Interactive Visual Pattern Search on Graph Data via Graph Representation Learning



Huan Song, Zeng Dai*, Panpan Xu*, Liu Ren

Fig. 1. The visualization interface of GraphQ contains: (1) A query editing panel to specify the subgraph patterns and initiate the search. (2.1) (2.2) Query result panels to display the retrieved results. The graph thumbnails can be displayed in overview and detail modes. (3) A statistics and filtering panel that helps users select a graph to construct example-based query, and visualizes the distribution of the query results in the database. (4) A query option control panel to specify whether fuzzy-pattern search is enabled and whether the node-match should be highlighted. (5) A popup window for pairwise comparison between the query pattern and the returned result. The figure shows a case study on program workflow graph pattern search and the details are described in Section 5.1.

Abstract— Graphs are a ubiquitous data structure to model processes and relations in a wide range of domains. Examples include control-flow graphs in programs and semantic scene graphs in images. Identifying subgraph patterns in graphs is an important approach to understand their structural properties. We propose a visual analytics system GraphQ to support human-in-the-loop, example-based, subgraph pattern search in a database containing many individual graphs. To support fast, interactive queries, we use graph neural networks (GNNs) to encode a graph as fixed-length latent vector representation, and perform subgraph matching in the latent space. Due to the complexity of the problem, it is still difficult to obtain accurate one-to-one node correspondences in the matching results that are crucial for visualization and interpretation. We, therefore, propose a novel GNN for node-alignment called NeuroAlign, to facilitate easy validation and interpretation of the query results. GraphQ provides a visual query interface with a query editor and a multi-scale visualization of the results, as well as a user feedback mechanism for refining the results with additional constraints. We demonstrate GraphQ through two example usage scenarios: analyzing reusable subroutines in program workflows and semantic scene graph search in images. Quantitative experiments show that NeuroAlign achieves 19%–29% improvement in node-alignment accuracy compared to baseline GNN and provides up to 100x speedup compared to combinatorial algorithms. Our qualitative study with domain experts confirms the effectiveness for both usage scenarios.

Index Terms—Graph, Graph Neural Network, Representation Learning, Visual Query Interface

1 INTRODUCTION

- Huan Song and Liu Ren are with Robert Bosch Research and Technology Center, USA. E-mail: huan.song, liu.ren@us.bosch.com.
- Zeng Dai is with ByteDance Inc. E-mail: zeng.dai@bytedance.com.
- Panpan Xu is with Amazon AWS AI. E-mail: xupanpan@amazon.com.
 Work done while the authors were with Robert Bosch Research and Technology Center, USA.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on

The graph data structure models a wide range of processes and relations in real-world applications. Examples include business processes [64], control flow graphs in programs [5], social connections [53, 78], knowledge graphs [35] and semantic scene graphs in image analysis [48]. Visually identifying and searching for persistent subgraph patterns is a common and important task in graph analysis. For example, searching

obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxx/TVCG.201x.xxxxxx for graph motifs such as cliques or stars in a social network reveals the community structures or influencers [17]; searching for similar work-flow templates helps streamline or simplify business processes; searching for images with similar scene graphs helps systematic retrieval of training/testing cases to develop models for computer vision tasks.

In this work, our goal is to support human-in-the-loop, examplebased graph pattern search in a graph database, which could contain hundreds to thousands of individual graphs. Supporting interactive, example-based visual graph pattern query is challenging. Previous graph motif/pattern finding algorithms, e.g. [54, 55, 75] often impose a strict limit on the size of query pattern and do not scale well as the size of the query pattern and the number or the size of the query targets increases. In fact, subgraph matching is a well-known NP-complete problem [70] and there is no known efficient solution so far. Furthermore, the complexity of the subgraph matching problem also makes it difficult to obtain accurate one-to-one node correspondence in the matching results. The node correspondences are crucial to enable visualization-based interpretation and verification of the model's finding. Besides that, it is quite often that domain knowledge is needed to further refine and adjust the results, which cannot be easily supported in algorithms with heavy computational costs.

To address those challenges, we propose a novel framework for interactive visual graph pattern search via graph representation learning. Our approach leverages graph neural networks (GNNs) to encode topological as well as node attribute information in a graph as fixed-length vectors. The GNNs are applied to both the query graph and the query targets to obtain their respective vector representations. The graph matching problem is therefore transformed into a high-dimensional vector comparison problem, which greatly reduces the computational complexity. In particular, we leverage two separate GNNs to address 1) the *decision problem* to determine whether a query pattern exists in a graph and 2) the node-alignment *problem* to find the one-to-one node correspondence between the query pattern and the query targets. We leverage NeuroMatch [44] for the decision problem. For the node-alignment problem, we propose a novel approach called NeuroAlign that can directly generate cross-graph node-to-node attention scores indicating the node correspondences. In most application scenarios we can precompute and store the vector representations of the query targets for efficient retrieval of the graph matching results. The visualization interface enables easy search and specification of the graph query patterns. Since the query engine could return a large number of matched graphs, we present the results with different levels-of-details that show the matched graphs in space-efficient, thumbnail style representations. They can also be sorted via a variety of criteria. Users can also interactively specify additional constraints to further filter the returned results based on their domain knowledge.

We develop the visual analytics system GraphQ based on the proposed framework. GraphQ goes beyond looking for a predefined set of graph motifs and the users can interactively specify and search for meaningful graph patterns in the respective application domain. The query pattern can include both topological structures and domain-specific node attributes to be matched in the query results. The specified query can be partially matched to enable fuzzy-pattern search.

We demonstrate GraphQ's usefulness with two example usage scenarios in different application domains. In the first usage scenario, we apply the system to analyze a large collection of engineering workflow graphs describing the diagnostics programs in automotive repair shops. The goal is to understand whether there are repetitive patterns in the workflow graphs which eventually serves two purposes - curate the workflows to reduce repetitive operations and reuse the patterns as templates for future workflow creation. In the second usage scenario, we apply GraphQ to analyze the semantic scene graphs generated from images, where the nodes are image regions (super-pixels) with semantic labels such as buildings and road, and the links describe the adjacency relations between regions. Searching for subgraph patterns in such semantic scene graphs can help retrieve similar test cases for model diagnostics in computer vision tasks. The example usage scenarios demonstrate that the framework is generalizable and can be applied to graphs of different nature.

Furthermore, we conduct quantitative experiments to evaluate the accuracy and the speed of both NeuroMatch and NeuroAlign. We show that for the node alignment problem, NeuroAlign can produce 19%-29% more accurate results compared to the baseline technique described in NeuroMatch [44]. The improvement greatly helps in validating and interpreting the query results in the visualization. We also compared the speed of the algorithm with a baseline combinatorial approach, the result shows that our algorithm gains up to $100 \times$ speed improvement. The speed improvement is the key that enables a human-in-loop, visual analytics pipeline.

To summarize, our contributions include:

- A visual analytics framework for human-in-the-loop, examplebased graph pattern search via graph representation learning. To the best of our knowledge, this is the first deep learning-based approach for interactive graph pattern query.
- A novel approach (NeuroAlign) for pairwise node-alignment based on graph representation learning which provides 10×– 100× speedup compared to baseline combinatorial algorithm [47] and 19%–29% more accurate results than existing deep learning based approach.
- A prototype implementation of the framework, GraphQ, with interactive query specification, query result display with multiple levels-of-detail, and user feedback mechanisms for query refinement. Two example usage scenarios illustrating the general applicability and effectiveness of the proposed system.

2 RELATED WORK

In this section, we focus on the most relevant research to our work in the areas of graph visualization, visual graph query, and graph representation learning for subgraph pattern matching.

