VINCENT: Towards Efficient Exploratory Subgraph Search in Graph Databases

Kai Huang ^{§,†}, Qingqing Ye[†], Jing Zhao [§], Xi Zhao [§], Haibo Hu[†], Xiaofang Zhou [§]

[§]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology [†]Department of Electronic and Information Engineering, Hong Kong Polytechnic University ustkhuang|xizhao|zxf@ust.hk,qqing.ye|haibo.hu@polyu.edu.hk,jzhaobq@connect.ust.hk

ABSTRACT

Exploratory search is a search paradigm that plays a vital role in databases, data mining, and information retrieval to assist users to get familiar with the underlying databases. It supports iterative query formulation to explore the data space. Despite its growing importance, exploratory search on graph-structured data has not received adequate attention in the literature. In this paper, we demonstrate a novel system called VINCENT that facilitates an efficient exploratory subgraph search in a graph database containing a large collection of small or medium-sized graphs. By automatically generating the content for panels in GUI and diversified patterns from databases and providing a visual result explorer, VINCENT supports data-driven visual query formulation, incremental subgraph processing, and efficient query result summarization.

PVLDB Reference Format:

Kai Huang, Qingqing Ye, Jing Zhao, Xi Zhao, Haibo Hu, Xiaofang Zhou. VINCENT: Towards Efficient Exploratory Subgraph Search in Graph Databases . PVLDB, 15(12): 3634-3637, 2022. doi:10.14778/3554821.3554862

1 INTRODUCTION

Graph databases for small or medium-sized data graphs have been extensively studied in the literature. It has become increasingly prevalent in a variety of real-life applications, such as pattern recognition, social networks and chemoinformatics. To retrieve valuable information from the underlying databases, many query primitives have been developed. Subgraph search is an important type of query as well as a fundamental task for graph data management. Given a query graph q and a graph database $D = \{G_1, G_2, ..., G_i, ..., G_n\},\$ subgraph search is to find all data graphs G_i that contain q as a subgraph (i.e., subgraph search) or approximately contain q allowing a few missing edges (i.e., subgraph similarity search). Although recent years have witnessed a rapid development of subgraph (similarity) search, the vast majority of these efforts have focused on "lookup" retrieval with the assumption that users have a clear intent and sufficient knowledge of the underlying graph database such that they can accurately specify their search goal in the form of a connected query graph. However, this assumption is clearly impracticable as graph databases grow rapidly in size.

One may resort to exploratory search to address this problem, as exploratory search provides a search paradigm that goes beyond such lookup retrieval and typically involves users who may not be familiar with the underlying data in a specific domain [1]. In particular, it can assist users to iteratively or progressively formulate queries, explore the query results, know the infrastructure of the underlying databases, and identify possible search directions. For example, an user wants to query a substructure q, but she does not have precise knowledge of the subgraph structure due to the topological complexity of data graphs. This hinders the query formulation, *i.e.*, precisely representing the query graph q. If there is an efficient tool that supports exploratory search, Mary can formulate an initial query graph (e.g., a subgraph of q) and then iteratively formulate the query and explore the query results, and finally identify the exact query q. In recent years, exploratory search in relational databases has attracted a great deal of attention [4], but exploratory search on graph-structured data has not received adequate attention in the literature. [5] is the first exploratory subgraph search framework, which has many disadvantages such as the inability to summarize query results for better exploring experience.

In this demonstration, we present a novel exploratory subgraph search engine called VINCENT, which has the following innovative features. First, it generates a set of diversified patterns called TED patterns (*i.e.*, <u>T</u>op-k <u>Edge-D</u>iversified patterns), which can summarize the characteristic of the underlying databases and the query results. In particular, TED patterns achieves a guaranteed approximation ratio of edge coverage and its generation process requires limited memory. Second, it provides a visual query interface that supports user-friendly query formulation with the aid of TED patterns, query rewrite, and efficient progressive query processing. Third, it embodies the *query results explorer* to analyze various features of the query results during the exploration process to better facilitate understanding of the data space. *Query results explorer* can take advantage of the TED patterns to guide the search directions.

