PLAYPEN: Plug-and-Play Visual Graph Query Interfaces for Top-down and Bottom-Up Search on Large Networks*

Zifeng Yuan
Fudan University
China
zfyuan16@fudan.edu.cn

Huey-Eng Chua
Nanyang Technological University
Singapore
hechua@ntu.edu.sg

Sourav S Bhowmick
Nanyang Technological University
Singapore
assourav@ntu.edu.sg

Zekun Ye
Fudan University
China
zkye16@fudan.edu.cn

Byron Choi
Hong Kong Baptist University
Hongkong SAR, China
bchoi@comp.hkbu.edu.hk

Wook-Shin Han
POSTECH
South Korea
wshan@dblab.postech.ac.kr

ABSTRACT

Visual graph query interfaces (VQI) facilitate non-programmers to query graph data effortlessly. The construction of these interfaces for large networks is typically not *data-driven*. That is, they do not exploit the underlying networks to *automatically* generate the contents of various panels of a VQI. Such data-driven construction has several benefits such as facilitating efficient *top-down* and *bottom-up* query formulation and portability of an interface across different application domains and sources. In this demonstration, we present a novel *plug-and-play* visual subgraph query interface construction engine called PLAYPEN that can be *plugged* on any large network G with a *plug specification* b to automatically generate the VQI for G that satisfies b by populating various components of the interface.

CCS CONCEPTS

• **Information systems** → *Query languages for non-relational engines.*

KEYWORDS

Visual query interface, graph search, data-driven, plug-and-play, cognitive load

ACM Reference Format:
Zifeng Yuan, Huey-Eng Chua, Sourav S Bhowmick, Zekun Ye, Byron Choi, and Wook-Shin Han. 2022. PLAYPEN: Plug-and-Play Visual Graph Query Interfaces for Top-down and Bottom-Up Search on Large Networks. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3514221.3520157

1 INTRODUCTION

A recent survey [9] reported that graph query languages and usability are some of the top challenges for graph processing. Although considerable efforts have been invested toward efficient and scalable processing of graphs, these issues have received relatively lesser attention from the data management community. Fortunately, graphs are more intuitive to draw than to compose them in textual format. Hence, a common starting point for addressing these challenges is the deployment of a direct manipulation-based visual query interface (VQI) to enable an end user to draw graph queries interactively instead of formulating them textually using a graph query language. Indeed, several industrial graph querying systems in a variety of domains provide such interfaces [1, 2]. Given that query formulation *precedes* query processing, VQIs naturally facilitate democratization of graph querying frameworks by empowering end users with no programming background to formulate queries and thereby exploit powerful graph processing techniques for their tasks.

Typically, end users follow a *top-down* or *bottom-up* search paradigm for formulating subgraph queries either textually or visually. The former refers to searching graph data based on the user's intuition of what the desired pattern (*i.e.,* query) should look like (*i.e.,* translating a pattern "in-the-head" to a query). The latter refers to search when a user does not have upfront knowledge of what her query graph should look like. She learns the key patterns that exist in the dataset through representative subgraphs to galvanise query formulation. Hence, any superior VQI should facilitate both these search paradigms. In particular, support for bottom-up search is paramount as a large network looks like a "hairball" visually and hence it is cognitively challenging to browse it to figure out which topological patterns one can use to trigger query formulation.

A useful component in a VQI that can facilitate both these search paradigms is a panel containing a set of *patterns* (*i.e.,* small subgraphs) representing the underlying network. Specifically, a pattern enables a user to construct multiple nodes and edges in a subgraph query by performing a *single* click-and-drag action (*i.e., pattern-at-a-time* mode) in lieu of iterative construction of edges (*i.e., edge-at-a-time* mode). Hence, these patterns potentially decrease the time taken to finish a visual query formulation task by reducing the number of formulation steps (steps for brevity) [3, 7, 11]. Importantly, an end user may use these patterns as representative objects to trigger query formulation in bottom-up mode.

