LICS: Towards Theory-Informed Effective Visual Abstraction of Property Graph Schemas

KASIDIS CHANTHATROJWONG, Nanyang Technological University, Singapore
SOURAV S BHOWMICK, Nanyang Technological University, Singapore
BYRON CHOI, Hong Kong Baptist University, Hong Kong SAR, China

Property graph schemas are essential for organizing property graph data, serving both prescriptive and descriptive roles. This has led to the recent development of property graph schema languages such as PG-Schema. While understanding of these languages requires familiarity with complex syntax, this poses usability challenges, particularly for domain experts who are not programmers. Current visual abstractions, such as the *labeled schema graph* (LSG), simplify representation but suffers from visual clutter and limited feature support. To address these challenges, we propose a novel, *generic*, and *extensible* visual abstraction, *labeled iconized composite schema* (LICS), whose design is informed by theories and principles from HCI, cognitive psychology, and visualization. A novel LICS-based visual interface coined PASCAL is also proposed to facilitate visualization of property graph schemas. Under the hood, it leverages the *Map-Paint* algorithm for creating the visual components of LICS. A user study demonstrates that LICS is superior to the traditional LSG abstraction w.r.t. usability, effectiveness, query formulation efficiency, and schema comprehension.

CCS Concepts: • **Human-centered computing** → *Visualization systems and tools*; • **Information systems** → *Query languages.*

Additional Key Words and Phrases: Visual abstraction, property graph schema, cognitive psychology, visualization, human-computer interaction, theories, principles, theory-informed design

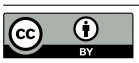
ACM Reference Format:
Kasidis Chanthatrojwong, Sourav S Bhowmick, and Byron Choi. 2025. LICS: Towards Theory-Informed Effective Visual Abstraction of Property Graph Schemas. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 180 (June 2025), 26 pages. https://doi.org/10.1145/3725410

1 Introduction

The property graph abstraction is increasingly used to represent relationships between real-world entities in fields such as social networks, finance, biology, cybersecurity, and transportation. It consists of nodes (representing entities) and relationships (indicating connections between entities). Nodes can have multiple optional labels, and relationships are directed, connecting a source node to a target node with a specific type. Additional data is stored as properties associated with both nodes and relationships.

Schemas are essential for structuring data systems, and a property graph schema language is crucial for organizing property graph data. This language can have a *prescriptive* role by enforcing constraints on data modifications and a *descriptive* role by guiding users or systems on data expectations. A recent survey [52] revealed that insufficient support for property graph schemas is a common challenge for developers. This has led to increased research in this area. Notably, Angles

Authors' Contact Information: Kasidis Chanthatrojwong, Nanyang Technological University, Singapore, Singapore, kasidis001@e.ntu.edu.sg; Sourav S Bhowmick, Nanyang Technological University, Singapore, Singapore, assourav@ntu.edu.sg; Byron Choi, Hong Kong Baptist University, Hong Kong, Hong Kong SAR, China, bchoi@comp.hkbu.edu.hk.



ACM 2836-6573/2025/6-ART180
https://doi.org/10.1145/3725410

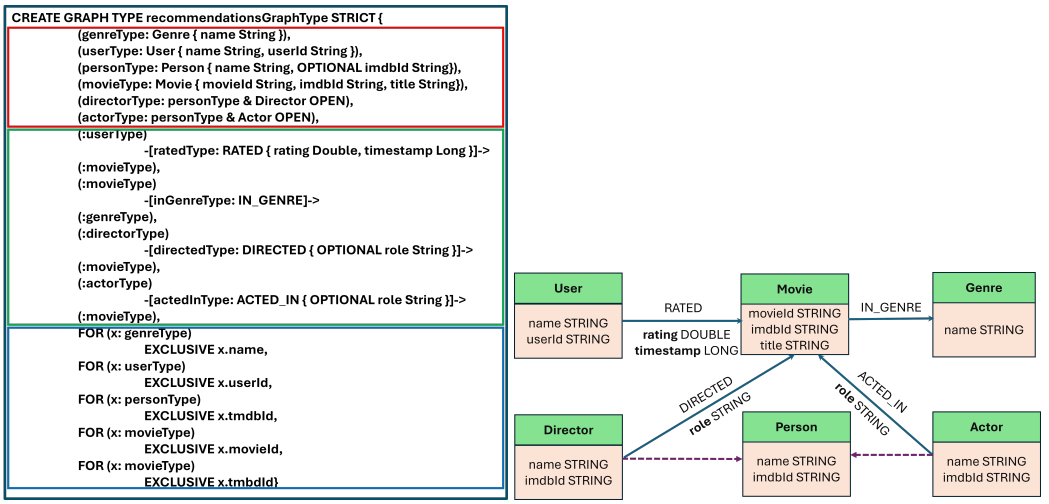


Fig. 1. [Best viewed in color] (left) A property graph schema; (right) A labeled schema graph (LSG) of the schema.

et al. [8] introduced the first unified property graph schema language, called PG-Schema, which provides a formal framework for defining property graph schemas. An example of PG-Schema is given in Figure 1 (left).

Understanding a property graph schema requires familiarity with the syntax and semantics of schema languages like PG-Schema, which poses a challenge since many end users are domain experts but not programmers. This makes it impractical to expect users to grasp complex schema languages. In fact, usability is a significant concern in graph processing as highlighted in [52]. Despite this, research on schema languages has mostly concentrated on improving expressiveness rather than usability.

A popular approach to address the usability challenges of schema languages is through visual abstraction. Sahu *et al.* [52] found that visualization was the most popular non-query tasks for users of graph data management systems. At first glance, it may seem that the popular *labeled schema graph* (LSG) abstraction [8] can serve the visualization need, where node types/labels are depicted using basic shapes (*e.g.,* rectangles), and edge types/labels and inheritance are shown with directed labeled edges. Properties and constraints for node types can be displayed as text within the shapes or accessed on demand, while edge properties are similarly represented.

Despite its simplicity, the LSG abstraction has two significant limitations. First, using text labels for nodes and edges can lead to visual congestion and clutter, undermining the theories and principles of visualization and cognitive psychology (*e.g.,* Gestalt principles [3, 49], visual complexity theories [40]). Additionally, hiding annotations by default limits users' understanding of the property graph schema. Second, the LSG only visually represents a limited subset of schema features, such as node and edge types and inheritance, while other features like constraints, *label expressions*, and the concept of *arbitrary labels/properties* lack corresponding visual encoding. This makes it challenging for users to achieve a comprehensive visualization of a property graph schema.

Example 1.1. Figure 1 (right) depicts the LSG representation of the schema in Figure 1 (left), using a modified version of the LSG abstraction from [8]. Node labels are displayed within rectangles, while edge labels and inheritance are represented by directed solid and dotted arrows, respectively. The visual encoding in this representation is limited to node/edge labels and inheritance, with other

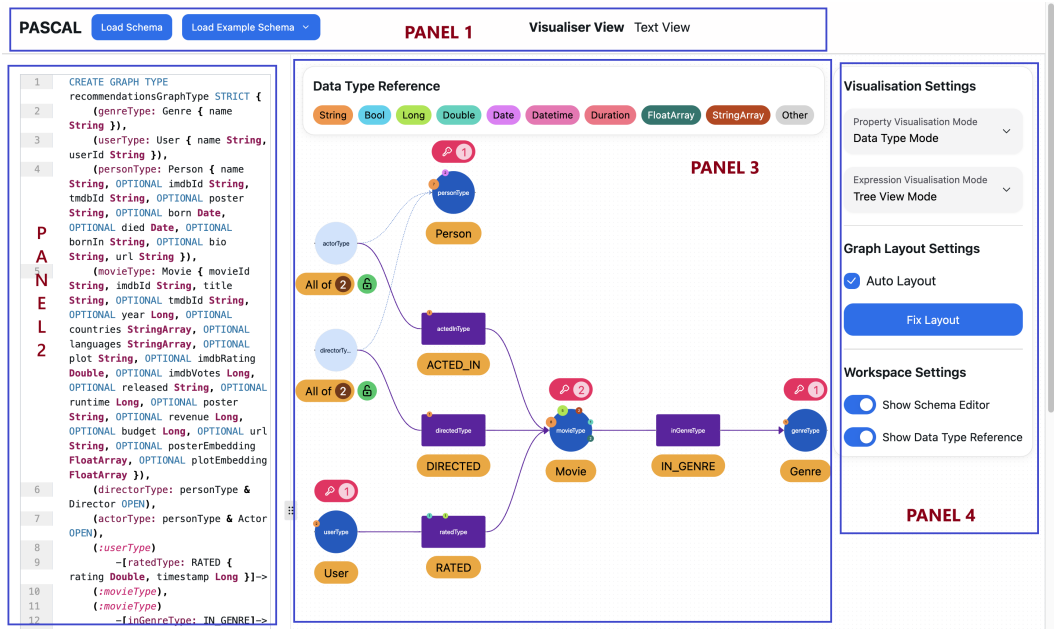


Fig. 2. [Best viewed in color] GUI of PASCAL.

schema features such as constraints, node and edge types, properties, and arbitrary labels/properties (OPEN keyword) not being visually encoded. Some features, like properties, are represented as text (*e.g.,* name STRING), while others (*e.g.,* node and edge types, constraints) are not included in the LSG. Notably, using text to represent these features in the LSG can lead to visual congestion, which worsens as the schema size and complexity grow. For example, displaying multiple properties of the Movie node or explicitly showing node/edge types (*e.g.,* movieType, ratedType) and constraints (*e.g.,* bottom rectangle in Figure 1 (left)) using text would significantly increase clutter in Figure 1 (right).

While there are recent initiatives to visualize relational and property graph queries [24, 25, 32, 34, 35, 38], we believe there has been no systematic study to visualizing property graph schema languages. Visualizing property graph schemas is particularly challenging compared to queries because schemas encompass a much richer set of features as highlighted in Example 1.1.

In this paper, we present a novel, *generic*, and *extensible* visual abstraction called *Labeled Iconized Composite Schema* (LICS) for property graph schemas to address the aforementioned challenges. Specifically, LICS is aimed at enhancing the representation of various schema features such as *node* and *edge types*, *inheritance*, *constraints*, and *label expressions*. Unlike the traditional LSG abstraction, LICS employs composite geometric *shapes* and *icons* to effectively depict these elements, guided by principles from human-computer interaction (HCI), visualization, and cognitive psychology. An example of the LICS abstraction is depicted in Figure 2 (Panel 3). Observe that in contrast to the LSG representation (Figure 1), it contains less textual data and a greater variety of visual encodings.

We present a novel algorithm, Map-Paint, which implements the LICS abstraction in a systematic way. It utilizes a set of *LICS operators* to create and manipulate the LICS representation of a property graph schema. At a high level, the *map* operator assigns different schema components from a schema language to their respective *shape buckets* based on the visual encoding applied. Following this, the *paint* operator "paints" these buckets according to established theories and principles to

fully realize the LICS abstraction. To demonstrate the effectiveness of this abstraction, we developed an innovative LICS-based visual interface for a representative property graph schema language (*i.e.,* PG-Schema) coined PASCAL (Property grAph SChema VisuALizer). We undertake a comprehensive user study to demonstrate the benefits of LICS and PASCAL.

Our work builds on the recent efforts by Ma *et al.* [34, 35], who introduced the *labeled composite graph* (LCG) abstraction for visually representing nodes, links, and predicates in conjunctive property graph queries. Their design is also informed by principles and theories from HCI, psychology, and visualization. The novelty of LICS compared to this foundational work is as follows: First, LICS visually encodes schema elements such as constraints, inheritance, label expressions, and arbitrary labels and properties, which are not considered in LCG (detailed in Sec. 3.4). Second, LICS incorporates *theories of icons* [26, 45, 50] to represent several features of property graph schemas, an approach not utilized in LCG. Third, LICS exploits a novel Map-Paint paradigm for generating its various components, whereas LCG relies on a visual shape definition language called VEDA to manipulate shapes within an LCG.