2 SYSTEM ARCHITECTURE

Figure 1 shows the architecture of VINCENT, which consists of five modules, *TED Pattern Generator*, *Query Editor*, *Index Constructor*, *Query Processor*, and *Query Results Explorer*. Given a graph database *D*, *TED Pattern Generator* module generates the diversified patterns (*i.e.*, TED Patterns) from *D* to summarize the database information. *Query Editor* module automatically generates the visual query interface to support the visual query formulation by displaying these TED Patterns and efficient query processing with the aid of *Query Processor*. As subgraph search is an NP-hard problem, VINCENT follows the "filter-and-verification" paradigm, *i.e.*, using indices

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097. doi:10.14778/3554821.3554862

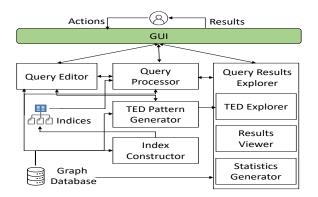


Figure 1: The architecture of VINCENT.

to filter some candidates to reduce verification. *Index Constructor* module implements such a paradigm by constructing indices from the database. Based on the generated indices, the *Query Processor* module can efficiently process subgraph queries in a progressive manner. To provide better exploratory experience, *Query Results Explorer* that utilizes TED Patterns for exploring and summarizing query results is presented. More details are discussed below.

TED Pattern Generator module. The TED Pattern Generator module proposes the TED algorithm to generate top-*k* TED Patterns $\mathcal{P} = \{g_1, g_2, ..., g_j, ..., g_k\}$ such that (nearly) maximum number of edges in $D = \{G_1, G_2, ..., G_i, ..., G_n\}$ can be covered/matched by \mathcal{P} .

DEFINITION 1 (SUBGRAPH ISOMORPHISM). Given two graphs G_1 and G_2 , a subgraph isomorphism is an injective function $f: V(G_1) \rightarrow V(G_2)$ such that $1) \forall v \in V(G_1), l(v) = l'(f(v))$ and $2) \forall (u, v) \in E(G_1), (f(u), f(v)) \in E(G_2)$ and l(u, v) = l'(f(u), f(v)) where l and l' are the labeling functions of graph G_1 and G_2 , respectively.

 G_1 is subgraph isomorphic to G_2 if there is at least one subgraph isomorphism f from G_1 to G_2 . We also say that G_2 is covered by G_1 (denoted by $G_1 \subseteq G_2$). Given a subgraph isomorphism f and the subgraph G' of G_2 consisting of vertices f(v) and edges (f(u), f(v))where $u, v \in V(G_1)$, we say that G' is a matching of G_1 in G_2 . The edge (f(u), f(v)) is the covered edge.

DEFINITION 2 (COVER SET AND COVERAGE). Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, if G_1 is subgraph isomorphic to G_2 and the matchings are \mathcal{F} , the cover set of G_1 over G_2 is $Cov(G_1, G_2) = \bigcup_{f \in \mathcal{F}} (f(u), f(v))$ and the coverage is $|Cov(G_1, G_2)|$.

The cover set of a set of graphs $\mathcal{G}_1 = \{G_1, G_2, ..., G_i, ..., G_n\}$ over graphs $\mathcal{G}_2 = \{G'_1, G'_2, ..., G'_j, ..., G'_m\}$ is defined as $Cov(\mathcal{G}_1, \mathcal{G}_2) = \bigcup_i \bigcup_j Cov(G_i, G'_j)$ where $i \in [1, n]$ and $j \in [1, m]$.