2.1 Graph Visualization

Graph visualization is an extensively studied topic [30, 51] for its application in a wide range of domains. Open source or commercial software for graph visualization (e.g. Gelphi [8] and Neo4j Bloom [3]) are also available for off-the-shelf use. Researchers in graph visualization typically focus on one or more of the following aspects: develop layout algorithms to efficiently compute readable and aesthetic visualizations (e.g. [9, 16, 22, 33, 34, 38]), design new visual encoding to display nodes and edges (e.g. [29, 30, 71]), develop graph simplification or sampling technique to avoid over-plotting and visual clutter (e.g. [17, 72]), and design novel user interaction scheme for exploratory analysis (e.g. [30, 56, 63, 67]). Depending on the nature of the graph data, they have developed a variety of systems and algorithms for directed/undirected graphs, multivariate graphs (with node/edge attributes) and dynamic network visualization to support a wide range of graph analytic tasks [40, 57].

In this work, we focus on supporting interactive, example-based visual query of graph patterns in a database and visualizing the results. This is a generic framework that can be applied to both directed or undirected graph and graphs with node/edge attributes, as demonstrated in the example usage scenarios. We utilize existing graph layout techniques for a detailed view of directed graphs [22] and design a compact visualization for summarizing graph structure to provide an overview of the query results.

2.2 Visual Graph Query

Graph patterns/motifs are frequently used to simplify the display of graphs and reduce visual clutter. Motif Simplification [17] was developed to identify graph motifs including *clique*, *fan*, and *d-connectors* based on topological information and visualized them as glyphs in the node-link display for more efficient usage of the screen space. More generally, cluster patterns, esp. "near-clique" structures are the most studied and visualized in the literature and various methods have been developed to compute and visualize them [75]. However, most of the patterns/ motifs here are predefined and can not be easily modified by users.

Graphite [13], Vogue [10], and Visage [55] support interactive, user-specified queries on graph data and Vigor [54] focuses on visualization of the querying results. In these systems, users can

© 2022 IEEE. This is the author's version of the article that has been published in IEEE Transactions on Visualization and Computer Graphics. The final version of this record is available at: 10.1109/TVCG.2021.3114857



Fig. 2. Visual illustration of the subgraph matching problem. We color-encode the node categorical features of both graphs. The example query graph is subgraph-isomorphic to the target graph with the correct node alignment indicated by dashed lines.

interactively specify node attributes as well as topological constraints in the form of a query graph and the system searches for matching subgraphs. However, the complexity of the query is usually limited, which reduces the expressive power of the specified patterns.

Our approach is also inspired by a number of existing visual query system on time series data, where the user can interactively specify the patterns they are searching for, by either drawing the pattern directly on a canvas or selecting the pattern from a data sample [12, 31, 32, 41, 79]. Supporting user-specified patterns gives the user great flexibility and power to perform exploratory analysis in various application domains. However, querying arbitrary patterns on a graph structure brings unique challenges in terms of the computation speed needed to support an interactive user experience, which we address with a graph representation learning-based approach.

2.3 Graph Representation Learning for Subgraph Pattern Matching

Graph neural networks (GNNs) have emerged as a generic approach for graph representation learning, which can support a variety of graph analytics tasks including link prediction, node classification, and community structure identification [27, 37, 60, 76, 80]. The recent development on GNN library further increases the popularity among researchers [19]. The success of GNN on diverse graph tasks also motivated researchers to address the comparison problem between different graphs, such as graph matching [42] and graph similarity learning [4]. A comprehensive survey on this topic is provided in [45]. Recently, GNNs have been shown to improve the performance on the challenging subgraph-isomorphism problems, including subgraph matching [44], subgraph isomorphism counting [43], maximum common subgraph detection [7], and graph alignment [20]. Powered by flexible representation learning, these approaches addressed issues of heuristic-based solutions [28, 65] in terms of accuracy and query scalability. Our objective is to utilize GNNs to facilitate fast user-interaction with graph queries, where the embeddings of the existing graphs can be precomputed and stored to enable efficient retrieval during the inference stage. Compared to [7, 20], our approach resolves subgraph isomorphism from the learned embedding space alone, without expensive iterative search [7] or embedding refinement aided by the additional network [20]. Our proposed framework utilizes NeuroMatch [44] as a core component to efficiently query matching graphs but involves a novel component NeuroAlign to resolve the issue of NeuroMatch on obtaining accurate node alignment. The capability to identify matching nodes is critical for intuitive user interaction with complex topologies.

There are relatively fewer works in the visual analytics domain utilizing graph representation learning. In [21], a contrastive learning approach is developed to visualize graph uniqueness and explain learned features. Graph representation learning-based algorithms have also been developed for graph layout/drawing [39, 77], evaluating graph visualization aesthetics [26], and sample large graphs for visualization [83]. Our framework addresses the important problem of subgraph matching and facilitates intuitive interaction. To the best of our knowledge, this is the first approach based on representation learning for interactive visual graph queries.

3 ALGORITHM

In this section, we first define the subgraph matching problem and describe our overall framework to resolve it. We then describe NeuroMatch and NeuroAlign, the two GNNs as the core components of the framework. Finally, we introduce an improved inference method and a simple extension to support approximate query matching.

3.1 Problem Definition

We first formally define the subgraph matching problems. We denote G = (V, E) as an undirected, connected graph with vertex set V and edge set E, X as the features associated with V (e.g. categorical attributes). Given a query graph G_Q and a target graph G_T , we consider the *decision problem* which determines whether there exists a subgraph $H_T \subseteq G_T$, such that G_Q is isomorphic to H_T . When H_T exists, i.e. G_Q is subgraph-isomorphic to G_T , we further consider the *node alignment problem* which looks for an injective mapping function $f : V_Q \to V_T$, such that $\{f(v), f(u)\} \in E_T$ if $\{v, u\} \in E_Q$. When the node features X exist, the matching requires equivalence of the feature too. Note that this defines *edge-induced* subgraph isomorphism, which is our focus in the paper. However, the system is general to apply on *node-induced* subgraph isomorphism [6] too.

An illustrative example is shown in Fig. 2, where the colors encode node categorical feature and letters are the node names. The example query graph G_Q is a subgraph of G_T with the correct node alignment of f(a) = A, f(b) = B, f(c) = C, f(d) = D. In this paper, we consider the practical case of a large database of target graphs, where the task is to solve the above decision problem and node-alignment problem for each of the target graphs.

3.2 Overall Framework

Our proposed framework consists of two core components: NeuroMatch (Fig. 3) and NeuroAlign (Fig. 4), which focus on solving the subgraph decision and node alignment problems respectively. Given a graph database and user-created query graph, we utilize the state-of-the-art NeuroMatch method [44] to efficiently retrieve matching target graphs which contain the query graph. NeuroMatch decomposes the graphs into small neighborhoods to make fast decision locally and then aggregates the results. After a matching target graph is found, the node alignment between the two graphs can still be ambiguous and misleading based on what we observe in the experimental results. This is due to the fact that the learning process of NeuroMatch relies entirely on small neighborhoods within the graphs. As a result, each query node could end up matched to multiple target nodes where many of them are actually false positives. To tackle these issues, we propose a novel model NeuroAlign, which directly predicts node alignment from query and target graphs, without segmenting them into small neighborhoods. It computes node-to-node attention based on graph node embeddings to obtain the alignment results. Finally, the matching target graphs and corresponding matching nodes are returned to the user for exploration and analysis.

NeuroMatch and NeuroAlign both employ GraphSAGE [27] as the backbone GNN for representation learning. For simplicity, we consider GraphSAGE as a general function that performs representation learning, where the input is a given graph and the output is a set of embeddings for every node in the graph. Optionally, a pooling layer can be added on top of the node embeddings to obtain a single embedding of the input graph. A more detailed description can be found in the appendix. We use h_v to denote the learned representation of node v at the final output layer, which will be used by NeuroMatch and NeuroAlign as described in the following sections.