In summary, this paper makes the following key contributions.

- We propose a new visual abstraction called *labeled iconized composite schema* (LICS) for property graph schemas, informed by cognitive psychology, HCI, and visualization principles (Section 3). To the best of our knowledge, it is the most comprehensive abstraction to date for visually representing property graph schemas. We also introduce a novel graphical user interface, PASCAL, that visualizes these schemas.
- We present a novel and efficient algorithm, Map-Paint, for systematically creating LICS components, focusing on balancing *visual complexity* with a richer support of schema features (Section 4). For the first time, it introduces a *map-paint* paradigm, where schema components are *mapped* to a specific data structure (*i.e.,* shape buckets), which are then *painted* onto a visual interface (a.k.a GUI) using a set of operators.
- We conduct a user study to demonstrate that the LICS abstraction enhances usability, schema comprehension and more efficient property graph query formulation compared to the traditional LSG abstraction (Section 5).

2 Background

In this section, we first give a brief introduction to PG-Schema [8], which we use as the representative schema language of property graphs as it is expressive, feature-rich, and versatile. Next, we briefly introduce relevant cognitive psychology, HCI, and visualization principles and theories that guide the design of the LICS abstraction.

2.1 PG-Schema: An Overview

PG-Schema is a property graph schema proposal aiming to influence the development of Data Definition Language (DDL) for the specification of schema of property graphs [8]. It mainly consists of two parts: PG-Types and PG-Keys. PG-Types define the specification for the combination of labels and properties in nodes and edges. It also supports abstraction and inheritance. PG-Keys [9], on the other hand, specify *key* and *participation constraints* in PG-Schema. We first briefly describe PG-Types which involves *node types*, *edge types*, *graph types*, *inheritance* and *abstract types*. The focus is on the types currently supported in LICS.

Figure 1 (left) illustrates a PG-Schema with two main components: Node Types (top box) and Edge Types (middle box). Node Types are categorized as *base* or *inherited*. A base Node Type, like PersonType, represents a basic entity, such as a person, with a label Person, required properties (*e.g.,* name of type STRING) and optional properties (*e.g.,* imdbId of the same type). Edge Types also come in *base* and *inherited* forms, with an example being the directedType edge that connects

Table 1. Principles and theories that inform the design of key features of the LICS abstraction and the PASCAL GUI.

ID	Principles & Theories	Representative features of LICS	Representative features of PASCAL GUI
Gestalt Principles [3, 49]			
T1	Law of proximity	(a) Labels and constraints are positioned closely to their Node or Relationship Type. (b) Data type or property circles are positioned along the Node Type's circumference or Relationship Type's perimeter.	The content in this cell matches the content in the adjacent left cell and involves the *Schema Canvas* panel. In addition, color codes of different data types are grouped together at the top of the *Schema Canvas* panel.
T2	Law of similarity	(a) Node Types are all blue circles. (b) Relationship Types are all purple rectangles. (c) Relationship Endpoints are all purple arrows. (d) Labels are all yellow pills. (e) Constraints are all red pills. (f) Data type and property circles are all circles and colour-coded based on their data type. (g) Node inheritances are all blue dotted arrows. (h) Relationship inheritance are all purple dotted arrows. (i) OPEN keywords are all green circles with an unlock icon.	The contents in these cells match the content in the corresponding adjacent left cells and involve the *Schema Canvas* panel.
T3	Law of closure	Circular and rectangular shapes for Node and Relationship Types.	
T4	Law of continuity	(a) Data Type or Property circles are positioned along the Node Type's circumference or Relationship Type's perimeter. (b) bézier curves are used for all arrows.	
T5	Law of symmetry	Data type and property circles are evenly spaced out along the Node Type's circumference or Relationship Type's perimeter.	
T6	Law of common fate	(a) Labels, properties, and OPEN keywords move along with their associated Node or Relationship Types. (b) Relationship endpoints and inheritances are all left-to-right by default.	
Visual Complexity Theories and Principles			
T7	Quantity of information [19, 22, 40, 43, 46, 58]	The numbers of property and data type circles and color codes are kept to minimum to sufficiently convey relevant information visually.	The content in this cell matches the content in the adjacent left cell and involves the *Schema Canvas* panel. In addition, details of properties/relationships/constraints are exposed on-demand.
T8	Variety of visual form [19, 40, 46, 48, 51]	All components of same type have a consistent shape (circle, rectangle, or pill). Colors are used to facilitate visual correlation. Properties are color coded based on their data types instead of semantics to mitigate the impact of color variety. Size dissimilarity is only used to inform differences in the number of properties of a specific data type.	The content in this cell matches the content in the adjacent left cell and involves the *Schema Canvas* panel. Consistent text format for property graph schema in the left panel.
T9	Spatial organization [40, 43, 46]	Positioning and alignment of property and data type circles along circumference/perimeter to maintain symmetry.	The content in these two cells match the contents in the corresponding adjacent left cells and involve the *Schema Canvas* panel.
T10	Perceivability of details [39, 40]	Visual congestion is avoided by maintaining sufficient distance between various visual components, concealment of details, color contrast w.r.t. the background.	
Visualization Theories and Principles [36]			
T11	Expressiveness criteria [36]	The circles, arrows, icons, and colors capture exactly the facts, no more, no less.	The contents in these three cells match the contents in the corresponding adjacent left cells and involve the *Schema Canvas* panel.
T12	Effectiveness criteria [17, 36]	Uses color, position, and shapes which are high/medium-ranked in perceptual tasks list of nominal data type.	
T13	Principle of Importance Ordering [36]	Area and nominal values are used to convey differences in the number of properties.	
Theories and Principles of Icons [26, 45, 50, 53]			
T14	Recognition over recall [26, 50]	The unlock, key, and the connect icons are easily recognizable and represents commonly understood function.	The contents in these four cells match the contents in the adjacent left cells and involve the *Schema Canvas* panel.
T15	Consistence and standard [26, 53]	The three icons are used consistently in all property graph schemas.	
T16	Simplicity and clarity [26, 50, 53]	The unlock, key, and the connect symbols are simple and not visually complex.	
T17	Affordance [45] & semantic understanding [26]	The unlock icon conveys "open", the key icon indicates constraints such as primary key, and the connect icon conveys relationship-related constraints.	
HCI Principles for Interface Design [53]			
T18	Strive for consistency	Consistent representation of all visual components of LICS.	Consistent representation of all visual components in the *Schema Canvas* panel.
T19	Offer informative feedback		The *Schema Editor* displays a list of error messages that show the cause of the parsing error.
T20	Support internal locus of control	Certain details are only exposed on-demand.	Users of PASCAL are the initiators of actions during schema visualization.
T21	Reduce short-term memory load		*Data Type Panel* eliminates the need to remember the color coding of each data type.

nodes of types directorType and movieType using the label DIRECTED. Node and Edge Types can have multiple labels defined using *label expressions*, combining mandatory and optional labels with operators like & (and) and | (choice). By default, types are *closed*, meaning only specific labels

are allowed, but the OPEN keyword can enable arbitrary labels (*e.g.,* directorType), including for properties.

PG-Schema supports inheritance, allowing new *inherited* types to reuse properties from parent types while adding new specifications. For example, directorType and actorType inherit from personType, gaining its labels and properties, and can also include additional labels using the OPEN keyword. If a base type is open, the inherited type is automatically open; otherwise, if the base types are closed, the inherited type must be explicitly declared as open.

PG-Schema extends PG-Keys [9] to define constraints. There are three qualifiers available in PG-Schema: EXCLUSIVE, MANDATORY, and SINGLETON. FOR $p(x)$ EXCLUSIVE $q(x, \overline{y})$ means two different x of $p(x)$ should not share the same $\overline{y}$ that satisfies $q(x, \overline{y})$. FOR $p(x)$ MANDATORY (*resp.* SINGLETON) $q(x, \overline{y})$ means for each x of $p(x)$, there should be at least (*resp.* most) one $\overline{y}$ that satisfies $q(x, \overline{y})$. A constraint can use one or more qualifiers together. These can be used to define key, foreign key, and participation constraints [8]. The bottom box in Figure 1 (left) shows constraints defined on the four types.

2.2 Relevant Theories & Principles

A visual representation of property graph schemas should enable users to quickly grasp and find information while also being visually pleasing. To achieve this, its design must draw on theories and principles from HCI, visualization, and cognitive psychology. Given the variety of theories available, the selection should focus on those that address visual perception, usability, and their relevance in related research and practice. This section outlines and justifies the chosen theories and principles for this work.

Gestalt Principles [3, 49]. They are well-known theories for visual perception and explain how humans organize similar elements, detect patterns, and simplify complex images. These principles are crucial for creating visually appealing and easily understandable designs. Gestalt principles are widely used in aesthetic web design [18] and more recently in query visualization [34, 38]. For instance, the law of *proximity* suggests that objects that are close to one another are perceived as a group. The law of *similarity* indicates that elements that resemble each other in shape, color, or shading are grouped together perceptually. The law of *closure* means that people tend to perceive incomplete objects as complete, focusing on the whole rather than the gaps. The law of *symmetry* states that the mind perceives objects as symmetrical and centered around a midpoint, which helps in connecting two symmetrical but separate elements into a unified shape. The law of *continuity* suggests that elements aligned within an object are grouped together, forming perceptual wholes. Finally, the law of *common fate* states that elements moving in the same direction or at the same speed are perceived as a group or a single unit.

Principles and Theories of Visual Complexity [40]. Visual complexity heightens cognitive load [27] while diminishing usability and aesthetics [39, 57]. As schemas are often displayed on visual interfaces, we draw on recent research in HCI and graphical user interface (GUI) design to define and characterize *visual complexity*. This complexity is operationally defined by a combination of factors, including the *amount of information*, *variety of visual form*, *spatial organization*, and the *perceivability of details* [40].

The *amount of information* is the most common aspect of visual complexity: more information units on the screen generally make it appear more complex to users, a phenomenon known in psychology as the *set-size effect* [58]. *Variety of visual form* includes elements such as diversity in shapes, colors, sizes, and textures [19, 46, 48]. An increase in these visual features contributes to greater complexities [51]. *Spatial organization* pertains to how structural repetition and regular positioning can simplify the presentation of information [40]. *Perceivability of details* concerns the

Table 2. Visual encoding schemes in LICS and comparison with LSG and LCG. 'N.A' means 'Not Applicable'. The symbol △ indicates partial visual support of the feature and ⊙ means the visual encoding has a different semantics from LICS.

ID	Schema Feature	Shape	On Hover	Color	Approach		
					LICS	LSG [8]	LCG [34]
F1	Node Type	Circle	N.A.	Blue (Base), Light Blue (Inherited)	✓	✘	✘
F2	Edge Type	Rectangle	N.A.	Purple (Base), Light Purple (Inherited)	✓	✘	✘
F3	Edge Source	Line or bézier curve	N.A.	Purple	✓	✓	✓
F4	Edge Target	Line or bézier curve	N.A.	Purple	✓	✓	✓
F5	Label	Pill	*Tree View*	Yellow	✓	△	△
F6	Key Constraint	Pill with key icon	NL desc.	Red	△	✘	✘
F7	Participation Constraint	Pill with connect icon	NL desc.	Red	△	✘	✘
F8	Property	Circle	Details	Data type color code	✓	✘	⊙
F9	Node Inheritance	Dotted line or bézier curve	N.A.	Light Blue	✓	✓	✘
F10	Edge Inheritance	Dotted line or bézier curve	N.A.	Light Purple	✓	✘	✘
F11	Arbitrary Labels	Circle with unlock icon	NL desc.	Green	✓	✘	✘
F12	Arbitrary Properties	Circle with unlock icon	NL desc.	Green	✓	✘	✘

limits of human visual perception. Understanding visual complexity requires considering all these features together, as relying on a single factor, like the amount of information, is neither sufficient nor practical [40].