Hence, TED patterns \mathcal{P} are graphs that maximize $|Cov(\mathcal{G}_1, \mathcal{G}_2)|$. In general, TED algorithm alternately performs subgraph enumeration and search process to generate these top-k patterns. For subgraph enumeration, it adopts a depth-first search (DFs) strategy to traverse the search space. For top-k pattern search on the enumerated subgraphs, a swapping-based strategy is adopted to maintain patterns with limited memory consumption. In particular, it first enumerates all 1-sized subgraphs (*i.e.*, edges) and appends them to the set S_p . Then an iterative process is performed to generate final patterns \mathcal{P} by taking S_p and D as inputs. Specifically, each subgraph $g \in S_p$ is first considered and removed from S_p . Then, the procedure PATMAINTAIN is performed to update top-k patterns \mathcal{P} with newly enumerated subgraph g. After that, the right-most extension method [6] extends each subgraph $g \in S_p$ with one more edge so that its supergraphs will be considered in the next iteration. The process repeats until S_p is empty. The details of PATMAINTAIN procedure are discussed below.

Pattern Maintenance (PATMAINTAIN). As the max k-cover problem is a subproblem of top-k TED patterns discovery, greedy searchbased solutions typically find entire subgraphs and store them in memory and hence cannot be effectively exploited for a large database. To address this, PATMAINTAIN maintains only k patterns in memory with a swapping-based method motivated by the existing maximum coverage solver in the context of a streaming scenario. We first introduce the concepts of loss score and benefit score below to facilitate exposition. Given a pattern set \mathcal{P} and a database D, the loss score of a pattern $p \in \mathcal{P}$ is the decrease in total coverage caused by removing p from \mathcal{P} , i.e., SCORE_L(p, \mathcal{P} , D) = $|\cup_{p \in \mathcal{P}} Cov(p, D) \setminus \cup_{p' \in \mathcal{P} \cup p} Cov(p', D)|$. The benefit score of a pattern $g \notin \mathcal{P}$ is the increase of total coverage caused by adding g to \mathcal{P} , i.e., SCORE_B(g, \mathcal{P} , D) = $|\cup_{p' \in \mathcal{P} \cup \{q\}} Cov(p', D) \setminus \bigcup_{p \in \mathcal{P}} Cov(p, D)|$.

PATMAINTAIN first greedily selects k patterns into the pattern set \mathcal{P} . When a new subgraph q is generated, a swapping-based process is developed to determine if *q* should be swapped into \mathcal{P} . Specifically, it first calculates and ranks the loss scores for each pattern $p \in \mathcal{P}$, and then records the pattern p_t and its pattern SCORE SCORE such that p_t has a minimum loss score. Meanwhile, the benefit score SCORE_B of q is also recorded. The subgraph qis considered as a promising candidate and swapped into \mathcal{P} if $\text{SCORE}_B > (1+\alpha)\text{SCORE}_L + (1-\alpha)|Cov(\mathcal{P}, D)|/k$ is satisfied, where $\alpha \in [0, 1]$ is a swapping threshold for balancing loss score Score_I and average coverage of patterns in \mathcal{P} . The pattern p_t is swapped out if *q* is swapped in. The pattern set \mathcal{P} is hence updated. Note that the approximation ratio (w.r.t., total coverage) of patterns $\mathcal P$ is bounded by $|Cov(\mathcal{P}, D)| / |Cov(\mathcal{P}_{opt}, D)| \geq \frac{1}{4}$ where \mathcal{P}_{opt} is the optimal solution and can be obtained by greedily enumerating all subgraphs and generating all possible combinations of k subgraphs.

THEOREM 1. Let \mathcal{P}_{opt} be an optimal solution to the TED patterns discovery problem. The approximation ratio of the patterns \mathcal{P} generated by TED is bounded by $\frac{|Cov(\mathcal{P},D)|}{|Cov(\mathcal{P}_{opt},D)|} \geq \frac{1}{4}$.

