CGQ: Relationship-Aware Contextual Graph Querying in Large Networks

Jithin Vachery[1] Akhil Arora[2] Sayan Ranu[3] Arnab Bhattacharya[4]
[1]Dept. of Computer Science and Engineering, Indian Institute of Technology, Madras, India
[2]Text and Graph Analytics, Xerox Research Centre India, Bangalore, India
[3]Dept. of Computer Science and Engineering, Indian Institute of Technology, Delhi, India
[4]Dept. of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India
[1]jithin@cse.iitm.ac.in [2]akhil.arora@xerox.com [3]sayanranu@cse.iitd.ac.in
[4]arnabb@cse.iitk.ac.in

ABSTRACT

The phenomenal growth of graph data from a wide-variety of real-world applications has rendered graph querying to be a problem of paramount importance. Traditional techniques use *structural* as well as node similarities to find matches of a given *query* graph in a *(large) target* graph. However, almost all previous research has tacitly ignored the presence of *relationships* and *context* (usually manifested in the form of node/edge label distributions) in the query. In this paper, we propose *CGQ – Relationship-aware Contextual Graph Querying* for real-world graphs. Given a query graph and a target graph, CGQ identifies the (top-k) maximal common subgraph(s) between the query and the target graphs with the highest contextual similarity. We prove that the problem is NP-hard and APX-Hard. To overcome this computational bottleneck, we propose a hierarchical index, *CGQ-Tree*, with its associated *CGQ search* algorithm. Empirically, the CGQ search algorithm is capable of achieving speed-ups of up to three orders of magnitude over a baseline strategy. Our experiments show that CGQ is *effective*, *efficient* and *scalable*.

1. MOTIVATION

Recent advances in scientific technologies generate a lot of data that are in the form of graphs such as protein-protein interaction networks [30], social networks [11], and the semantic web[1]. Consequently, graph-based searching and querying have received significant interest in both academia [16, 25, 26, 36] and industry (Ex. Facebook's Graph Search[2] and Google's Knowledge Graph[3]).

One of the most common frameworks in graph querying is to find similar embeddings of a query graph q in a much larger target graph G. More formally, for a query graph q and a distance function $d(q, g)$ between two graphs q and g, the goal is to identify the k most similar subgraphs $g_1, \cdots, g_k \subset G$ in the target graph G. The notion of "similar" graphs depends on the efficacy of the distance function $d(q, g)$. Typical distance (or similarity) functions for graphs range from maximal common subgraphs and graph edit distance [48] to more recent developments in fuzzy graph matching [25, 26]. While these distance functions extend the state-of-the-art in graph querying, they *lack* the ability to cope with several requirements posed by real-world graphs of today.

1. Querying based on relationships: Traditional similarity functions consider two graphs as similar if they are structurally similar and they contain similar nodes [16, 25, 26, 48]. Two nodes are similar if they are represented by similar feature vectors (the most common form is just a node label). This definition of similarity function is completely oblivious to the relationships encoded in the graphs.

To illustrate, assume our target graph is a collaboration network among researchers, such as DBLP. In a collaboration network, each node is an author, and two nodes are connected if they have co-authored a paper. In addition, each node is tagged with a feature vector that characterizes the corresponding author. On this network one wants to query using the graph q_1 shown in Fig. 1. It describes a collaboration pattern among three authors. Several relationships are encoded in this query. First, the authors are of similar repute (based on H-index). Second, they work in the same university (Stanford) and country. Third, this is a multi-disciplinary collaboration.

The graph t_1 shows a collaboration pattern that is considered a "good" match using traditional distance functions. Structurally, t_1 is identical to q_1. The nodes in t_1 also match well with those in q_1 across most features. For example, Hector G. Molina matches well with Sebastian Thrun across all features except Area. Similarly the graduate students match well with their counterpart nodes in q_1 in all features except H-Index. However, despite this good correspondence in structure and node descriptions, these two collaboration patterns are much different semantically; while q_1 is a multi-disciplinary collaboration among established authors, t_1 is a typical scenario of an established author guiding two research students. The semantics are lost due to two important relationships encoded in q_1 not being captured in t_1; neither is this a collaboration among established authors, nor is this a truly multi-disciplinary effort.

In contrast, consider the collaboration pattern g_1 in Fig. 1. Notice that, although, none of the nodes in g_1 match well with the nodes in q_1, g_1 does represent a multi-disciplinary collaboration pattern among established authors working in the same organization. In other words, *to capture similar relations, we need to move beyond*

[1]https://www.w3.org/standards/semanticweb/
[2]https://en.wikipedia.org/wiki/Facebook_Graph_Search
[3]https://www.google.com/intl/es419/insidesearch/features/search/knowledge.html

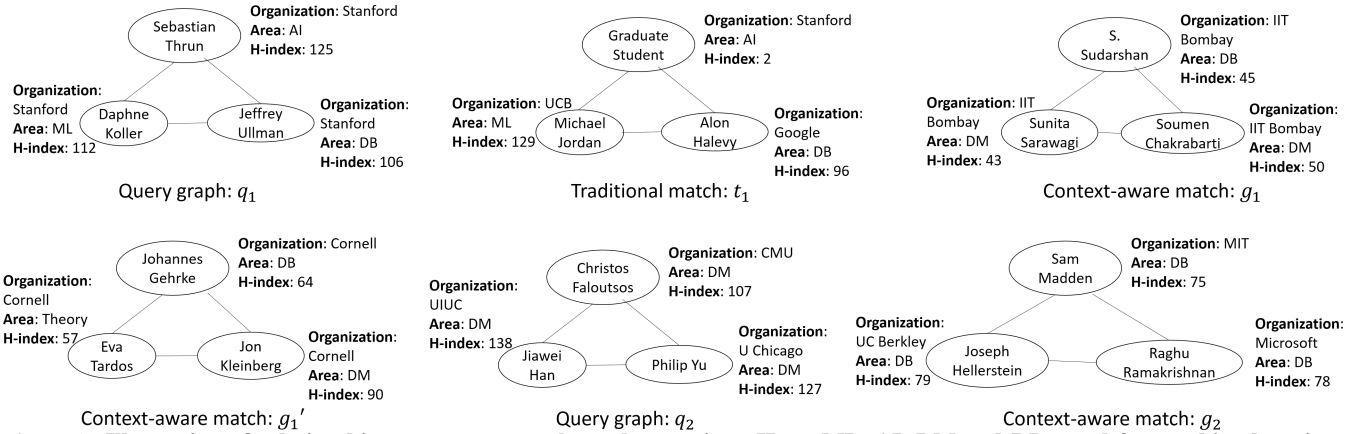


Figure 1: **Illustration of relationship-aware contextual graph querying. Here, ML, AI, DM and DB stand for machine learning, artificial intelligence, data mining, and databases respectively.**

node similarity.

2. Learning the context: Revisiting q_1 in Fig. 1, is it as important to preserve the relationship that all authors are from the same country compared to the relationship that all have such high H-index? More precisely, *do all relationships contribute equally to similarity and semantics?*

One obvious approach to answer this question is to simply let the user define the importance of each relationship. Such an approach however is neither practical nor user-friendly. In our work, we learn the importance of each relationship based on the *context* manifested in the query. To illustrate, the relationship that all authors are from USA does not stand out since this could happen just by chance as USA contributes a large number of scientific papers to the research community. On the other hand, a collaboration among authors of such high H-index is rare and is therefore likely to have been expressed in the query on purpose by the user. Consequently, it is important to preserve this relationship on the matched subgraphs. In other words, a relationship is more important if it has been expressed on purpose by the user rather than being present as a co-incidence. We call the rarity of the expressed relations as *context* and use it to rank the importance of each relationship in the query graph.

Note that recently there has been work on *relationship queries* [44]. Although they deal with relationships, the proposed problem is much different. For further details please refer to Sec. 5.

Challenges: Several challenges emerge while matching subgraphs based on the relationships they encode.

- **Quantifying relationship:** Each feature corresponds to a relation and an edge encodes the relationships between the two nodes it connects. How do we quantify these relationships? A relationship may indicate diversity, such as in a multi-disciplinary collaboration, or a property being preserved such as authors from the same organization, country, or similar H-index collaborating.
- **Quantifying context:** Context defines the relative importance of each relationship. This task is non-trivial. How can we quantify the fact that collaboration among authors of high H-index is rare and, therefore, more important for retaining the semantics? Furthermore, how high an H-index is really *high enough*? We answer all these questions, by computing the *statistical significance* of each feature (or relationship).
- **Context is dynamic:** The context *changes* with each query. For example, in q_1, H-index is an important factor since a collaboration among authors of such high repute is rare. However, given a query portraying a collaboration among three authors with H-indices of 2, 3, and 4, H-index would cease to be an important relationship since they are found in abundance and, therefore, likely to be manifested due to chance than on purpose. In other words, context changes from query to query.
- **Computational challenge:** Graph querying is computationally challenging since the number of subgraphs in the search space is exponential with respect to the size of the target graph. Consequently, without the support of an index structure, answering the proposed contextual queries is not feasible.

Contributions: Owing to the challenges outlined above, we need to design an *adaptive similarity function* and a *flexible index structure* for answering relationship-aware contextual graph queries. The key contributions of our work are as follows:

- We introduce the notion of *context* in the field of graph search, which although popularly used for querying web data, is the first-ever effort for querying graph data. Our novel (sub)-graph searching paradigm facilitates learning of the context prevalent in the query graph, which is further used to devise a context-aware similarity function (Sec. 2).
- We design a similarity function that incorporates the notion of *relationship* and *context* in the field of graph search (Sec. 2).
- Under the proposed searching paradigm, we propose a novel problem of retrieving the top-k most *contextually similar subgraph(s)* of a query graph (Sec. 2.5).
- We propose a *flexible* CGQ index structure that possesses the capability to adapt to dynamic similarity functions along with an *efficient* CGQ querying algorithm (Sec. 3).
- Through an in-depth empirical analysis on real datasets, we establish the superiority of the proposed similarity metric using both qualitative and quantitative results (Sec. 4). We also show that the proposed CGQ index and its associated querying algorithm is *scalable*: we achieve a speed-up of up to three orders of magnitude over a baseline strategy with a small memory footprint (Sec. 4). The executables and datasets are available at http://bit.ly/2fishkH.

2. RELATIONSHIP-AWARE CONTEXTUAL GRAPH QUERYING

In this section, we formulate the problem of context-aware graph

Item	Definition
$g_1 \subseteq g_2$	g_1 is subgraph isomorphic to g_2
$f(v) = [f_1(v), \ldots, f_d(v)]$	d-dimensional feature vector for node v
$\mathcal{N}(f_i, e)$	Neighborhood vector for feature f_i around edge e
$w(e) = [w(f_1, e), \ldots, w(f_d, e)]$	Weight vector of edge e
$s(e) = [s(f_1, e), \ldots, s(f_d, e)]$	Relationship vector corresponding to edge e
$cs(e, e')$	Contextual similarity between edges e and e'
$\phi : V_q \rightarrow V$	(Sub)graph injection function
$CGS_\phi(q, G)$	Contextual similarity between q and G

Table 1: **Summary of the notations used.**

querying and define the concepts central to it. The basic objective of our problem is to identify similar contextual embeddings of a query graph q in a (large) target graph G. A graph is *contextually* similar to q if it preserves the same relationships and context expressed in q. The notations used in this paper are summarized in Table 1.