Theories and Principles of Icons [26]. Icons play a crucial role in helping users navigate and interact with interfaces. Several key theories and principles guide the effective design and use of icons [26, 45, 50, 53]: (a) *Recognition Over Recall:* Icons should be easily recognizable, resembling real-world objects or familiar symbols to aid quick identification of their functions. (b) *Consistency and Standards:* Using icons consistently and following established standards helps users predict their functions more intuitively. (c) *Simplicity and Clarity:* Icons should be simple and clear, focusing on core concepts without unnecessary detail to avoid confusion. (d) *Affordance:* Icons should visually suggest their functionality, making their purpose evident through design. (e) *Semantic Understanding:* Icons must convey meaning through their visual elements, allowing users to infer their purpose based on shape, color, and context.

Principles and Theories of Visualization [36]. Visual abstractions rely on graphical representations of data, making the theories of *expressiveness* and *effectiveness* crucial to this work [36]. Expressiveness assesses whether a graphical language can convey the intended information, while effectiveness evaluates which languages best leverage the output medium and the human visual system in specific contexts. These criteria have been applied in recent studies on query visualizations [24, 34, 38].

Mackinlay [36] identified and ranked perceptual tasks for encoding different types of data–quantitative, ordinal, and nominal. For instance, position is ranked the highest for all data types, while color is highly effective for ordinal and nominal data. Area is moderately effective for encoding quantitative data. From this ranking, Mackinlay proposed the *Principle of Importance Ordering* to prioritize the accurate encoding of more critical information by using the most effective perceptual tasks for that information.

Shneiderman's Eight Golden Rules of Interface Design [53]. These rules are key principles intended to improve the usability and effectiveness of user interfaces and are widely used for interface design [59]. We list the ones that are relevant to this study: (a) *Strive for Consistency:* Ensure that similar actions and elements are consistent throughout the interface to make it easier for users to learn and use the system. This includes consistent terminology, layout, and design conventions. (b) *Offer Informative Feedback:* Give users clear and immediate feedback on their actions. (c) *Support Internal Locus of Control:* Design interfaces so that users feel in control of the system. This involves giving users control over their interactions and avoiding unexpected changes. (d) *Reduce Short-Term Memory Load:* Minimize the cognitive load on users by reducing the amount of information they need to remember at one time. Use visual cues, reminders, and logical grouping to help users navigate and understand the interface.

3 The LICS Visual Abstraction

In this section, we present our novel visual abstraction of property graph schema coined LICS. The design and implementation of LICS is informed by aforementioned theories and principles. We begin by summarizing the features of PG-Schema that we aim to visualize using various encodings in LICS. Next, we introduce LICS and discuss its components in detail. Finally, we present a novel LICS-based GUI called PASCAL for visualizing PG-Schema.

3.1 Features of Property Graph Schema

We utilize the features of PG-Schema [8] introduced in Section 2.1 as key property graph schema elements for our proposed LICS abstraction. An overview of these features is provided in Table 2. The second column outlines the features, while the third, fourth, and fifth columns summarize the corresponding visual encoding schemes in LICS. The *"On Hover"* column shows the output displayed when the mouse hovers over a visual object. The sixth column summarizes whether a specific feature is *completely* or *partially* visually represented in LICS. Specifically, except constraints, all features are completely supported. In LICS, we leverage visual encoding schemes to identify key and participation constraints on given types. Recall that in PG-Schema, these constraints are expressed using the EXCLUSIVE, MANDATORY, and SINGLETON qualifiers. For instance, these three qualifiers can be employed to define a key. MANDATORY can being utilized to represent foreign keys. Instead of adding more visual encoding schemes to represent these qualifiers for these complex constraints, LICS uses natural language descriptions on mouse hover to maintain a balance between visual complexity and usability. As noted in Section 2.2, the variety in visual forms affects visual complexity, and introducing more encoding schemes with finer granularity for representing these qualifiers would increase that complexity. As we shall see in Section 5, this design choice did not result in negative feedback in our user study. Nevertheless, further research on finer visual representation of complex constraints is earmarked for future work.

3.2 Overview of LICS

Intuitively, the *labeled iconized composite schema (LICS)* is a directed graph and comprises of two *types* of visual shapes, *elementary* and *derived*, for visually representing the above features of a property graph schema. Specifically, *circle* and *arrow* belong to the elementary shape type whereas the *composite circle* and *arrow* are derived shapes. The nodes in a LICS represent Node Types (*base* and *inherited*). They are color-coded (blue or light blue) and represented by *circle* and *composite circle*. Specifically, the circle represents a Node Type without any properties. The composite circle models a Node Type with one or more properties. These properties are represented using a set of smaller color-coded circles overlaid *systematically* on its circumference. The color codes of these smaller circles have a one-to-one mapping with the colors associated with the *data types* of the

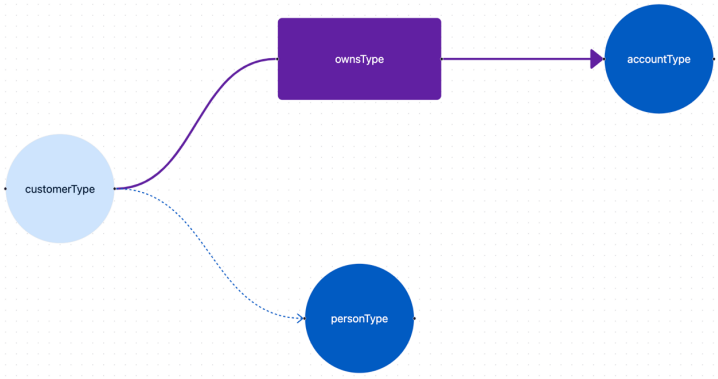


Fig. 3. [Best viewed in color] Visual encoding of Node Types, Edge Source and Target.

properties. The label of a Node Type is visually represented using a yellow pill positioned in the proximity of the corresponding (composite) circle. A node may be optionally associated with one or more *icons* encapsulated in a pill or circle to visually encode information related to key constraints and arbitrary labels or properties (*e.g.,* OPEN Label, OPEN Property in PG-Schema).

Similarly, the Edge Types and inheritances are represented using a solid or dotted, color-coded (purple or blue) *arrow* and a *composite arrow*. Specifically, blue dotted straight line and the purple dotted bézier curve are used to represent the Node inheritance and Edge inheritance edges, respectively. A rectangle is overlaid on the arrow of the Edge Type to distinguish itself from inheritance edges. The sides of a rectangle may be overlaid with a set of smaller color-coded circles to represent properties associated with the Edge Type. In this case, it is categorized as a composite arrow. On the other hand, relationships with no properties associated with it are represented using an arrow. Note that an (composite) arrow may be associated with an icon encapsulated in a pill to represent a participation constraint. Lastly, the label of an Edge Type is represented using the aforementioned yellow pill positioned in the vicinity of the corresponding Edge Type.

Figure 2 (Panel 3) depicts an example of the LICS representation of the schema in Figure 1.

3.3 Components of LICS

We now elaborate on the visual encoding schemes of LICS. Table 1 elaborates on how their designs are informed by theories and principles from HCI, cognitive psychology, and visualization.

Node Types. Figure 3 depicts the visual encoding of the *base* Node Type (blue circle) and *inherited* Node Type (light blue circle). We ensure that they are all circles with blue hue (informed by $T2$, $T3$, $T8$, $T11$, $T12$, and $T18$ in Table 1). The circle shape chosen is inspired by the parentheses in the ASCII-art format of *Cypher* [23] (*i.e.,* ()-[]->()), which represents nodes in a property graph.

Edge Types. Figure 4 depicts the representation of the *base* Edge Type (purple rectangle) and *inherited* Edge Type (light purple rectangle). These components are all rectangles with purple hue (informed by $T2$, $T3$, $T8$, $T11$, $T12$, and $T18$). The rectangle shape chosen is inspired by the brackets in the ASCII-art format of *Cypher* (*i.e.,* ()-[]->()), which represents relationships in a property graph.

Figure 3 illustrates the visual representation of edge endpoints. The edge source (purple edge at the left of Edge Type) and target (purple edge at the right of Edge Type) are shown. They employ $T2$ and $T18$ (Table 1) by using purple color to indicate that they are related to the Edge Type (*i.e.,* Similarity) and by using a bézier curve to ensure that the edge source and target can be seen as the same curve despite being separated by the Edge Type (*i.e.,* $T4$).

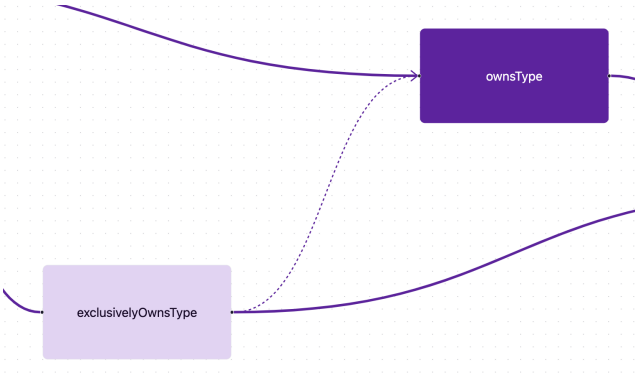


Fig. 4. [Best viewed in color] Edge Types.

Node and Edge Inheritances. The blue and purple dotted lines in Figures 3 and 4 represent the Node and Edge inheritance edges, respectively. These *inheritance edges* are designed to have a lower visual hierarchy than the endpoint edges since they are not as crucial for querying tasks. Therefore, they are depicted using dotted lines with a smaller width. Blue and purple colors are utilized to distinguish between them but also convey the fact that they are related to Node and Edge Types (*i.e.,* they share the same color). That is, their design is influenced by $T2$, $T4$, $T11$, and $T18$ (Table 1).

Labels. Figure 5 (left) shows the visualization of the label (yellow pill) component when the Node Type only has a single label. Since it is one of the most important components for performing various tasks (*e.g.,* querying), the yellow colour was chosen since it is a distinct primary colour. This makes them highly distinguishable from other components and make them easily identifiable. In addition to $T2$, $T7$, $T8$, and $T18$ (yellow pill), $T1$ and $T6$ are exploited here in its design as each label is positioned right below (*i.e.,* $T1$) and moves along with (*i.e.,* $T6$) its associated Node or Edge Type.

Recall that the Node or Edge Type of a property graph schema may be associated with *multiple* labels expressed using label expressions. Hence, we visually represent multiple labels using two modes, namely the *Expression Mode* and the *Tree View Mode*. Figure 5 (middle) depicts an example of the *Expression Mode*. Observe that in this visual mode the entire label expression is encoded in the pill in textual form. This may be challenging for lay users to interpret it as they may not be familiar with label expressions. To alleviate this, we visualize multiple labels using the *Tree View Mode* where the corresponding expression is visually represented using a tree structure and textual labels (Figure 5 (right)).

Properties. Guided by $T1 - T3$, $T11$, and $T18$, properties are depicted as color-coded, labeled circles on the circumference of a Node Type. We visualize them in two modes: *Data Type* and *Property*. Properties are represented by *property circles*, while data types are represented by *data type circles*. Examples of these visualizations are shown in Figure 6 (left) and 6 (middle).