We can prove this theorem by reducing it to the *Max k-cover problem*: given a number *m* and a collection of sets *S*, the *Max k-cover* problem aims to find a set $S' \subset S$ such that |S'| = m and the number of covered elements is maximized. This problem has a $\frac{1}{4}$ -approximation solution when the swapping strategy is adopted [7]. Observe that the problem has the same setting as our problem if all promising patterns are generated. In addition, the same swapping strategy [7] is adopted by default. Hence, the approximation ratio of patterns \mathcal{P} is bounded by $|Cov(\mathcal{P}, D)|/|Cov(\mathcal{P}_{opt}, D)| \ge \frac{1}{4}$, which is the best known bound for the maximum k-cover problem in the context of a streaming scenario.

Query Editor module. Figure 2 depicts the screenshot of the visual query interface of VINCENT, which consists of four panels. Panel 1 allows us to choose a dataset, load the database into memory, create a query, build an index, and execute query processing. When

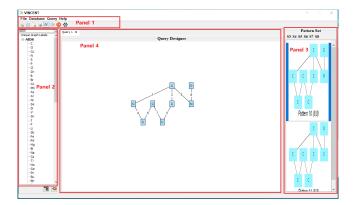


Figure 2: The interface of VINCENT.

users load a database, Panel 2 that lists all the labeled vertices is automatically generated by a depth first search on the graphs. Panel 3 displays the TED patterns and groups them by their size. Panel 4 provides a canvas where users can formulate the query graph by dragging-and-dropping TED patterns from Panel 3 and labeled vertices from Panel 2 as well as adding edges to connect disconnected parts. Moreover, VINCENT supports query rewrite on Panel 3 and progressively executes the evolving queries by clicking the "Run" button in Panel 1.

Index Constructor module. This module aims to build indices for the underlying database to facilitate efficient exploratory visual subgraph search. The indices in our context should not only process the partially formulated query graph, but also utilize previous query results to quickly generate the query results for the current query. To this end, this module implements an action-aware indexing framework called PRAGUE [9]. PRAGUE index is built upon *frequent fragments* and *discriminative infrequent fragments* (DIF for short). Frequent fragments is also known as frequent subgraphs, which is generated by gSpan algorithm [6]. The data graph G_i in D that contains a frequent fragment g is called the *frequent support fragment* (FSG for short) of g. For ease of presentation, we also use fsgIds(g) to denote the set of identifiers of FSGS of g.

Based on frequent fragments and discriminative infrequent fragments, the indices consisting of the action-aware frequent index (A^2F) and the action-aware infrequent index (A^2I) are constructed. A^2F is used to prune the graphs that contain none of the frequent fragments which are contained by the query. While A^2I is designed for pruning infrequent spaces with the aid of DIFS.

Query Processor module. The query processor module is to utilize the A^2F and A^2I indices to improve query efficiency. Given a query q, this module generally consists of two procedures, the offline computation and the online processing. The former is to filter the data graphs that are definitely not the supergraph of g and return the filtered results R_q . This can be cast as an offline process as it is can be done by utilizing the GUI latency. The latter is to further verify the query results over R_q by executing subgraph isomorphism testing after the "Run" button is clicked.

Observe that users involved in exploratory subgraph search tend to iteratively reformulate and re-execute a query fragment by adding new query graphs (*i.e.*, TED patterns from Panel 3) or nodes

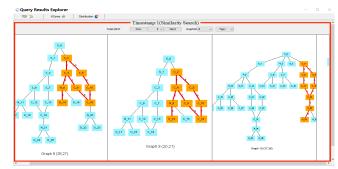


Figure 3: Query Results Explorer.

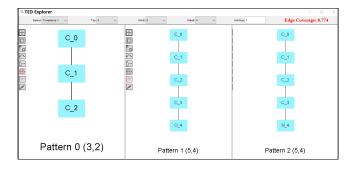


Figure 4: TED Explorer.