3.3 Subgraph Decision via NeuroMatch

Conducting subgraph matching in the embedding space can facilitate efficient retrieval. However, considering the scale of the database and the large size of certain graphs, it is challenging to build the predictive model to encode the subgraph relationships. NeuroMatch resolves this issue by decomposing the given query and target graphs into many small regions and learns the subgraph relationship in these small regions first. In particular, for each node q in the query graph, it extracts a small k-hop neighborhood graph g_q . For each node t in the target graph, it also extracts their k-hop neighborhood g_t . Then the problem of determining whether $G_Q \subseteq G_T$ transforms into many local subgraph matching



Fig. 3. NeuroMatch determines whether G_Q is a subgraph of G_T by looking for local matches first and then aggregate the results. In this figure, we highlight the 1-hop local neighborhoods at anchor nodes b, cin the query graph as an example (in green and orange outlines). The NeuroMatch algorithm compares these 1-hop neighborhoods with those in the target graph. It finds that the 1-hop neighborhood graph of b is a subgraph of the 1-hop neighborhood of B (highlighted in green) and the neighborhood of c is a subgraph of the neighborhood of C (highlighted in orange). Since for each query node (a, b, c, d), we can find a matching 1-hop neighborhood graph in the target graph (A, B, C, D), the algorithm concludes that indeed G_Q is a subgraph of G_T .



Fig. 4. NeuroAlign algorithm obtains accurate node-to-node correspondence. It extracts the embeddings of each node in the query graph and the target graph by directly feeding them through GNN. It then uses an attention network to compare every pair of node embeddings between the query and target graphs. For the convenience of computation, these pair-wise comparison results are formed as a matrix. The rows correspond to query nodes and columns correspond to target nodes. The matrix is then transformed into a probability matrix through softmax on each row. A greedy assignment algorithm resolves potential conflicts (black outlined block) during inference (Section 3.6).

decisions about whether $g_q \subseteq g_t$. To find potential local matches, Neuro-Match compares all pairs of nodes between the query and target graphs. Finally, the ensemble decision can be made by checking whether every query neighborhood can find a matching target neighborhood. Figure 3 shows a simple example to illustrate the main idea of NeuroMatch. In order to determine the local subgraph relationship, i.e. whether the *k*-hop neighborhood graph g_q is a subgraph of g_t , the algorithm feeds g_q and g_t into GNN with the pooling layer to extract the respective anchor node embedding at q and t. A comparator function then takes each pair of these embeddings and predicts the subgraph relationship, as shown in Fig. 3. We describe the method in the appendix and refer readers to the NeuroMatch paper for more detail [44].

When the model is trained, we pre-compute and store embeddings of all graphs in the database. The inference process simply iterates through all pairs of query and target nodes, and utilizes the (trained) comparator to make local subgraph decisions. The aggregated decision is then made by checking whether each query neighborhood finds a match. This process has linear complexity in terms of both query and target number of nodes, thus facilitates efficient retrieval at the front-end interface.

3.4 Node Alignment via NeuroAlign

NeuroMatch determines whether the query is a subgraph of the target graph. When a matching target graph is retrieved and visualized, it is still difficult for the user to extract insights when the target graph is large and the topology is complex. In this case, showing the corresponding nodes can provide intuitive and explainable visual cues. We propose NeuroAlign, to obtain improved node alignment performance. We formulate the prediction problem as a classification task, where query nodes are examples and the target nodes correspond to labels. This architectural change is crucial to enable more accurate alignment by accounting for much larger areas on both graphs. However, for different target graphs, the number of classes (i.e. target nodes) varies. This creates a challenge for predictive models. We resolve it by employing a flexible, cross-graph attention mechanism.

As shown in Fig. 4, NeuroAlign directly takes the node embeddings obtained from GNN on the entire graphs G_Q and G_T . These embeddings are denoted as $\{h_q, \forall q \in G_Q\}$ and $\{h_t, \forall t \in G_T\}$. We then compute the similarity between each query embedding and every target embeddings through an attention network. This process can be considered as creating an attention matrix $\mathbf{A} \in \mathbb{R}^{\|V_Q\| \times \|V_T\|}$, where the element $\mathbf{A}_{q,t}$ contains the attention from node q to t. We then directly transform the similarity matrix to a probability matrix $\mathbf{P} \in \mathbb{R}^{\|V_Q\| \times \|V_T\|}$ using row-wise softmax and use them in the cross-entropy loss. Formally,

$$\mathbf{A}_{q,t} = \boldsymbol{\psi}(h_q \parallel h_t)$$

$$\mathbf{p}_q = \operatorname{softmax}(\mathbf{a}_q)$$

$$L(G_Q, G_T) = -\sum_{q \in G_Q} \mathbf{y}_q \log(\mathbf{p}_q)$$
(1)

where Ψ denotes the attention network, \mathbf{a}_q is the *q*-th row of \mathbf{A} , and \mathbf{y}_q is the one-hot ground-truth label for node *q*, indicating which node in G_T is the corresponding node of *q*. The prediction \mathbf{p}_q contains the probabilities of matching query node *q* to every target node. We implement the attention network as a multi-layer perceptron, which takes a pair of embeddings produced by the GNN, concatenate them and return a similarity score between a node *q* in the query graph and a node *t* in the target graph. In case G_T is too large, the computation of $\mathbf{A}_{q,t}$ could consume too much memory, and needs to be constrained to a subgraph at *t*. In practice, we specify a maximum size that covers most target graphs in the database.

Similar to NeuroMatch, when the model is trained, we can pre-compute all graph embeddings generated by NeuroAlign to make the retrieval process efficient. In addition, NeuroAlign works subsequently to NeuroMatch and only activates when a subgraph relationship is predicted, thus creating minimal computational overhead for visualization and interaction.

3.5 Algorithm Training

The training of NeuroMatch and NeuroAlign are conducted separately. Training NeuroMatch (and its backbone GraphSAGE GNN) involves sampling large amounts of mini-batches containing both positive and negative pairs. A positive pair consists of two neighborhood graphs g_q and g_t that satisfy the subgraph relationship, while a negative pair consists of neighborhood graphs where the relationship is violated. To sample a positive pair, we first randomly sample a k-hop neighborhood as g_t , and then sample a subgraph within g_t as the query neighborhood g_q . To sample negative pairs, we start with the obtained target neighborhood g_t above, and sample a smaller neighborhood from a different graph as g_q (query neighborhood). Note that g_q needs to be verified with exact matching protocol [14] to ensure $g_q \nsubseteq g_t$. In practice, we find that hard negatives are necessary to achieve high precision, which are obtained by perturbing the above positive pair ($g_q \subseteq g_t$) such that the subgraph relationship no longer exists. We perturb the positive pair by randomly adding edges to g_q and verify the success with exact matching [14]. As can be seen, negative sampling extensively invokes exact matching algorithm, which is slow to compute. To keep the training tractable, we set small neighborhood hop k = 3 and also limit the number of nodes to sample from the neighborhood to 30.

© 2022 IEEE. This is the author's version of the article that has been published in IEEE Transactions on Visualization and Computer Graphics. The final version of this record is available at: 10.1109/TVCG.2021.3114857

Training NeuroAlign (and its backbone GraphSAGE GNN) is much simpler. It involves sampling only positive pairs, since its objective is to improve node alignment when the subgraph decision has already been made that $G_Q \subseteq G_T$. Therefore, the sampling involves extracting random queries from the graphs in the database. For each target graph G_T in the database, we randomly sample a subgraph within it as G_Q . The ground-truth injection mapping is acquired directly in the sampling process, and it is converted to \mathbf{y}_q to indicate which node in G_T is the corresponding node of q. NeuroAlign can be trained efficiently through this simple sampling process and without invoking the expensive exact matching algorithm.