The selection of appropriate patterns to be displayed on a VQI is typically performed *manually* based on domain knowledge [3]. Unfortunately, such manual selection is very challenging as it demands a comprehensive knowledge of different topologies in the

*Work done when Zifeng Yuan and Zekun Ye were in Nanyang Technological University.

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA.

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3520157

underlying network data. As a result, the selected patterns may not be diverse enough to expedite the formulation of a wide range of subgraph queries especially in bottom-up mode [7, 11]. Naturally, such manual selection also limits the portability of VQIs across different domains and graph data sources [3].

In this demonstration, we present a novel *data-driven* visual subgraph query interface construction system called PLAYPEN (**PL**ug-**A**nd-pla**Y** subgra**P**h qu**E**ry i**N**terface). Given a large graph or network G, PLAYPEN *automatically* populates various panels (*e.g.,* node labels, patterns) of the VQI from G. Although the set of labels of nodes/edges for populating the *Attribute* panel can be easily generated by traversing the underlying graph, automatically selecting useful patterns is an NP-hard problem [7, 11]. To this end, PLAYPEN selects patterns [11] that have high *coverage* of G and are highly *diverse* and useful for query formulation. Furthermore, it preferentially selects patterns that have potentially low *cognitive load* on end users as patterns with a high load may adversely impact the visual search time [6].

Data-driven selection of VQI components paves the way for *plug-and-play* VQIs [11], which are like a plug-and-play device that can be plugged into any kind of socket (*i.e.,* graph data) and used. A plug-and-play VQI is dynamically built from a high-level specification of pattern properties known as the *plug* [11]. Consequently, PLAYPEN is highly portable as it can automatically construct a VQI for *any* application (*e.g.,* social networks, road networks, biological networks) centered around large networks. Note that PLAYPEN focuses on the interface for query formulation and not for visualization of query results.

It is worth noting that PLAYPEN goes against the traditional mantra of VQI construction. VQIs for graphs are traditionally manually constructed for each data source or domain. We argue that as more and more graph data sources become prevalent in a wide variety of domains, the plug-and-play approach minimizes the cost of development and maintenance of VQIs. PLAYPEN paves the way for a *single* framework to automatically construct the query formulation interface for any domain or source involving large networks.

2 DESIGN PHILOSOPHY

Construction of existing VQIs for large networks is typically performed by developers coding various features of a VQI. In PLAYPEN, we take a fundamentally different approach. It is designed to give end users the freedom to easily and quickly construct a VQI for any network data without resorting to coding by simply "plugging" it on the data. Its design is based on the following four principles.

(1) Work with independent data sources. Our data-driven approach should be able to work with any data source or application domain involving large graphs. PLAYPEN will offer sufficient benefits to developers and end users by making VQI generation effortless with the growing number of sources.

(2) Useful pattern selection. In theory, there are numerous patterns with different topologies that can be selected from a given network. However, many of these patterns may not be useful to end users in supporting top-down and bottom-up search effectively. Hence, it is paramount to select patterns that are potentially "useful" for subgraph query formulation.

(3) Cognitive load-aware pattern selection. A key issue in exposing patterns to support query formulation is that a user should

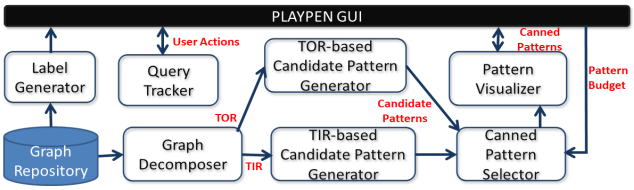

Figure 1: Architecture of PLAYPEN.

be able to visually interpret a pattern (*i.e.,* edge relationships) quickly so that she can determine if it is useful for her query. Such efficient interpretation naturally reduces the overall query formulation time [11]. Hence, the PLAYPEN framework needs to select subgraphs that are not only potentially useful but also impose a low cognitive load on users.

