GFocus: User Focus-based Graph Query Autocompletion

Peipei Yi, Byron Choi, Zhiwei Zhang, Sourav S Bhowmick, and Jianliang Xu

Abstract—Graph query autocompletion (GQAC) generates a small list of ranked query suggestions during the graph query formulation process in a visual environment. The current state-of-the-art of GQAC provides suggestions that are formed by adding subgraph increments to arbitrary places of an existing (partial) user query. However, according to the research results on human-computer interaction (HCI), humans can only interact with a small number of recent software artifacts in hand. Hence, many of such suggestions could be irrelevant. In this paper, we present the GFocus framework that exploits a novel notion of *user focus of graph query formulation* (or simply *focus*). Intuitively, the focus is the subgraph that a user is working on. We formulate *locality principles* inspired by the HCI research to automatically identify and maintain the focus. We propose novel monotone submodular ranking functions for generating *popular* and *comprehensive* query suggestions only at the focus. In particular, the query suggestions of GFocus have high result counts (when they are used as queries) and maximally cover the possible suggestions at the focus. We propose efficient algorithms and an index for ranking the suggestions. Our results show that GFocus saves 12%-32% more mouse clicks and is 35× more efficient than the state-of-the-art competitor.

Index Terms—Subgraph query, Query autocompletion, Database usability.

1 Introduction

Query formulation, typically, using a graph query language, is the first key step of querying graph data. Several graph query languages (*e.g.*, SPARQL, and Cypher) have been proposed to realize the query formulation step. While most of these languages can express a wide variety of graph queries, their syntaxes can be too complex to be used by ordinary users. A popular approach toward making query formulation tasks palatable to end users is to build visual graph query interfaces (a.k.a GUIs) that facilitate the drawing of query graphs in an easy and intuitive manner. In fact, several such visual graph query interfaces are already offered by the industry to facilitate query construction (*e.g.*, PUBCHEM[1], CHEMSPIDER[2], and SCAFFOLD HUNTER[3]). This reflects a real demand for visual graph query interface.

Despite these efforts, composing graph queries in a visual environment may still be cumbersome. In particular, during query formulation, a user needs to precisely draw the (edge) relationships between nodes in a query graph to construct the topological structure he/she is interested in. Due to the topological complexity of the underlying graph data, it is often unrealistic to assume that an end user can easily specify such relationships precisely.

This paper studies the problem of *graph query autocompletion* (GQAC) to alleviate the burden of visual query formulation. We start with a typical scenario that a user intends to formulate a query graph q iteratively using a visual query interface. *Given an existing (partial) visual query graph q_0, the aim of* GQAC *is to suggest a subgraph increment Δq to q_0, such that adding Δq to q_0 yields a query suggestion toward q.*[4] Since it is hard to accurately predict the individual query graph a user intends to draw, GQAC may return a ranked list of query suggestions for users to choose from.

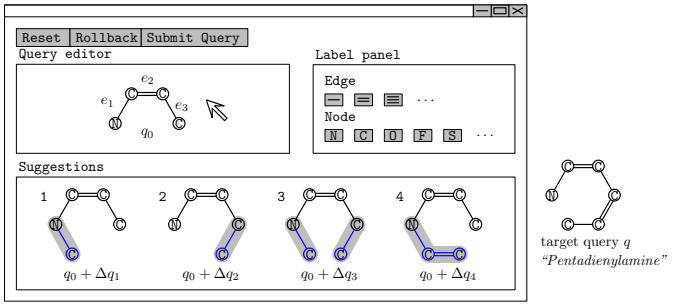


Fig. 1. Left: A typical GUI and suggestions of GQAC; and Right: the target query of Example 1.1

Example 1.1. Consider the visual interface in Fig. 1 for querying a chemical compound database. The query formalism of the query is subgraph isomorphism. Suppose Mike wishes to search for compounds containing the *"Pentadienylamine"*[5] substructure (as shown in Fig. 1). The partial subgraph query constructed by him is depicted in the `Query editor` panel. It is helpful to Mike if the query system can suggest top-k possible query fragments (subgraphs) that he may add to his query. An example of such top-4 suggestions (*i.e.*, $q_0 + \Delta q_1$ to

- *P. Yi, B. Choi, Z. Zhang and J. Xu are with the Department of Computer Science, Hong Kong Baptist University, Hong Kong. P. Yi is a data scientist at Lenovo Machine Intelligence Center, Hong Kong.*
  E-mail: {csppyi, bchoi, cszwzhang, xujl}@comp.hkbu.edu.hk
- *S. S. Bhowmick is with School of Computer Science and Engineering, Nanyang Technological University, Singapore.*
  E-mail: assourav@ntu.edu.sg

Manuscript received MM DD, YYYY; revised MM DD, YYYY.

1. https://pubchem.ncbi.nlm.nih.gov/search/
2. http://www.chemspider.com/
3. http://scaffoldhunter.sourceforge.net/
4. We consider a "query suggestion toward q" to be a subgraph of q and supergraph of q_0. By adopting the suggestion, the user adds fewer nodes/edges by manual formulation to formulate q.
5. https://pubchem.ncbi.nlm.nih.gov/compound/59750537

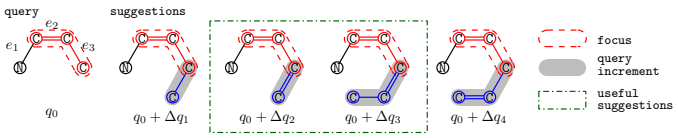


Fig. 2. Suggestions (example) of GFOCUS

$q_0 + \Delta q_4$) is shown in the `Suggestions` panel. Observe that each suggestion is composed by adding small increments (highlighted in blue with gray background) to the query graph. Mike may select a useful suggestion (if present) by clicking on it, thus saving his mouse clicks to formulate the new nodes and edges manually. He may then continue formulating the final query graph in subsequent steps by leveraging the query suggestion capability iteratively. However, none of the suggestions in Fig. 1 can be further extended to the target query. By using GFOCUS, Mike obtains some useful suggestions, as shown in Fig. 2. □

Despite the increasing research efforts on visual graph querying in recent years to enhance both usability and performance of graph databases [1], graph query autocompletion has received relatively little attention in the literature. To the best of our knowledge, there are only two related studies on GQAC [2], [3]. VIIQ [2] demonstrates edge increments to the user's existing query, whereas AUTOG [3] returns the top-k subgraph increments that yield query suggestions with high selectivities and structural diversity.