3.6 Greedy Assignment for Inference

During inference of node alignment, different nodes in the query graph could be mapped to the same node on the target graph. This is likely to occur among nodes with highly similar topological and attribute features. The prediction conflict can be resolved with a task assignment algorithm. Instead of resorting to the combinatorial Hungarian algorithm [47], we further develop a simple greedy assignment approach. Specifically, given the predicted probability matrix **P**, we iterate the probabilities in descending order and record the corresponding matching pair only when both the query and target nodes have not been assigned. The iteration stops when all query nodes have been assigned. This simple process resolves conflicting assignment to the same target node and improves the overall node alignment performance (experimental results in Section 5.3.1).

3.7 Approximate Query Matching

In addition to the retrieval results obtained from the query graph, we provide the option to perform approximate query matching. This method perturbs the query graph slightly, in order to obtain similar, but different matching graphs. Specifically, denote the set of obtained matches from the original query graph G_Q as R. We remove one node from G_Q and its associated edges to obtain the perturbed query G'_Q . Then we conduct the search with NeuroMatch on G'_Q and add the novel matches R. We continue the iteration by removing a node from the perturbed query, until either a prespecified maximum number of steps is reached or G'_Q becomes disconnected. To lower the chance of getting a disconnected graph, each time we remove the node with the lowest degree in G'_Q .

4 VISUALIZATION AND INTERACTION

In this section, we first evaluate the design goals of GraphQ (Section 4.1). We then describe the GraphQ system with details on its visualization and interaction components (Section 4.2.1), and technical implementation (Section 4.2.2).

4.1 Design Goals

GraphQ's principle design goal is to provide a generic solution for interactive graph pattern search on a graph database based on user-specified examples. The basic requirement is that the user needs to be able to interactively select and refine graph patterns and analyze the retrieved results. In the meanwhile, the system should display the matching instances as well as explaining the results by highlighting the node correspondences.

We further enrich and refine the design goals by collecting requirements for domain-specific usage scenarios. We analyzed two example usage scenarios including workflow graph pattern analysis and semantic scene graph analysis in image understanding. For the first usage scenario (details in Section 5.1) we worked closely with the domain experts who provided the workflow graph data and who are also the end-user of the system. In the second usage scenario, we reference the relevant literature in computer vision on semantic scene graphs. Semantic scene graph is a commonly used graph structure that describes not only the objects in an image but also their relations [36]. They are frequently used to retrieve images with the same semantics. By analyzing the commonalities of the two usage scenarios we identified the following user analysis tasks to support in GraphQ:

- **T1** Browse/search the graph database. To start the query process, the user needs to be able to select from hundreds to thousands of graphs. Therefore, the system should provide graph search and filtering functionalities based on the category, the name, or graph statistics such as the number of nodes/links. Besides that, a visualization showing an overview of all graphs in the database will be useful to help locate interesting graphs or clusters.
- **T2** Interactively construct the query pattern by selecting on a graph visualization. To minimize user effort, the system should support both bulk selection mechanisms such as brushing the graph regions as well as query refinement methods to add/delete individual nodes/edges from the pattern.
- **T3** Interpret and validate the matched graphs via highlighted similarities and differences. To help users interpret the matching results, the node correspondences, as well as differences in the query results, should be highlighted. Furthermore, since the subgraph matching and node correspondence calculation algorithms are not 100% accurate, the results need to be presented in a meaningful way for easy verification.
- **T4** Explore the distribution of the matching instances. After the matched graphs are returned, the system should indicate how frequently the query pattern occurs in the entire database, and provide the distribution of the pattern among different categories of graphs in the database.
- **T5 Refine query results.** A flexible query system should further support query refinement mechanism where the users can apply their domain knowledge to filter the results with additional constraints, such as matching additional node attributes or limiting the results to a certain category of graphs.

4.2 GraphQ System

We design GraphQ to support the user analysis tasks (**T1-5**) described in Section 4.1 with the architecture and user workflow featured in Fig. 5. The user can start with an overview of the graph database (**T1**), brush, and select a graph to create example-based query patterns (**T2**). The query pattern (along with optionally perturbed query pattern for approximate query matching) will be sent to the back-end, its node representations will be computed and compared with the precomputed node embeddings to obtain a set of matching graphs containing the query pattern. The matching results along with the query pattern will go through NeuroAlign to compute one-to-one node correspondence. The query results will be displayed in the front-end with multiple levels-of-detail (**T3**) and can be refined further by adding node-attribute constraints interactively in the query panel (**T5**). The distribution of the matching graphs will be highlighted interactively in the database overview panel (**T4**).

4.2.1 Components

The user interface of GraphQ is composed of four main components:

Overview and filters. In the overview panel (Fig. 1(3)) the system displays the distribution of key graph statistics such as the number of the nodes/edges as well as domain-specific attributes such as the category of the graph. Both univariate distributions and bivariate distributions can be displayed as histograms or scatterplots. Users can brush the charts and select a subset of graphs to create example-based query patterns. To provide an overview of the graph structural information and help users navigate and select a graph to start the query (**T1**), we further precompute the graph editing distance [23] which roughly captures the structural similarities between all pairs of graphs. A 2-D projection coordinates of the graph can then be precomputed using t-SNE [73] based on the distance matrix and stored as additional graph attributes (Fig. 1(a)).

After the query result is obtained, the charts will be updated to provide a contextual view of how the subgraph pattern occurs in the database. For example, the user can observe whether the pattern occurrence concentrate on a small subset of graph categories or it is a generic pattern that appears in many different categories (**T4**) (Fig. 1(d)).

Furthermore, the overview panel is a customizable module that can be configured through a json file specifying the attributes to be displayed and the chart to display it. Users can also interactively fold each



Fig. 5. System architecture of GraphQ. The back-end precomputes and stores the graph representations to support efficient matching graph retrieval through the NeuroMatch algorithm. After the matching graphs are obtained, we use NeuroAlign to obtain accurate node-to-node correspondence to be displayed in the visualization for the user to verify the results. Users can start from an overview of all the graphs in the database and select one to construct example-based query pattern. The query pattern can be slightly perturbed to retrieve approximate matching results from the database. After the results are returned, the user can use a variety of views to explore the returned results.

chart and hide it in the display, such that space can be used for keeping important attribute information on the screen. The system also displays a popup window to show detailed information for selected charts.

Graph query panel. In the graph query panel (Fig. 1(1)), the user can interactively select from a graph instance to construct the query pattern. The color of the nodes encodes the key node attribute to be matched in the subgraph pattern query. The system currently supports categorical node attributes. This can be extended to numerical attributes by quantizing the values. Additional node attributes are displayed in attachment to the nodes or in tooltips. As discussed in Sect. 4.1, we need to support fast, interactive query construction (T2). In this panel, the user can quickly select a group of nodes and the subgraph they induce by brushing a rectangular area on the visualization. They can also construct the pattern in a more precise manner by clicking the + and - button on the top right corner of each node. A minimap on the bottom right of the panel allows the user to easily navigate and explore graphs of larger size. The layout of the graph is computed with existing layout algorithms. such as the algorithm described in [22] for directed graphs. When the nodes have inherent spatial locations, they are used directly for display.

Query results. After the sub-graph pattern matching results are returned, the query results panel will be updated to display all the matching graphs as a small multiples display (Fig. 1(2.1) and (2.2)). Since the number of returned results could be large, the system supports sorting the returned graphs with graph attribute values such as the number of nodes (Fig. 1(f)). To support **T3**, the matching nodes are highlighted based on the results returned by the node alignment module. The graphs can be displayed either in a node-link diagram with the same layout as the graph in the query panel (Fig. 1(2.2)) or in a thumbnail visualization designed to display the graph in a more compact manner (Fig. 1(2.1)). In particular, we use topological sorting of the nodes for directed acyclic graphs to order the nodes, layout them vertically, and route the links on the right to obtain a compact view (Fig. 1(2.1)).