2.1 Preliminaries

DEFINITION 1 (TARGET GRAPH). *A target graph is a labeled graph $G = (V, E, \mathbb{F})$, with the node set V, edge set E, and a set of feature vectors $\mathbb{F}$, where (1) each target node $v \in V$ represents an entity, (2) each target edge $e = (u, v) \in E$ denotes the relationship between the two entities u and v, and (3) $\mathbb{F}$ is a set of d-dimensional feature vectors $f(v) = [f_1(v), \ldots, f_d(v)]$ associated with each node $v \in V$.*

In practice, the features correspond to various node attributes, such as price, category, company, etc., in a co-purchase network, or H-index, area of research, etc., in a collaboration network.

DEFINITION 2 (QUERY GRAPH). *A query graph $q = (V_q, E_q, \mathbb{F}_q)$ is a labeled graph with a set of query nodes V_q, a set of query edges E_q, and a set of feature vectors $\mathbb{F}_q$.*

Both the target graph and query graph can be either directed or undirected. In our framework, the user or application only provides the structure of the query graph, i.e., the nodes and edges, and does not require any knowledge of the feature vectors. The feature vectors of the nodes are extracted automatically from the data repository. For example, DBLP could serve as the data repository in co-authorship networks.

Since our goal is to find contextually similar embeddings of a query graph q in a target graph G, next, we define the notions of (sub-)graph isomorphism and maximal common subgraph between two graphs.

DEFINITION 3 (GRAPH ISOMORPHISM). *Graph $g_1 = (V_1, E_1, \mathbb{F}_1)$ is isomorphic to $g_2 = (V_2, E_2, \mathbb{F}_2)$ if there exists a bijection ϕ such that for every vertex $v \in V_1$, $\phi(v) \in V_2$ and for every edge $e = (u, v) \in E_1, \phi(e) = (\phi(u), \phi(v)) \in E_2$.*

The concept of *subgraph isomorphism* is defined analogously by using an *injection* instead of a bijection. We use the notation $g_1 \subseteq g_2$ to denote that g_1 is subgraph isomorphic to g_2.

DEFINITION 4 (MAXIMAL COMMON SUBGRAPH). *A subgraph $g = (V_g, E_g, \mathbb{F}_g)$ is a maximal common subgraph (MCS) of q and G if there exists no other common subgraph g' of q and G such that the vertex and edge sets of g are proper subsets of that of g'. Formally, g is an MCS of q and G if $g \subseteq G$, $g \subseteq q$ and $\nexists g' = (V_{g'}, E_{g'}, \mathbb{F}_{g'})$ such that $g' \subseteq G$, $g' \subseteq q$, $V_{g'} \supset V_g$ and $E_{g'} \supset E_g$.*

There are two key components in defining a context-aware similarity function: *quantifying the context* and capturing *similar relationships* between nodes. Thus, we first develop a model to quantify context and then devise a mechanism to capture similar relationships. Finally, we tie them together to formulate the context-aware similarity function.

2.2 Capturing Relationships

Each edge encodes the relationships between the two nodes it connects. A relationship exists corresponding to each feature. For example, in q_1, every edge connects authors from the same organization. We capture these relationships in form of a *relationship vector* $s(e)$ corresponding to each edge $e = (u, v)$.

DEFINITION 5 (RELATIONSHIP VECTOR). *The relationship vector, $s(e)$, of an edge $e = (u, v)$ captures the features that are preserved in the relationship between u and v. Formally, $s(e) = [s(f_1, e), \ldots, s(f_d, e)]$ where*

$$s(f_i, e) = \begin{cases} \Gamma(f_i(u), f_i(v), & \text{if } f_i \text{ is real valued} \\ 1, & \text{if } f_i \text{ is categorical and } f_i(u) = f_i(v) \\ 0, & \text{if } f_i \text{ is categorical and } f_i(u) \neq f_i(v) \end{cases} \quad (1)$$

$$\text{where } \Gamma(x, y) = \begin{cases} 1, & \text{if } x = y = 0 \\ \min\{x, y\}/\max\{x, y\}, & \text{otherwise} \end{cases} \quad (2)$$

EXAMPLE 1. *The relationship vector $s(e)$ for the edge e connecting Sebastian Thrun and Daphne Koller in q_1 is $[1, 0, 0.90, 1]$ while the relationship vector $s(e')$ for the edge e' connecting S. Sudarshan and Sunita Sarawagi in g_1 is $[1, 0, 0.96, 1]$.*

From the definition, it can be see that each feature f_i takes a value in the range $[0, 1]$, where a value close to 1 indicates that both endpoints of the edge share similar values in f_i.

Intuitively, the relationship vectors for both edges in Ex. 1 are similar. However, as discussed in Sec. 1, all relationships may not be equally important. Our next goal is therefore to learn the importance of each relationship based on the context prevalent in the query.

2.3 Learning the Context

Context can be loosely defined as "the circumstances that form the setting for an event". In our problem, the user expresses the setting implicitly through latent query characteristics. For example, in q_1 of Fig. 1, the user wants to find other collaborations among authors of similar reputes and employed in the same organization. To a human being, this context is easy to decipher since in a random collaboration subgraph it is unlikely for all three authors to belong to the same organization and have such high H-indices. We capture this intuition by computing the *statistical significance* of each feature where the higher the significance, the more is its importance in setting the context. For example, in q_1, H-index should have the highest statistical significance followed by organization. Similarly, in q_2, H-index and area are the most significant features. In the discussion that follows, we develop a mathematical model to accomplish this task.

Formally, our goal is to learn a weight $w(f_i)$ for each feature f_i where $w(f_i)$ corresponds to the statistical significance of the i^{th} feature in the query graph. Statistical significance tests quantify whether an event is likely to occur due to chance alone or due to some factor of interest. Several models such as chi-square test, g-test and log-likelihood ratio exist to quantify statistical significance. In our work, we use the *chi-squared statistic* [31].

The chi-square statistic measures the deviation of the observed frequencies (O_x) from their expected values (E_x) given by the null model, and is mathematically defined as:

$$X^2 = \sum_{\forall x} \frac{(O_x - E_x)^2}{E_x}. \quad (3)$$

The higher the chi-square, the more statistically significant the observed event is.

Let $\mathcal{V}(f_i)$ represent the set of unique feature values assumed by the feature f_i in a graph G. Then the observed frequency for any $\nu \in \mathcal{V}(f_i)$ is simply the number of times ν is encountered in a graph.

EXAMPLE 2. *In q_1 of Fig. 1, the feature "f1: Organization" and "f4: Country" has taken only one value 'Stanford' and 'USA' respectively; and they both have an observed frequency of* 3*. On the other hand, both "f2: Area" and "f3: H-index" have taken three unique values, with each value possessing an observed frequency of* 1*.*

Generally, a context is set by the query when a statistically significant pattern is observed in the feature distribution of nodes. The straightforward approach to model context, therefore, is to learn the chi-square value for each feature in the graph. However, learning a global chi-square value for each feature imposes the generalization that the context is invariant across all local regions in the query. This assumption is too strong, particularly when the query graph is large. Hence, We build a contextual model for *each edge* in the query graph. Each contextual model summarizes the context in the neighborhood around the corresponding edge. If the query graph is small, or the context is homogeneous across the entire query, then the contextual model for each edge should converge to the same values. On the other hand, if there are localized contexts in the query, the individual edge-centered context models should learn them.

Given an edge $e = (u, u')$, for each feature f_i, we learn a *neighborhood vector* $\mathcal{N}(f_i, e)$.

DEFINITION 6 (NEIGHBORHOOD VECTOR). *For an edge $e = (u, u')$, let $R_h(e)$ be the set of nodes reachable within h hops from either u or u'. Let $\mathcal{V}(f_i, e)$ be the set of all unique values in feature f_i among the nodes in $R_h(e)$. The neighborhood vector, $\mathcal{N}(f_i, e)$, is a set of tuples of the form*

$$\mathcal{N}(f_i, e) = \{\langle \nu, \mathcal{P}(\nu)\rangle | \nu \in \mathcal{V}(f_i, e)\} \quad (4)$$

$$\mathcal{P}(\nu) = \sum_{\forall v \in R_h(e), f_i(v) = \nu} \delta^{\min\{l(v,u), l(v,u')\}} \quad (5)$$

where $l(v, u)$ is the length of the shortest path from u to v, and δ is a decay parameter.

EXAMPLE 3. *Let us revisit q_1 in Fig. 1 that contains the features $F = \{f_1$: Organization, f_2 : Area, f_3 : H-Index, f_4 : Country$\}$. Let $h = 1$ and $\delta = 0.1$. We will construct the neighborhood vector for f_1 : Organization on edge $e =$ (Sebastian Thrun, Daphne Koller). Since the graph is a triangle, all nodes are within 1-hop from every edge. Thus, $R_h(e) = V_q$. Since all nodes have 'Stanford' as the organization, $\mathcal{V}(f_i, e) = \{$Stanford$\}$. Thus, $\mathcal{P}($Stanford$) = 0.1^0 + 0.1^0 + 0.1^1 = 2.1$ and $\mathcal{N}(f_1, e) = \{\langle$Stanford$, 2.1\rangle\}$. Similarly, for feature f_2 : Area, $\mathcal{V}(f_i, e) = \{$ML, AI, DB$\}$, and $\mathcal{N}(f_2, e) = \{\langle$AI$, 1\rangle, \langle$ML$, 1\rangle, \langle$DB$, 0.1\rangle\}$.*

Intuitively, the neighborhood vector $\mathcal{N}(f_i, e)$ is similar to a frequency vector of all feature values that occur in the neighborhood of an edge. However, through decay factor δ, nodes closer to e have a larger say than those further away.

Null model and expected frequency: With the formalization of the neighborhood vector of a feature f_i, we next focus on its chi-square value. While $\mathcal{N}(f_i, e)$ provides the observed frequency of each feature value in e's neighborhood, we still need to compute its expected frequencies. Towards that, we assume that the feature values for f_i in the query graph q are drawn independently from the distribution of f_i in the target graph G.

Formally, let the probability distribution of the feature values for feature f_i in the target graph G, denoted by $P(f_i) = \{p_1, \ldots, p_{|\mathcal{V}(f_i)|}\}$, represents the null model. Here, $p_j \in P(f_i)$ represents the probability that $f_i(v) = \nu_j \mid v \in V$ for some $\nu_j \in \mathcal{V}(f_i)$. Note that, unlike $\mathcal{V}(f_i, e)$ which is associated with the neighborhood of an edge e in the query graph q, $\mathcal{V}(f_i)$ is independent of such association and represents the set of unique values for the feature f_i in the target graph G. Since each feature value is drawn randomly and independently, the expected values of unique features in the neighborhood vector of the query are $\mathbb{E}[\nu] = \hat{n}.p_\nu, \forall \nu \in \mathcal{V}(f_i, e)$ where

$$\hat{n} = \sum_{\nu \in \mathcal{V}(f_i, e)} \mathcal{P}(\nu) \quad (6)$$

The chi-square value, therefore, is

$$X^2 = \sum_{\nu \in \mathcal{V}(f_i, e)} \frac{(\mathcal{P}(\nu) - \hat{n}.p_\nu)^2}{\hat{n}.p_\nu} \quad (7)$$

EXAMPLE 4. *Following on from Ex. 3, we compute the chi-square value of "Organization" on edge $e =$ (Thrun, Koller). Let the null model on "Organization" be as follows:*

$$\mathcal{V}(f_1) = \{\text{Stanford, Cornell, UCB, Google, Microsoft, MIT, IIT Bombay, UIUC, CMU, U Chicago}\}$$

$$P(f_1) = \{0.30, 0.15, 0.10, 0.05, 0.05, 0.05, 0.15, 0.05, 0.05, 0.05\}$$

Since $\mathcal{N}(f_1, e) = \{\langle$Stanford$, 2.1\rangle\}$, $\hat{n} = 2.1$. Thus, over the edge $e =$ (Sebastian Thrun, Daphne Koller), the chi-square value for f_1 : Organization is

$$X^2(f_1, e) = \sum_{\nu \in \mathcal{V}(f_i, e)} \frac{(\mathcal{P}(\nu) - \hat{n}.p_\nu)^2}{\hat{n}.p_\nu} = (2.1 - (2.1 \times 0.3))^2/(2.1 \times 0.3) = 3.43.$$

To summarize, a high X^2 value for an edge indicates that its neighborhood vector deviates from what is expected according to the null model. This deviation is too large to occur by chance and, therefore, sets the context for similarity queries.