Nevertheless, these studies suffer from the following drawbacks. First, the returned query suggestions may be too diversified, *i.e.*, they are extended from the existing query graph at *seemingly arbitrary subgraphs*, which may lead to a *cognitive overload* to users. Following up Mike's example, users typically focus their attention[6] on a specific subgraph of the query at each query formulation operation due to the limited short-term/working memory [5], [6]. Suppose Mike formulates q_0 in a clockwise way, as shown in Fig. 1 (*i.e.*, first add e_1, then e_2 and e_3), he may continue to add edges connecting with e_3 or e_2 but not e_1. In Fig. 2, the left-hand side shows the current query q_0, the red colored portion of q_0 (surrounded by the dashed line) is the more recently added edges and hence is more likely at Mike's focus. Note that the increments of all the four suggestions showed in Fig. 2 are now connected to the red portion *only*. Mike obtains two useful suggestions, which connect a carbon node (`C`) to the red portion via an edge of the *double* chemical bond, *i.e.*, `C=C`, in the increments.

Second, a "user focus-unaware" strategy, adopted by existing GQAC techniques, may adversely impact not only the quality of the suggestions but also the response time of GQAC. In Example 1.1, since the user pays little attention to e_1 at the current step, three suggestions in Fig. 1 are deemed to be irrelevant.[7] Additionally, since these existing techniques search for useful suggestions from a large number of candidate suggestions, they may take a long time. This makes users wait for suggestions to appear, disrupting their flow of query construction.

In this paper, we address the drawbacks above. We propose GFOCUS, a *user focus-based graph query autocompletion* framework. Intuitively, *user focus* (or simply *focus*) refers to a small subgraph of a query graph that has the highest user attention, from where GQAC generates query suggestions.[8] While there has not been an agreed-upon definition of attention, it has been generally understood as being *selective* and *reactive* and having a limited capacity (*e.g.*, [4], [9]). While users may explicitly mark their focus, this approach requires extra human intervention [10]. Instead, we propose to *automatically* derive the focus from users' query formulation operations. That is, the computation of user focus is transparent to users.

The main ideas are to propose user's attention weight for each query edge and to maintain the weights of a query graph by proposing two locality principles. Specifically, we define a set of operators to capture users' actions for query formulation, *e.g.*, adding edges and adopting suggestions. We introduce *user attention weights* to edges/subgraphs that involve these operators. The weights are maintained by the following locality principles. We propose the *temporal locality* of query formulation to capture human's memory decay as time passes [11]. The human's memory decay is modeled by a parameter τ. We then formulate the *structural locality* to capture the attention propagation to limited neighboring structures around the attended edges. The *user focus* is then the (proper) connected subgraph of the query with the maximal normalized weight. Next, we propose an algorithm to automatically identify the user focus.

To address the limitation related to both suggestion quality and efficiency, we propose new techniques for generating suggestions at the focus. In particular, we propose a *structural union* of suggestions to compactly represent all possible candidate suggestions and then generate suggestions that maximally cover the union. We remark that it is infeasible to compute suggestion coverage without the user focus as the number of candidate suggestions can be huge. The previous work [3] resorts to computing suggestion diversity only. We propose new linear submodular ranking functions of suggestions that involve not only *query suggestions' selectivities* but also *the coverage of the union*. It is not surprising that the ranking problem is NP-HARD. We propose a greedy algorithm and an index for structural unions to efficiently compute the top-k suggestions.

We observed from user studies and extensive simulations on popular datasets that GFOCUS shows significant improvements of suggestion quality over the current state-of-the-art (*i.e.*, AUTOG). Moreover, GFOCUS almost always generates suggestions within a second. Such efficiency has not been achieved before. In all, this paper makes the following contributions.

- We formalize *query formulation sequence* that users carry out in constructing their queries visually. We define the user attention weight of each edge of the query and locality principles of query formulation. Based on these, for the first time in the context of GQAC, we formally

6. Readers interested in visual attention may refer to a nice chapter by Chun and Wolfe [4].

7. Users may not even "see" the suggestions if they appear in the portion users are not paying attention to. Such a phenomenon has been called inattentional blindness or amnesia [7], [8].

8. Humans may shift their focus at arbitrary times. Modeling such arbitrary shifts is beyond the scope of this paper.

TABLE 1
Frequently used notations

Symbol	Meaning
q	the current query or existing query
q'	a query suggestion or simply suggestion
Δq	query increment (adding Δq to q yields q')
Q'	query suggestions
$q \subseteq_\lambda q'$	q is a subgraph of q' and λ is the embedding of q in q'
f	the user focus, a (proper) connected subgraph of the query q

introduce the concept of user focus. We present a PTIME algorithm to determine user focus.

- We propose *structural union* of suggestions. It is compact and yet contains all the suggestion structures. It can also be efficiently computed.
- We propose new *ranking functions* for generating suggestions at the focus, where efficient ranking algorithms are possible. To optimize the online ranking, we propose the *Structural-Union-of-Suggestions* DAG index (SUDAG) to support the covering semantics for suggestion ranking.
- We investigate the usefulness and efficiency of GFOCUS by an exhaustive experimental study including user studies and simulations. The results show that GFOCUS saves 12%-32% more mouse clicks and speeds up the suggestion process by 35X on average when compared to the state-of-the-art competitor (*i.e.*, AUTOG).

2 BACKGROUND ON GRAPH QUERY AUTOCOMPLETION (GQAC)

We first provide some background on graph databases and visual graph query formulation that are necessary to describe graph query autocompletion. We summarize common user behaviors of query formulation in visual environments. Frequently used notations are listed in Table 1.

Graph databases. We consider a graph database D as a large set of data graphs $\{g_1, g_2, \ldots, g_n\}$. Each graph is a 3-ary tuple $g = (V, E, l)$, where V and E are the vertex and edge sets of g, respectively, and l is the label function of g. For nodes/edges that contain attribute values, we use labels to denote those values. The size of a graph is defined by $|E|$. In this paper, we illustrate the techniques with a large collection of data graphs of modest sizes (*e.g.*, chemical compounds). The query formalism adopted by this paper is subgraph isomorphism. Intuitively, the query graph q retrieves the data graphs that contain q as a subgraph. The definition of subgraph isomorphism is recalled below.