Comparison view. To support **T3** and **T5**, we further visualize the query and selected matching graphs side-by-side in a popup window. The user can click on the zoom-in button on each small multiple to bring out the comparison view (Fig. 1(5)) and review each matching graph in detail. The matched nodes are highlighted for verification.

4.2.2 Implementation

GraphQ's implementation uses a typical client-server architecture. The frontend UI framework is implemented in Javascript with React [18] and AntD UI [15] libraries. The visualizations are drawn using D3.js [11] on svg within the React framework. We use dagre [1] to compute directed graph layout in the front-end. The backend server is implemented in Python with Flask [24]. The graph data are stored as json documents in the file system and modeled with NetworkX [25]. We use Py-Torch [52] for graph representation learning for both subgraph matching and node correspondence learning. More specifically, we use PyTorch Geometric [19] and DeepSNAP [2] to batch graph data (including their topological structures and node features) for training and inference.

5 EVALUATION

Our evaluation of the proposed system consists of two example usage scenarios (Section 5.1 and 5.2), quantitative experiments on various datasets (Section 5.3), and interview with domain experts on both usage scenarios (Section 5.4).

5.1 Example Usage Scenario: Program Workflow Analysis

In the first usage scenario, we apply GraphQ to analyze a collection of graphs describing the workflows in a vehicle diagnostics software program. The software program uses prescripted workflow graphs to check the functionalities of the system and locate the problem in the vehicles. The workflows are modeled as directed graphs where each node represents an individual procedure in the workflow and the link represents their sequential orders. We convert the graphs to undirected graphs as input for the query algorithms. In total, there are \sim 20 different types of procedures in the workflow, and we use node colors in the system to distinguish them (Fig. 1) (all the names of the nodes are anonymized). In both NeuroMatch and NeuroAlign, the type of the procedures is considered as a node attribute.

The workflows are manually created and it is a time-consuming process. The goal of analyzing workflow graphs is to identify subroutines in the workflow that are reused frequently and therefore can be used as templates, or submodules in the future to facilitate the workflow editing process or to simplify the workflow descriptions. However, identifying such frequent subroutines cannot be easily automated – substantial domain knowledge in automotive hardware and software system is needed to curate meaningful patterns, therefore a human-in-the-loop approach is well-suited.

Through an initial data exploration together with the domain experts, we found that pairwise comparison of workflows using graph editing distance [23] can provide an overview of the graph similarities in the dataset. This overview can help the user to select interesting workflows as the starting point for exploration. Our system integrates a t-SNE projection [73] of all the graphs based on the graph editing distance matrix which reveals several clusters (Fig. 1(a)). The user can use the brushing function to select one cluster and the selected graphs will be updated in the table (Fig. 1(b)). The user could then select any graph from the table to be displayed in the query editor (Fig. 1(1)) to create example-based queries. In Fig. 1(c), a subroutine with a branching structure is selected by brushing on the visualization. The user can invoke the context menu and search for the query pattern in the graph database. With approximate matching disabled (Fig. 1(4)), the system returns 45 matched graphs in the database. In the graph types histogram, we can see that most of the matched graphs belong to two types (Fig. 1(d)). For an overview of the matching results (Fig. 1(2.1)), the user could toggle minimize in the query results display (Fig. 1(f)) and highlight the node matches returned by NeuroAlign (Fig. 1(e)). The result shows that indeed most of the graphs returned contain the nodes in the query pattern, indicating that the algorithm is returning reliable results. To





Fig. 6. The user selects a fan-like pattern (a). Exact subgraph matching returns 21 results (b). After enabling approximate search (Fig. 1(4)), the back-end returns 172 graphs (d) containing fan-like patterns, although some of them are simpler than the query. The query results indicate that such structure can be reused as a template to reduce the manual effort for future workflow creation.



Fig. 7. To obtain a semantic scene graph from an image in the MSRC-21 dataset, we use the Quickshift [74] algorithm which segments the image into partitions, i.e. super-pixels; then we derive each semantic label as the most frequent ground-truth label of all pixels inside the corresponding super-pixel. Each super-pixel is mapped to a graph node with the semantic attribute.

further view the details, the user turns off the minimize toggle, and the graphs are displayed in a similar layout as in the query panel and the user can review more details about each graph including the graph name, number of nodes, and links, etc (Fig. 1(2.2)). To facilitate the inspection of more detail about the returned matches and aligned nodes, we design the side-by-side display of the query graph and returned matching graph (Fig. 1(5)). The display is activated as a popup window when the user clicks on the zoom button (Fig. 1(g)). Users can also add additional node attribute constraints by clicking on the corresponding node attribute (Fig. 1(h)) to be matched in the query results. In this example there is no workflow satisfying the specified attribute constraint. After verifying the results the user can save the query pattern in a json file to be reused when manually creating workflows in the future.

Fig. 6 shows the query results for a fan-like structure selected from a graph (Fig. 6(a)). The system returns 21 matched results with approximate search disabled. Indeed most of the returned graphs contain the fan-like structure (Fig. 6(b)), indicating another reusable submodule in the workflow creation process. In the t-SNE plot, the graphs with matching fan-like patterns are highlighted in orange, showing the graphs are scattered in different clusters according to graph editing distance (Fig. 6(c)). This finding indicates our method can uncover meaningful patterns in the sub-regions of the graphs that are missed by graph-level similarities. To further extend the search to graphs that may contain similar, but not exact the same patterns, the user toggles the button to enable approximate search (Fig. 1(4)), the returned result contains much more graphs (172 graphs) than in exact matching (Fig. 6(d)). The user sorts the results based on the number of nodes and found that the graphs with approximate matches contain a simpler fan-like structure with fewer nodes. Based on the analysis the user concludes

that the fan-like pattern can be used as a template in the future.

5.2 Example Usage Scenario: Scene Graph Search

In the second usage scenario, we apply GraphQ to semantic scene graph search in computer vision applications to find images with similar objects and relationships that resemble our query subgraph structure. It can be useful for many computer vision tasks such as image retrieval [59, 82], visual question answering, relationship modeling, and image generation. We follow the procedures described in [49] to extract a semantic scene graph from each image. Each node in the graph represents a super-pixel extracted from the image using a segmentation algorithm and the links between nodes encode the adjacency information between those super-pixels. Each node is annotated with a semantic label, as one of its attributes and the whole extracted graph from an image is an undirected, planar graph [69]. In this study, we use a public image segmentation dataset (MSRC-21 [62]) to illustrate this approach. Each image contains ground-truth labels such as tree, grass, wall and unlabeled void, etc. We illustrate the process to extract the scene graph from each image in Fig. 7.

To perform scene graph search, the user starts with the overview of all graphs in the database. The user picks a graph to work on and brushes a subgraph, for example, three connected nodes (Fig. 8(a)) including sky, building and road. This subgraph structure could indicate a typical city environment (with buildings and road). The backend, with approximate search disabled, returns matched result of 25 graphs and most of them contain the same subgraph: street view: interconnected super-pixels of sky, building and road as shown in (Fig. 8(b)). Note in histogram overview (Fig. 8(c)), all of these resulted images come from the same row (17th) in MSRC-21 dataset that belongs to the category "road/building". The user can also sort by different metrics and filter by different node information such as area range, or even super-pixel location, etc. Through these interactions, the user eventually finds interesting images tailored to needs.

5.3 Quantitative Evaluation

We evaluate the performance of the proposed system on 4 graph datasets in various domains: program workflow dataset (vehicle diagnostic), MSRC-21 (image processing), COX2 (chemistry) and Enzymes (biology). The workflow dataset contains \sim 500 individual workflow graphs with the number of nodes ranging from 5 to 150.