In both modes, the circles are evenly spaced out around the circumference of the Node Type circle and do not overlap the left and right handles. This follows $T4$, $T5$, and $T9$. Also, $T2$, $T7$, and $T8$ are employed by grouping identical data types with the same color (*e.g.,* all strings are shown using yellow circle). This mitigates visual complexity by avoiding visual cluttering of objects. Moreover, based on $T13$, the area of a data type circle increases with the number of properties associated with that data type. Since area may not be sufficient to make users perceive quantitative data

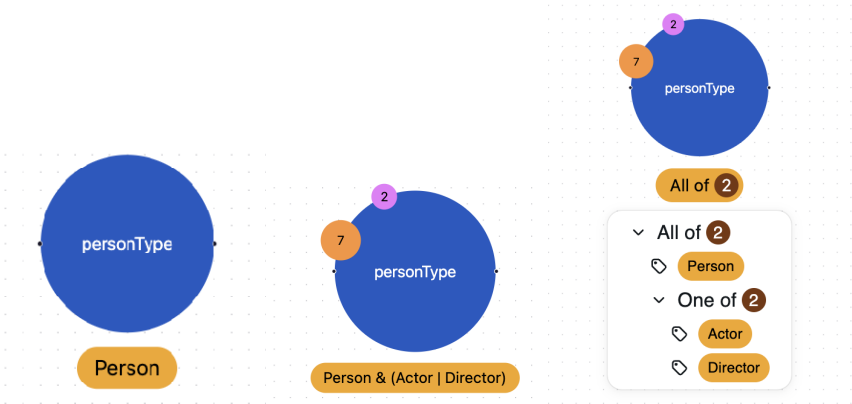


Fig. 5. [Best viewed in color] Visual encoding of labels: (left) single label; (middle) label expression; (right) *Tree View* of label expression.

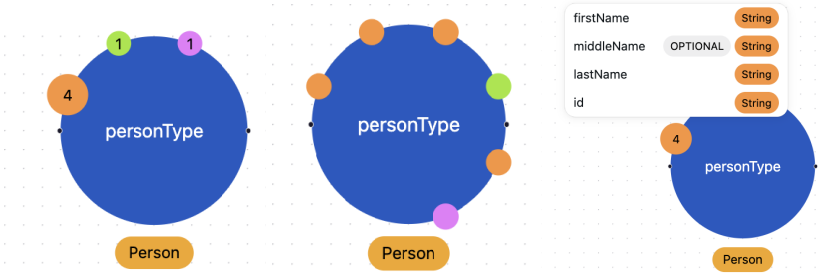


Fig. 6. [Best viewed in color] Visual encoding of properties of Node Type: (left) *Data Type* mode; (middle) *Property* mode; (right) Visualizing properties in the *Data Type* mode.

accurately [36], a data type circle is labeled with the number of properties of the corresponding type. For example, in Figure 6 (left) the STRING data type circle is labeled with 4 as there are four properties with type STRING associated with the PersonType. Observe that its area is larger than the other data type circles. As we shall see later, in the PASCAL GUI one can hover the mouse on a data type circle to view the full list of properties (Figure 6 (right)) (*i.e.,* $T20$). If a property is not required, an OPTIONAL pill will be displayed along with the colour-coded *Data Type* pill.

The Edge Type is also supported by these property and data type circles. Figure 7 (left) depicts an example when the *Data Type Mode* is enabled. In this case, the data type circles are placed along the top and bottom perimeter of the Edge Type rectangle and are also evenly spaced out to satisfy the above theories and principles.

Note that we do not label property circles with properties as it will increase visual cluttering of a Node or Edge Type (*i.e.,* $T10$ in Table 1). Specifically, placing a label *inside* a property circle compromises visual clarity, while positioning an integer value inside a data type circle is easier and maintains clarity.

Lastly, in addition to the above two modes, we also support a *Hidden Mode* that enables a user to hide all properties on-demand (informed by $T7$, $T10$, and $T20$).

In summary, these three modes are created to target different use cases. For most use cases, the *Data Type Mode* is potentially most effective. The *Property Mode* is more effective when there are

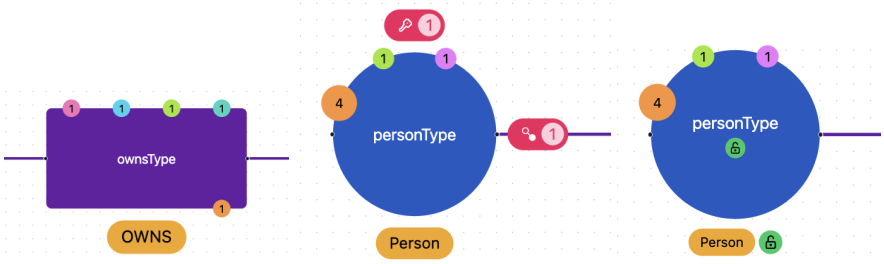


Fig. 7. [Best viewed in color] (left) Properties on Edge Type; (middle) Visual encoding of key and participation constraints; (right) Visual encoding of OPEN label.

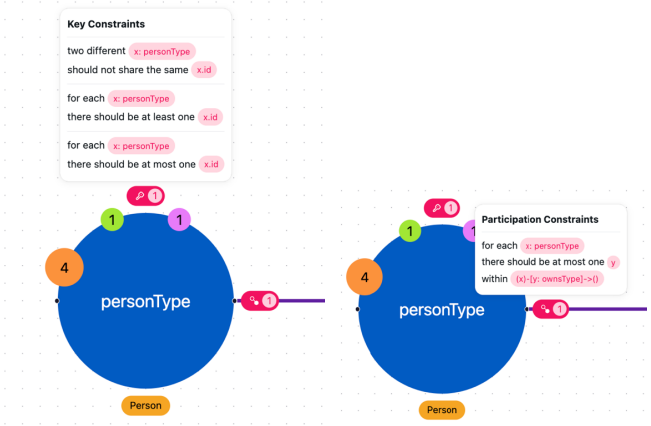


Fig. 8. [Best viewed in color] NL description: (left) key constraints; (right) participation constraints.

only a few properties on the Node and Edge Types since every property can be displayed without clutter. Lastly, the *Hidden Mode* is suitable for tasks that do not involve properties.

Key and Participation Constraints. We exploit $T14 - T17$ (Table 1) to represent key and participation constraints. Figure 7 (middle) depicts the key constraint (red pill with key icon at the top of the Node Type) and participation constraint (red pill with connect icon at the right of the Node Type). Appropriate icons (*i.e.,* key, connect) are utilized to convey the semantics of the constraints ($T14$, $T17$). In addition to consistent colour (red) and shape (pill) that satisfy $T2$, $T8$, and $T18$ to make the constraints easily identifiable, these components also employ $T1$ and $T6$ as they are placed closely (Proximity) and move along (Common Fate) with their associated Node or Edge Type. Note that since they may involve multiple Node and Edge Types, they cannot be attached to a specific property or relationship without increasing visual complexity ($T16$).

Based on $T10$ and $T20$, when a user hovers the cursor over a key or participation constraint, a pop up with natural language (NL) description of the constraint is displayed (Figure 8). The property keys, Node Types, or Edge Types involved are highlighted in red for ease of identification (informed by $T2$, $T18$). We chose to describe in NL format for reasons discussed in Section 3.1.

Arbitrary Labels and Properties. Lastly, we also exploit $T14 - T17$ (Table 1) to represent arbitrary labels and properties (*i.e.,* OPEN label and property). Figure 7 (right) illustrates the OPEN label (green circle with unlock icon at the right of label) and OPEN property (green circle with unlock

icon below type name) components. Since OPEN label (resp. property) indicates that the Node or Edge Type can accept additional arbitrary labels (resp. properties), the green colour and unlock icon are used to communicate openness w.r.t. these components ($T14$, $T17$). Upon hovering on them, one would be able view their semantics in text (informed by $T10$, $T20$).

Remark. Observe that LICS is *generic* and *extensible*. We support features that are essential for *any* property graph schema language and expected to be supported by languages beyond PG-Schema. That is, it can be deployed with a variety of property graph schema languages. Furthermore, we can easily *extend* LICS by adding new visual encodings for new schema features.

3.4 Comparison with LSG & LCG

In this section, we compare LICS with the LSG [8] and LCG [34] (recall from Section 1) abstractions. The last two columns in Table 2 report the schema features supported by these two abstractions. The LICS abstraction offers a richer visual encoding than the LSG, which primarily uses directed graphs to represent Node and Edge Types and their labels, leading to potential visual clutter. In contrast, LICS incorporates composite shapes and icons to visually represent a broader range of schema features. Specifically, the edge source and targets and node inheritance are visually encoded in LSG. The label feature is only partially supported, as it lacks visual encoding for label expressions. Importantly, LSG is not designed based on principles from cognitive psychology, HCI, and visualization.

More germane to LICS is the recently proposed LCG abstraction [34] for conjunctive property graph queries. Both are influenced by theories from psychology, HCI, and visualization, yet LCG focuses on visual encoding for query nodes, links, and predicates using composite circles and arrows. In LCG, there are no corresponding visual encodings for Node and Edge Types, constraints, inheritances, and arbitrary labels/properties (Table 2). Like LSG, it only partially represents labels as it does not support label expressions. While both LICS and LCG use composite circles and arrows to represent properties and predicates on properties, the semantics of these shapes differ. In LICS, the color coding of properties reflects data types rather than property names. Lastly, LCG allows for varying sizes of its circles and arrows based on the number of associated predicates, whereas LICS only changes the size of data type circles. Since a property graph schema may have a larger number of nodes and edges compared to a property graph query in practice, different sizes of Node and Edge Types may increase visual complexity of LICS due to size constraint of a GUI.

3.5 PASCAL: A LICS-based Visual Interface

Figure 2 depicts the PASCAL GUI, which has four panels. *Panel 1* is the navigation panel, allowing users to load and switch between *views* of a property graph schema. *Panel 2* (resp. *Panel 3*) is the *Schema Editor* (resp. *Schema Canvas*), displaying the schema in text (resp. LICS) form. Note that if the schema results in a syntax error while parsing, PASCAL provides appropriate error messages to users (informed by $T19$ in Table 1). *Panel 4* enables a user to set various modes for schema visualization (informed by $T20$). We now elaborate on *Panels* 1, 3, and 4. Table 1 reports the how the design is guided by the aforementioned theories and principles.

Users can click on the *Load Schema* button in *Panel 1*, upload a schema file that contains a plaintext PG-Schema, and click save to populate the *Schema Editor*. If the schema is valid, its LICS abstraction is rendered in the *Schema Canvas*. PASCAL enables two views: *Visualizer View* and *Text View*. To navigate between the views, the navigation links at the centre of *Panel 1* can be used (informed by $T20$). The *Visualizer View* contains the *Schema Editor* and the *Schema Canvas* whereas the *Text View* only involves the former.

Panel 3 visualizes the schema in *Panel 2* by exploiting the LICS abstraction. The top of the panel exposes the *Data Type Reference* subpanel whose design is guided by $T21$. It contains the color codes of different data types used in LICS. This reduces the load on limited working memory as users do not need to remember the color codes.

Panel 4 enables users to interactively visualize in different settings, namely *Visualisation*, *Graph Layout*, and *Workspace* (informed by $T20$). The *Visualisation Settings* allows one to choose one of the three property visualization modes (recall from Section 3.3). Users can toggle between the *Data Type*, *Property*, and *Hidden* Modes with the drop down input (the default mode is *Data Type*). The *Expression Visualisation Mode* enables users to choose one of the two modes for displaying labels. The default option is the *Tree View Mode*. Expert users may switch to the *Expression Mode* where they can view the label expression at a glance.

The *Graph Layout Settings* allows a user to set the layout of the LICS. By default, the *auto layout* feature is enabled which automatically adjusts the coordinates of Node and Edge Types based on Table 1. It also positions them on the canvas such that all arrows can flow from left to right (*i.e.,* $T6$ and $T18$). Disabling the auto layout feature enables the users to control the positioning of these components on the canvas ($T20$). PASCAL also allows users to manually fix the layout at any time by clicking on the *Fix Layout* button. Lastly, the *Workspace Settings* allows users to hide the *Schema Editor* or *Data Type Reference* by toggling the corresponding switches.

4 Implementing LICS

Intuitively, any implementation aims to facilitate automatic *transformation* of a property graph schema to a color-coded LICS on the *Schema Canvas*. We begin by identifying a set of *operations* that form the basis for generating the LICS abstraction. Then, we describe the Map-Paint algorithm that exploits them for visualizing a PG-Schema.