Definition 2.1. [Subgraph isomorphism] Given two graphs $g = (V, E, l)$ and $g' = (V', E', l')$, g is a *subgraph* of g', denoted as $g \subseteq_\lambda g'$, *iff* there is an injective (or *embedding*) function $\lambda : V \mapsto V'$ such that

1) $\forall u \in V$, $\exists \lambda(u) \in V'$ such that $l(u) = l'(\lambda(u))$; and
2) $\forall (u,v) \in E$, $\exists (\lambda(u), \lambda(v)) \in E'$ and $l(u,v) = l'(\lambda(u), \lambda(v))$. □

Multiple subgraph isomorphic embeddings of g may exist in g', denoted as $\lambda^0_{g,g'}$, $\lambda^1_{g,g'}$, $\ldots$, $\lambda^m_{g,g'}$. For succinct presentation, we often refer each $\lambda^i_{g,g'}$ to as an *embedding* λ, when the subscripts and superscripts are clear from or irrelevant to the context. The embeddings specify the locations of subgraphs in a query and GQAC requires them to analyze how suggestions can be composed.

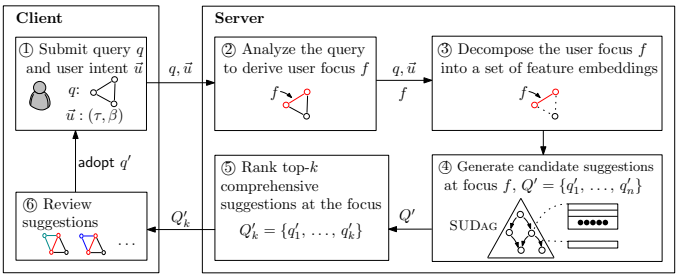


Fig. 3. The major steps in user focus-based graph query autocompletion

Definition 2.2. [Subgraph query] Given a graph database $D = \{g_1, g_2, \ldots, g_n\}$ and a query graph q, the answer (or result set) of q is $D_q = \{g \mid q \subseteq_\lambda g, g \in D\}$. □

Graph feature representation of queries. Graph features (or simply *features*) are generally understood as subgraphs that carry important characteristics of graph data. Features have been extensively used in subgraph query indexing [12]. Recently, Yi et al. proposed indexing c-prime features [3] for GQAC. c-prime features are frequent features that can be constructed from other smaller features in no more than c ways. Non c-prime features are not indexed as they are more likely to be suggested by GQAC. The paper follows this approach because, without prior knowledge, users may prefer to include such important characteristics in their queries. The current query is decomposed into a set of features and then another feature is added to form query suggestions.

Visual formulation of queries. Subgraph queries are graphs. It is therefore natural to provide a visual interface for drawing the queries, such as Fig 1. In visual query formulation, edge addition is a fundamental operation of *manual query composition*. That is, each manual operation adds *one edge* to the query graph q. Some visual interfaces may provide subgraph templates that can be added to an existing query as a whole. This is equivalent to adding a set of edges at a time. Moreover, it is handy for users to return to a previous state of the query graph.

Consistent with the findings of human-computer interactions [13], [14], users may continue with their tasks at hand. Users may compose a *connected query subgraph*. However, users may shift their attention occasionally, such as starting a new query subgraph (which is disconnected from the existing query) at some empty space in the query editor and connecting them later. Disconnected queries can be considered independent queries and passed *individually* to GQAC for autocompletion [3]. Such details are minor and omitted for a concise presentation.

Graph query autocompletion (GQAC). GQAC aims at alleviating users from the cumbersome actions needed to compose a visual query. The details of the user focus-based GQAC process can be illustrated with the sketch shown in Fig. 3. The autocompletion process is extended from the previous work [3], which is *not* focus-aware. Here, we summarize only the GQAC process but postpone some technical details for later sections.

1) GQAC takes a user's existing partial query and preference in a visual environment as input. The query is analyzed and decomposed into a set of graph features. Query suggestion candidates are generated and ranked.

A suggestion is a graph that is formed by adding a *subgraph increment* (or *increment*) to the existing query. An increment may contain *multiple edges*. A small set of query suggestions (or *suggestions*) are returned as output.

2) A user may further compose the query by either adopting a suggestion or adding an edge manually. This step repeats until the desired query completes.

Comparison with previous work. Recently, a GQAC framework [3], namely AUTOG, provides the top-k suggestions according to a user's preference on suggestion selectivity and diversity. AUTOG proposed *c-prime features* as logical units to be added to the user's query. AUTOG proposed algorithms to rank candidate suggestions and an index called *feature* DAG (FDAG) to optimize the ranking. Due to the user's preference for structural diversity, the returned suggestions can be subgraph increments connected to different subgraphs of the query. As reported in [3], AUTOG saved approximately 40% of clicks in query formulation. Hence, there are a notable amount of clicks not yet saved.

To enhance the quality of query suggestions, this paper is the first work that proposes a notion of user focus (of users' query formulation) and provides comprehensive query suggestions at the focus. In particular, as motivated in Sec 1, new suggestion ranking definitions are needed; the suggestion candidates are localized and compactly represented; and new ranking algorithms and index are required.

Remarks. We follow the general phenomenon (*i.e.*, in web searches and AUTOG [3]) that query autocompletion does not give suggestions involving *infrequent* items of a database. The analogy is that in web searches, infrequent keywords are not suggested; similarly, in GFOCUS, infrequent increments are not suggested. By definition, infrequent edges lead to small answer sets. In this case, users may simply run their queries to obtain the few answers.

3 QUERY FORMULATION SEQUENCE

The process (steps) that users go through to build their queries in a visual query formulation environment provides useful *implicit* information for GQAC. In this section, we formalize a query formulation sequence that aims at capturing major user interactions with the visual environment, which are then used to derive the user focus.

Formulation operators. The set of fundamental query formulation operators in this paper is OP = {add, adopt, select, rollback}. For better readability, we illustrate these essential operators by providing their narrative definitions.

- add(q, s): The add operator denotes that a user manually adds a structure s to an existing query q, and returns the augmented query.
- adopt(q, Δq): The adopt operator specifies that a user adopts a query suggestion, which augments query q with an increment Δq.
- select(q, e): The select operator denotes that a user indicates the edge of the query that he/she may work on. The select operator does not change the structure of the query, but indicates the working region of q.

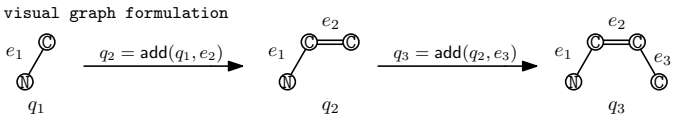


Fig. 4. An example of visual graph formulation

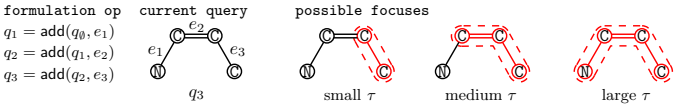


Fig. 5. User focuses by varying τ

- rollback(q, k): The rollback operator denotes that a user revokes the last k operators. It is a special form of deletion.[9]

We remark that other interesting operators (*e.g.*, mouse hover) may be introduced according to the application. The add and adopt operators formalize the major user interactions in the current state-of-the-art and are sufficient for user studies of GFOCUS. rollback is proposed for users' convenience. select is proposed for manually indicating the user focus when GFOCUS fails to detect users' desired one.