Fig. 8. Case study 2, searching by brushing a subregion (a chain of sky, building, and road nodes) on the (MSRC-21) scene graph and find the matching results (b), most of which contain the same chain of such three nodes as in (a). The three nodes' relationship resembles a typical street view image.

Table 1. Subgraph decision performance using NeuroMatch.

Dataset	Precision	Recall	F1	
Workflow	87.0	89.9	88.4	
MSRC-21	83.6	91.6	87.4	
COX2	87.4	90.9	89.1	
Enzymes	81.8	73.0	77.1	

 \sim 20 different types of nodes correspond to different diagnostic procedures. MSRC-21 [62] contains natural scene images with 21 object semantic labels. After the super-pixel extraction and processing steps as described in Section 5.2 and Fig. 7, the resulting graph dataset includes 544 graphs with 11 to 31 nodes. COX2 [46, 66] consists of 467 chemical molecule graphs with the number of nodes ranging from 32 to 56. Enzymes dataset [46, 58] contains 600 graphs of protein tertiary structure with 3 to 96 nodes. The last 3 datasets are public.

We utilize an 8-layer GraphSAGE in training and the hidden dimension for node embeddings is 64. For NeuroAlign, the attention network has two hidden layers of dimensions 256 and 64. We use ReLU activation. The learning rate is fixed at 0.0001 without weight decay and Adam optimizer is utilized.

The training data is generated on the fly by randomly sampling the positive and negative pairs, as described in Sect. 3.5. Note that the ground-truth label for a positive pair is obtained automatically during sampling, and for a negative pair is calculated by exact matching algorithm [14]. The batch size is fixed to 128. For validation data, we sample the dataset following the same process, prior to training. For testing data, we sample based on the evaluation tasks as described in the following sections.

All experiments are conducted on a single GeForce GTX 1080 Ti GPU. We measure the performance of the system in terms of prediction correctness and runtime efficiency. For all evaluations, the approximate query matching is turned off. The detailed description of the evaluation setup and experimental results are presented below.

5.3.1 Prediction Accuracy

To construct the testing dataset for evaluation of the prediction accuracy, we randomly extract 5 queries from each graph, and obtain their ground-truth subgraph-isomorphism labels. The evaluation is conducted on the problem of subgraph decision and node alignment separately. For subgraph decision, we measure the precision and recall, commonly used in the information retrieval domain, to measure how well NeuroMatch retrieves the ground-truth matching target graphs from the graph database.

For node alignment, the objective is to measure how well the algorithm predicts the correct matching nodes on the retrieved target graphs. Since the wrong retrieval does not have ground-truth node alignment, we conduct the evaluation on the set of correctly retrieved target graphs. For this task, we compare our proposed NeuroAlign with NeuroMatch, which provides node correspondence through the matched anchor nodes. Greedy assignment (Section 3.6) is applied on both NeuroMatch and NeuroAlign to improve the inference. The details on utilizing the greedy assignment on NeuroMatch can be found in the appendix. To measure the performance, we calculate the top-k ($k \in \{1, 2, 3\}$) accuracy along with the accuracy after the greedy assignment on each query, and report the average among all queries. In case multiple matches exist in the ground truth, we only consider the one closest to algorithm prediction to measure the accuracy. The identification of multiple subgraph isomorphisms [43] is a more challenging research topic and we provide a discussion in Section 6.

The performance of subgraph decision is shown in Table 1. The results show that the system is able to retrieve around 90% matching target graphs for both datasets while maintaining high precision. Note that achieving high precision is much more challenging than high recall since a matching target graph is rare as compared to non-matching graphs. The excellent precision and F1 score of the system demonstrate the model's capability to learn embeddings that correctly reflect the subgraph relationship.

The comparison between NeuroMatch and our proposed algorithm NeuroAlign on the node alignment task is shown in Table 2. Neuro-Match performed poorly on this task due to multiple predicted matches for many query nodes. We achieve significant improvement over Neuro-Match (e.g. 27.3% improvement on top-1 acc. and 22.2% improvement after assignment for Workflow, 18.7% improvement on top-1 acc. and 28.7% improvement after assignment for MSRC-21). We also observe that MSRC-21 is much more challenging than Workflow dataset due to the dense connectivity and a large number of similar adjacency nodes. Interestingly, although NeuroAlign makes many wrong decisions from the top-1 predictions, its top-3 predictions contain most labels. As a result, the simple assignment approach successfully resolves many predicted conflicts and significantly improves the accuracy. Contrarily the assignment does not make much improvement for NeuroMatch predictions. In addition, we experimented with the optimal Hungarian assignment algorithm and observe that, as compared to our greedy approach, the improvement is negligible for NeuroAlign, but higher for NeuroMatch (e.g. achieves 73.1% acc. on Workflow and 55.4% acc. on MSRC-21) due to more conflicting predictions.

5.3.2 Runtime Efficiency

Next, we measure the runtime efficiency in comparison with the VF2 baseline [14] to evaluate the speed gain. VF2 is the state-of-the-art exact matching algorithm based on backtracking procedure. Although it calculates true subgraph-isomorphism results, the computation is expensive, especially for larger graphs. In addition, we also compare with a similar system where NeuroAlign component is removed to evaluate the added computational overhead of NeuroAlign. For this evaluation,

© 2022 IEEE. This is the author's version of the article that has been published in IEEE Transactions on Visualization and Computer Graphics. The final version of this record is available at: 10.1109/TVCG.2021.3114857 Table 2. Node alignment performance. NeuroAlign achieves averaged 25% improvement on the final accuracy.

Method	Dataset	top-1 acc.	top-2 acc.	top-3 acc.	acc. w/ assignment	Dataset	top-1 acc.	top-2 acc.	top-3 acc.	acc. w/ assignment
NeuroMatch NeuroAlign (Ours)	Workflow	64.2 91.5	85.6 97.7	93.4 98.7	68.6 95.2	COX2	42.2 65.3	56.5 81.6	65.9 92.0	44.1 70.4
NeuroMatch NeuroAlign (Ours)	MSRC-21	40.9 59.6	62.7 84.2	77.0 95.1	52.6 81.3	Enzymes	41.7 53.6	56.6 75.3	67.4 86.3	47.5 66.7



Fig. 9. Runtime comparison with VF2 [14] and NeuroMatch [44] on the Workflow dataset. Runtime in seconds is shown on the *y*-axis as logarithm scale and the exact number is above the bar. Compared to VF2, our system provides $10 \times -100 \times$ speedup starting from 10 query nodes and therefore enables interactive query. Our proposed NeuroAlign component adds little to none computational overhead as compared to NeuroMatch, while providing much more accurate node-alignment results.

we consider the number of query nodes ranging from 5 to 30 with an increment of 5 on the Workflow dataset, and randomly extract 2000 corresponding queries for each number. We measure the averaged runtime in seconds for the matching with the entire database. The results are visualized in Fig. 9. We observe that the runtime of VF2 increases exponentially with the increase in query nodes and reaches close to 6 minutes with just 25 query nodes. With further increased query nodes they become larger than many target graphs and cannot be matched, thus creating a runtime drop at node size 30. In contrast, our runtime increases linearly with query node size. Compared to NeuroMatch, the added NeuroAlign component induces little to none computational overhead. Surprisingly it is slightly faster than NeuroMatch in some cases. We conjecture this is due to the easier assignment task generated by NeuroAlign (i.e. fewer conflicts), such that the greedy algorithm can terminate early.