The X^2 values for all other features $f_i \in F$ are similarly calculated for the edge e. The weight of the feature value for the edge is the normalized X^2 value:

$$w(f_i, e) = \frac{X^2(f_i, e)}{\sum_{j=1}^{d} X^2(f_j, e)} \quad (8)$$

The *weight vector*, $w(e)$, for each edge e is the collection of d weights that each feature plays in defining the context around e:

$$w(e) = [w(f_1, e), \cdots, w(f_d, e)] \quad (9)$$

Null model for continuous-valued features: The null model, that essentially captures the distribution of feature values in the target graph, is easy to learn when the feature is discrete-valued since it only involves computing frequency counts. For continuous-valued features, we use kernel density based estimators to learn the underlying data distribution and discretize the feature space into appropriate bins [8].

2.4 Relationship-Aware Contextual Similarity Measure

Having formalized an approach for learning the context, the next task is to formalize the similarity between two graphs under the contextual matching paradigm.

With the foundation to quantify relationships and their importances by learning the context, we next define the contextual similarity between two graphs.

First, we formulate the *contextual* similarity, $cs(e, e')$, between two edges e and e' is defined as follows, where e is an edge in the

query graph and e' is from the target graph.

$$cs(e, e') = \sum_{i=1}^{d} w(f_i, e) \cdot \Gamma\left(s(f_i, e), s(f_i, e')\right) \quad (10)$$

More simply, $cs(e, e')$ is a weighted min-max similarity between the two relationship vectors. $\Gamma\left(s(f_i, e), s(f_i, e')\right)$ is defined in Eq. 2.

EXAMPLE 5. *To illustrate contextual similarity, we revisit query graph q_1 and its context-aware match g_1 in Fig. 1. The relationship vector $s(e)$ for the edge e connecting Sebastian Thrun and Daphne Koller in q_1 is $[1, 0, 0.90, 1]$ while the relationship vector $s(e')$ for the edge e' connecting S. Sudarshan and Sunita Sarawagi in g_1 is $[1, 0, 0.96, 1]$. Assume that the learned weight distribution for the edge e is $w(e) = [0.35, 0.3, 0.39, 0.01]$. Then, the similarity between the edges e and e' is $0.35 \times \frac{1}{1} + 0.3 \times 1 + 0.34 \times \frac{0.90}{0.96} + 0.01 \times \frac{1}{1} = 0.98$..*

With the power to compute the contextual similarity between two edges, we are now ready to define contextual similarity between two graphs.

DEFINITION 7 (CONTEXTUAL SIMILARITY FUNCTION). *For a subgraph $g \subseteq G$, which is an MCS of target graph G and query q, let $\phi : V_g \rightarrow V_q$ be the injection from the vertices of g to q such that $e_g = (u_g, v_g) \in E_g$ if $\phi(e_g) = (\phi(u_g), \phi(v_g)) \in E_q$. The contextual similarity between g and q is defined as:*

$$CGS(g, q) = \sum_{\forall e_g = (u_g, v_g) \in E_g} cs(e_g, \phi(e_g))) \quad (11)$$

To summarize, the context-aware similarity measure aggregates the contextual similarity between the mapped edges. We next highlight some of key properties of the similarity function.

2.4.1 Properties

- **Range of the similarity value:** The similarity, $cs(e, e')$, between two edges e and e', ranges between 0 and 1. Consequently, the similarity between two graphs $g_1 = (V_1, E_1)$ and $g_2 = (V_2, E_2)$ ranges between 0 and $\max\{|E_1|, |E_2|\}$.
- **Monotonicity with size of subgraphs:** Let $g_1 = (V_1, E_1)$ and $g_2 = (V_2, E_2)$ be two graphs such that $E_2 \supseteq E_1$ (and, therefore, $V_2 \supseteq V_1$). For any query graph q, it is guaranteed $CGS(q, g_2) \geq CGS(q, g_1)$. This follows from the fact that having more edges in g_2 can never hurt the similarity since edge similarity is always a non-negative value. Consequently, searching only among maximal common subgraphs (MCS) of the query and the target graph are enough to identify the best matches.
- **Statistical significance:** Although we use chi-square statistic, X^2, any other method of computing statistical significance can also be used to define the weight vector (Eq. 9). We use X^2 since it is well studied in literature [5, 33] and efficient to compute.

Why Chi-Square? There are two standard techniques of graph mining based on either frequencies or statistical significance. While frequency-based methods such as tf-idf are simpler, they cannot adapt well to different distributions and often require specifying a frequency threshold either explicitly or implicitly. Hence, we use statistical significance based methods as they are more robust, can estimate the parameters empirically from the data itself, and can directly compare the importance of different outcomes in terms of how likely they are. The exact formulation of statistical significance depends on the statistical model used and the metric. Between the two most common metrics, *p-value* and *z-score*, in most practical cases, p-value provides more precise and accurate results [13]. However, computing the p-value exactly requires analyzing a potentially exponential number of outcomes, thereby making the computation impractical. It was showed that for large samples, asymptotic approximations are accurate enough and easier to calculate [31]. The *Pearson's chi-square statistic*, X^2, that asymptotically converges to the χ^2 distribution, provides a robust and efficient way to compute the p-value. Hence, in this paper, we use the chi-square statistic.

2.5 Problem Formulation

Our goal is to identify the k maximal common subgraphs between q and G having the largest *relationship-aware contextual similarity*.

PROBLEM 1 (TOP-k CONTEXTUALLY SIMILAR MCS). *Find the top-k contextually similar maximal common subgraphs of query graph q and target graph G.*

Formally, given a query graph $q = (V_q, E_q, \mathbb{F}_q)$, a target graph $G = (V, E, \mathbb{F})$, and a positive integer k, identify the subgraphs $\mathbb{A} = \{g_1 \subset G, \cdots, g_k \subset G\}$, such that

1. *$\forall g_i \in \mathbb{A}$, g_i is an MCS of G and q, and*
2. *$\nexists g' \subseteq G$ such that g' is an MCS of G and q and $CGS(g', q) > CGS(g_i, q)$ for some $g_i \in \mathbb{A}$.*

Searching for MCSs of q and G has several advantages over restricting to only exact embeddings of q in the target graph G, i.e, subgraphs of G that are isomorphic to q. First, we are able to extract good matches even when q is not a subgraph of G. Second, even when isomorphic embeddings of q are present in G, they may not have contextual similarity to q and, therefore, it is preferable to identify MCSs that have higher contextual similarity than the exact embeddings. Overall, MCSs allows us to be expand the search space and extract better contextual matches.

While we mostly discuss top-k queries, our similarity function as well as the proposed index structure can easily be applied to range queries. A discussion on range queries is included in Appendix A. We focus on top-k query since specifying k is more intuitive for the end user than a similarity threshold.

2.6 Complexity Analysis

THEOREM 1. *Top-k context-aware querying is NP-hard.*

PROOF. We prove that the problem is NP-hard by reducing the *maximum common subgraph* (MCG) isomorphism problem, which is NP-hard [20], in polynomial time to it. Given a query graph $q = (V_q, E_q, \mathbb{F}_q)$ and a target graph $G = (V, E, \mathbb{F})$, without loss of generality, assume that all the node feature values associated with the query and the target graph are identical. Under this setting, all the subgraph isomorphic matchings (if existent) of q in G possess the same and the largest contextual similarity. Moreover, any smaller sized (maximal) common subgraph between q and G will be of lower similarity when compared to the maximum common subgraph (the second property in Sec. 2.4.1). Thus, identifying the maximum common subgraph between q and G reduces to finding the most contextually similar subgraph of G to q, i.e., context-aware top-k querying with $k = 1$. Thus, proved. □

COROLLARY 1. *Top-k context-aware querying is APX-hard.*

PROOF. Since *maximum common subgraph* problem is APX-hard [4, 24, 41], our problem is APX-hard as well. □

3. CGQ FRAMEWORK

Sec. 2.6 establishes that top-k context-aware querying is not only NP-hard but also hard to approximate. To overcome this computational challenge, we focus on building an index structure. The individual problems of indexing subgraph queries and indexing high-dimensional points (feature vectors) have both been well studied [21–23, 45]. However, for situations where *both* the structure and the feature vectors matter, such as in CGS, very few techniques exist. Furthermore, in CGS, the importance of each dimension changes according to the weight vector (Eq. 9), which makes the existing index structures inapplicable. In summary, there are three primary challenges: (a) matching structures, (b) indexing the feature space, and (c) adapting to dynamic weight vectors. We next show how to handle theses challenges.

However, as a benchmark and to understand the problem better, we first analyze the naïve algorithm.

3.1 The Naïve Algorithm

The naïve algorithm simply enumerates each possible maximal common subgraph of the target and the query graph, and computes its contextual similarity. If the similarity is among the top-k, then the subgraph is part of the answer set.

MCSs are enumerated using the following bottom-up approach. First, we pick an arbitrary edge $e_q \in E_q$ from the query graph q and map it to an arbitrary edge e of the target graph. This forms an 1-edge common subgraph of q and G. Next, we try to grow into a 2-edge subgraph by extending both e_q and e through one of their neighboring edges such that the two extended subgraphs are isomorphic and, therefore, a common subgraph of q and G. This procedure of growing through a neighboring edge continues iteratively till the grown subgraphs can no longer be isomorphic. The isomorphic subgraph where we stop is an MCS. This entire process is repeated by trying all possible edge combinations for the initial 1-edge common subgraph.

There are several bottlenecks in the above approach. First, the naïve algorithm evaluates an exponential number of subgraphs with respect to the target graph size. Second, the above procedure is completely oblivious to the feature vectors. More specifically, if a query edge e_q is mapped to an edge e in the target graph such that e_q and e capture highly dissimilar relationships, then this mapping would contribute little to the contextual similarity. More generally, to achieve a high contextual similarity, the mapped edges must denote similar relationships.

The straightforward way to rectify the above problem is to start the growth from a pair of edges that are similar and then extend them with other similar edges in the neighborhood. The proposed index structure *CGQ* builds on this idea. Instead of enumerating all possible subgraphs of G, CGQ identifies regions in the target graph that potentially contains the best match to the query. By processing only these promising regions, CGQ drastically prunes the search space. The promising regions are identified by carefully studying the properties of the contextual similarity function and employing these properties to compute an upper bound on the similarity value. We begin the development of CGQ by deriving these properties.

3.2 Properties of Edge Similarity

Given an edge e, the relationship vector $s(e)$ (Def. 5) captures the features preserved in the relationship between its two end vertices. For a query edge e_q, our goal is to find the edge e in the target graph having the most similar relationship vector. Since a relationship vector is essentially a high-dimensional point, an index structure such as R-tree [22] could have been used. However, R-Tree (or its variants [7]) makes several assumptions that do not hold in our problem. Thus, we cannot use an R-tree as is, although our index structure is inspired from its design. Similar to R-trees, we use the concept of *minimum bounding rectangles (MBR)* to summarize the relationship vectors of a set of edges. The key aspects in which CGQ differs from R-tree are as follows.