(4) Independence from query logs. Although query logs can provide rich information of the topology of past queries posed on a specific data source, in practice such information is often not publicly available [11]. Hence, PLAYPEN should be able to generate patterns from a specific source without demanding its query logs as input. In the case query logs are available, it should be easily extensible to incorporate them.

These principles enable PLAYPEN to be easily integrated into any network data source. One may plug it on the data to generate a VQI and then install it on top of the query engine.

3 SYSTEM OVERVIEW

Figure 1 depicts the architecture of PLAYPEN. It consists of the following components. The reader may refer to [11] for details.

The GUI module. Figure 2(a) depicts a screenshot of the PLAYPEN visual interface. *Panel 1* enables a user to select a network data source, specify a *plug* [11] for it, and load previously generated patterns. Note that the plug enables her to customize the interface according to her need by specifying the number and sizes of patterns she needs. Different users may specify different plugs for the same or different graph data (socket). *Panel 2* contains a list of distinct vertex labels in the selected dataset. *Panel 3* tracks statistics related to a subgraph query formulation activity. *Panel 4* is used for query formulation. PLAYPEN supports two types of patterns, *default* and *canned*. The former are small-sized patterns that are basic building blocks of networks [8] (*e.g.,* an edge, 2-path, triangle, rectangle) and the latter are those of larger size (*i.e.,* size greater than 3) [11]. *Panels 5* and *6* display the default and canned patterns, respectively.

Observe that the contents of Panels 1 and 4 are provided by a user and the contents of Panel 3 depend on users' actions in Panel 4. On the other hand, the contents of Panels 2, 5, and 6 depend on the graph data. Hence, the goal of PLAYPEN is to populate Panels 2, 5, and 6. In particular, since the default patterns (Panel 5) occur in any large graph, they are invariant for all data sources. The contents of Panels 2 and 6 vary with data sources and hence are generated from the data.

Label generator module. This module traverses the underlying network to generate the set of unique node/edge labels in G, which are then displayed on Panel 2.

Graph decomposer module. A recent analysis of large volumes of SPARQL query logs reveals that the topology of many real-world subgraph queries maps to chains, trees, stars, cycles, petals, and

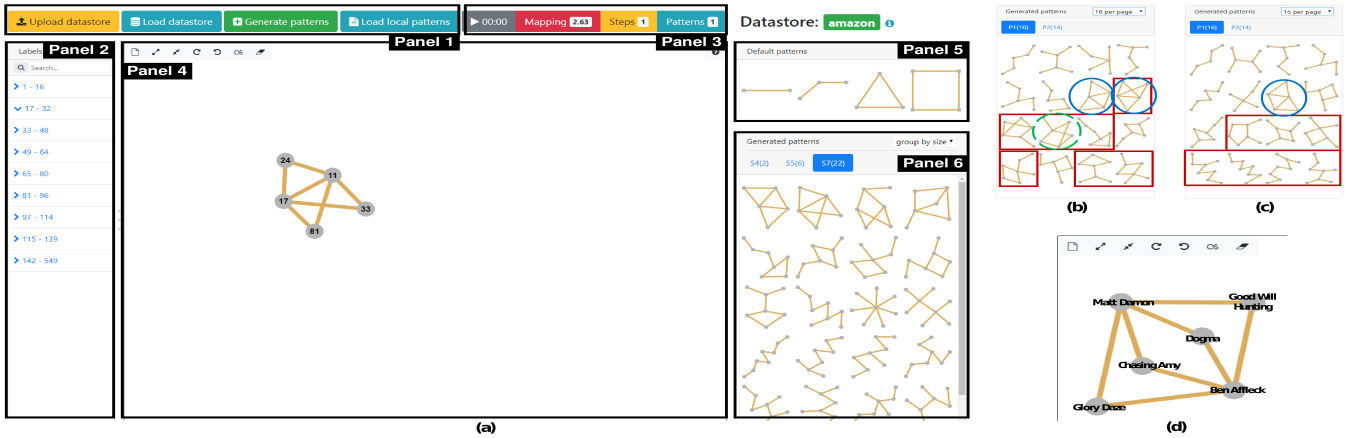

Figure 2: Visual interface constructed by PLAYPEN.