5.4 Expert Interview

To evaluate the usability of the system, we conducted a semi-structured interview involving three industrial experts working on program work-flow construction and review for the first usage scenario, as well as three researchers working in the computer vision domain for the second usage scenario. We introduced the system with a walk-through of the interactive features and visual encodings and then explored the system together through a remote call. We report a brief summary of the findings here as an initial validation of the usability and utility of the system.

For the first usage scenario, the domain experts considered the visual analytic system easy to understand and fits their current usage scenario very well: identifying reusable workflow modules to simplify future workflow creation. They can easily create new patterns and search for matching graphs in the database and validate the results in the visualization interface. They even proposed new usages such as using the visualization to review newly created workflows. One of them commented, "The abstraction and searching of custom queries open up a lot of opportunities". In addition, they requested that the returned workflows to be grouped by additional node features for fine-grained analysis. We are currently working with the experts to deploy the system for larger-scale use, and are expecting more feedback after long-term usage.

For the second usage scenario, the domain experts appreciated the usefulness of the system by commenting, "It's great to perform query so fast and see results interactively. It's certainly very powerful for many computer vision problems". They showed great interest in applying the system for diagnosing computer vision models to answer questions such as: does an object detection model performs worse when the object is placed on the road instead of in a room? One of them is interested in retrieving images containing similar semantic structure as some failure cases of the model to perform further analysis and model refinement. Another expert is interested in utilizing the tool for computer vision problems with a heavy focus on object relationships, such as image captioning and visual question answering. For improvement, they mentioned that the graph edge could encode additional information such as the relative positions (up, down, left, right) of the superpixels to retrieve similar images. In addition, a ranking of the matched images could be provided based on the closeness of visual appearance to the query image.

6 DISCUSSION, LIMITATIONS AND FUTURE WORK

We introduced a novel system GraphQ to perform interactive visual pattern queries on graph databases based on user-created query patterns. To facilitate interactive query, we utilize graph representation learning to resolve the problem of subgraph decision and node alignment. The intuitive and explainable visual cues provided by NeuroAlign are paired with novel visual and interaction designs to help users navigate the retrieval results and extract insights. Due to the complexity of the subgraph matching problem, there are still many open questions we have not addressed yet:

Node alignment for multiple subgraph isomorphism. Currently, the training and inference of NeuroAlign focus on a single instance of subgraph isomorphism. However, in practice, the query nodes could be mapped to multiple sets of nodes in the same matching target graph. Counting and enumerating all these instances is a very challenging problem and requires future research. Besides that, multiple pattern matches in a large graph bring additional challenges for interaction and scalable visual representations.

Scalability to very large query graphs. During training of Neuro-Match, we observe that hard negative samples are crucial to achieving high precision rate. However, sampled or perturbed queries need to be verified with exact matching algorithms to ensure the subgraph relationship does not exist. These algorithms are slow to compute especially when the query and target neighborhood graphs become larger and the connectivity becomes denser. A potential approach to alleviate the issue is to assign large weights to these hard negatives and reduce the overall need to invoke these algorithms during training.

Handling directed or disconnected query patterns. Currently, our algorithm works with using undirected, connected graphs as the query pattern. For directed graphs, we converted them into undirected graphs as input for NeuroMatch and NeuroAlign. To account for the direction of connectivity, the backbone GNN model needs to be modified. For example, GraphSAGE can be modified by distinguishing the in-node and out-node neighborhoods during the aggregate-update process and other GNNs specifically designed for directed graphs such as [61, 68] can be considered. On the other hand, for disconnected query patterns, a potential workaround is to consider each connected component separately and make an ensemble of the individual predictions. However, the performance still needs to be investigated.

In the future, besides addressing the aforementioned limitations, we plan to investigate database index applied on the embeddings of the large graph database to allow even more efficient retrieval at sub-linear time. Furthermore, considering the wide variety of graph-structured data, we plan to extend the current work to more usage scenarios including social network analysis [81] and 3-D point clouds [50].

REFERENCES

- dagre graph layout for javascript. https://github.com/dagrejs/ dagre.
- [2] Deepsnap. https://github.com/snap-stanford/deepsnap.
- [3] Neo4j bloom: Friendly graph visualization, exploration and collaboration tool. https://neo4j.com/product/bloom/.
- [4] R. Al-Rfou, B. Perozzi, and D. Zelle. Ddgk: Learning graph representations for deep divergence graph kernels. In *The World Wide Web Conference*, pp. 37–48, 2019.
- [5] F. E. Allen. Control flow analysis. In *Proceedings of a Symposium on Compiler Optimization*, p. 1–19. Association for Computing Machinery, New York, NY, USA, 1970. doi: 10.1145/800028.808479
- [6] S. Bachl. Isomorphic subgraphs. In International Symposium on Graph Drawing, pp. 286–296. Springer, 1999.
- [7] Y. Bai, D. Xu, K. Gu, X. Wu, A. Marinovic, C. Ro, Y. Sun, and W. Wang. Neural maximum common subgraph detection with guided subgraph extraction. 2019.
- [8] M. Bastian, S. Heymann, and M. Jacomy. Gephi: an open source software for exploring and manipulating networks. In *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 3, 2009.
- [9] C. Bennett, J. Ryall, L. Spalteholz, and A. Gooch. The aesthetics of graph visualization. In *CAe*, pp. 57–64, 2007.
- [10] S. S. Bhowmick, B. Choi, and S. Zhou. Vogue: Towards a visual interaction-aware graph query processing framework. In *Cidr*. Citeseer, 2013.
- [11] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Dec. 2011. doi: 10.1109/TVCG.2011.185
- [12] P. Buono, A. Aris, C. Plaisant, A. Khella, and B. Shneiderman. Interactive pattern search in time series. In *Visualization and Data Analysis 2005*, vol. 5669, pp. 175–186. International Society for Optics and Photonics, 2005.
- [13] D. H. Chau, C. Faloutsos, H. Tong, J. I. Hong, B. Gallagher, and T. Eliassi-Rad. Graphite: A visual query system for large graphs. In 2008 IEEE International Conference on Data Mining Workshops, pp. 963–966. IEEE, 2008.
- [14] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.
- [15] A. Design. Ant design an enterprise-class ui design language and react ui library. https://github.com/ant-design/ant-design.
- [16] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. ACM Computing Surveys (CSUR), 34(3):313–356, 2002.
- [17] C. Dunne and B. Shneiderman. Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 3247–3256, 2013.
- [18] Facebook. React a javascript library for building user interfaces. https: //github.com/facebook/react.
- [19] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs* and Manifolds, 2019.
- [20] M. Fey, J. E. Lenssen, C. Morris, J. Masci, and N. M. Kriege. Deep graph matching consensus. arXiv preprint arXiv:2001.09621, 2020.
- [21] T. Fujiwara, J. Zhao, F. Chen, and K.-L. Ma. A visual analytics framework for contrastive network analysis. In 2020 IEEE Conference on Visual Analytics Science and Technology (VAST), pp. 48–59. IEEE, 2020.
- [22] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.
- [23] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. Pattern Analysis and applications, 13(1):113–129, 2010.
- [24] M. Grinberg. Flask web development: developing web applications with python. "O'Reilly Media, Inc.", 2018.
- [25] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
- [26] H. Haleem, Y. Wang, A. Puri, S. Wadhwa, and H. Qu. Evaluating the readability of force directed graph layouts: A deep learning approach. *IEEE Computer Graphics and Applications*, 39(4):40–53, 2019.
- [27] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. arXiv preprint arXiv:1706.02216, 2017.
- [28] M. Heimann, H. Shen, T. Safavi, and D. Koutra. Regal: Representa-

tion learning-based graph alignment. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pp. 117–126, 2018.