(i.e., labeled vertices from Panel 2), the edge list eList is adopted to store all newly added edges including those of TED patterns. These edges are gradually added to the query q such that q always remains connected and a dynamic index called SPIG set [9] is constructed on the fly. The identifiers of data graphs (i.e., FSG identifiers) that contains the query fragment q (denoted by R_q) are then obtained. As q can be a frequent fragment, DIF, and non-DIF, different steps are introduced. If q is a frequent fragment, the fsgIds(q) can be retrieved from A^2I ; if it is a DIF, then fsgIds(q) can be retrieved from A^2F ; otherwise, the SPIG set and the action-aware indices are teamed to generate the candidate set. If R_q is empty at a specific step, a similarity search process are implemented by using the SPIG set to identify relevant subgraphs of q, which need to be matched for retrieving candidates. Once a user clicks on the "Run" button, the online processing is activated. If q is a frequent fragment or DIF, R_q is directly derived; otherwise, subgraph isomorphism testing, e.g., VF2, for exact search or mccs-based similarity verification [9] is used to verify the query results in R_q .

Query Results Explorer module. This module enables users to analyze and explore the query results to identify possible search directions. To this end, it includes three submodules, *Results Viewer*, *TED Explorer*, and *Statistics Generator*. As depicted in Figure 3, *Results Viewer* (the part circled in red, Figure 3) provides a multistream results viewer to view and analyze former and current results during an exploratory search in a user-friendly manner. The multi-stream results viewer enables users to view the query results, highlighted matched parts, total number of matchings, etc. *TED Explorer* module is to generate and display the TED patterns for the query results. As TED patterns can summarize the characteristic of graphs, this module can enable users to better understand the results space. When users click the "TED" button (top left, Figure 3), the TED Explorer is activated and TED patterns are generated (see Figure 4). As shown in this figure, users are allowed to specify parameters such as minimum/maximum number of edges (*i.e.*, MinE and MaxE), value of k, minimum support, etc. The total edge coverage is also displayed accordingly. *Statistics Generator* module can generate different topological and statistical properties for different search streams, and allow users to view and compare them by clicking the "KCores" and "Distribution" buttons (next to "TED" button, Figure 3).

3 RELATED SYSTEMS AND NOVELTY

There is extensive work in classical subgraph search and subgraph similarity search, involving not only a set of small or medium-sized graphs [13] but also a single large graph [8, 11]. They focus on efficient subgraph query processing methods instead of exploratory subgraph search. Although [3, 9, 12] can better support exploratory subgraph search or visual query formulation, they are orthogonal to our work as they are necessary steps in VINCENT framework.

Research prototypes on subgraph search such as GBLENDER [10] and AURORA [14] have recently been demonstrated in data management venues. However, these efforts do not focus on exploratory search. GBLENDER [10] is such a demonstration system that blends the formulation and processing of visual subgraph queries. The query graph was executed once in contrast to iterative query reformulation and execution during an exploratory search. AURORA [14] is designed for the construction of data-driven visual subgraph query interfaces.

VINCENT is built on top of the first exploratory subgraph search framework PICASSO [5]. However, PICASSO has the following disadvantages: 1) the visual patterns PICASSO generated have no quality guarantee in terms of edge coverage. In contrast, the TED patterns (Panel 3, Figure 2) generated by VINCENT can achieve a guaranteed approximation ratio of edge coverage (see *Theorem* 1) and the generation process requires limited memory; 2) PICASSO cannot generate summary information for the query results while VINCENT utilizes *TED Explorer* to facilitate identifying possible search directions.

4 DEMONSTRATION OVERVIEW

VINCENT is implemented in Java JDK 1.8. It will be loaded with several real datasets including AIDS¹, and EMOLECULES². Example query graphs will be presented for exploratory search. Users can also write their own ad-hoc queries through our GUI. The key objective of the demonstration is to enable the audience to interactively experience multiple reformulations of the initial subgraph query in a progressive manner to learn about the underlying data space to identify possible search directions. In particular, it enables the audience to interactively experience the following.