- The distance function is assumed to be Euclidean. In our problem, the distance between two edges is computed using the weighted min-max similarity defined in Eq. 10.
- The computation of minimum and maximum similarity to an MBR is different in our problem due to the change in similarity function.
- Ours is a graph problem and thus we need to incorporate structural aspects in our search algorithm.

DEFINITION 8 (MINIMUM BOUNDING RECTANGLE (MBR)). *Given a set of d-dimensional relationship vectors $\mathbb{D} = \{s(e_1), \cdots, s(e_m)\}$, an MBR $\mathcal{H}$ on $\mathbb{D}$ is the smallest axis-parallel d-dimensional hyper-rectangle that encloses all vectors in $\mathbb{D}$.*

An MBR $\mathcal{H}$ can be uniquely represented by the co-ordinates of its *lowest* and *largest* diagonal points h^{min} and h^{max} respectively. More specifically, $h^{min} = [h_1^{min}, \ldots, h_d^{min}]$,where

$$h_i^{min} = \min_{\forall s(e) \in \mathbb{D}} \{s(f_i, e)\}$$

h^{max} is defined analogously. We use the notation $e \in \mathcal{H}$ to denote that the relationship vector of e is contained within MBR $\mathcal{H}$. Mathematically, this means $\forall i,\ h_i^{min} \leq s(f_i, e) \leq h_i^{max}$.

Next, we define the similarity between a relationship vector $s(e_q)$ of an edge e_q and an MBR $\mathcal{H}$ as follows.

$$\begin{aligned} cs(e_q, \mathcal{H}) &= \max_{\forall e \in \mathcal{H}} \{cs(e_q, e)\} \\ &= \max_{\forall e \in \mathcal{H}} \sum_{i=1}^{d} w(f_i, e_q) \Gamma(s(f_i, e_q), s(f_i, e)), \\ &= \sum_{i=1}^{d} w(f_i, e_q) \max_{s(f_i,e) \in [h_i^{min}, h_i^{max}]} \Gamma(s(f_i, e_q), s(f_i, e)). \end{aligned}$$

More simply, $cs(e, \mathcal{H})$ provides the maximum similarity between e and any possible edge contained within $\mathcal{H}$. This similarity value can now be used to provide the following upper bound on $\mathbb{D}$.

THEOREM 2. *If $\mathcal{H}$ is an MBR on a set of d-dimensional relationship vectors $\mathbb{D}$ and e_q is a query edge then,*

$$\max_{\forall s(e) \in \mathbb{D}} \{cs(e_q, e)\} \leq cs(e_q, \mathcal{H}).$$

Theorem 2 allows us to upper bound the maximum similarity between a query edge and a set of edges in the target graph. We next build on this platform to develop a tree that facilitates rapid pruning of the search space.

3.3 CGQ-Tree

We perform hierarchical partitioning on the entire edge set E of the target graph G in a top-down recursive manner. Each partition is summarized by a d-dimensional MBR constructed over edges contained in the partition. An MBR is represented using its two diagonals. It is worthwhile to note that although the edge-sets between two sibling partitions are disjoint, their corresponding MBRs may overlap. Edges are stored only at the leaf nodes along with the MBRs constructed over them. Non-leaf nodes only store the MBRs. During the tree building phase, we ensure that each node has b children. This controls the depth of the tree.

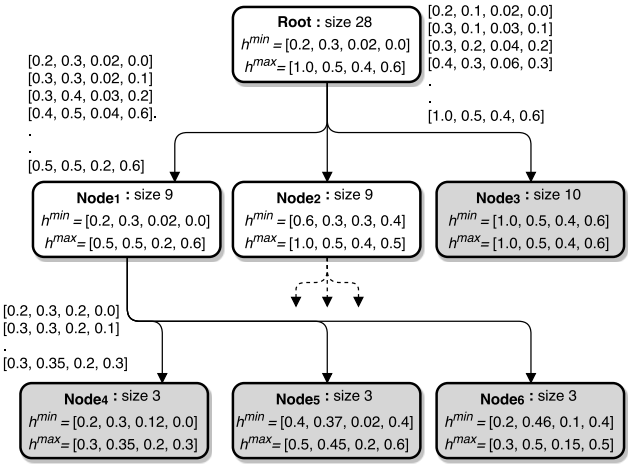


Figure 2: **The structure of the CGQ-tree. The leaf nodes are colored in grey. We do not explicitly show the relationship vectors in the leaf nodes due to space limitations.**

Alg. 1 presents the pseudocode of the tree construction algorithm. Before we initiate the partitioning procedure, we construct d orderings of E by sorting them based on their values in each dimension. The partitioning procedure starts at the root node, which contains the MBR on the relationship vectors of the entire edge set E. The edges in the root node are partitioned and distributed among b child nodes. To partition a set of m edges, we first choose the feature f_i having maximum variance (line 6). Each of the b child nodes receive $\frac{m}{b}$ edges from their parent node, where the first child node receives the vectors with the smallest $\frac{m}{b}$ values in feature f_i and so on (lines 7-11). [4] After receiving the relationship vectors from the parent, the MBR on them is constructed and the process is recursively repeated on each of the newly created child nodes (line 9). The recursion stops when a node receives less than n edges, where n is a threshold, or all edges have an identical relationship vector (lines 4-5). Edges are stored only in the leaf nodes. At internal nodes, edges received from the parent are only used to create the MBR.

EXAMPLE 6. *Fig. 2 illustrates the structure of the tree built on a set of 28 edges. The root node contains the entire set and is represented using the corresponding MBR. The branching factor, b is set to 3 and n to 4. At the second level, we partition the root on dimension 3 and three more nodes are formed. Here, note that although $Node_3$ has 10 edges, since all edges have the same relationship vector, we do not partition it any further. A recursive partitioning is performed on $Node_1$ and $Node_2$ to create level 2. Since all nodes at this level contain less than 4 vectors, no further partitions are performed.*

Time Complexity: We start by sorting all edges based on their values in each dimension. This consumes $O(d|E|\log|E|)$ time. Next, at each level of the tree at most $|E|$ edges are processed, where we compute the variance in each dimension, and thus requiring $O(d|E|)$ time. Since, the height of the tree is at most $\log_b|E|$, the overall complexity is bounded by $O(d|E|\log|E|)$.

Space Complexity: There are $O(|E|)$ nodes in the tree. Since each internal node stores only the diagonals of the MBR and all edges are stored only once in the leaf nodes, the total space complexity is $O(|E|)$.

[4]Note that the rightmost node in a level may contain more than $\frac{m}{b}$ nodes when m is not divisible by b.

Algorithm 1 ConstructTree(E, b)

Input: Edge-set E. E is sorted based on their values in each dimension.
Input: branching factor b, minimum edges in a node n
1: $\mathcal{H} \leftarrow MBR(E)$
2: $node \leftarrow$ a new node with MBR $\mathcal{H}$
3: **if** $|E| < n$ **or** $h^{max} = h^{min}$ **then**
4: Add E to $node$
5: **else**
6: $f_i \leftarrow$ feature with highest variance
7: **for** $j \in [1, b]$ **do**
8: $E_j = \{e \in E | s(f_i, e)$ is among the top $\frac{|E|}{b}$ smallest values$\}$
9: $c_j \leftarrow$ConstructTree(E_j, b)
10: add c_j as a child node of n
11: $E \leftarrow E - E_i$
12: **return** node

CGQ-tree allows us to quickly search edges that are similar to a query edge. Intuitively, we need to the follow a branch that is likely to lead to the best matching edge. Thus, starting from the root node, we prioritize all child nodes based on the distance of their MBRs to the query edge. The best child node is then chosen to explore next and the process is recursively applied till we reach a leaf node. The search procedure stops when all nodes that remain to be explored have a maximum possible similarity smaller than a target edge we have already found. In essence, we employ the *best-first search* algorithm. We discuss this more formally in Sec. 3.5.

3.4 Avoiding Local Maxima

While the above algorithm is efficient in locating a similar edge, it is prone to getting stuck in local minima. We illustrate this issue with Fig. 3. In this figure, each node is characterized by only one feature, which is its color. Suppose we want to find the best matching edge to e_q of query graph q in the target graph G. Consider e_1 and e_2 in the target graph. Both these edges are good matches since they preserve the color of their endpoints. Recall that in contextual similarity, node values are not matched; rather, we match features conserved in an edge. At this juncture, notice that e_2's neighborhood is significantly different from the neighborhood of e_q. More specifically, while e_q's neighborhood is homogeneous in terms of color, e_2's neighborhood is a mixture of different colors. Consequently, even though e_2 is a good match to e_q, it would be hard to find good matches for the neighboring edges of e_q with those in the neighborhood of e_2. As a result, matching e_q with e_2 is unlikely to lead to a graph with high contextual similarity. In contrast, e_1 is not only similar to e_q, but is also situated in a similar neighborhood with homogeneous colors. Consequently, the relationship encoded by edges in e_q's neighborhood are likely to be similar to the relationships captured by edges in e_1's neighborhood. Therefore, e_1 should be preferred over e_2.

To summarize the above discussion, searching for similar edges may lead us to a good match in a bad neighborhood (local maxima), and such leads should be avoided. A neighborhood is bad if the distribution of features is different from the neighborhood around the query edge. Therefore, an important question arises. *How can we capture similar neighborhoods without explicitly performing edge-to-edge matches?* To answer this question, let us recall the formulation of the weight vector (Eq. 9). A weight vector $wv(e) = [w(f_1, e), \cdots, w(f_d, e)]$ captures the statistical significance of the feature distributions in the neighborhood of an edge e. Now, if two neighborhoods are similar, then their statistical significances are likely to be similar as well. Certainly, it is possible to have dissimilar neighborhoods with similar weight vectors. However, such false positives can be pruned out in a later verification stage. Overall, weight vectors serve as good low-cost signatures of

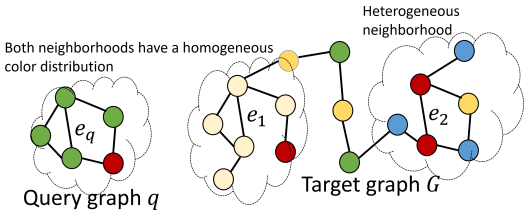


Figure 3: **An illustration of why the structural aspect and feature distribution need to be incorporated in the CGQ index.**

edge neighborhoods.

3.4.1 Searching for similar neighborhoods

Equipped with the above intuition, we leverage weight vectors to better guide our search strategy. Mathematically, given a query edge e_q, if $|w(f_i, e_q) - w(f_i, e)|$ is low, then the neighborhood of e_q and e are similar with respect to feature f_i. To identify edges located in similar neighborhoods, we calculate the weight vectors $w(e)$ for all the edges e in G. Next, in each leaf node of the CGQ-tree, we construct d sorted lists. Each list generates an ordering of all edges contained in the leaf node. List i corresponds to feature f_i and sorts edges based on $w(f_i, e)$. Owing to this arrangement, we can identify the target edge e minimizing $|w(f_i, e_q) - w(f_i, e)|$ by simply performing a binary search in list i and therefore requiring only $O(\log m)$ time, where m is the number of edges in this node.

Time Complexity: The weight vector calculations are done for each edge and it requires exploration of the h hop neighborhood around an edge. If η is the average degree in the target graph, the time complexity is $O(d\eta^h)$ on average for a single edge. h is typically 1 or 2. Sorting across d dimensions requires $O(d|E|log|E|)$ cost. After incorporating the cost of building CGQ-tree, the total index construction of CGQ is $O(d|E|log|E| + d\eta^h|E|)$.

Space Complexity: Weight vectors incur a storage of $O(d|E|)$ cost. The overall cost of the entire CGQ index is also $O(d|E|)$.