4.1 A Set of LICS Operations

The LICS abstraction uses the following set of operations:

- map. Maps the elements in a plaintext schema $\mathbb{S}$ to corresponding *shape buckets* representing elementary shapes and icons in LICS.
- paint. Generates the visual encodings of the shape buckets.
- join. "Join" data type or property circles with a Node Type or Edge Type shape to create the derived shape by positioning them systematically (Section 3.3) on the circumference/perimeter of the Node Type or Edge Type).
- up. Increases the area of a data type circle in accordance to $T13$ (Table 1). It is invoked by the join operator.
- iconize. Generates the icon object of a constraint or arbitrary labels/properties based on icon theories.
- position. Positions specific shapes or icon objects on the GUI based on Table 1.
- translate. Generates a natural language description of a constraint or label expression.
- delete. Removes a component from the LICS abstraction.

Note that the join, iconize, position, and translate are invoked by the paint operation. Hence, the map and paint are the core operators for implementing the LICS abstraction.

4.2 The Map-Paint Algorithm

Algorithm 1 outlines the procedure for generating the LICS abstraction of an input PG-Schema $\mathbb{S}$. First, it parses $\mathbb{S}$ (Line 1), sets the error messages, if any, in the PASCAL GUI, and generates the LICS abstraction for visualizing $\mathbb{S}$ if the parsing result is valid (Lines 6-21). Line 6 initializes the

Algorithm 1 The Map-Paint Algorithm.

Input: Plaintext property graph schema $\mathbb{S}$, icon map $\mathbb{I}$, label mode $M_\ell \in$ {'Tree View', 'Expression'}, property mode $M_p \in$ {'Data Type', 'Property', 'Hidden'}
Output: The LICS representation $\mathbb{L}$

```
1:  D_s, isValid, errors ← PgsParse(𝕊) /* Parse schema */
2:  if not isValid then
3:      SetErrors(errors) /* Parsing error message */
4:      return
5:  end if
6:  ℂ ← InitializeDTColorCodeMap() /* Hash map for data type color code */
7:  B_N, B_E, B_C ← []
8:  for elem ∈ D_s do
9:      if isNodeType(elem) then
10:         B_N.Map(elem) /* Map operation for Node Types */
11:     else if isEdgeType(elem) then
12:         B_E.Map(elem) /* Map operation for Edge Types and inheritances */
13:     else if isConstraint(elem) then
14:         B_C.Map(elem) /* Map operation for constraints */
15:     end if
16: end for
17: 𝕃.Paint(B_N, 𝕀, ℂ, M_ℓ, M_p) /* Paint operation for Node Types (Algorithm 2) */
18: 𝕃.Paint(B_E, B_N, 𝕀, ℂ, M_ℓ, M_p) /* Paint operation for Edge Types and inheritances */
19: 𝕃.Paint(B_C, 𝕀) /* Paint operation for constraints */
20: return 𝕃
```

one-to-one map of color code to various data types (*Panel 3* in Figure 2). Then the visualization of the LICS abstraction is achieved by two main operations, map and paint. Intuitively, the map operation maps the Node Types, Edge Types, inheritance, and constraints in $\mathbb{S}$ to their corresponding *shape buckets* (Lines 8-16). Then, the paint operation operates on these buckets to generate the LICS abstraction $\mathbb{L}$ and render on the GUI (Lines 17-19). We elaborate on these two operations now.

The Map Operation. The map operation $map(\cdot)$ assigns each element in $\mathbb{S}$ to a specific *shape bucket* B_i, where each bucket corresponds to a particular elementary shape (*e.g.,* circle or arrow) or icon in the LICS abstraction. For instance, Node Types are mapped to the circle bucket, Edge Types, inheritance, and constraints are mapped to arrow and icon buckets, respectively. Note that the map operation does not create new buckets for shapes associated with labels, property and data type circles, or OPEN keywords. These are handled by the paint operation on the corresponding Node or Edge Types to generate the derived shapes.

The Paint Operation. Intuitively, the paint operation $paint(\cdot)$ is responsible for "painting" the LICS abstraction of $\mathbb{S}$ on the PASCAL GUI. This translates to the generation of visual encodings of the key components in a property graph schema. Algorithm 2 outlines the procedure for creating Node Types. For each Node Type n_i, it generates a circle shape and assigns a color (blue for base, light blue for inherited) (Line 3). It then retrieves associated properties and generates color-coded data type or property circles based on the *property mode* (Lines 4-6), using the *data type color map* $\mathbb{C}$. These circles are positioned around the main circle s_i using a join operation [34] based on Gestalt principles and visual complexity theories (Line 7). It also adjusts the size of relevant data type circles in *Data Type Mode* (using the up operation). A color-coded pill shape for the label/label expression is generated and positioned near s_i, with movement coordinated using the Principle of Common Fate (Lines 10, 13, 15). In *Tree View Mode*, the translate method uses a rule-based approach to convert label expressions into a NL-based tree view (Line 12). Finally, if n_i is associated with arbitrary labels or properties (OPEN keyword), the corresponding icon is generated by utilizing the icon map $\mathbb{I}$ and positioned following the aforementioned theories and principles (Lines 18-23).

The process of generating visual representations for Edge Types and inheritance (Line 18 in Algorithm 1) is similar to that of Node Types in Algorithm 2. It draws solid or dotted edges between

Algorithm 2 The Paint Algorithm for Node Types.

Input: Node Type bucket B_N, data type color map $\mathbb{C}$, icon map $\mathbb{I}$, label mode M_ℓ, property mode M_p
Output: The LICS representation of B_N

```
1:  circleShapes ← [ ]
2:  for n_i ∈ B_N do
3:      s_i ← NodeShape(n_i)
4:      P_{n_i} ← GetProperties(n_i) /* Retrieve node type properties */
5:      if |P_{n_i}| > 0 then
6:          C_i ← PropShape(P_{n_i}, ℂ, M_p)
7:          s_i.Join(C_i) /* Join data type or property circle */
8:      end if
9:      if n_i.label then
10:         s_ℓ ← LabelShape(n_i.label)
11:         if M_ℓ == 'Tree View' then
12:             T_i ← Translate(n_i.label) /* Transform label expression to NL-based tree */
13:             s_i.Position(T_i)
14:         else
15:             s_i.Position(n_i.label, s_ℓ)
16:         end if
17:     end if
18:     if n_i.isOpen() then
19:         for each arbitrary item o_i in n_i do
20:             I_o ← Iconize(o_i, 𝕀) /* Generate icon */
21:             s_i.Position(I_o)
22:         end for
23:     end if
24:     circleShapes.add(s_i)
25: end for
26: return circleShapes
```

the previously generated Node Types and creates the corresponding labels and properties in the same manner.

The paint operation for constraints processes each constraint in the bucket B_C. It first identifies the associated elements and determines whether the constraint is a key or participation constraint. For key constraints, a key icon is created, while for participation constraints, a connect icon is created. The constraint c_i is then translated into a natural language representation using a rule-based approach and added as an attribute to I_{elem}. Finally, the visual representation $\mathbb{L}$ is updated by positioning the icon I_{elem} according to the discussion in Section 3.3 (*e.g.,* Figure 8).

Complexity Analysis. The time complexity of Algorithm 2 is dominated by the operations of generating and joining data type or property circles (Lines 6-7), translating label expression in the *Tree View Mode* (Line 12), and iconizing OPEN keywords (Lines 19-22) as generating single 2D shapes, retrieving properties/elements/icons, and positioning them based on aforementioned principles and theories take constant time [28, 34, 37]. In the worst case, up to $|P|$ properties (Line 4) must be rendered per node. Hence, it takes $O(|P|)$ time for the join for each Node Type $n_i \in B_N$ [34]. Similarly, the cost for generating icons for arbitrary labels/properties for each Node Type is $O(|I|)$ where $|I|$ is the maximum number of elements associated with OPEN keyword. Let us assume the worst-case time complexity of NL-based translation (Line 12) is $\mathcal{T}(\cdot)$. Hence, the complexity of is $O(|B_N| \cdot (|P| + \mathcal{T}(\cdot) + |I|))$.

Similarly, the paint operation for Edge Types and inheritance (Line 18 in Algo. 1) has a complexity of $O(|B_E| \cdot (|P| + \mathcal{T}(\cdot) + |I|))$. The paint for constraints (Line 19 in Algo. 1) requires $O(|B_C| \cdot (\mathcal{T}(\cdot)))$.

Now consider Algorithm 1. Assume that the worst-case parsing cost of $\mathbb{S}$ (Line 1) is $\mathcal{P}(\mathbb{S})$. Lines 8-16 require $O(|D_s|)$ to map parsed results into three buckets. Hence, the overall time complexity for Algorithm 1 is $O(\mathcal{P}(\mathbb{S}) + |D_s| + (|B_N| + |B_E|) \cdot (|P| + \mathcal{T}(\cdot) + |I|) + |B_C| \cdot (\mathcal{T}(\cdot)))$. Since $|B_N| + |B_E| \leq |B_N| + |B_E| + |B_C| \leq |D_s|$, it can be simplified to $O(\mathcal{P}(\mathbb{S}) + |D_s| \cdot (\mathcal{T}(\cdot) + |P| + |I|))$.

Table 3. Datasets. $|N|$, $|E|$, $|T|$, and $|C|$ denote the number of Node Types, Edge Types, inheritances, and constraints, respectively.

Name	Complexity	$\|N\|$	$\|E\|$	$\|T\|$	$\|C\|$	Generation Time (msec)	
						Avg.	S.D.
Northwind	Low	5	4	0	0	46.1	3.16
Recommendations	Medium	6	4	2	4	77.35	6.67
Neoflix	High	13	13	2	12	367.64	9.06

The worst-case space complexity of Algorithm 2 is $O(|B_N|)$, as $|B_N|$ base and inherited Node Types are "painted" and added to the *circleShapes* list. Similarly, the paint operations for Edge Types and constraints require $O(|B_E|)$ and $O(|B_C|)$ space, respectively.

Assuming the space usage of error message and data type color map $\mathbb{C}$ is constant, the parsing results in Line 1 take $O(|D_s|)$ space. Additionally, the three mapped buckets require $O(|B_N| + |B_E| + |B_C|)$ space. Incorporating the three paint operations (Lines 17-19), the total space complexity of Algorithm 1 becomes $O(|D_s| + |B_N| + |B_E| + |B_C|)$, which can be further simplified to $O(|D_s|)$.

Remark. Algorithm 1 is *generic* and *extensible*. The map operation assigns schema components to shape buckets, and the paint operation processes these buckets. This approach can be easily adapted to any property graph schema language by modifying the parser (Line 1). Additionally, new schema components can be incorporated by extending the map (Lines 8-16) and paint (Lines 17-19) operations.

Updating the Visualization. Recall that a user may interactively change the visualization by changing the settings of the *Visualisation Settings* or *Expression Visualisation Mode* in the PASCAL GUI. Under the hood, this is performed by the removal of the current objects using the delete operation and then invoke the relevant part of the paint operation to create new visual objects.

5 User Study

PASCAL is implemented with React. In this section, we report the performance of PASCAL using a user study. Specifically, we investigate the following research questions:

RQ1: Which abstraction do participants prefer for schemas of varying complexities, PASCAL (*i.e.,* LICS) or LSG?

RQ2: How effective is PASCAL in aiding the understanding of property graph schemas? More specifically, do the principles and theories behind the design of LICS positively influence participants' comprehension of the schema components?

RQ3: How effective is PASCAL in aiding efficient property graph query formulation as an application of these schemas?

All experiments are performed on a 64-bit Windows machine with Intel(R) Xeon(R) W-2235 CPU (3.80GHz) and 32GB of main memory.

5.1 Experimental Setup

Datasets. We use three property graph schemas from *Neo4j* [5] as reported in Table 3. These schemas are described using the PG-Schema language. We classify them into three categories, *low*, *medium*, and *high* based on the complexity of the schemas. Intuitively, a *low complexity schema* has only a few Node and Edge Types and no constraints and inheritance. On the other hand, a *high complexity schema* contains tens of Node and Edge Types, inheritance or constraints.