Query formulation sequence. A query formulation sequence is a sequence of query formulation operators, which specifies how a user composed the current query. A query formulation sequence S_t at time t is denoted as follows:

$$q_t = \mathsf{op}_t(\ldots \mathsf{op}_k(\ldots \mathsf{op}_1(q_0))), \quad (1)$$

where q_t is the current formulated query, op_t is the user's last operator, and q_0 is the initial query.

Example 3.1. Suppose a user is composing a query graph q_3 manually from q_1, as shown in Fig. 4. He/She may obtain the query q_3 via the formulation sequence below:

$$q_3 = \mathsf{add}(\mathsf{add}(q_1, e_2), e_3),$$

where e_1, e_2, and e_3 are edges in the query. The labels of the nodes are marked in the circles (*i.e.*, "C"), and the labels of the edges are represented by "=" and "–" connecting the circles. We remark that users may formulate the same query by going through different sequences. □

At any time point t, the current query q_t and its formulation sequence S_t are the least information for GFOCUS to automatically determine the user focus, and then produce suggestions at the focus.[10]

4 LOCALITY PRINCIPLES FOR GQAC

Madison and Batson [15] show that both human cognitive processes and program executions exhibit localities. A survey by Denning [16] reports that the locality principle has influenced a wide range of applications. Because of limited human short-term memory [5], [6], [9], it is intuitive that locality principles apply to query formulation. Inspired by the classical work on locality principles, we formulate the

9. It is an open issue that an arbitrary deletion might lead to arbitrary focus shift. We consider arbitrary deletion as if it is emulated by rollbacks followed by add operators.

10. There is a stream of work on exploiting the human gaze to predict human-computer interactions. However, such work requires users to install special hardware and is beyond the scope of this paper.

temporal and structural locality principles of user attention of the query formulation and propose a decay-and-propagation algorithm that runs in $O(|q|)$ at each query formulation step to compute the (implicit) user attention. It is possible to adopt other efficient approaches to derive the attention. In Sec. 5, we define user focus to be the subgraph of the highest user attention.

4.1 Temporal locality principle

Berman et al. [11] observe that memory fades due to the mere passage of time. In the context of query formulation, the temporal locality principle can be stated as follows: *human's attention on the edges that he/she is working on fades as time passes*. In other words, his/her attention localizes on the small number of newly operated edges. When this principle is applied to graph query autocompletion, the suggestions are more likely to be useful if they are connected to the small number of recently operated edges. Hence, the user focus is localized in the newly operated edges.

Temporal locality can be modeled by the *decay theory* [11]. To model user attention during the query formulation, we propose an *attention weight* for each edge e, denoted as w_e^t, where t is the t-th formulation operator in the query formulation sequence. We adopt the *exponential decay* function, as a forgetting function of w_e^t, which has a wide range of applications, *e.g.*, [17].

$$w_e^{temp} = w_e^{t-1}\mathsf{e}^{-1/\tau}, \tag{2}$$

where w_e^{temp} is the attention weight of w_e^t after the decay, e is the Euler's number, e is an edge in the query, and τ is a constant that controls the decay rate. τ conveys the semantic of the relative strength of the user's memory. The intuition is that the edge weight decays more slowly for a user with a strong memory (*i.e.*, a large τ) than a user with a weak memory (*i.e.*, a small τ).

The edge weight is set to a constant w_0 ($w_0 = 1$ by default) when the edge is just involved in a formulation operator. The edge weight decays by $\mathsf{e}^{-1/\tau}$ after each operator (*i.e.*, the recently operated edges have higher weights).

Example 4.1. At this point, assume user focus is intuitively the subgraph having the highest attention weight. Fig. 5 shows an example of the possible users focuses based on the temporal locality principle. The formulation sequence of query q_3 is shown in the leftmost, in which the edges e_1, e_2, and e_3 are added one by one. The graphs on the right show three possible focuses (highlighted in red) for users with different memory strengths. □

4.2 Structural locality principle

When humans are working on a software artifact, their attention is naturally localized on, and then propagated to a small number of neighboring artifacts [6], which is generally known as the *spatial spread of attention*. In the context of graph query formulation, we translate the structural locality principle as follows: *the edges that are close to the newly operated edges receive closer user attention and are more likely to be a part of the user focus.*

We capture the structural locality by Formula 3. It propagates some attention weight from the newly operated edges to their neighboring edges.

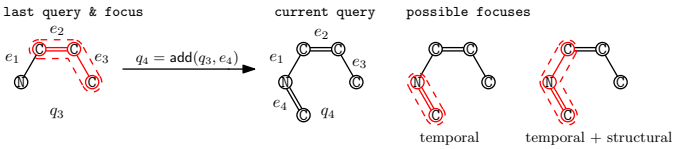


Fig. 6. Illustration of focuses by applying the locality principles

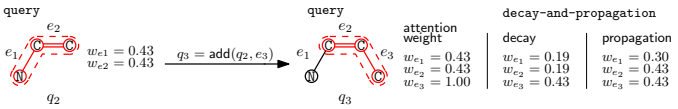


Fig. 7. An example of decay and propagation

$$w_e^t = w_e^{temp} + \Delta w\mathsf{e}^{-h/\tau}, \tag{3}$$

where e is an existing edge in the query before the last operation; h is the smallest number of edge hops from e to the last operated edge(s), that quantifies the *structural proximity* between them. Δw is the constant for controlling the maximum propagated weights. GFOCUS sets Δw to the decayed weight of the last operated edge(s), *i.e.*, $\Delta w = w_0 - w_0\mathsf{e}^{-1/\tau}$, because the last operated edge(s) has (have) an initial weight w_0, and a weight of $w_0\mathsf{e}^{-1/\tau}$ after one decay according to Formula 2. Such a Δw guarantees that the last operated edges have the highest weights (see Appx. E of [18]), after applying the locality principles (see Sec. 4.3).

Formula 3 states that each neighboring edge of the last operated edges receives some amount of weight that is positively proportional to their structural proximity h, and inversely exponentially proportional to a parameter τ, which models the user's relative memory strength. With a stronger memory (*i.e.*, a larger τ), the user's last operation propagates more attention weight to its neighboring structures. While τ in Formulae 2 and 3 has the same semantics of memory strength, they could be of different values.