The incorporation of the weight vectors completes the CGQ index. In the next section, we discuss how the CGQ-tree in conjunction with the sorted weigh vectors are employed to answer context-aware top-k queries.

3.5 CGQ Search Algorithm

In the naïve algorithm discussed in Sec. 3.1, we start from a common 1-edge subgraph, and grow them through neighboring edges to form an MCS. The initial 1-edge subgraph is formed by mapping an arbitrary edge in the query graph to an arbitrary edge in the target graph. With the help of CGQ index structure, the goals of our searching algorithm are two-fold.

1. **Detect promising leads:** Instead of mapping two arbitrary edges, identify a promising mapping such that the resultant 1-edge common subgraph leads to an MCS with high contextual similarity.
2. **Prune early:** Instead of discovering that the contextual similarity of an MCS is low after fully growing it, prune them early in their initial stages of formation.

With these objectives in mind, we design our searching algorithm. Fig. 4 outlines the flowchart and Alg. 2 presents the pseudocode. As visible in Fig. 4, there are three major phases: selecting the most promising query edge e_q for exploration, identifying the best matching target edges, and growing MCSs from these initial seeds. We explain each of these phases next. Before we execute any of the phases, two operations are performed. First, we compute the weight vector for each edge in the query graph (line 1). Second, we initialize a priority queue called Ans, which stores the k most similar MCSs identified till now (line 2). Initially, Ans is empty.

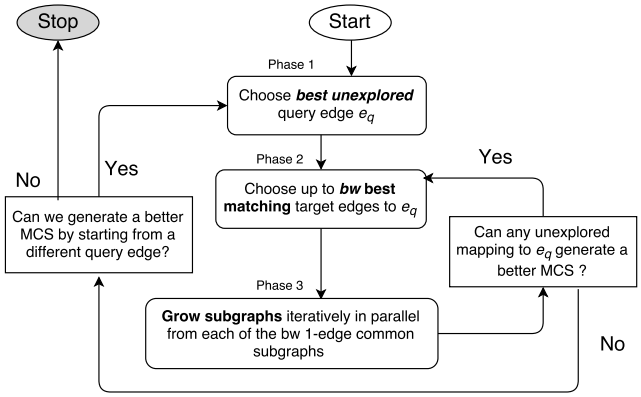


Figure 4: **Flowchart of the searching algorithm in the CGQ framework.**

3.5.1 Phase 1: Selecting the best query edge

Given a query graph q, we select the unexplored query edge $e_q \in E_q$ that has the highest similarity to the root node of CGQ-tree (line 5). An edge is unexplored if it has not already been selected in this phase in some earlier iteration.

3.5.2 Phase 2: Selecting the best target edges

In this phase, our goal is to find the best matching edges in target graph G to e_q. Towards that, we use *best-first search*. Specifically, we initialize a priority queue, *Cands* (line 6). $Cands$ stores MBRs corresponding to nodes in CGQ-tree and orders them in descending order of their similarity to the query edge e_q (Theorem 2). Initially, $Cands$ contains the root node. We iteratively pop the top node (or MBR) from $Cands$ and insert all its children to $Cands$ till a leaf node is reached. In this leaf node, we perform a *beam stack search* [49] with beam width bw (lines 13-20) . Beam stack search selects the bw best matching target edges to e_q and explores them further to identify contextually similar MCSs (Phase 3).

An edge in the target graph is a good match to e_q if it has high contextual similarity and it is located in a neighborhood similar to that of e_q's. Since the selected leaf node has the highest similarity to e_q among all unexplored leaf nodes in CGQ-tree, it is likely that all target edges in this node are similar to e_q. Thus, we shift our focus to neighborhood similarity and select bw edges having similar weight vectors to k. For that purpose, we first pick feature f_i with probability proportional to its statistical significance. Mathematically,

$$p(f_i) = \frac{w(f_i, e_q)}{\sum_{j=1}^{d} w(f_j, e_q)} \tag{12}$$

If f_i is the selected feature, we choose the target graph edge e that minimizes $|w(f_i, e_q) - w(f_i, e)|$. Recall from our discussion in Sec. 3.4.1, that this requires $O(\log m)$ time, where m is the number of edges in this node. The same process is then repeated till bw target edges are selected.

3.5.3 Phase 3: Grow from bw seeds to identify MCSs

Each of the bw edges selected in the previous phase generates a common 1-edge subgraph of the query and the target graph. We now grow them to form MCSs using the standard algorithm explained in Sec. 3.1 (line 19). However, we impose a prioritization scheme. Suppose $g = (V_g, E_g)$ is a common subgraph of the query and the target graph, but not necessarily an MCS. On g, we can apply the following bound.

Algorithm 2 CGQ Search Algorithm

Input: $q(V_q, E_q)$, k, $CGQIndex$
Output: Top-k maximal common subgraphs.
1: Compute weight vector for each edge in q
2: $Ans \leftarrow \emptyset$ // Priority queue of maximum size k
3: $Cands \leftarrow \emptyset$ // Priority Queue to store MBRs
4: **while** $\exists e \in E_q$, $\widehat{CGS}(e, rootNode.MBR) > Ans.leastValue()$ and e has not been explored **do**
5: $e_q \leftarrow$ unexplored query edge with highest similarity to MBR in root node
6: Initialize $Cands$ with the root node of CGQ-tree.
7: **while** $Cands$ is not empty **do**
8: $n \leftarrow Cands.poll()$. // Most similar node (MBR) to e_q
9: $max \leftarrow \widehat{CGS}(e_q, n.MBR)$ // Corollary 3.
10: **if** $max \leq Ans.leastValue()$ **then**
11: break // answer set computation terminates
12: **if** n is a leaf node **then**
13: **while** $\widehat{CGS}(e_q, n.MBR) > Ans.leastValue()$ and n contains unexplored edges **do**
14: $PQ \leftarrow$ Priority Queue containing bw unexplored edges with similar neighborhoods to e_q // Phase 2 (Sec. 3.5.2).
15: **while** PQ is not empty **do**
16: $g \leftarrow PQ.poll()$ // subgraph g that maximizes $\widehat{CGS}(g, q)$
17: **if** $\widehat{CGS}(g, q) \leq Ans.leastValue()$ **then**
18: break // computation ends for this batch of bw edges
19: $\mathbb{G} \leftarrow$ Grow g through all possible 1-edge extensions to form common subgraphs
20: Add all MCSs in $\mathbb{G}$ to Ans and remaining subgraphs in $\mathbb{G}$ to PQ
21: **else**
22: Add all children of n to $Cands$.
23: **return** Ans

THEOREM 3. *The maximum similarity of any MCS M formed by growing g further is $\widehat{CGS}(q, g) = CGS(q, g) + |E_q| - |E_g|$.*

PROOF. The maximum similarity between any two edges is 1. Furthermore, the maximum number of edges in any MCS of q is $|E_q|$. Since at most $|E_q| - |E_g|$ edges can be added to any MCS M formed through g, $CGS(q, M) \leq \widehat{CGS}(q, g)$. □

COROLLARY 2. *Let e_q be a query edge and $\mathcal{H}$ be an MBR on some node in the CGQ-tree. The maximum similarity of any MCS M formed by mapping e_q to some target edge contained with $\mathcal{H}$ is $\widehat{CGS}(e_q, \mathcal{H}) = cs(e_q, \mathcal{H}) + |E_q| - 1$.*

Equipped with the above upper bounds, we prioritize each of the initial 1-edge common subgraphs based on $\widehat{CGS}(q, g)$ (Theorem 3). More specifically, we initialize a third priority queue PQ and insert all 1-edge subgraphs in PQ (line 14). The subgraph g with the highest upper bound is popped from PQ and we check if $\widehat{CGS}(q, g)$ is larger than the k^{th} highest similarity score in Ans (line 17). If yes, then we explore all possible single edge extensions of g to create common subgraphs with one more edge (line 20). If an extended graph of g is an MCS, then we add it to Ans. Otherwise, we insert it back in PQ (line 20). On the other hand, if the check fails, then we are guaranteed that none of the unexplored subgraphs in PQ can lead to a better solution than what we already have and hence Phase 3 completes (lines 7-18).

3.5.4 Tying all the phases together

As noted in Fig. 4, after Phase 3 completes, we go back to Phase 2 if there is a chance of identifying better solutions (line 13). If this check passes we perform Phase 3 again with a new batch of bw edges. Otherwise, we pop the next node from $Cands$ and find the way to the next most promising leaf node, where the iterative cycle of Phase 2 and Phase 3 is repeated. Once processing e_q terminates, we repeat the entire pipeline from Phase 1 to Phase 3 on the next most promising unexplored query edge, provided that this edge can potentially lead to a more similar MCS than the k^{th} most similar MCS till now (top MCS in Ans). This check is performed by applying Corollary 2 on each unexplored query edge (line 4).

3.5.5 Summary

In summary, we alter the generic technique to identify MCSs in two ways. We identify the best initial seeds to start our search from. This enables us to identify a region in target graph that has a high chance of having sub-graphs similar to the query graph. To further optimize the search algorithm, we adopt *beam-stack search*. Under this search scheme, instead of exploring one edge at a time, we simultaneously explore bw (beam width) edges and the related growths. This enables us to reach to a more promising subgraph sooner. The second differentiating factor comes from the application of Theorem 3, which allows us to prune off candidates that are not promising enough.

Correctness Guarantee: The CGQ framework provides the *optimal answer set*. We do not prune off any possibility without applying Theorem 3 or Corollary 2. Consequently, we do not lose out on an MCS that can be in the top-k answer set.

4. EXPERIMENTS

In this section, we benchmark the proposed techniques on real graph datasets and demonstrate their efficacy and efficiency.

4.1 Datasets

We consider a mix of directed and undirected real (large) graphs, as summarized in Table 2.

- The **Northeast (NE)** biodiversity dataset [5, 32] comprises of 1434 spatial points, with each node possessing 5 types of information (features): (i) Bio-diversity richness index, (ii) Disturbance index, (iii) Medicinal property, (iv) Economical property, and (v) Forest type. If two nodes are spatially close, they are connected by an edge. Forest type is a categorical feature possessing 27 unique categories, while all other features are numerical.
- The **DBLP** dataset [2] represents the citation network of research papers published in computer science. Each node is a paper and a directed edge $e = (u, v)$ from u to v denotes paper u being cited by paper v. Each node possesses 5 types of information (features): (i) publication venue, (ii) the set of authors, (iii) the year of publication, (iv) the rank of the venue, and (v) the subject area of the venue. The publication year and venue rank are numerical features while the rest are categorical in nature. The first three features are obtained from the DBLP database while the last two are added from the CORE rankings portal [1].
- The **DBLP (co-author)** dataset [2] represents the co-authorship network of the research papers. Each node is an author of a paper and edges denote collaboration between two authors. Each node possesses 5 features: (i) number of papers published by the author, (ii) the number of years for which the author has been active (calculated as the difference between the latest and the first paper), (iii) the set of venues in which the author has published, (iv) the set of subject area(s) of the venues in which the author has published, and (v) the median rank of the venue (computed as the median of the ranks of all the venues where the author has published). The number of papers published, number of years active, and the median rank are numerical features, while the rest are categorical. Similar to the DBLP dataset, the first three features are from the DBLP database while the last two are from the CORE rankings portal [1].
- The **Pokec** dataset [3] presents data collected from an on-line social network in Slovakia. Each node is a user of the social network and a directed edge $e = (u, v)$ denotes that u has a friend

Dataset	# Nodes	# Edges	Type	Mean Degree	# Features
Northeast	1.4K	15K	Undirected	10.50	5
DBLP	3.2M	6.8M	Directed	2.13	5
DBLP (co-author)	1.8M	7.4M	Undirected	8.44	5
Pokec	1.6M	30.6M	Directed	19.13	6

Table 2: **Summary of datasets used in our experiments.**

v (but not necessarily vice versa). Each node is characterized by six features: (i) gender, (ii) age, (iii) joining date, (iv) timestamp of last login (v) public: a boolean value indicating whether the friendship information of a user is public, and (vi) profile completion percentage. While public and gender are categorical features, the rest are numerical in nature.