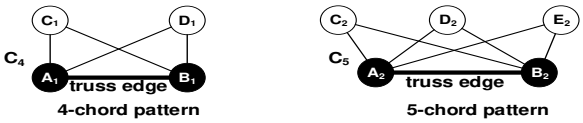

Figure 3: k-chord and composite chord patterns.

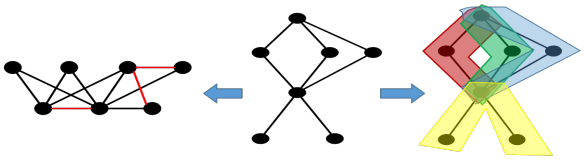

Figure 4: Formulation of flower topology.

flowers [5]. These topologies can be broadly represented by *triangle-like* and *non-triangle-like* substructures [11]. Hence, this module breaks down a large network G into *dense* (containing trusses) and *sparse* (containing non-trusses) regions by leveraging k-trusses [10]. The former region is referred to as *truss-infested region* (TIR graph) and the latter as *truss-oblivious region* (TOR graph), and are denoted by G_T and G_O, respectively. Note that triangle-like structures (*e.g.,* petals, flowers) can be visually formulated from canned patterns extracted from G_T whereas non-triangle-like structures (*e.g.,* chains, stars, cycles) can be efficiently constructed by using canned patterns generated from G_O [11]. Furthermore, the canned pattern generation from G_T and G_O instead of G leads to a more efficient generation process as well as superior quality of patterns [11].

TIR-based candidate pattern generator module. This module is responsible for generating *candidate* canned patterns from a TIR graph. Specifically, PLAYPEN extracts substructures that are more "relaxed" than k-trusses so that they can be efficiently used to formulate queries containing triangle-like structures (*e.g.,* flower, petal) with minimal modifications [11].

PLAYPEN extracts two types of k-truss-based structures as candidate patterns from TIR, namely, *k-chord patterns* (k-CP) and *composite chord patterns* (CCP). Intuitively, a k-CP is a connected graph containing a *truss edge* e (*i.e.,* edge belonging to a k-truss) and k-2 triangles of e. A k-CP can be considered as a building block of k-trusses since it is found with respect to each edge in a given k-truss. Examples of k-CPs (4-CP and 5-CP) are illustrated in Figure 3. In order to facilitate the selection of larger canned patterns with greater structural diversity, PLAYPEN *combines* k-CPs to yield CCPs that occur in G_T. In particular, it generates a CCP by merging a *single* edge of two k-CPs as it reduces the complexity of CCP generation. Consider the flower topology of a query in Figure 4 (middle). The edge-at-a-time construction mode requires 15 steps. In contrast, the pattern-at-a-time mode using a CCP and deleting three edges (highlighted in red) from it takes 4 steps. Hence, it may take a significantly lesser number of steps if truss-like structures such as CCPs are utilized.

TOR-based candidate pattern generator module. This module extracts stars, paths, cycles, and small connected subgraphs of unique topology from a TOR graph [11]. First, it extracts *k-star* and *asterism* patterns using a breadth-first search. A *k-star* is a connected subgraph containing a vertex r (*center vertex*) where the remaining vertices are connected only to r and $k \geq \epsilon$. *Asterism* patterns are formed by *merging* k-stars on a pair of edges. Figure 5 depicts a combination of a 4-star with a 5-star by merging them on the bold edges. Next, it extracts all k-paths ($k > 2$) and k-cycles ($k > 4$), leaving small connected components in the resultant graph, which are then harvested.

In summary, queries containing trees and forests can be easily formulated by combining k-paths, k-stars, and asterism patterns and by growing them using one or more edges (*i.e.,* default patterns) when required. Similarly, chain queries can be formulated using k-paths. Petal and flower structures can be efficiently formulated using k-CP/CCP or a set of k-paths.