Formula 3 has some advantages over existing propagation methods, *e.g.*, PAGERANK. Formula 3 spreads some weights from the newly operated edges to their neighboring edges *once* after each query operation. This well reflects the users' spatial spread of attention based on the query formulation sequence and the evolving structure of the query graph. In comparison, the propagation of PAGERANK repeats until it converges to a stationary distribution over the whole graph based on the last structure only.

Example 4.2. Fig. 6 shows an example of the possible focuses based on the locality principles. The leftmost graph shows the last query q_3 and the user focus. q_3 is formulated in three consecutive add operators, adding e_1, e_2, and e_3. The user then adds edge e_4 to obtain query q_4. The last operated edge e_4 is the user focus after the temporal locality is applied. After applying the structural locality, the first edge e_1 can be in the focus. □

4.3 Decay-and-propagation algorithm

In this subsection, we propose a unified decay-and-propagation algorithm to maintain the attention weights *in background*. The algorithm of the attention weight is presented in Algo. 1. Specifically, an edge obtains its initial attention weight (*i.e.*, w_0) when it appears in a formulation operator op (Lines 6-11), and its weight decays after each

Algorithm 1 Decay-and-propagation Algorithm for Maintaining Attention Weights

Input: query formulation sequence $q_t = \mathsf{op}_t(q_{t-1}, ...)$
Output: updated query graph
1: **switch** op_t **do**
2: **case** rollback(q_{t-1}, k)
3: revoke q_{t-1} to the query structure q_{t-1-k}
4: $w_e^t \leftarrow w_e^{t-1-k}$ for $e \in q_{t-1}$
5: **return** q_{t-1}
6: **case** add(q_{t-1}, s)
7: add s to q_{t-1}, and set $w_e^{t-1} \leftarrow w_0$ for all $e \in s$
8: **case** adopt(q_{t-1}, Δq) // Δq is the suggested increment
9: add Δq to q_{t-1}, and set $w_e^{t-1} \leftarrow w_0$ for all $e \in \Delta q$
10: **case** select(q_{t-1}, e) // e is the clicked edge
11: set $w_e^{t-1} \leftarrow w_0$
12: //the decay step
13: **for all** $e \in q_{t-1}$, $w_e^{temp} \leftarrow w_e^{t-1}\mathsf{e}^{-1/\tau}$ //Formula 2
14: //the propagation step
15: denote C to be the last operated edge(s)
16: $R \leftarrow q_{t-1}.E \setminus C$ // the remaining edges
17: $h \leftarrow 0$
18: **while** $R \neq \emptyset$ **do**
19: increase h by 1
20: **for all** $e \in R$ and $e_c \in C$ **do**
21: **if** e is connected to e_c **then**
22: $w_e^t \leftarrow w_e^{temp} + \Delta w\mathsf{e}^{-h/\tau}$ //Formula 3
23: $C \leftarrow C \cup \{e\}$ //C: completed propagation
24: $R \leftarrow R \setminus \{e\}$
25: **return** q_{t-1}

operator. rollback is an exception, which simply revokes the changes of the query structure *and* weights of the last k steps, in Lines 2-5. In Lines 12-13, the edge weights decay as time passes, according to Formula 2. Next, the algorithm proceeds to the propagation step (Lines 14-24). C is a set of edges whose weights have completed the propagation step. Initially, C is the last operated edge(s) and R is the remaining edges. The algorithm updates the weights (in Line 22) in a breadth-first traversal manner (Lines 18-24). The time complexity of maintaining the attention weights is $O(|q|)$ since each of the decay step and the propagation step takes $O(|q|)$ time to update the edge weights. The space complexity is also $O(|q|)$ since both C and R contains $O(|q|)$ edges at most.

Example 4.3. Consider an existing query q_2 with two edges e_1 and e_2 added in op_1 and op_2, respectively, illustrated with Fig 7. The attention weights of e_1 and e_2 are 0.43. Then, we add another edge e_3 with an initial weight of 1.0 in op_3. Suppose we have τ set to 1.2. In the decay step, the weights of the three edges are reduced to 0.19, 0.19, and 0.43, respectively. In the propagation step, the weights of e_1 and e_2 increase. After one decay-and-propagation step, the weights of e_1, e_2, and e_3 are 0.30, 0.43, and 0.43, respectively. □

5 User Focus of GQAC

In this section, we define the user focus, derived from the user attention weights (Sec. 4), and propose a polynomial time algorithm to compute the focus. Query graphs have been modeled by edge-weighted graphs with attention weight associated with each edge. We then define the user focus as a subgraph of the query.

Definition 5.1. [User focus] The *user focus* f is the (proper) connected subgraph of the query q with the maximal normalized weight w_f, defined as follows:

Algorithm 2 GREEDY for Determining the User Focus

Input: the current query q
Output: user focus f
1: $C_f \leftarrow q.E$ // each edge as a candidate focus
2: initialize user focus $f_{\max}$ to the edge with the highest weight
3: **repeat**
4: $C'_f \leftarrow \emptyset$ // initialize the next cand. focuses
5: **for all** $c_f \in C_f$ **do** // current candidate focus
6: $e_{\max} \leftarrow \mathsf{argmax}_{e \in q.E \setminus c_f}(w_{c_f \oplus e})$ // $\oplus$: edge adding
7: $c'_f \leftarrow c_f \oplus e_{\max}$ // next candidate focus
8: **if** $w_{c'_f} > w_{c_f}$ **then**
9: $C'_f \leftarrow C'_f \cup \{c'_f\}$
10: **if** $w_{c'_f} > w_{f_{\max}}$ **then**
11: $f_{\max} \leftarrow c'_f$ // maintain the user focus
12: $C_f \leftarrow C'_f$ // candidates updated in this iteration
13: **until** $C_f = \emptyset$ // no more candidates to be expanded
14: **return** $f_{\max}$

$$w_f = \frac{\sum w_e}{|f.E|^{\omega(\tau)}}, \tag{4}$$

where $e \in f.E$ and $f \subseteq_\lambda q$ and when an edge is added to f, the normalized weight is reduced. □

The user's relative memory strength τ is an input of the function ω that models the relevant importance between the sum of the weights and the size of the subgraph f. For illustration purposes, we define $\omega(\tau)$ to be $1/\tau$.

A user with a strong memory (*i.e.*, a large τ) can focus on a larger subgraph than a user with a weak memory, which is consistent with intuitions. In extreme cases, when τ is set to 1, the formula defines a user focus containing only the last operated edges. In case τ is set to a sufficiently large value, the formula defines the focus to be the whole query.