4.2 Experimental Setup

All experiments are performed using codes written in Java on an Intel(R) Xeon(R) E5-2609 8-core machine with 2.5 GHz CPU and 512 GB RAM running Linux Ubuntu 14.04.

Query graph: Query graphs are generated by selecting an edge uniformly at random from the target graph and then growing the graph to a desired size. Unless specifically mentioned, the default query graph size is 7 edges. All the reported results were averaged over 30 random query graphs.

Baseline: As discussed in Sec. 1, none of the existing techniques in the literature can be used for either answering context-aware queries or index them. Thus, the baseline technique for us is the naive algorithm discussed in Sec. 3.1.

Parameters: The number of hops (h) used to construct the neighborhood vector of an edge is fixed to 2 (Eq. 4). The decay parameter is set to $\delta = 0.25$. Unless otherwise stated, the branching factor in CGQ-tree is $b = 4$. While building CGQ-tree, we set a node as leaf node if it contains less than 100 edges. Unless specifically, mentioned, the default value of k is set to 10.

4.3 Quality

We initiate the empirical evaluation by analyzing the quality of the results produced by CGQ. To demonstrate the efficacy of our proposed contextual similarity function – *CGS* (defined in Sec. 2.4) – we perform top-10 context-aware graph querying on the DBLP co-author dataset using queries comprising popular authors. In this experiment, each node is restricted to contain three features, namely – *number of papers published*, *median venue rank*, and *subject area*. The first two features are calculated as defined in Sec. 4.1, while the *subject area* corresponds to the area of the venue where the author has published most of her works. Note that, we do not necessarily present the rank-1 matches for the queries. Rather we present the results containing *authors popular in the database community* and thereby being relevant to the usual readers of a database paper. Nevertheless, the top-5 matches of each query are provided in Appendix in Figs. 10-12 for the readers' reference.

The first query presented in Fig. 6a represents a collaboration pattern comprising prolific authors – *Jiawei Han, Philip S. Yu, and Charu C. Aggarwal* – working in the field of *data mining*. Fig. 6b presents the result ranked 2 in the CGQ framework. It is clearly evident that the proposed contextual graph similarity function is able to *capture the context*. More specifically, *subject area* and *median rank* are prevalent in the query graph and provide the context. Consequently, the rank-2 match of query 1 represents a collaboration pattern among prolific authors – *Magdalena Balazinska, Ugur Çetintemel, and Michael Stonebraker* – working in the field of *databases*. As can be seen, all the authors in Fig. 6b possess the *median rank* as "A*" and the *subject area* as "Databases".

To further highlight the importance of context, we compute the rank of the shown result (Fig. 6b) under traditional graph matching. Generally, traditional graph matching quantifies the quality of correspondence between the mapped nodes across two graphs [23,26]. Thus, given an MCS g between a query graph q and a target graph G, the traditional similarity between q and g is calculated as the summation of *min-max* similarity (similar to Eq. 1) between the feature-values of the mapped nodes. For further details on the traditional similarity function please see Appendix B.

As can be seen, there is negligible similarity between features of the nodes in the two graphs. Consequently, regardless of the similarity function used to capture node similarity, these two graphs would always be considered dissimilar. It is therefore not surprising to observe that Fig. 6b has a rank beyond 10000 under the traditional graph matching paradigm

As discussed in Sec 1, the importance of features is dynamic and must be learned from the query itself. The query in Fig. 5a demonstrates this ability of CGQ. More specifically, while the subject area provided context in query 1, in query 2, the context is provided by the high paper counts and Rank of the authors. The result, Fig 5b, is a collaboration pattern among authors from diverse backgrounds, but all prolific and publishing in top venues.

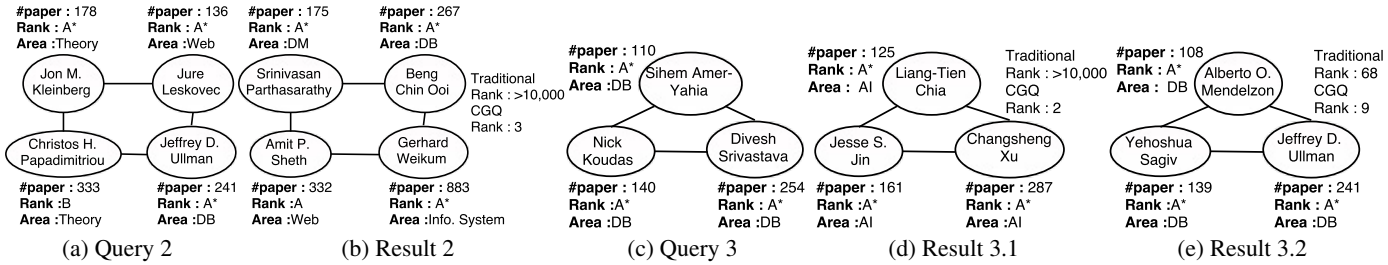
Finally, a third query is shown in Fig. 5c. Similar to the previous two queries, the result, shown in Fig. 5d, is intuitive and meaningful. The second result of this query, shown in Fig. 5e unearths a different property of the contextual similarity function: *backward compatibility*. More specifically, this is one of the rare results, where traditional matching provides a relatively high rank of 68. This high rank is not a consequence of traditional matching capturing the context, rather a coincidence where the node feature vectors between query 3 and result 3.2 happen to be similar (*Sihem-Alberto, Nick-Yehoshua, Divesh-Jeffrey*). Note that when the mapped nodes across two graphs are similar, from the formulation of relationship vector, it is guaranteed that the mapped edges would also be similar. Consequently, contextual similarity is backward compatible with traditional similarity. However, the same cannot be said about traditional similarity.

As already observed, traditional matching, ranks majority (Figs. 6b, 5b, and 5d) of the shown results beyond 10000. This behavior is not a coincidence. It is *practically impossible* for traditional (context-oblivious) similarity metrics to retrieve matches at ranks similar or even close to that produced by the contextual graph similarity. Our next experiment substantiates this further by quantifying this uniqueness of *CGQ*. We perform a top-5 query on random query graphs with sizes varying from 3 to 10. For each of the retrieved CGQ results we find its rank using the traditional similarity function. Next, we plot the average traditional rank for each of the query graph sizes. Since it is hard to scale beyond top-10000 in traditional similarity function, if a CGQ result does not appear within top-10000 of traditional search, we set its rank to 10000. The result of this experiment is presented in Fig. 7a. It is clearly evident that the traditional rank of the top-5 contextually similar matches is greater than $10,000$ on average. This highlights the utility of CGQ in surfacing useful results that would otherwise remain hidden. Consequently, CGQ provides a significant step-forward in the domain of graph querying and is the key contribution of our work.

4.4 Efficiency and Scalability

In this section, we evaluate the efficiency and scalability of CGQ against various data input parameters and properties.

First, we compare the performance of CGQ index with the baseline technique (Sec. 3.1). We perform top-10 queries on the Northeast dataset using random query graphs of sizes varying from 3 to 12 edges. The result is shown in Fig. 7b. As is evident, CGQ is up to 9 times faster for query graphs of size till 5 and more than 1000 times faster for larger sizes. The poor performance of base-



(a) Query 2 (b) Result 2 (c) Query 3 (d) Result 3.1 (e) Result 3.2

Figure 5: **Results of top-k querying in CGQ for the shown query graphs.**

line technique is because of the complex nature of our problem. The problem by itself is NP-hard, and in addition, since we are looking for maximal common subgraphs, we cannot employ any sophisticated search scheme involving the structure of graph. In addition, the non-importance of labels further increases the search space. CGQ is able to overcome this bottleneck by exploiting the signals hidden in the feature space and weight vectors, using which it identifies regions in the target graph that contain the best matches to the query. Hence, large portions of the search space is pruned off. An alternative view of the same result in terms of speed-up against query graph size is provided in the Appendix in Fig. 9b

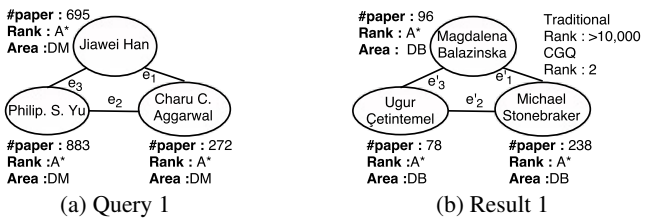
Fig. 7c demonstrates the growth of querying time against query graph size. As expected, the growth of search time is exponential in nature. This is natural since the search space grows exponentially with query graph size. Nonetheless, CGQ successfully mitigates the effects of exponential growth through effective pruning mechanisms, which limits the running time to a maximum of only 80ms.

Notice that CGQ is fastest in Pokec even though Pokec is the largest network containing more than 30 million edges. The reason behind this behavior is that the variance in the features in Pokec is low. Consequently, the chance that a randomly picked subgraph is identical to the query is larger. Owing to this property, the search for the query graph converges quickly. On the other hand, the DBLP datasets are more heterogeneous and therefore provide a tougher challenge. Overall, these results establish beyond any doubt that CGQ is a fast search mechanism. In this three datasets, the baseline technique is unable to scale beyond query graph size of 5. The comparison with baseline for smaller query sizes is provided in Fig. 9a in Appendix. Even for smaller query sizes, CGQ is more than three orders of magnitude faster.

The second most important factor influencing the querying time is the target graph size. We study its effect on the two largest datasets: DBLP co-author and Pokec. Figs. 7d-7e present the results. For this experiment, we extract four different subgraphs of the entire network covering 25%, 50%, 75% and 100% of the nodes (and all edges between them) of the entire network. To pick a subgraph covering $X\%$ of all nodes in the entire network, we start from an arbitrary node, and perform a random walk till at least $X\%$ of the nodes have been visited at least once. On each of the subgraphs, we plot the growth rate of querying time across various query graph sizes. As anticipated, the querying time increases with increase in target graph size. This is natural since the search space increases exponentially with increase in target graph size. Notice that the growth rate is higher in Pokec than in DBLP co-author dataset. Since Pokec is much more dense, the search space increases more drastically with addition of nodes in the target graph. Figs. 7f-7g further drills into the impact of target graph size and studies its impact as a function of query graph size. Here, we observe a more linear growth rate similar to the results of Fig. 7c.

The final factor influencing querying time is k. Fig. 7h presents the growth of querying time across various values of k. As expected, the querying time increases with increase in k. What is more interesting to observe however is that the querying time flattens out very rapidly. The relatively flat growth rate of the querying time stems from the fact the number of subgraphs explored till the answer set converges remains relatively same even for larger values of k. Hence, the flat running time.

After a thorough investigation of the querying time, we shift our focus to offline indexing costs. As a first step, we plot the time taken to construct the CGQ-Tree index structure in Figs. 7i-7j and even on Pokec, it completes within 11 minutes. Note that index construction is a one-time offline activity and thus not a critical aspect. However, the memory foot print of the index structure is a relatively more important aspect. In Sec 3.4.1 we proved that CGQ-Tree has a linear space complexity with respect to the number of edges in the target graph size. This result is reflected in Fig. 8a where we calculate the memory footprint against the number of edges in the target graph. To extract a subgraph of desired size from Pokec, we use the same random walk based method employed earlier in Figs 7f-7g. The linear growth rate of memory footprint is a highly desirable characteristic, which allows CGQ to be applied on million-scaled datasets. As can be seen, even for a graph having 30 Million edges, we consume less than $3GB$ of memory. Overall, these results establish that CGQ is not only scalable in querying time, but also in its index construction costs.