Canned pattern selector module. Given a user-specified *plug* $b = (\eta_{min}, \eta_{max}, \gamma)$ where η_{min} (resp. η_{max}) is the minimum (resp. maximum) size of a canned pattern and γ is the number of patterns to be displayed on Panel 6, the aim of this module is to select a set of canned patterns $\mathcal{P}$ satisfying b from the set of candidate patterns. Note that it selects patterns with $\eta_{min} > 3$. To this end, it implements a selection algorithm that guarantees $\frac{1}{e}$-approximation and exploits a novel *pattern set score* that is sensitive to coverage, diversity, and cognitive load (*i.e.,* memory demand or mental effort required to perform pattern selection visually) of patterns. Particularly, it is selected by maximizing coverage and diversity and minimizing the cognitive load of $\mathcal{P}$. Although coverage of $\mathcal{P}$ is intuitive, the need for structurally diverse patterns is to make efficient use of the limited display space on a VQI. For example, the

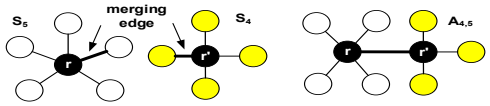

Figure 5: Star and asterism patterns.

4-star and 5-star patterns in Figures 5 are structurally very similar and hence both should not be selected for display. Note that a user views the canned patterns in Panel 6 during query formulation and determines the ones that are relevant. Hence, given two candidate patterns p_1 and p_2, PLAYPEN prefers p_1 to p_2 if p_1 has a lower cognitive load than p_2. The computation of coverage, diversity, and cognitive load is detailed in [11].

Pattern visualizer module. This module facilitates visualization of patterns on the VQI. All patterns are displayed using the force-directed layout. A user may select different options (group-by-size, single-page, x-per-page) to display the canned patterns.

Query tracker module. Finally, this module tracks various information related to subgraph query formulation activities (*e.g.,* number of steps taken, query formulation time).

4 RELATED SYSTEMS

Classical visual subgraph querying systems [1, 2] do *not* construct a VQI in a data-driven manner. The work most germane to PLAYPEN are efforts in [4, 7, 12] that focus on the data-driven construction of VQIs on a large collection of small- or medium-sized data graphs (*e.g.,* chemical compounds). PLAYPEN focuses on *large* networks and differs from them both *externally* and *internally*. Under the hood, the strategy of clustering data graphs and then summarizing each cluster as realized in [4, 7, 12] is ineffective and inefficient for large graphs [11]. Furthermore, labelled patterns are used in [4, 12] whereas unlabelled patterns are more useful in large networks as nodes/edges in a network may have many attribute-value pairs [11]. Hence, the key modules related to the core engine of PLAYPEN do not exist in [4, 7, 12].

Externally, the user experience with PLAYPEN is different from [4, 7, 12]. First, the canned patterns exposed to users in PLAYPEN are topologically different as topologies of small data graphs (*e.g.,* chemical compounds) are often very different from large networks. Second, the cognitive load inflicted by large networks during bottom-up search plays a substantially more significant role compared to a small- or medium-sized data graph containing only a few (10-200) nodes. The large size of a network makes it prohibitive to undertake bottom-up search effectively without the aid of canned patterns.

5 DEMONSTRATION OVERVIEW

PLAYPEN is implemented in Java JDK 1.8 and Javascript 1.6. Our demonstration will be loaded with a few real networks (*e.g., Amazon*, *RoadNet-TX*) with millions of nodes and edges. Example query graphs that can be constructed using patterns will be presented for formulation. Users can also draw ad hoc visual queries. A **video** to illustrate the main features using example use cases is available at https://youtu.be/nS_nFQQN_Ck. The key objectives of the demonstration are the followings.