Example 5.1. Continuing with Fig. 7, τ is set to 1.2. When formulated q_2, the user focus is the subgraph that consists of e_1 and e_2. After adding e_3 with the operator add, the attention weights are updated. The new focus is the subgraph of e_2 and e_3 (highlighted in q_3). □

The problem of determining the user focus is closely related to a classical NP-complete problem. Identifying the user focus of the size k (with the *maximum normalized weight*) is equivalent to find the node-optimal connected k-subgraph problem [19], which is NP-complete. In particular, given an instance of the node-optimal k-subgraph problem, we can convert it into its line graph in linear time [20]. And the node weights are converted to the edge weights. Then, suppose we find the query focus of the size k, which is the one with the largest sum of edge weights. The query focus of the line graph can be converted back to the solution of the node-optimal connected k-subgraph problem in linear time. However, we note that user focus is transparent to users. The definition does not require a fixed user focus size and the maximum normalized weight, it only requires a *maximal normalized weight*.

Greedy algorithm for computing the focus. Users may compose a query in arbitrary formulation sequence. We propose a bottom-up greedy algorithm (called GREEDY) to enumerate the subgraph with the maximal normalized weight w.r.t Formula 4. The most important design of GREEDY is that *every single edge in the query is initially a candidate focus*. GREEDY then iteratively expands each candidate focus with its neighboring edge which *maximizes* Formula 4. The iterations terminate when the candidate focuses cannot

be expanded and meanwhile their normalized weights are increased. Then, the candidate with the largest weight is selected as the user focus. In Appx. F of [18], we further elaborate the pseudocode.

The complexity of determining user focus depends on $\omega(\tau)$. We show that when $\omega(\tau)$ is a constant greater than 0, GREEDY correctly determines the user focus. The time complexity of GREEDY is simply $O(|q|^3)$ since i) there are at most $O(|q|)$ candidate focuses; ii) it takes $O(|q|)$ to expand each candidate focus in one iteration; and iii) there are at most $O(|q|)$ iterations before GREEDY terminates. The space complexity of GREEDY is $O(|q|^2)$ since i) there are at most $O(|q|)$ candidate focuses, and ii) each focus can expand to at most $O(|q|)$ edges.

Proposition 5.1. GREEDY determines the maximal user focus when $\omega(\tau) \geq 0$. □

The proof is presented in Appx. A of [18].

6 GQAC AT USER FOCUS

Sec. 5 presents how GFOCUS determines a user focus f (Step ② of Fig. 3). Next, GFOCUS decomposes f into a set of feature embeddings (Step ③). GFOCUS then performs the following two main steps. First, in a candidate generation step, we generate possible candidate suggestions at the focus. This step determines the increments to be attached to the current query to form suggestions. In Sec. 6.1, we propose structural union and efficiently compute it to compactly represent the "universe" of all candidate suggestions. Second, in Sec. 6.2, we propose several new ranking functions for the top-k suggestions.

6.1 Candidate suggestions at user focus

We observed from our experiments that the numbers of possible subgraph increments to an existing query in the presence of a user focus typically ranged from hundreds to thousands. Without user focus, such numbers are huge. Hence, at the user focus, it is feasible to compute the "universe" of all candidate suggestions. To provide comprehensive suggestions (*i.e.*, Sec. 6.2), we rank the top-k candidate suggestions that cover such universe the most.

6.1.1 The Universe of Candidate Suggestions – Structural Union

We propose the *structural union* of a set of graphs. For simplicity, we first consider the "union" of a pair of suggestions (graphs) and then generalize it to a set of suggestions. Intuitively, a structural union is a graph that preserves the structures of the individual graphs. In particular, it combines the two suggestions by sharing the maximum common edge subgraph (mces) between them.[11] We recall the function [3] that combines two graphs in Def. 6.1.

11. There are existing notions of union of two graphs. One popular approach is to consider the graphs as finite state automata (FSA). Such techniques preserve the accepting languages or behaviors of the FSA but may eliminate states/transitions of the FSAs. However, query graphs are not state machines. If applied, the structures of the individual graphs cannot be preserved.

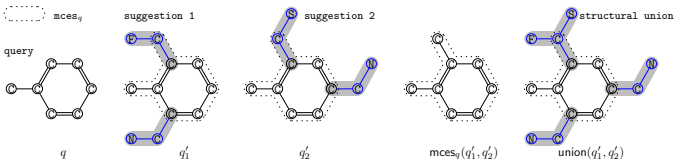


Fig. 8. Structural union of a pair of suggestions (q'_1 and q'_2) for q

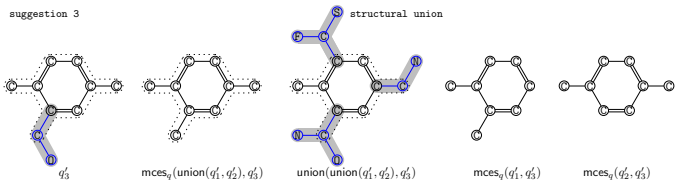


Fig. 9. Structural union of a set of suggestions (q'_1, q'_2 and q'_3) for q

Definition 6.1. [Query composition] compose [3] takes two graphs, q_1 and q_2, and the corresponding embeddings, λ_1 and λ_2, of a common subgraph cs as input, returns the graph g that is composed by q_1 and q_2 via λ_1 and λ_2 of cs, respectively, *i.e.*, g = compose(q_1, q_2, cs, λ_1, λ_2). □

Definition 6.2. [Structural union of a suggestion pair] Given a pair of suggestions q'_1 and q'_2 for a query q, the *query-centric* mces of q'_1 and q'_2 (denoted as mces$_q(q'_1, q'_2)$) satisfies the following properties:

- $q \subseteq$ mces$_q(q'_1, q'_2)$; and
- mces$_q(q'_1, q'_2) \subseteq_{\lambda_1} q'_1$ and mces$_q(q'_1, q'_2) \subseteq_{\lambda_2} q'_2$, where λ_1 and λ_2 are the corresponding subgraph isomorphism embeddings.

The *structural union of* q'_1 *and* q'_2, denoted as union(q'_1, q'_2), is

- compose(q'_1, q'_2, mces$_q(q'_1, q'_2)$, λ_1, λ_2). □

Structural union (Def. 6.2) is space-efficient since q'_1 and q'_2 are combined via their *maximum* common subgraph (see Def. 6.1 for compose). Note that q appears in mces$_q$. That is, the mces$_q$ of q'_1 and q'_2 is a supergraph of q. Further, union(q'_1, q'_2) *contains* the structures of q'_1 and q'_2, *i.e.*, q'_1 and q'_2 are subgraphs of union(q'_1, q'_2).