(a) Query 1 (b) Result 1

Figure 6: **Result of top-k query in CGQ for the shown query graph.**

4.5 Optimization

The CGQ framework has three major components: *CGQ-Tree* for similar edge search, *weight-vectors* for identifying similar neighborhoods and the *beam-stack search* during query time. Each of these components are not required for the correctness of the search algorithm but contribute immensely to the efficiency of the operation. In this section we demonstrate the importance of each of these components in performing an efficient search. In addition, we look into the effects of the internal parameters used by CGQ.

To demonstrate the importance of CGQ-tree, we compare the time taken for top-k search with CGQ-tree and without the tree. Since the running times for top-k queries without CGQ-tree on DBLP datasets and Pokec are prohibitively large, we restrict this experiment to the Northeast dataset. We generate random query graphs of sizes varying from 3 to 12 and perform top-10 query on them using both the search techniques. The results are shown in Fig. 8b. The use of CGQ-tree provides up to 74 times speedup over

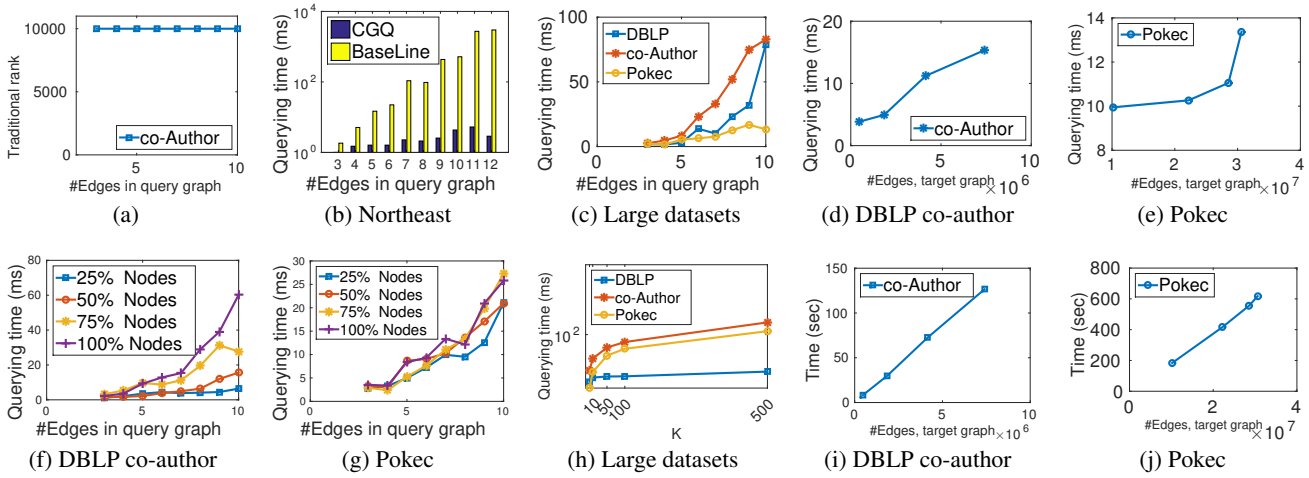


Figure 7: **(a) Comparison of traditional graph search with the proposed contextual similarity function. (b) Comparison of querying time between CGQ and the baseline technique. (c) Growth of querying time against query graph size and (d-e) target graph size. (f-g) Impact of target graph size on queries of various sizes. (h) Growth of querying time against k. (i-j) Growth of index construction time against network size.**

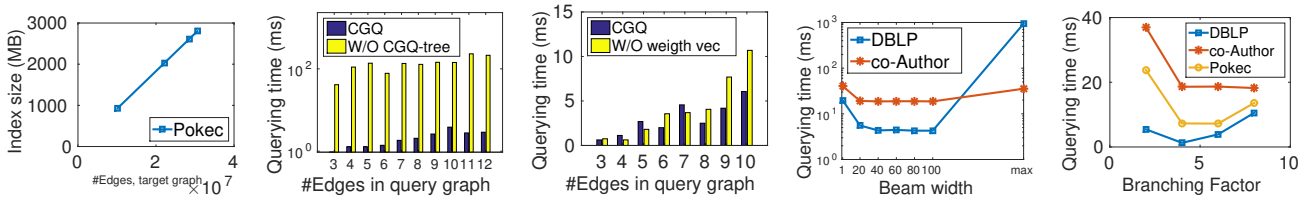


Figure 8: **(a) Growth of CGQ's memory footprint against network size. Impact of (b) CGQ-tree, (c) neighborhood search through weight vectors, (d) beam width, and (e) branching factor of CGQ-tree on querying time.**

a search without CGQ-tree, thus establishing its importance.

Next, we study the importance of weight vectors in identifying similar neighborhoods. Using the same design of the previous experiment, we compare the time taken for top-k search using two techniques, CGQ and CGQ without neighborhood-based prioritization. Fig. 8c presents the results in the DBLP co-author dataset. We observe that for smaller query graphs, CGQ without neighborhood optimization performs better. However, for larger query graphs neighborhood optimization helps. This is due to the fact that for smaller query graphs, the neighborhood of a query edge itself is very small and thus they lack much discriminative information. When the query size grows, the neighborhood of an edge is larger and can therefore be used to discriminate between various regions of the target graph and prune the search space more effectively.

Beam-width controls the number of simultaneous subgraphs we grow to form maximal common subgraphs. A beam-width of one would make the search work like a depth-first search. On the other hand, an infinite beam width would degenerate to best-first search. To study the effect of this parameter, we run top-10 query with beam-width varying from 1 to ∞. The results of the experiment are in Fig. 8d. Generally, a beam width between 40 and 60 provides the highest speed-up. In comparison to a beam width of 1, for DBLP dataset beam search performs up to 4 times faster and for DBLP co-author dataset, beam-search is up to 2 times faster. This observation follows from the fact that in depth-first search we have no chance for retracting from an ineffective search. On the contrary, beam-search provides us enough room to explore the more promising results. The benefits of multiple search is negated in the case of ∞ beam-width because our algorithm now spends more time switching between the best results, which is analogous to the case of thrashing in operating systems.

Finally, we analyze the impact of *branching factor* of CGQ-tree on the querying time performance. The results of the experiment are in Fig. 8e. Across all three datasets, the best performance is observed at a branching factor of 4. All datasets show relatively higher querying times for a branching factor of 2. This could be attributed to an increase in the height of the tree. In addition at lower values, each node contains more edges, and as a result the similarity upper bounds are loose. Owing to these reasons, the querying time initially decreases with an increase in the branching factor. Eventually, the querying time either stabilizes or starts to increase. This is attributed to the fact that the node size reduces with the increase in branching factor. Since our beam-stack search is largely restricted to each node, we eventually loose out its benefits.

5. RELATED WORK

The general problem of graph querying has been studied extensively over the past decade. Here, we overview the existing works that overlap with our problem.

Traditional Similarity Measures: Research work done in both exact and approximate (sub-)graph querying have employed the use of a plethora of similarity measures. The most prominent of them being graph edit distance [47, 48], maximum and minimum common subgraph [9, 18], edge misses [37], structural similarity [35, 36, 38], node-label mismatches [35, 36]. However, all of these distance methods operate oblivious to the presence of *context in the query graph*, thereby ignoring its impact on the eventual query-target graph similarity computation.

Information Propagation based Similarity Measures: Information propagation is a popular approach for capturing the interac-

tions represented in the h-hop neighborhoodneighborhood of each node. This concept significantly increases the expressive power of the similarity functions so designed, and has been used by NESS [25], NeMa [26] and DeltaCon [16]. However, despite capturing the neighborhood in a better way, all these techniques lack the ability to *adapt* the distance function based on the *query context*.

Relationship-aware Querying: Relationship-queries [44] are typically used in knowledge bases, where the query is a relationship and the output are linked entities satisfying that relationship. For example, a sample query would be "South-American football (soccer) players playing in a Spanish football club". One possible answer to this query is the tuple Messi and Barcelona F.C. This problem is much different from ours. First, the query in our problem is a graph and not a relationship. Second, we need to automatically identify and rank the relationships encoded in the query. Finally, we need to devise a similarity function that incorporates relationships and context.

Context-Aware Querying: As discussed in Section 1, the importance of *query context* has been widely studied and appreciated in the domain of text databases facilitating web-searches. With wide-spread research in the use of context for effective querying [14, 15, 42, 43], query suggestions and auto-completion [6, 10] and recommendations [46], this concept has become quotidian for web search. However, its use has been surprisingly absent in the field of (sub-)graph querying. Although [12] claims to employ the use of context for object connection discovery in large graphs, this claim is not completely true as the idea is to just to use the concept of *modularity* to create communities, which are eventually used as the context of a node. Thus, [12] just captures the structural/topological context and is clearly not extensible to employ the use of node labels present in labeled graphs to understand the context. Moreover, the problem of object connection discovery is simpler when compared to that of (sub-)graph querying. In sum, this work (*CGQ*) presents the first-ever effort in bringing the power of context to facilitate (sub)-graph querying in large networks.

Note that although literature has witnessed a plethora of indexing techniques for (sub-)graph querying [17, 19, 25, 26, 28, 29, 34–37, 39, 40, 50], we do not compare against any of these methods owing to the reasons that are three fold. First, the notion of context has never been studied in the broad domain of graph querying, and thus, the requirements for index-structures to be flexible and adapt to dynamically changing distance functions with queries has not been met by the existing techniques. Secondly, the index-structures are usually designed keeping a certain requirement of the problem formulation or similarity function in mind, i.e. incorporating capabilities for – structural matching, label matching etc.; these methods are therefore fine-tuned according to a specific use-case(s) and hence are inherently inextensible to newer similarity functions. Lastly, while one may argue that a host of state-of-the-art methods for searching subgraph isomorphic embeddings of a query graph in a target graph [27] (and the references therein) can be easily extended to new similarity functions and should be used as a baseline, these methods are not capable of finding maximal common subgraphs, which is central to our problem formulation.

6. CONCLUSIONS

In this paper, we addressed the problem of graph querying in real-world networks under a novel *relationship-aware contextual* paradigm, where we capture the relationships and learn the context prevalent in the query graph. Since majority of the works in the literature operate oblivious to the existence of relationships and context in the query graph, there exist limitations in graph querying in real-world scenarios. Consequently, we bridged this gap by introducing a novel *CGQ* (Relationship-aware Contextual Graph Querying) framework; by coming up with a novel similarity function and a subgraph matching technique which is context-aware. The CGQ framework employs a similarity function to capture the relationships and context prevalent in the query graph. Moreover, to address the computational challenges posed by graph querying we designed a flexible and hierarchical CGQ-Index and corresponding efficient querying algorithms. Empirical studies on real-world graph datasets showed that our algorithms are *effective*, *efficient* and *scalable*, thereby providing a speed-up of up to *three orders* of magnitude over a baseline strategy. In future, we would like to explore more detailed models of context.

7. REFERENCES

[1] Core rankings portal. http://portal.core.edu.au/conf-ranks/.

[2] Dblp: Computer science bibliography. http://dblp.uni-trier.de/.

[3] SNAP Datasets. https://snap.stanford.edu/data/.

[4] Subgraph isomorphism and related problems. http://www.cs.upc.edu/~valiente/graph-00-01-a.pdf.