Data-driven construction of VQIs. Through PLAYPEN's visual interface (Figure 2(a)), the audience will be able to select a graph data source and plug using Panel 1 to automatically construct the contents of the panels *within a few minutes* due to the realization of an efficient canned pattern selection strategy [11]. One will also be able to interactively select different graph repositories as well as plugs (through Panel 1) to appreciate the portable and data-driven nature of PLAYPEN. For example, Figures 2(b) and 2(c) depict some of the patterns selected from *Amazon* and *RoadNet-TX*, respectively. Specifically, we generate 30 canned patterns of sizes between 4 and 15 from each dataset. In particular, patterns encapsulated by blue ellipses (solid lines) in Figures 2(b) and (c) are examples of some that are derived from G_T. Observe that the patterns are different (topologically) for different datasets. Specifically, the patterns in red rectangle boxes in Figures 2(b) are not found in Figures 2(c) and vice versa. This emphasizes the fact that different datasets may expose different collections of canned patterns. Second, the audience can experience a low cognitive load associated with the displayed patterns, enabling them to easily recognize a topology with a quick glance during query formulation.

Efficient top-down and bottom-up subgraph search. After generating the VQI for the chosen network data, the audience can experience the benefits of PLAYPEN for top-down and bottom-up search. In particular, we shall request them to first formulate a query without using any canned pattern. As remarked earlier, this may be challenging to them, especially for bottom-up search as without a clear "pattern-in-head" beyond default patterns (*e.g.,* triangle, wedge, rectangle), it is hard to initiate a meaningful search. After appreciating the difficulty of subgraph query formulation without the aid of canned patterns, we shall ask them to use Panels 2, 5, and 6 to formulate queries that they may be interested in. For instance, suppose one is interested in making a new movie involving *Ben Affleck* and *Matt Damon*. She would like to identify other actors that have prior working experiences with them in the *Amazon* dataset. She may be unsure of the topology of her query for finding some valid results. Hence, she may undertake a bottom-up search and browse Panel 6 to select a k-chord pattern (highlighted by a green ellipse with a broken line in Figure 2(b)) to initiate her query formulation. She may then construct the query in Figure 2(d) by utilizing it and an edge. Specifically, the query formulation takes 8 steps ($\sim$ 20s), which includes steps for vertex label assignment. In comparison, the edge-at-a-time mode requires 20 steps ($\sim$ 39s).

Acknowledgements. The first four authors were supported by the AcRF Tier-2 Grant MOE2015-T2-1-040. Byron Choi is supported by HKRGC GRF 12201119 and 12201518. Wook-Shin Han was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2021R1A2B5B03001551).

REFERENCES

[1] Neo4j Bloom. https://neo4j.com/bloom.
[2] PubChem VQI. https://pubchem.ncbi.nlm.nih.gov/edit3/index.html.
[3] S. S. Bhowmick, et al. Data-driven visual graph query interface construction and maintenance: challenges and opportunities. *PVLDB*, 9(12), 2016.
[4] S. S. Bhowmick, et al. AURORA: Data-driven Construction of Visual Graph Query Interfaces for Graph Databases.*In SIGMOD*, 2020.
[5] A. Bonifati, et al. An Analytical Study of Large SPARQL Query Logs. *In PVLDB*, 11(2), 2017.
[6] W. Huang, P. Eades, S.H. Hong. Measuring effectiveness of graph visualizations: A cognitive load perspective. *Inf. Vis.*, 8(3), 2009.
[7] H. Kai, H.-E Chua, et al. CATAPULT: data-driven selection of canned patterns for efficient visual graph query formulation. *In SIGMOD*, 2019.
[8] R. Milo, et al. Network motifs: simple building blocks of complex networks. *Science*, 298, 2002.
[9] S. Sahu, et al. The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. *PVLDB*, 11(4), 2017.
[10] J. Wang, J. Cheng. Truss decomposition in massive networks. *In PVLDB*, 5(9): 812-823, 2012.
[11] Z. Yuan, et al. Towards Plug-and-play Visual Graph Query Interfaces: Data-driven Canned Pattern Selection for Large Networks. *PVLDB*, 14(11), 2021.
[12] J. Zhang, et al. DaVinci: Data-driven visual interface construction for subgraph search in graph databases. *In ICDE*, 2015.