The last column in Table 3 reports the average time required to generate the LICS abstraction for these three datasets. Each schema is run five times, with the average time and standard deviation (S.D.) of the last four runs being recorded. Observe that the Map-Paint algorithm takes less than a second to generate the abstractions.

Table 4. Experiment groups.

Group	Size	Querying Task	Survey
A (Control)	4	*Visualiser View* for all querying tasks.	N.A.
B (Control)	4	*Text View* for all querying tasks.	N.A.
C	8	(a) *Visualiser View* for the first half of the querying tasks; (b) *Text View* for the second half of the querying tasks.	Participated
D	8	(a) *Text View* for the first half of the querying tasks; (b) *Visualiser View* for the second half of the querying tasks.	Participated

Baselines. To the best of our knowledge, we are not aware of any publicly-available visualization framework for property graph schemas. Typically, directed graph view (*i.e.,* LSG) is used as the default visualization abstraction (*e.g.,* [8]). Hence, we created a directed graph representation of the schema where nodes are labeled with Node Types and relationships are labeled with Edge Types. All other details of the schema are hidden in this view to avoid visual clutter and only available on-demand (through clicks) in textual format. The corresponding PG-Schema syntax of the entire schema is shown in parallel as plaintext in a side panel. We refer to this view as the *Text-Graph View* (*Text View* for brevity). In our user study, we compare LICS with this "text-graph" view of property graph schemas to answer **RQ1**-**RQ3**. In the sequel, we refer to the LICS-based visualization of the schemas as the *Visualizer View*.

Specifically, to answer **RQ2**, we compare *Cypher* query formulation aided by *Text View* and *Visualizer View* using the followings: (a) Neo4j Browser: We use Neo4j Browser as a representative framework for formulating *Cypher* queries in textual form. In particular, it can be used by participants who are familiar with the syntax and semantics of *Cypher*. (b) SIERRA: Since many users are unfamiliar with a property graph query language, we leverage SIERRA [34, 35], a state-of-the-art visual query interface for constructing property graph queries in *Cypher*. In particular, SIERRA enables a user to visually construct conjunctive property graph queries without using any textual query languages and is reported to be more efficient than *Neo4j*'s Bloom [34].

Participants. 24 unpaid volunteers, within the age group of 21 to 35 years old, participated in the study in accordance to HCI research that recommends at least 10 participants [21, 31]. They are from different professional backgrounds such as backend engineers, data analysts, graduate students, etc. They have computing-related educational background. All participants have prior experience in working with SQL or NoSQL databases. Hence, they are familiar with the notion of schema. Eight of them indicated that they have worked with graph databases and among them only one is proficient in *Cypher*. None were familiar with PG-Schema prior to this study.

We informed the participants about the purpose of study. After consenting to have their feedback used for research, we gave them a tutorial on the key features of PG-Schema and property graphs and demonstrated how to use PASCAL, SIERRA, and Neo4j Browser. We allowed them to explore these interfaces and example property graph schemas and answered any queries related to them. Once done, we gave them a series of tasks to complete.

5.2 Tasks and Procedure

We describe our framework of user study to answer the research questions (**RQ1**-**RQ3**). For ease of exposition, we begin with **RQ3**.

RQ3: Query Formulation. Our broad goal is to study the impact of the *Visualizer View* and the *Text-Graph View* (*i.e., Text View*) on the query formulation time for property graph queries. To this end, the participants were divided into 4 groups as reported in Table 4. *Groups A* and *B* were control groups that used either the *Visualizer View* or *Text View* to perform the querying tasks, but they did not participate in the subsequent survey as it involves comparison questions between these

Table 5. Querying tasks and features. $|V|$, $|L|$, and $|P|$ denote the number of vertices, edges, and predicates, respectively.

Task ID	Schema	$\|V\|$	$\|L\|$	$\|P\|$	Structure	Schema Features
1	*Northwind*	3	2	3	Chain	F1-F5, F8
2	*Northwind*	3	2	2	Chain	F1-F5, F8
3	*Northwind*	5	4	3	Tree	F1-F5, F8
4	*Northwind*	5	4	3	Tree	F1-F5, F8
5	*Recommendations*	4	3	3	Tree	F1-F5, F8, F11
6	*Recommendations*	5	4	4	Chain	F1-F6, F8, F11
7	*Recommendations*	6	6	5	Graph	F1-F5, F8, F11
8	*Recommendations*	6	6	4	Graph	F1-F6, F8, F9
9	*Neoflix*	5	4	4	Chain	F1-F6, F8
10	*Neoflix*	4	4	3	Graph	F1-F6, F8
11	*Neoflix*	8	7	7	Tree	F1-F5, F8
12	*Neoflix*	8	7	8	Tree	F1-F6, F8

two views. Conversely, participants in *Groups C* and *D* used both the *Visualizer View* and *Text View*. Specifically, *Group C* was presented with the *Visualizer View* first and *Group D* was presented with the *Text View* first to reduce bias towards a specific view of property graph schemas.

Each participant was presented with either the *Visualizer View*, *Text View*, or both, depending on the group he/she belongs to, and was requested to construct 12 *Cypher* queries using either SIERRA or Neo4j Browser. The query set is available at [6]. These queries were presented to the participants in natural language (NL) form. They were asked to refer to the corresponding schemas in order to construct the graph queries. We recorded the time taken to formulate a query by the subjects. Specifically, a participant was allowed to clarify any details related to the NL description of a query. To mitigate the impact of interpretation time for queries given in natural language, the query formulation time (QFT) is recorded only when a subject starts constructing it *after* perusing and interpreting the description. A subject then constructs the given queries visually or textually using a mouse and a keyboard using the corresponding property graph schemas as a guide. They were informed to undertake their tasks at leisure. If a participant constructs an erroneous query then it is discarded and he/she has to construct it again from scratch.

The characteristics of the queries are reported in Table 5. The last column summarizes the relevant schema features from Table 2. Note that the selection of queries is constrained by the capabilities of SIERRA that we utilized. The current version supports only conjunctive queries with simple boolean predicates. As a result, we have restricted our study to using only these types of queries [6].

To mitigate any bias, the following measures were also taken: (a) All participants were presented with the same number of tasks in low (*Northwind*), medium (*Recommendations*), and high (*Neoflix*) complexity schemas. (b) Subjects in *Groups C and D* were presented with the same number of simple and complex querying tasks for each schema and view. (c) Participants in each group were equally divided to begin with either a low complexity schema or a high complexity schema.

RQ1, RQ2: Preference & Schema Comprehension. Participants in *Groups C* and *D* were asked of their preferences (**RQ1**). They were also tasked to comprehend the structure of the property graphs by referring to the corresponding schemas (**RQ2**). They were given specific keys, properties, node and edge types, inheritance, participation constraints, OPEN labels, and label expressions (randomly selected) to find them in these schemas.

Survey. After finishing the aforementioned tasks, each participant was requested to fill out a survey form to indicate their preferences between the *Visualizer View* and *Text View* (**RQ1**), rate various PASCAL/LICS features (**RQ2**), and provide unstructured feedback on its best and worst features. We survey them on their overall experience with the three schemas, as survey fatigue and the

Table 6. Features of PASCAL. The symbols D and SC in the **Target** column denote the feature is due to theory-informed design and useful for schema comprehension, respectively.

ID	Question	Target	Likert Scale 1	2	3	4	5
Node and Edge Types							
Q1	*The consistent colour coding on the Node (Blue) and Edge (Purple) Types is helpful in understanding the schema.*	D, SC	0	0	2	10	4
Q2	*The consistent shape of Node (Circle) and Edge (Rectangle) Types is helpful in understanding the schema.*	D	0	0	0	4	12
Q3	*The Edge Types' source and target are easy to find when there is a single source and target.*	D, SC	0	0	0	5	11
Q4	*The Edge Types' sources and targets are easy to find when there are multiple sources or targets.*	D, SC	0	2	6	6	2
Q5	*The use of inheritance is helpful in understanding the schema.*	D, SC	0	0	1	5	10
Labels and Label Expressions							
Q6	*The consistent colour coding on the labels (Yellow) is helpful in understanding the schema.*	D, SC	0	0	0	3	13
Q7	*Each label pill is well positioned below its associated Node or Edge Type.*	D	0	0	0	1	15
Q8	*The label expression Tree Views are easy to understand.*	D, SC	1	3	4	8	0
Properties							
Q9	*The colour coding on the data types is helpful in finding the properties.*	D, SC	0	0	0	5	11
Q10	*The Data Type Reference panel is helpful in finding properties.*	D	0	0	0	2	14
Q11	*The number of Properties on Data Type circles is helpful in finding the Properties.*	D, SC	0	3	5	3	5
Q12	*The dynamic sizing of data type circles based on the number of properties is helpful in finding the properties.*	D, SC	0	2	4	6	4
Q13	*The data type circles are well positioned around the Node Types.*	D	0	0	1	5	10
Q14	*The data type circles are well positioned around the Edge Types.*	D	0	0	1	5	10
Constraints							
Q15	*The consistent colour coding on the constraints (Red) is helpful in understanding the schema.*	D, SC	0	0	0	3	13
Q16	*Each key constraint pill is well positioned above its associated Node or Edge Type.*	SC	0	0	3	6	7
Q17	*The natural language translation of constraint expressions is easy to understand.*	D, SC	0	0	3	2	11
Arbitrary Labels/Properties							
Q18	*It is easy to identify the OPEN labels (Green unlock icon).*	D, SC	0	0	0	5	11
Usefulness of Icons							
Q19	*The icons are appropriately applied and are helpful in understanding the schema.*	D, SC	0	0	3	9	4
Layout of LICS in PASCAL GUI							
Q20	*It is easy to identify Node and Edge Types when zoomed out.*	SC	0	2	2	5	7
Q21	*The consistent left-to-right flow of relationship and inheritance edges is helpful in understanding the schema.*	D, SC	0	0	0	3	13
Q22	*It is easy to identify the direction of relationship and inheritance edges when zoomed out.*	D, SC	0	2	4	8	2

challenge of objectively isolating feedback for each individual schema prevent us from gathering feedback for *each* one separately.

5.3 Experimental Results

Results of RQ1. For low complexity schemas (*Northwind*), 14 out of 16 participants (*Groups C* and *D*) preferred the *Visualiser View*. Most of them mentioned that understanding various components of the schema at a glance and finding relationships were faster and more convenient in the *Visualizer View*. Interestingly, one participant who voted for *Text View* was an experienced graph database user who used plaintext *Cypher* to perform the querying task and found the syntax of PG-Schema easy to understand for such schemas.

When working with the medium complexity *Recommendations* schema with more features such as inheritance, constraints, and OPEN keyword, all participants preferred the *Visualiser View*. In particular, several participants mentioned that the inheritance edge visualization in PASCAL simplified their understanding of the parent-child relationship between Node Types.

For the high complexity *Neoflix* schema with most Edge Types having multiple sources or targets, the proportion of participants who preferred the *Visualiser View* over *Text View* decreased to 12 out of 16. Although majority of the participants still prefer PASCAL, several pointed out that edges were relatively more difficult to identify compared to the other two schemas.

Table 7. Subjective feedback.

Positive Feedback
(P1) *"Labels, shapes, properties are clear."* **(P2)** *"Left to right direction."* **(P3)** *"Shape to differentiate relationships and nodes."* **(P4)** *"Able to see overall structure at a glance."* **(P5)** *"Intuitive interface."* **(P6)** *"Dynamic sizing of data type circles."* **(P7)** *"Color coded data types."* **(P8)** *"Nodes and relationships are easy to read, parent/child easy to understand, fix layout."* **(P9)** *"Very easy to find relationships (first two schemas)."*
Negative Feedback
(N1) *"Relationships overlapping in complex schema."* **(N2)** *"Label expression is a bit hard to understand at first."* **(N3)** *"The minimum size of data type circles should be bigger."* **(N4)** *"Plain expressions are easier to understand than Tree View."* **(N5)** *"Cannot find arrow heads when the graph is small."*

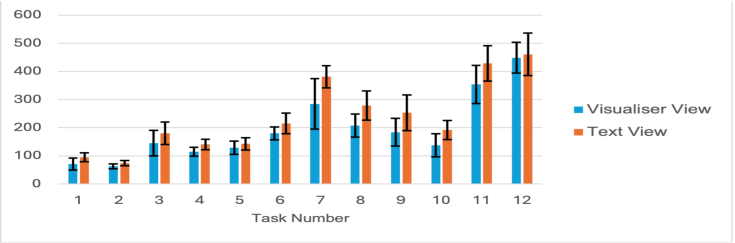


Fig. 9. Mean query formulation times (in sec.).