[5] A. Arora, M. Sachan, and A. Bhattacharya. Mining Statistically Significant Connected Subgraphs in Vertex Labeled Graphs. In *SIGMOD*, pages 1003–1014, 2014.

[6] Z. Bar-Yossef and N. Kraus. Context-sensitive query auto-completion. In *WWW*, pages 107–116, 2011.

[7] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[8] M. Biba, F. Esposito, S. Ferilli, N. Di Mauro, and T. M. A. Basile. Unsupervised discretization using kernel density estimation. In *IJCAI*, pages 696–701, 2007.

[9] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3):255–259, 1998.

[10] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD*, pages 875–883, 2008.

[11] V. Chaoji, S. Ranu, R. Rastogi, and R. Bhatt. Recommendations to boost content spread in social networks. In *WWW*, pages 529–538, 2012.

[12] J. Cheng, Y. Ke, W. Ng, and J. X. Yu. Context-aware object connection discovery in large graphs. In *ICDE*, pages 856–867, 2009.

[13] A. Denise, M. Régnier, and M. Vandenbogaert. Assessing the statistical significance of overrepresented oligonucleotides. In *WABI*, pages 85–97, 2001.

[14] L. R. A. Derczynski, B. Yang, and C. S. Jensen. Towards context-aware search and analysis on social media data. In *EDBT*, pages 137–142, 2013.

[15] J. Etzold, A. Brousseau, P. Grimm, and T. Steiner. Context-aware querying for multimodal search engines. In *ICMM*, pages 728–739, 2012.

[16] C. Faloutsos, D. Koutra, and J. T. Vogelstein. DELTACON: A principled massive-graph similarity function. In *SDM*, pages 162–170, 2013.

[17] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. In *VLDB*, pages 264–275, 2010.

[18] M.-L. Fernández and G. Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*,

22(6):753–758, 2001.
[19] B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAIFS*, 2006.
[20] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.
[21] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.
[22] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.
[23] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, 2006.
[24] V. Kann. On the approximability of the maximum common subgraph problem. In *STACS*, pages 375–388, 1992.
[25] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *SIGMOD*, pages 901–912, 2011.
[26] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. Nema: Fast graph search with label similarity. *PVLDB*, 6(3):181–192, 2013.
[27] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. In *PVLDB*, pages 133–144, 2013.
[28] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo. Capturing topology in graph pattern matching. In *VLDB*, pages 310–321, 2012.
[29] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *PAMI*, 26(10):1367–1372, 2004.
[30] S. Ranu, M. Hoang, and A. K. Singh. Mining discriminative subgraphs from global-state networks. In *KDD*, pages 509–517, 2013.
[31] T. R. C. Read and N. A. C. Cressie. *Goodness-of-fit Statistics for Discrete Multivariate Data*. Springer, 1988.
[32] P. S. Roy and S. Tomar. Biodiversity characterization at landscape level using geospatial modelling technique. *Biological Conservation*, 95(1):95–109, 2000.
[33] M. Sachan and A. Bhattacharya. Mining statistically significant substrings using the chi-square statistic. In *PVLDB*, pages 1052–1063, 2012.
[34] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. In *PVLDB*, pages 364–375, 2008.
[35] R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. In *PNAS*, pages 12763–12768, 2008.
[36] Y. Tian, R. C. Mceachin, C. Santos, D. J. States, and J. M. Patel. Saga: A subgraph matching tool for biological graphs. *Bioinf.*, 23(2):232–239, 2007.
[37] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. In *ICDE*, pages 963–972, 2008.
[38] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, pages 737–746, 2007.
[39] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, pages 737–746, 2007.
[40] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.
[41] O. Verbitsky. On the largest common subgraph problem. Technical report, Ulm University, 1995.
[42] R. W. White, P. N. Bennett, and S. T. Dumais. Predicting short-term interests using activity-based search context. In *CIKM*, pages 1009–1018, 2010.
[43] B. Xiang, D. Jiang, J. Pei, X. Sun, E. Chen, and H. Li. Context-aware ranking in web search. In *SIGIR*, pages 451–458, 2010.
[44] M. Yahya, D. Barbosa, K. Berberich, Q. Wang, and G. Weikum. Relationship queries on extended knowledge graphs. In *WSDM*, pages 605–614, 2016.
[45] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, 2004.
[46] W. Yao, J. He, G. Huang, J. Cao, and Y. Zhang. A Graph-based Model for Context-aware Recommendation using Implicit Feedback Data. *WWW*, 18(5):1351–1371, 2015.
[47] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. In *PVLDB*, pages 25–36, 2009.
[48] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao. Graph similarity search with edit distance constraint in large graph databases. In *CIKM*, pages 1595–1600, 2013.
[49] R. Zhou and E. A. Hansen. Beam-stack search: Integrating backtracking with beam search. In *ICAPS*, pages 90–98, 2005.
[50] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao. gstore: Answering sparql queries via subgraph matching. In *PVLDB*, pages 482–493, 2011.

APPENDIX

A Range Query

Besides top-k query another important search operation is the *range query*. Given a query graph $q(V_q, E_q)$ and a similarity threshold r, the objective of the range query is to find all maximal common subgraphs of q and the target graph G, possessing a similarity value of at least r with q. It is well known that range query is a simpler problem when compared to top-k query. The most promising reason for that being a fixed pruning criterion r in the former. With each identified match the similarity threshold has to change for a top-k query (line 4 in Alg. 2), however, it remains constant for a range query.

To this end, we alter the CGQ search algorithm to perform range queries by using r as a pruning criterion, instead of $Ans.leastValue()$ used in the top-k version. Since, simply substituting $Ans.leastValue()$ in line 4 of Alg. 2 by r gives us the range search algorithm, we omit its pseudo-code.

B Traditional Similarity

DEFINITION 9 (TRADITIONAL NODE SIMILARITY). *The traditional similarity between two nodes u and v having d features is defined as,*

$$ts(u,v) = \frac{1}{d}\sum_{i=1}^{d} s_i(u,v) \qquad \textit{where} \tag{13}$$

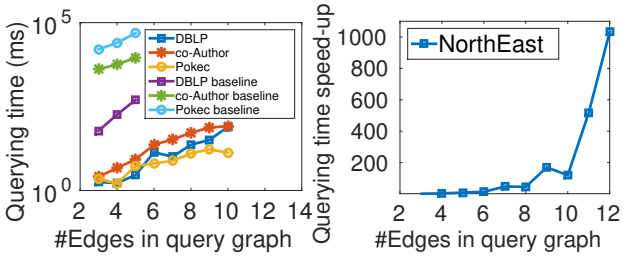
$$s_i(u,v) = \begin{cases} \frac{\min\{f_i(u),f_i(v)\}}{\max\{f_i(u),f_i(v)\}} & \textit{if} \quad f_i \textit{ is real valued} \\ 1 & \textit{if} \quad f_i \textit{ is categorical and } f_i(u) = f_i(v) \\ 0 & \textit{if} \quad f_i \textit{ is categorical and } f_i(u) \neq f_i(v) \end{cases} \tag{14}$$

DEFINITION 10 (TRADITIONAL GRAPH SIMILARITY). *For a subgraph $g \subseteq G$, which is an MCS of target graph G and query q, let $\phi : V_g \rightarrow V_q$ be the injection from the vertices of g to q. The traditional similarity between g and q is defined as:*

$$TS(g,q) = \sum_{\forall u \in E_g} ts(u, \phi(u)) + \sum_{\forall e_g=(u,v) \in E_g} I_{\phi(e_g) \in E_g} \quad (15)$$

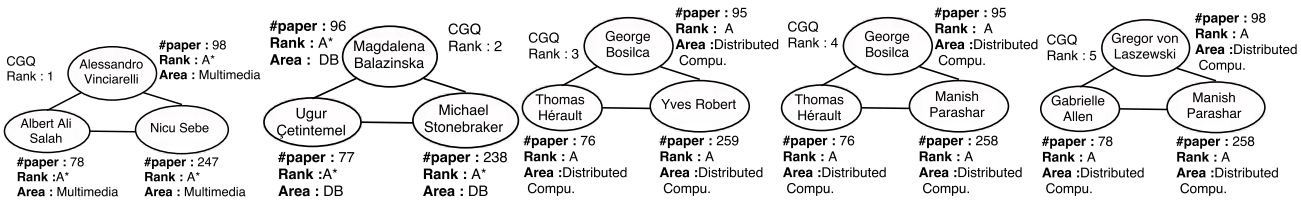
where I is the indicator function: $I = 1$ if $\phi(e_g) \in E_g$ and 0 otherwise; and $\phi(e_g) = (\phi(u), \phi(v))$.

To summarize, the traditional similarity measure captures both: node-level similarity using the feature-values and structural similarity using the number of edge matches. It can be observed that, the maximum similarity a graph g can have to any other graph is $|E_g| + |V_g|$.



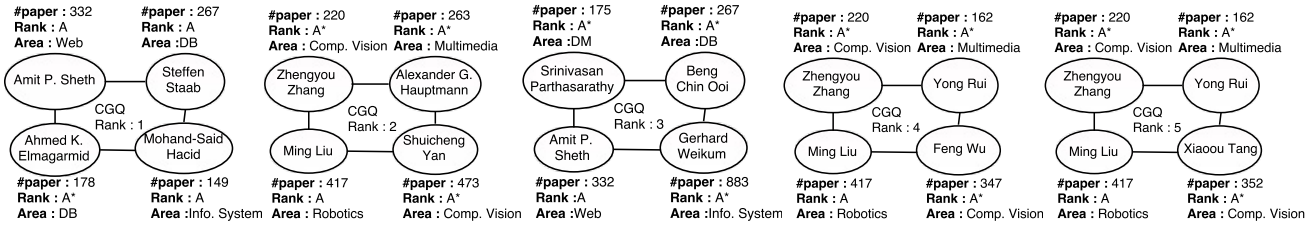
(a) Large datasets (b) Northeast

Figure 9: **(a) Comparison of querying time for the baseline algorithm against CGQ framework. Note that we were unable to scale baseline beyond query graph size of 5. (b) Speed-up achieved by CGQ in the Northeast dataset against query graph size.**



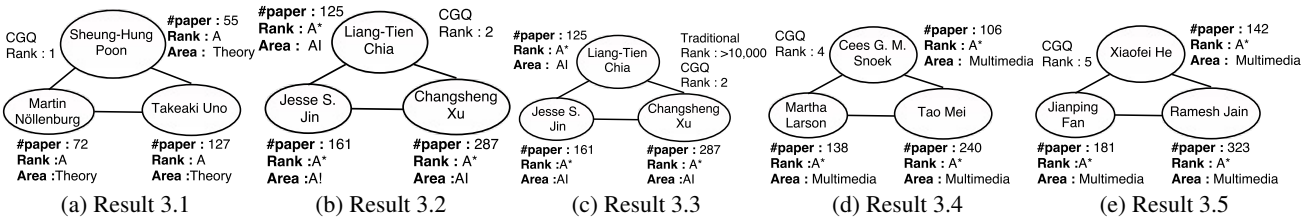
(a) Result 1.1 (b) Result 1.2 (c) Result 1.3 (d) Result 1.4 (e) Result 1.5

Figure 10: Top-5 results for Query 1 in Sec. 4.3



(a) Result 2.1 (b) Result 2.2 (c) Result 2.3 (d) Result 2.4 (e) Result 2.5

Figure 11: Top-5 results for Query 2 in Sec. 4.3



(a) Result 3.1 (b) Result 3.2 (c) Result 3.3 (d) Result 3.4 (e) Result 3.5

Figure 12: Top-5 results for Query 3 in Sec. 4.3