Scenario 1: Data-driven visual query formulation. The visual query interface of VINCENT (Figure 2) enables the audience to load databases into memory (Panel 1, Figure 2), generate diversified patterns (*i.e.*, TED patterns) from the databases to display on the GUI (Panel 3), create a canvas to visually formulate and reformulate

queries (Panel 4) by dragging-and-dropping TED patterns from Panel 3 and labeled vertices from Panel 2. Consequently, one will be able to formulate visual subgraph queries effortlessly over different graph databases.

Scenario 2: Incremental subgraph processing. VINCENT enables the audience to gain a better query experience by exploring the underlying data graphs through iterative refinement of a subgraph query, incrementally generating results of a query fragment in real time by clicking the "Run" button in Panel 1 (Figure 2) to leverage the Query Processor module.

Scenario 3: Efficient query result summarization. VINCENT enables users to analyze and explore query results to identify possible search directions. When users click the "TED" button (top left, Figure 3), the TED Explorer is activated and TED patterns are generated (see Figure 4). The audience is able to view the summary information and specify the parameters such as minimum/maximum number of edges to obtain more details. They can also acquire different topological and statistical properties for different search streams, view and compare them by clicking the "KCores" and "Distribution" buttons (Figure 3).

A demonstration video is publicly available at https://www. youtube.com/video/rP9Csi2oJTo/.

ACKNOWLEDGMENTS

The research work described in this paper was partially conducted in the JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust. This work was also supported by the National Natural Science Foundation of China (Grant No: 62072390 and 62102334), and the Research Grants Council, Hong Kong SAR, China (Grant No: 15222118, 15218919, 15203120, 15226221, 15225921 and C2004-21GF).

REFERENCES

- White R W, Roth R A. Exploratory search: Beyond the query-response paradigm. Synthesis lectures on information concepts, retrieval, and services, 1(1): 1-98, 2009.
- [2] Ahn J, Brusilovsky P. Adaptive visualization for exploratory information retrieval. Info. Proc. & Man., 49(5): 1139-1164, 2013.
- [3] Huang K, Chua H E, et al. MIDAS: Towards Efficient and Effective Maintenance of Canned Patterns in Visual Graph Query Interfaces. In SIGMOD, 2021.
- [4] Idreos S, Papaemmanouil O, Chaudhuri S. Overview of data exploration techniques. In SIGMOD, 2015.
- [5] Huang K, Bhowmick S S, Zhou S, et al. PICASSO: exploratory search of connected subgraph substructures in graph databases. *In PVLDB*, 2017.
- [6] Yan X, Han J. gspan: Graph-based substructure pattern mining. In ICDE, 2002.
 [7] Feige U. A threshold of ln n for approximating set cover. Journal of ACM, 45(4): 634-652, 1998.
- [8] Huang K, Hu H, Zhou S, et al. Privacy and efficiency guaranteed social subgraph matching. The VLDB Journal, 31(3): 581-602, 2022.
- [9] Jin C, Bhowmick S S, Choi B, et al. prague: A practical framework for blending visual subgraph query formulation and query processing. *In ICDE* 2012
- visual subgraph query formulation and query processing. *In ICDE*, 2012.
 [10] Jin C, Bhowmick S S, Xiao X, et al. GBLENDER: visual subgraph query formulation meets query processing. *In SIGMOD*, 2011.
- [11] Bi F, Chang L, Lin X, et al. Efficient subgraph matching by postponing cartesian products. In SIGMOD, 2016.
- [12] Huang K, Chua H E, Bhowmick S S, et al. CATAPULT: data-driven selection of canned patterns for efficient visual graph query formulation. In SIGMOD, 2019.
- [13] Katsarou F, Ntarmos N, Triantallou P. Performance and Scalability of Indexed Subgraph Query Processing Methods. In PVLDB, 2015.
- [14] Bhowmick S S, Huang K, Chua H E, et al. AURORA: data-driven construction of visual graph query interfaces for graph databases. In SIGMOD, 2020.

¹https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data ²https://www.emolecules.com/info/plus/download-database