In summary, at least 75% of participants prefer the *Visualizer View* to the *Text View*. Since more than 60% of participants favor the former (success criterion), the answer to **RQ1** is LICS.

Results of RQ2. Next, we report the impact of the theory-informed design choices we made in PASCAL/LICS on the participants in comprehending property graph schemas. In our survey of participants from *Groups C* and *D*, we asked questions about the features of PASCAL. These questions were grouped into six categories: *Node and Edge Types*, *Labels and Label Expressions*, *Properties*, *Constraints*, *Arbitrary Labels/Properties*, *Usefulness of Icons*, and *Layout of LICS in PASCAL*. The first five categories address the main features of property graph schemas, while the *Usefulness of Icons* category focuses on feedback about the effectiveness of icons for visually depicting certain schema features. Recall that this is the first time icon theories are being applied in this space. The final category seeks input on the overall layout of the LICS abstraction in PASCAL.

Each subject gave a rating in the Likert scale of 1-5 (1 = strongly disagree and 5 = strongly agree). We consider the theory-informed design choices in a specific category to have a positive impact if the majority of participants rated most of the questions above 3, which serves as the success criterion.

Table 6 reports the results. We can make the following observations: **(1)** Most respondents found the consistent color coding of Node and Edge Types, labels, constraints, and data types helpful ($Q1$, $Q6$, $Q9$, $Q15$). Additionally, the positioning of all components was deemed effective by the majority of respondents ($Q7$, $Q13$, $Q14$, $Q16$). **(2)** Respondents found the consistent shapes of Node and Edge Types helpful ($Q2$), with most indicating that shapes were more effective than color for identifying them. The source and target of Edge Types were easy to locate when there was only one, but less effective when multiple sources or targets were involved ($Q3$, $Q4$). The inheritance edges were considered useful ($Q5$). **(3)** The *Tree View* mode for label expressions was easy to understand for half of the respondents ($Q8$). **(4)** The *Data Type Reference Panel* for identifying property data types was useful to all ($Q10$), while only a small number found the number of properties and the dynamic sizing of data type circles unhelpful ($Q11$, $Q12$). **(5)** Most respondents understood the natural language translation of constraint expressions ($Q17$), supporting the design choice discussed in Sec. 3.1. **(6)** The OPEN Labels were easily identifiable ($Q18$). **(8)** The use of icon theories was effective, as most respondents felt the icons were appropriately applied ($Q19$). **(9)** Lastly, most respondents found the layout of LICS in the PASCAL GUI acceptable. The consistent left-to-right

Table 8. p-values of query formulation tasks.

Task ID	p-value	Task ID	p-value	Task ID	p-value
1	0.0246	*5*	0.2329	*9*	0.0296
2	0.0294	*6*	0.0402	*10*	0.0129
3	0.1274	*7*	0.0201	*11*	0.0374
4	0.0104	*8*	0.0094	*12*	0.7194

Table 9. Experience vs QFT (in sec.) and QFE (in brackets).

Schema Type	No GDB Exp (N-GD)		GDB Exp (GD)	
	Visualizer	Text	Visualizer	Text
Low	103.35 *[0]*	123.26 *[0.06]*	90.08 *[0]*	120.27 *[0]*
Medium	201.54 *[0.22]*	259.44 *[0.31]*	194.63 *[0.19]*	243.42 *[0.25]*
High	277.05 *[0.56]*	351.18 *[0.72]*	289.15 *[0.53]*	279.7 *[0.69]*

Table 10. Survey on query formulation task.

ID	Question	Likert Scale 1	2	3	4	5
QF1	*The data type circles around the Node and Edge Types are not distracting during query construction.*	0	1	1	0	6
QF2	*The inheritance edges do not distract the relationship edges during query construction.*	0	0	3	1	4
QF3	*Fixing the size of label and constraint pills when zooming in or out is helpful in query construction.*	0	1	1	0	6
QF4	*The OPEN labels are not distracting in query construction.*	0	0	1	1	6

flow of edges was helpful for all ($Q21$), and the edges were easily identifiable. However, some respondents felt the arrowhead was too small when zoomed out ($Q22$). In summary, all questions meet our success criterion, except for $Q8$ and $Q11$.

We also asked the participants to provide unstructured feedback on what they liked (resp. disliked) about PASCAL. Table 7 reports some representative feedback. They support majority of the design choices made in PASCAL. Based on the feedback and also our observations of the user study sessions, subjects appear to enjoy using PASCAL. Some of the negative feedback can be easily mitigated (discussed in Section 5.4).

Results of RQ3. Finally, we report the query formulation times (QFT) taken by the participants to complete the queries (*i.e.,* tasks) in Table 5. In total, each task was performed 12 times with the *Visualizer View* and 12 times with the *Text View*. Participants were given the option to formulate queries using either SIERRA (visual) or Neo4j Browser (Cypher). In practice, only one participant used the latter due to familiarity and experience with *Cypher*.

Figure 9 plots the mean QFT and its standard deviation for the *Visualizer View* and *Text View*. The QFT increases with more nodes, links, and predicates in a query. Participants are able to formulate queries faster using the *Visualizer View* compared to the *Text View*, especially for more complex queries in the *Recommendation* and *Neoflix* tasks. A two-sample unequal variance test (Welch's t-test) was used to calculate p-values. The p-value was less than 0.05 for 9 out of 12 tasks (Table 8), indicating that QFT is lesser in the *Visualizer View* for these tasks is statistically significant. We note that the choice between visual and textual query formulation modes may influence the QFT. However, since most participants used SIERRA for constructing queries, this impact is minimal.

As eight participants have prior experience with graph databases, we now compare the QFTs between those with previous exposure to graph databases (**GD**) and those without (**N-GD**), in order to examine the impact of familiarity on QFTs. For fair comparison, we randomly selected eight participants from the rest for **N-GD**. Table 9 reports the average QFTs for the three schema categories. Subjects in **GD** exhibit lower average QFTs than **N-GD** for most schema types. Similarly, the *Visualizer View* outperforms the *Text View* for both groups. The only exception is **GD** for complex schemas, likely because familiarity may outweigh the advantages of LICS for complex schemas (see Section 5.4).

We report the *query formulation error (QFE)* for both groups across schema categories (Table 9). QFE is calculated as the ratio of the number of reformulated queries due to errors (*e.g.,* incorrect

node/link construction, wrong property selection) to the total number of tasks performed by the subjects in each schema category ($8 \times 4 = 32$). For instance, if 10 tasks in the **N-GD** group required reformulation, the QFE would be 0.31. QFE increases with schema complexity, with most errors occurring in tasks 7, 8, and $10 - 12$ (Table 5). The *Visualizer View* consistently shows lower QFE, indicating that the LICS abstraction helps reduce these errors.

Lastly, we conducted a survey to evaluate participants' experiences with query formulation in PASCAL, asking eight of them to rate whether key visual components were distracting (Table 10). Participants used a Likert scale from 1 to 5 (1 = strongly disagree, 5 = strongly agree). The success criterion was same as that of **RQ2**. Overall, most participants reported that these design features did not interfere with their query construction.

5.4 Open Challenges of LICS

The user study highlights the effectiveness and advantages of LICS and PASCAL, but it also uncovers certain limitations and challenges that warrant further investigation. Consider Tables 6 and 7. Comments **P9** and **N1** suggest that as property graph schemas increase in size and complexity, edge crossings has the potential to significantly impact cognitive load, similar to challenges seen in graphs [61]. This effect is more pronounced in LICS due to its wider variety of visual encodings compared to LSG. This is also evident in Table 9. These findings point to the need for future work that integrates additional psychological theories, such as *cognitive load theory* [55], to refine visual abstractions for large schemas.

On the other hand, consider $Q11$, **N3**, and **N5**. These limitations can be addressed by allowing users to customize settings to their preferences (*e.g.,* adjusting circle size, arrowhead size, or the number of properties displayed). Additionally, preferences for label expressions ($Q8$, **N4**) over tree views are anticipated among participants who are familiar with the syntax and semantics of the expressions (some are from the **GD** group).

6 Related Work

Schema Visualization. The most relevant work for this study pertains to schema visualization whose broad goal is to make complex database structures more understandable for design, maintenance, and querying. The key areas of research in this area primarily focused on *visual representation* [16, 20, 30], *schema exploration* [14, 15], and *schema evolution* [11, 13, 33, 41]. Our work emphasizes visual representation for property graph schemas. Existing work on visualizing property graph schemas primarily use variants of LSG which suffer from limitations discussed earlier. To the best of our knowledge, no research utilizes rich visual encoding like LICS or systematically incorporates theories from HCI, psychology, and visualization to inform design decisions.

Since Entity-Relationship (ER) diagrams represent the lower bound of expressiveness in PG-Schema [8], research on their visualization is relevant to our work. We compare LICS with graphical query languages for ER diagrams [7, 29, 42, 54, 62], which have been explored for decades. In most studies, entity types are depicted as nodes (*e.g.,* rectangles) with names, and relationship types are shown as links (*e.g.,* arrows) with cardinality encodings. In contrast, LICS uses composite circles and arrows. Moody [42] presents a "user-friendly" ER diagram representation with visual images, color, and abstraction levels, though its user-friendliness has not been validated. Designs like GRAQULA [54] adopt Shneiderman's golden rules similar to LICS, but none of these efforts systematically incorporate psychological theories.

Query Visualization. Research in *visual query representation*, which involves visualizing a query independently of its results [24, 25, 32, 34, 35, 38, 54, 56], primarily focuses on SQL and property graph queries. *SQLVis* [38], for instance, supports query formulation by depicting tables and relationships as nodes and edges of a graph, respectively, guided by Gestalt and visualization principles.

SQLVis maintains uniformity in node shape, size, and color, while edges are labeled to indicate join conditions and types. *QueryVis* [24, 32] and *Relational Diagrams* [25] are efforts to visually represent the first-order logic and "query patterns" underlying an SQL query, respectively. The former approach adheres to certain principles of visualization (*e.g.,* minimality, effectiveness [36]).

Although design informed by theories and principles is essential for creating technology that serves human needs [12, 47], theory-driven design in HCI is surprisingly uncommon [47]. Recently, Ma *et al.* [34, 35] introduced the theory-informed *labeled composite graph* (LCG) abstraction for visually representing conjunctive property graph queries. The design of LCG is guided by principles from HCI, psychology, and visualization. In contrast, LICS and PASCAL offer richer visual encoding, as property graph schemas contain a wider variety of information than the property graph queries addressed in these earlier works (recall from Section 3.4).

Network Visualization. The visualization community has employed composite circles to represent network properties in various ways [10, 44, 60]. For instance, *Graph Thumbnails* [60] use circle packing to summarize hierarchical network structures, aiding navigation through numerous graphs. *Hybrid layouts* [44] combine different strategies for depicting network topology, integrating *Treemap encoding* [10] with composite circles to represent hierarchies. In contrast, the composite circles in the LICS abstraction are specifically designed for representing Node Types and properties in property graph schemas, arranged only along the circumference. Moreover, its design is informed by principles from cognitive psychology and visualization, distinguishing it from other approaches.

7 Conclusions & Future Work

Visual frameworks for property graph schemas are still in the early stages of development. This paper introduces a new visual abstraction, labeled iconized composite schema (LICS), which goes beyond traditional node-link diagrams (*e.g.,* LSG). LICS enhances usability and effectiveness by incorporating principles from HCI, visualization, and psychology. It is implemented using the Map-Paint algorithm, which constructs the visual representation of property graph schemas through a novel map-paint paradigm. Additionally, we present a new visual interface, PASCAL, and demonstrate the usability and effectiveness of LICS through a user study.