- [29] N. Henry, J.-D. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1302–1309, 2007.
- [30] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.
- [31] H. Hochheiser and B. Shneiderman. Interactive exploration of time series data. In *The Craft of Information Visualization*, pp. 313–315. Elsevier, 2003.
- [32] H. Hochheiser and B. Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.
- [33] Y. Hu. Efficient, high-quality force-directed graph drawing. *Mathematica journal*, 10(1):37–71, 2005.
- [34] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PloS one*, 9(6):e98679, 2014.
- [35] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu. A survey on knowledge graphs: Representation, acquisition and applications. *arXiv preprint* arXiv:2002.00388, 2020.
- [36] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. Shamma, M. Bernstein, and L. Fei-Fei. Image retrieval using scene graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3668–3678, 2015.
- [37] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.
- [38] O.-H. Kwon, T. Crnovrsanin, and K.-L. Ma. What would a graph look like in this layout? a machine learning approach to large graph visualization. *IEEE transactions on visualization and computer graphics*, 24(1):478–488, 2017.
- [39] O.-H. Kwon and K.-L. Ma. A deep generative model for graph layout. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):665– 675, 2019.
- [40] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI workshop* on *BEyond time and errors: novel evaluation methods for information* visualization, pp. 1–5, 2006.
- [41] F. Lekschas, B. Peterson, D. Haehn, E. Ma, N. Gehlenborg, and H. Pfister. Peax: Interactive visual pattern search in sequential data using unsupervised deep representation learning. *Computer Graphics Forum*, 39(3):167– 179, 2020. doi: 10.1111/cgf.13971
- [42] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International Conference on Machine Learning*, pp. 3835–3845. PMLR, 2019.
- [43] X. Liu, H. Pan, M. He, Y. Song, X. Jiang, and L. Shang. Neural subgraph isomorphism counting. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1959–1969, 2020.
- [44] Z. Lou, J. You, C. Wen, A. Canedo, J. Leskovec, et al. Neural subgraph matching. arXiv preprint arXiv:2007.03092, 2020.
- [45] G. Ma, N. K. Ahmed, T. L. Willke, and P. S. Yu. Deep graph similarity learning: A survey. arXiv preprint arXiv:1912.11615, 2019.
- [46] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.
- [47] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.
- [48] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2):209–245, 2016.
- [49] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. Propagation kernels: efficient graph kernels from propagated information. *Mach. Learn.*, 102(2):209–245, 2016.
- [50] M. Neumann, P. Moreno, L. Antanas, R. Garnett, and K. Kersting. Graph kernels for object category prediction in task-dependent robot grasping. In Online Proceedings of the Eleventh Workshop on Mining and Learning with Graphs, pp. 0–6, 2013.

© 2022 IEEE. This is the author's version of the article that has been published in IEEE Transactions on Visualization and Computer Graphics. The final version of this record is available at: 10.1109/TVCG.2021.3114857

- [51] C. Nobre, M. Meyer, M. Streit, and A. Lex. The state of the art in visualizing multivariate networks. In *Computer Graphics Forum*, vol. 38, pp. 807–832. Wiley Online Library, 2019.
- [52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.
- [53] A. Perer and B. Shneiderman. Balancing systematic and flexible exploration of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):693–700, 2006.
- [54] R. Pienta, F. Hohman, A. Endert, A. Tamersoy, K. Roundy, C. Gates, S. Navathe, and D. H. Chau. Vigor: interactive visual exploration of graph query results. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):215–225, 2017.
- [55] R. Pienta, A. Tamersoy, A. Endert, S. Navathe, H. Tong, and D. H. Chau. Visage: Interactive visual graph querying. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pp. 272–279, 2016.
- [56] A. Pister, P. Buono, J.-D. Fekete, C. Plaisant, and P. Valdivia. Integrating prior knowledge in mixed-initiative social network clustering. *IEEE Transactions on Visualization and Computer Graphics*, 2020.
- [57] J. Pretorius, H. C. Purchase, and J. T. Stasko. Tasks for multivariate network analysis. In *Multivariate Network Visualization*, pp. 77–95. Springer, 2014.
- [58] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.
- [59] B. Schroeder and S. Tripathi. Structured query-based image retrieval using scene graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 178–179, 2020.
- [60] U. S. Shanthamallu, J. J. Thiagarajan, H. Song, and A. Spanias. Gramme: Semisupervised learning using multilayered graph attention models. *IEEE Transactions on Neural Networks and Learning Systems*, 31(10):3977–3988, 2019.
- [61] L. Shi, Y. Zhang, J. Cheng, and H. Lu. Skeleton-based action recognition with directed graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7912–7921, 2019.
- [62] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *European conference on computer vision*, pp. 1–15. Springer, 2006.
- [63] A. Srinivasan, H. Park, A. Endert, and R. C. Basole. Graphiti: Interactive specification of attribute-based edges for network modeling and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):226–235, 2017.
- [64] A. Streit, B. Pham, and R. Brown. Visualization support for managing large business process specifications. In *International Conference on Business Process Management*, pp. 205–219. Springer, 2005.
- [65] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. arXiv preprint arXiv:1205.6691, 2012.
- [66] J. J. Sutherland, L. A. O'brien, and D. F. Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structureactivity relationships. *Journal of chemical information and computer sciences*, 43(6):1906–1915, 2003.
- [67] C. Tominski, J. Abello, F. Van Ham, and H. Schumann. Fisheye tree views and lenses for graph visualization. In *Tenth International Conference on Information Visualisation (IV'06)*, pp. 17–24. IEEE, 2006.
- [68] Z. Tong, Y. Liang, C. Sun, D. S. Rosenblum, and A. Lim. Directed graph convolutional network. arXiv preprint arXiv:2004.13970, 2020.
- [69] R. Trudeau. Introduction to Graph Theory. Dover Books on Mathematics. Dover Publications, 2013.
- [70] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.
- [71] S. van den Elzen, D. Holten, J. Blaas, and J. J. van Wijk. Reducing snapshots to points: A visual analytics approach to dynamic network exploration. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):1–10, 2015.
- [72] S. Van den Elzen and J. J. Van Wijk. Multivariate network exploration and presentation: From detail to overview via selections and aggregations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2310–

2319, 2014.

- [73] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. Journal of machine learning research, 9(11), 2008.
- [74] A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. In *European conference on computer vision*, pp. 705–718. Springer, 2008.
- [75] C. Vehlow, F. Beck, and D. Weiskopf. Visualizing group structures in graphs: A survey. In *Computer Graphics Forum*, vol. 36, pp. 201–225. Wiley Online Library, 2017.
- [76] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. arXiv preprint arXiv:1710.10903, 2017.
- [77] Y. Wang, Z. Jin, Q. Wang, W. Cui, T. Ma, and H. Qu. Deepdrawing: A deep learning approach to graph drawing. *IEEE Transactions on Visualization* and Computer Graphics, 26(1):676–686, 2019.
- [78] S. Wasserman, K. Faust, et al. Social network analysis: Methods and applications. 1994.
- [79] M. Wattenberg. Sketching a graph to query a time-series database. In CHI '01 Extended Abstracts on Human Factors in Computing Systems, CHI EA '01, p. 381–382. Association for Computing Machinery, New York, NY, USA, 2001. doi: 10.1145/634067.634292
- [80] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826, 2018.
- [81] P. Yanardag and S. Vishwanathan. Deep graph kernels. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1365–1374, 2015.
- [82] S. Yoon, W. Y. Kang, S. Jeon, S. Lee, C. Han, J. Park, and E.-S. Kim. Image-to-image retrieval by learning similarity between scene graphs. *arXiv preprint arXiv:2012.14700*, 2020.
- [83] Z. Zhou, C. Shi, X. Shen, L. Cai, H. Wang, Y. Liu, Y. Zhao, and W. Chen. Context-aware sampling of large networks via graph representation learning. *IEEE Transactions on Visualization and Computer Graphics*, 2020.