Example 6.1. Fig. 8 shows a partially formulated query q and a pair of query suggestions, q'_1 and q'_2, for q. The increments of q'_1 and q'_2 are highlighted in blue with the gray background. The maximum common subgraph of q'_1 and q'_2 (*i.e.*, mces$_q(q'_1, q'_2)$) are surrounded by dotted lines. The structural union of the two suggestions is the composed graph of q'_1 and q'_2 via mces$_q(q'_1, q'_2)$, shown as union(q'_1, q'_2). □

We then generalize Def. 6.2 to the structural union of a set of query suggestions and remark its properties.

Definition 6.3. [Structural union of a suggestion set] Given a set of query suggestions Q': $\{q'_1, q'_2, \ldots, q'_n\}$ for a query q, the *structural union of* Q' is the graph formed by a sequence of union of the suggestions, denoted as

union(Q') = union(...union(q'_1, q'_2), ..., q'_n). □

Proposition 6.1. [Constant minimum structural union size] Given a set of graphs,

- the structural union(s) may not be unique; and
- all structural unions are of the same minimum size. □

Given a set of query suggestions Q', its structural union(s) may not be unique, since the mces$_q$ of a pair of query suggestions may not be unique and each mces$_q$ may

have multiple embeddings. The second part of Prop. 6.1 can be established from a proof by contradiction. In essence, graphs are combined via mces_q in union and the union graph contains the original graphs. If a union graph is larger than the others, this graph must not be composed via mces_q.

6.1.2 Efficient Structural Union Computation

The structural union defined in Def. 6.3 allows us to propose an optimization for computing them. In particular, the costly mces_q computation of the intermediate union graph and suggestions can be optimized.

Given a set of query suggestions $Q' = \{q'_1, q'_2,...,q'_n\}$, when computing the $\mathsf{union}(Q')$, $\mathsf{union}(...\mathsf{union}(q'_1, q'_2),...,q'_n)$, as defined in Def. 6.3, is a left-deep approach. This approach is inefficient because as union being called, the sizes of the intermediate union graphs increase and the runtimes of mces_q increase significantly. The time complexity of computing the structural union is exponential to the graph sizes since it involves computing mces_q of the intermediate union graphs and suggestions. In contrast, Prop. 6.2 states a *decomposition* property of computing structural union. Prop. 6.2 states that to compute the mces_q of $\mathsf{union}(Q')$ and a suggestion graph q'_{n+1}, we can compute the structural union of the mces_q of q'_{n+1} and each suggestion graph in Q'. The intermediate union graph size is much smaller.

Proposition 6.2. [Decomposition of structural union] Given a query q and its query suggestions Q': $\{q'_1, q'_2, \ldots, q'_n\}$ and a new query suggestion q'_{n+1}, $\mathsf{union}(Q' \cup \{q'_{n+1}\})$ is:

$$\mathsf{compose}(\mathsf{union}(Q'), q'_{n+1}, \mathsf{mces}_q(\mathsf{union}(Q'), q'_{n+1}), \lambda_i, \lambda_j), \quad (5)$$

where λ_i (resp.λ_j) is the embedding of the mces_q in $\mathsf{union}(Q')$ (resp. q'_{n+1}); and

$$\mathsf{mces}_q(\mathsf{union}(Q'), q'_{n+1}) = \mathsf{union}(\{\mathsf{mces}_q(q'_{n+1}, q') | q' \in Q'\}). \quad (6)$$

The proof is provided in Appx. A of [18].

Example 6.2. We use Fig. 9 to illustrate the decomposition of the structural union of a set of query suggestions. Recall that we obtained the structural union of two query suggestions q'_1 and q'_2 in Example 6.1, shown as $\mathsf{union}(q'_1, q'_2)$ in Fig. 8. We further consider another query suggestion q'_3 (shown in Fig. 9). To obtain $\mathsf{union}(\mathsf{union}(q'_1, q'_2), q'_3)$, we first need to compute $\mathsf{mces}_q(\mathsf{union}(q'_1, q'_2), q'_3)$. To avoid directly computing $\mathsf{mces}_q(\mathsf{union}(q'_1, q'_2), q'_3)$ since the size of the intermediate union graph $\mathsf{union}(q'_1, q'_2)$ is large, we first compute $\mathsf{mces}_q(q'_1, q'_3)$ and $\mathsf{mces}_q(q'_2, q'_3)$ (shown in Fig. 9), then obtain $\mathsf{mces}_q(\mathsf{union}(q'_1, q'_2), q'_3)$ by computing the structural union of $\mathsf{mces}_q(q'_1, q'_3)$ and $\mathsf{mces}_q(q'_2, q'_3)$. □

Analysis. The time complexity of the structural union computation using Prop. 6.2 is $O(|Q'| \times T_{\mathsf{compose}_1} + |Q'|^2 \times (T_{\mathsf{mces}_q} + T_{\mathsf{compose}_2}))$. (a) The first term is the time for combining the intermediate union graphs and suggestions, and T_{compose_1} is the time for combining the union graph $\mathsf{union}(Q')$ and a suggestion graph q'_{n+1}. This is invoked $|Q'|$ times. (b) The second term is for the mces_q computation between the intermediate union graph and suggestions. Each mces_q computation between a pair of suggestions is denoted as T_{mces_q}. There are $|Q'|^2$ pairs. The time of T_{compose_2} is for the union of the computed mces_q graphs. The union is called $|Q'|^2$ times. In practice, the terms $|Q'|$, T_{compose_1} and T_{compose_2} are often small.

6.2 Suggestion ranking at user focus

This section presents the ranking that determines the top-k suggestions that cover the structural union, the suggestion universe, the most, namely *structural cover* (Def. 6.4). We also present some additional ranking functions can be plugged into GFOCUS.

Definition 6.4. [Maximum Structural Cover] Given a query q, a set of query suggestions Q', and the structural union U of Q', a user-specified constraint k, the *structural cover* of Q' is to determine k suggestions Q'' , where $Q'' \subseteq Q'$, $|Q''| \leq k$ and there is no other $Q''' \subseteq Q'$, such that the size of $\mathsf{union}(Q''')$ is larger than $\mathsf{union}(Q'')$, i.e. $|\mathsf{union}(Q''').E| > |\mathsf{union}(Q'').E|$. □

The problem of determining the maximum structural cover is NP-hard. The hardness can be established by a reduction from the MAXIMUM COVERAGE problem.