Creating an ER diagram is a standard practice for designing relational databases, but this approach has not been widely adopted for property graph databases yet due to the recency of property graph schemas. We believe that LICS has significant potential in this area. Specifically, LICS can be used to design a property graph database intuitively, which can then be automatically transformed into a property graph schema language (*e.g.,* PG-Schema) to serve as the database schema. This also opens opportunities for research on how effectively users understand LICS and its role in minimizing errors in schema formulation.

In our user study, we focused on conjunctive queries with simple predicates to assess how LICS improves property graph query formulation, as existing visual interfaces like SIERRA only support such queries. However, exploring how LICS can enhance the formulation of complex queries would be valuable. This, requires advancements in visual interface development, as we believe it is unrealistic to expect many domain-specific users to become proficient in *Cypher* in the near future.

Lastly, it is interesting to explore how LICS can be extended by incorporating additional schema features (*e.g.,* range constraints [8], cyclic relationship constraints) and psychological theories (*e.g.,* cognitive load theory [55]) through a theory-informed design approach that balances visual complexity and usability.

Acknowledgment

Sourav S Bhowmick is partially supported by Ministry of Education (MOE) AcRF Tier 1 Grant No. RG19/24.

References

[1] The Aesthetic-Usability Effect. https://www.nngroup.com/articles/aesthetic-usability-effect/ (Last Accessed: Apr 3, 2025).

[2] Cypher Query Language. https://neo4j.com/developer/cypher/.

[3] Gestalt psychology. https://en.wikipedia.org/wiki/Gestalt_psychology.

[4] Neo4j Bloom. https://neo4j.com/bloom.

[5] Neo4j Example datasets. https://neo4j.com/docs/getting-started/appendix/example-data/.

[6] Query Set for User Study. https://personal.ntu.edu.sg/assourav/research/PIANO/LicsQuerySet.pdf.

[7] M. Angelaccio, T. Catarci, G. Santucci. QBD*: A Graphical Query Language with Recursion, *IEEE Trans. Soft. Engin.*, 16(10), 1990.

[8] R. Angles, A. Bonifati, et al. PG-Schema: Schemas for Property Graphs. *Proc. ACM Manag. Data*, 1(2): 198:1-198:25, 2023.

[9] R. Angles, A. Bonifati, et al. PG-Keys: Keys for Property Graphs. *In SIGMOD*, 2021.

[10] D. Archambault, T. Munzner, D. Auber. GrouseFlocks: Steerable Exploration of Graph Hierarchy Space. *IEEE Trans. Vis. Comput. Graph.*, 14(4): 900-913, 2008.

[11] C. Athinaiou, H. Kondylakis. VESEL: VisuaL Exploration of Schema Evolution using Provenance Queries. *In EDBT/ICDT Workshops*, 2019.

[12] S. S. Bhowmick, S. H. Annabel Chen, D. Srivastava. Cognitive Psychology Meets Data Management: State of the Art and Future Directions. *In SIGMOD*, 2024.

[13] T. Bleifuß, L. Bornemann, D. V. Kalashnikov, F. Naumann, D. Srivastava. DBChEx: Interactive Exploration of Data and Schema Change. *In CIDR*, 2019.

[14] A. Bonifati, S.-G. Dumbrava, et al. DiscoPG: Property Graph Schema Discovery and Exploration. *Proc. VLDB Endow.*, 15(12): 3654-3657, 2022.

[15] K. Chen, A. Kannan, J. Madhavan, A. Y. Halevy. Exploring schema repositories with schemr. *SIGMOD Rec.*, 40(1): 11-16, 2011.

[16] C. Anutariya, R. Dangol. VizLOD: Schema Extraction And Visualization Of Linked Open Data. *In JCSSE*, 2018.

[17] W. S. Cleveland, R. McGill. Graphical Perception: Theory, Experimentation and Application to the Development of Graphical Methods. *J. Am. Stat. Assoc.*, 79 (387), 1984.

[18] W. Craig. Gestalt Principles Applied in Design, https://www.webfx.com/blog/web-design/gestalt-principles-applied-in-design/.

[19] L. Deng, M. S. Poole. Affect in Web Interfaces: A Study of the Impacts of Web Page Visual Complexity and Order. *Mis Quarterly*, 34(4), 2010.

[20] S. Devare, M. Koupaee, et al. SAGEViz: SchemA GEneration and Visualization. *In EMNLP*, 2023

[21] L. Faulkner. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3), 2003.

[22] A. Forsythe, N. Sheehy, M. Sawey. Measuring Icon Complexity: An Automated Analysis. *Behaviour Research Methods, Instruments, & Computers*, 35(2), 2003.

[23] N. Francis, A. Green, et al. Cypher: An evolving query language for property graphs. *In SIGMOD*, 2018.

[24] W. Gatterbauer, C. Dunne, H. V. Jagadish, M. Riedewald. Principles of Query Visualization. *IEEE Data Engineering Bulletin*, 45(3), September 2022.

[25] W. Gatterbauer, C. Dunne. On The Reasonable Effectiveness of Relational Diagrams: Explaining Relational Query Patterns and the Pattern Expressiveness of Relational Languages. *Proc. ACM Manag. Data*, 2(1): 61:1-61:27, 2024.

[26] David Gittins. Icon-Based Human-Computer Interaction. *Int. J. Man Mach. Stud.*, 24(6): 519-543, 1986.

[27] S. Harper, C. Jay, E. Michailidou, H. Quan. Analysing the Visual Complexity of Web Pages using Document Structure. *Behaviour & Information Technology*, 32(5), 2013.

[28] S. Jarukasemratana, T. Murata. Recent Large Graph Visualization Tools: A Review. *Information and Media Technologies*, 8(4), 2013.

[29] H.-J. Kim. Graphical Interfaces for Database Systems: A survey. *In the Proceedings of 1986 Mountain Regional ACM Conf.*, 1986.

[30] L. Lace, A. Romane, M. Grasmanis, K. Cerans. A Method of Visual Presentation of Data Schemas. *VOILA@ISWC*, 2023.

[31] J. Lazar, J.H. Feng, H. Hochheiser. Research Methods in Human-Computer Interaction. John Wiley & Sons, 2010.

[32] A. Leventidis, J. Zhang, C. Dunne, W. Gatterbauer, H. V. Jagadish, M. Riedewald. QueryVis: Logic-based Diagrams help Users Understand Complicated SQL Queries Faster. *In SIGMOD*, 2020.

[33] C.-T. Liu, S.-K. Chang, P. K. Chrysanthis. Database schema Evolution Using EVER Diagrams. *In Advanced Visual Interfaces*, 1994.

[34] J. Ma, S. S. Bhowmick, B. Choi, L. Tay. Theories and Principles Matter: Towards Visually Appealing and Effective Abstraction of Property Graph Queries. *In SIGMOD*, 2023.

[35] J. Ma, S. S. Bhowmick, L. Tay, B. Choi. SIERRA: A Counterfactual Thinking-based Visual Interface for Property Graph Query Construction. *In SIGMOD*, 2024.
[36] J. Mackinlay. Automating the Design of Graphical Presentations of Relational Information. *ACM Trans. on Graphics*, 5(2), 1986.
[37] T. Matsubayashi, K. Nishida, T. Hoshiide, K. Fujimura. Graph Visualization of Streaming Data using HTML5. *In Proceedings of 8th Neteco Symposium*, 2012.
[38] D. Miedema, G. Fletcher. SQLVis: Visual Query Representations for Supporting SQL Learners. *In VL/HCC*, 2021.
[39] A. Miniukovich, A. De Angeli. Quantification of Interface Visual Complexity. *In AVI*, 2014.
[40] A. Miniukovich, S. Sulpizio, A. De Angeli. Visual complexity of graphical user interfaces. *In AVI*, 2018.
[41] S. R. Monk. A Graphical User Interface for Schema Evolution in an Object-Oriented Database. *In IDS*, 1994.
[42] D. L. Moody. Graphical Entity Relationship Models: Towards a More User Understandable Representation of Data. *In ER*, 1996.
[43] M. Nadal, E. Munar, G. Marty, C. J. Cela-Conde. Visual Complexity and Beauty Appreciation: Explaining the Divergence of Results. *Empirical Studies of the Arts*, 28(1), 2010.
[44] C. Nobre, M. D. Meyer, M. Streit, A. Lex. The State of the Art in Visualizing Multivariate Networks. *Comput. Graph. Forum*, 38(3): 807-832, 2019.
[45] D. A. Norman. Affordance, conventions, and design. *Interactions*, 6(3): 38-43 (1999).
[46] A. Oliva, M. L. Mack, M. Shrestha, A. Peeper. Identifying the Perceptual Dimensions of Visual Complexity of Scenes. *In Proc. of the 26th Annual Meeting of the Cognitive Sc. Society*, 2004.
[47] A. Oulasvirta, K. Hornbaek. Counterfactual Thinking: What Theories Do in Design. *Int. Journal of Human-Computer Interaction,* 38(1), 2022.
[48] R. Pieters, M. Wedel, R. Batra. The Stopping Power of Advertising: Measures and Effects of Visual Complexity. *Journal of Marketing*, 74(5), 2010.
[49] A. S. Reber. Gestalt psychology. *The Penguin Dictionary of Psychology*, Viking, ISBN 9780670801367, 1985.
[50] Y. Rogers. Icons at the Interface: Their Usefulness. *Interact. Comput.*, 1(1): 105-117, 1989.
[51] R. Rosenholtz, Y. Li, L. Nakano. Measuring Visual Clutter. *Journal of vision*, 7(2), 2007.
[52] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, M. T. Özsu. The ubiquity of large graphs and surprising challenges of graph processing. *PVLDB*, 11(4), 420-431, 2017.
[53] B. Shneiderman, C. Plaisant. Desigining the user interface: Strategies for effective human-computer interaction. 5th Ed., Addison-Wesley, 2010.
[54] G. H. Sockut, L. M. Burns, A. Malhotra, K.-Y. Whang. GRAQULA: A Graphical Query Language for Entity-Relationship or Relational Databases. *Data Knowl. Eng.*, 11(2): 171-202, 1993.
[55] J. Sweller. Cognitive load during problem solving: Effects on learning. *Cognitive science,* 12(2):257–285, 1988.
[56] B. Thalheim. Visual SQL: Towards ER-Based Object-Relational Database Querying. *In ER*, 2008.
[57] A. N. Tuch, E. E. Presslaber, M. Stocklin, K. Opwis, J. A. Bargas-Avila. The Role of Visual Complexity and Prototypicality Regarding First Impression of Websites: Working Towards Understanding Aesthetic Judgments. *International Journal of Human-Computer Studies*, 70, 2012.
[58] J. M. Wolfe. Guided Search 2.0: A Revised Model of Visual Search. *Psychon Bull Rev*, 1: 202–238, 1994.
[59] E. Wong. Shneiderman's Eight Golden Rules Will Help You Design Better Interfaces. https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces, 2021.
[60] V. Yoghourdjian, T. Dwyer, K. Klein, K. Marriott, M. Wybrow. Graph Thumbnails: Identifying and Comparing Multiple Graphs at a Glance. *IEEE Trans. Vis. Comput. Graph.*, 24(12): 3081-3095, 2018.
[61] V. Yoghourdjian, Y. Yang, T. Dwyer, L. Lee, M. Wybrow, K. Marriott. Scalability of Network Visualisation from a Cognitive Load Perspective. *IEEE Trans. Vis. Comput. Graph.*, 27(2): 1677-1687, 2021.
[62] Z.-Q. Zhang, A. O. Mendelzon. A Graphical Query Language for Entity-Relationship Databases. *In ER*, 1983.

Received October 2024; revised January 2025; accepted February 2025