domains and keys. *ACM Trans. Database Syst.*, 6(3):387–415, 1981.

15. W. Fan. Dependencies for graphs: Challenges and opportunities. *ACM J. Data Inf. Qual.*, 11(2):5:1–5:12, 2019.
16. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.*, 33(2):6:1–6:48, 2008.
17. W. Fan, C. Hu, X. Liu, and P. Lu. Discovering graph functional dependencies. *ACM Trans. Database Syst.*, 45(3):15:1–15:42, 2020.
18. W. Fan and P. Lu. Dependencies for graphs. *ACM Trans. Database Syst.*, 44(2):5:1–5:40, 2019.
19. W. Fan, Y. Wu, and J. Xu. Functional dependencies for graphs. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 1843–1857, 2016.
20. M. Hannula, Z. Zhang, B. Song, and S. Link. Discovery of cross joins. *IEEE Trans. Knowl. Data Eng.*, 35(7):6839–6851, 2023.
21. E. E. Ian Robinson, Jim Webber. *Graph Databases*. O'Reilly Media, 2015.
22. C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Extending existing dependency theory to temporal databases. *IEEE Trans. Knowl. Data Eng.*, 8(4):563–582, 1996.
23. H. Köhler and S. Link. SQL schema design: Foundations, normal forms, and normalization. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 267–279, 2016.
24. H. Köhler and S. Link. SQL schema design: foundations, normal forms, and normalization. *Inf. Syst.*, 76:88–113, 2018.
25. S. Kolahi and L. Libkin. An information-theoretic analysis of worst-case redundancy in database design. *ACM Trans. Database Syst.*, 35(1):5:1–5:32, 2010.
26. M. Levene and G. Loizou. Database design for incomplete relations. *ACM Trans. Database Syst.*, 24(1):80–125, 1999.
27. M. Levene and M. Vincent. Justification for inclusion dependency normal form. *IEEE Trans. Knowl. Data Eng.*, 12(2):281–291, 2000.
28. S. Link, H. Köhler, A. Gandhi, S. Hartmann, and B. Thalheim. Cardinality constraints and functional dependencies in SQL: taming data redundancy in logical database design. *Inf. Syst.*, 115:102208, 2023.
29. S. Link and H. Prade. Relational database schema design for uncertain data. *Inf. Syst.*, 84:88–110, 2019.
30. S. Link and Z. Wei. Logical schema design that quantifies update inefficiency and join efficiency. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 1169–1181, 2021.
31. H. Mannila and K. Räihä. *Design of Relational Databases*. Addison-Wesley, 1992.
32. S. Mennicke. Modal schema graphs for graph databases. In *Conceptual Modeling - 38th International Conference, ER 2019, Salvador, Brazil, November 4-7, 2019, Proceedings*, pages 498–512, 2019.
33. W. Y. Mok. Utilizing nested normal form to design redundancy free JSON schemas. *Int. J. Recent Contributions Eng. Sci. IT*, 4(4):21–25, 2016.
34. W. Y. Mok, Y. Ng, and D. W. Embley. A normal form for precisely characterizing redundancy in nested relations. *ACM Trans. Database Syst.*, 21(1):77–106, 1996.
35. S. L. Osborn. Testing for existence of a covering boyce-codd normal form. *Inf. Process. Lett.*, 8(1):11–14, 1979.
36. T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proc. VLDB Endow.*, 8(10):1082–1093, 2015.
37. L. C. Shimomura, G. Fletcher, and N. Yakovets. Ggds: Graph generating dependencies. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 2217–2220, 2020.
38. P. Skavantzos, U. Leck, K. Zhao, and S. Link. Uniqueness constraints for object stores. *ACM J. Data Inf. Qual.*, 15(2):13.1–13.29, 2023.
39. P. Skavantzos and S. Link. Normalizing property graphs. *Proc. VLDB Endow.*, 16(11):3031–3043, 2023.
40. P. Skavantzos, K. Zhao, and S. Link. Uniqueness constraints on property graphs. In *Advanced Information Systems Engineering - 33rd International Conference, CAiSE 2021, Melbourne, VIC, Australia, June 28 - July 2, 2021, Proceedings*, pages 280–295, 2021.
41. Z. Tari, J. Stokes, and S. Spaccapietra. Object normal forms and dependency constraints for object-oriented schemata. *ACM Trans. Database Syst.*, 22(4):513–569, 1997.
42. M. Vincent. A corrected 5NF definition for relational database design. *Theor. Comput. Sci.*, 185(2):379–391, 1997.
43. M. Vincent and M. Levene. Restructuring partitioned normal form relations without information loss. *SIAM J. Comput.*, 29(5):1550–1567, 2000.
44. R. D. Virgilio, A. Maccioni, and R. Torlone. Model-driven design of graph databases. In *Conceptual Modeling - 33rd International Conference, ER 2014, Atlanta, GA, USA, October 27-29, 2014. Proceedings*, pages 172–185, 2014.
45. Z. Wei, S. Hartmann, and S. Link. Discovery algorithms for embedded functional dependencies. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 833–843, 2020.
46. Z. Wei, S. Hartmann, and S. Link. Algorithms for the discovery of embedded functional dependencies. *VLDB J.*, 30(6):1069–1093, 2021.
47. Z. Wei, U. Leck, and S. Link. Discovery and ranking of embedded uniqueness constraints. *Proc. VLDB Endow.*, 12(13):2339–2352, 2019.
48. Z. Wei and S. Link. Embedded functional dependencies and data-completeness tailored database design. *Proc. VLDB Endow.*, 12(11):1458–1470, 2019.
49. Z. Wei and S. Link. Embedded functional dependencies and data-completeness tailored database design. *ACM Trans. Database Syst.*, 46(2):7:1–7:46, 2021.
50. Z. Wei and S. Link. Towards the efficient discovery of meaningful functional dependencies. *Inf. Syst.*, 116:102224, 2023.
51. Z. Zhang, W. Chen, and S. Link. Composite object normal forms: Parameterizing boyce-codd normal form by the number of minimal keys. *Proc. ACM Manag. Data*, 1(1):13:1–13:25, 2023.