6.2.1 Baseline ranking function

Given the hardness of determining the maximum structural cover, one may be tempted to design greedy algorithms from covering problems. A greedy algorithm may favor suggestions of large increments that greedily cover the universe. Such suggestions are large queries but they do not often retrieve many answers, which may not be useful to users. In addition, to provide suggestions with prior knowledge about the users, we propose a baseline ranking function to balance the user's preference on popular (*e.g.*, high selectivities) and comprehensive (*e.g.*, high coverage) suggestions, presented in Def. 6.5. The preference of popular suggestions simply reflects users' intent to retrieve more answers, whereas the preference of comprehensive suggestions recognizes the importance of covering all candidate suggestions. These two preferences can be quantified as the following objective functions:

1) $\mathsf{sel}(q)$: the selectivity of q on D is defined as $|D_q|/|D|$, where $|D_q|$ is estimated using the candidate answer set (using techniques such as [3]), $|D|$ is for normalization.
2) $\mathsf{coverage}(Q)$: the coverage of Q over the structural union of all candidate suggestions is defined as $|\mathsf{union}(Q)|/|U|$, where U is the structural union of all candidate suggestions. The coverage of suggestions quantifies how comprehensive a set of suggestions are.

Definition 6.5. [Utility of query suggestions] Given a set of query suggestions $Q' : \{q'_1, q'_2, \ldots, q'_k\}$, the structural union U of all candidate suggestions and a user preference component β, the utility of Q' is defined as follows:

$$\mathsf{util}(Q') = \frac{\beta}{k} \sum_{q' \in Q'} \mathsf{sel}(q') + (1 - \beta)\mathsf{coverage}(Q'),$$

where $\beta \in [0, 1]$. □

The bi-criteria ranking function combines the selectivity and the coverage of the query suggestions. β is for balancing the two criteria and k is the constant denominator for normalization. Furthermore, the two objectives of util are competing: in practice, the selectivities of smaller queries are often larger as more data graphs contain smaller queries, whereas smaller queries cover a smaller portion of the structural union of all candidate suggestions.

The ranking task is then to find the top-k suggestions that have the highest util value. The ranking problem of query suggestions (presented in Def. 6.6) is also NP-hard.

Definition 6.6. [Ranked Subgraph Query Suggestions (RSQ)] Given a query q, a set of query suggestions Q', the ranking function util, a user preference component β, and a user-specified constraint k, the *ranked subgraph query suggestions* problem is to determine a subset Q'', util(Q'') is maximized, *i.e.*, $Q'' \subseteq Q'$, $|Q''| \leq k$ and there is no other $Q''' \subseteq Q'$ such that util(Q''') $>$ util(Q''). □

Proposition 6.3. The RSQ problem is NP-hard. □

The proof is presented in Appx. A of [18].

6.2.2 Additional ranking functions

The coverage function presented in Def. 6.5 counts the number of edges in the structural union of all candidate suggestions that the top-k suggestions cover. Depending on users' applications, users may prefer more sophisticated covering semantics. Hence, we illustrate that other functions can be seamlessly plugged into the second component of util.[12] In the following, we propose three intuitive monotone submodular functions. The function takes a suggestion q' ($q' \in Q'$) as input and returns a weight as output. Then, coverage(Q') is simply the sum of these weights.

1) overlap(q') returns the normalized weights of the edges in the increments of q'. The weight of each edge e is the number of candidate suggestions that cover e, in the structural union of all candidate suggestions. The edges having high weights are those that are covered by many candidate suggestions. That is, suggestions that cover popular edges among all candidate suggestions have high weights.

2) freq(q') returns $\Sigma_{e \in \Delta q'} |D_e|/|D|$ (after normalization) as the weight of q', where $\Delta q' = q'.E \setminus q.E$ and $|D_e|$ is the number of data graphs in database D containing e. That is, suggestions that contain frequent edges have high weights.

3) sel$_\Delta$(q') returns the number of supergraphs of the increments ($\Delta q'$) in the database D. That is, suggestions that may retrieve many data graphs have high weights.

Proposition 6.4. util is monotone submodular. □

The proof is presented in Appx. A of [18]. A natural approximation algorithm for solving RSQ is greedy algorithm of iteratively adding the element with the maximum marginal gain, because the problem of maximizing a monotone submodular function subject to a cardinality constraint admits a $1 - 1/\mathrm{e}$ approximation algorithm.

6.2.3 Greedy ranking algorithm

We propose a greedy algorithm (Algo. 3) to rank the candidate suggestions according to the user preference. Since the user focus is yet another graph, we can also decompose it into a set of feature embeddings M_q. Algo. 3 then greedily determines the overall top-k suggestions Q'_k from all candidate suggestions Q'. More specifically, Line 1 computes the overall structural union of all candidate suggestions using U_fs, where f is a feature at the focus. We remark that U_f is retrieved by a lookup of the SUDAG index (to be proposed next). Hence, it avoids computing the structural union of all possible candidate suggestions online. Lines 4-5 generate possible candidate suggestions Q' using each feature embedding (f, λ) at the focus. In each iteration of Lines 6-9, the algorithm *greedily adds* the composed query suggestion q' to Q'_k that makes the util function the largest. This step repeats until it obtains k query suggestions in Q'_k.

12. Users may not modify the first component of util (*i.e.*, sel) as query autocompletion generally requires the relative importance of the selectivities of the query suggestions.

Algorithm 3 Ranking Candidate Suggestions

Input: a query q and the set of feature embeddings M_q at the user focus, user preference component β, number of suggestions requested k and max. increment size δ
Output: the top-k suggestions Q'_k

1: Let $U \leftarrow$ union($\{U_f | f \in M_q\}$) be the overall structural union, where U_f is indexed *offline* in SUDAG (see Def. G.1)
2: Let Q' be an empty set // initialize candidate suggestions
3: Let Q'_k be an empty set // initialize top-k suggestions
4: **for all** $(f, \lambda) \in M_q$ **do**
5: $Q' \leftarrow Q' \cup Q'_\mathcal{C}$ //$Q'_\mathcal{C}$ is the possible suggestions composed by adding another feature to f, where f is embedded in q via λ, $q'_c \in Q'_\mathcal{C}$ implies $|q'_c|$-$|q| \leq \delta$.
6: **for** $i = 1 \ldots k$ **do**
7: $q'_{max} \leftarrow$ argmax(util($Q'_k \cup \{q'\}$)), where $q' \in Q'$
8: $Q'_k \leftarrow Q'_k \cup \{q'_{max}\}$
9: $Q' \leftarrow Q' \setminus \{q'_{max}\}$
10: **return** Q'_k

Analysis. Denote the time to generate a candidate suggestion as $O(T_{\mathsf{compose}})$. Denote $O(|Q'_\mathcal{C}|)$ to be the number of possible suggestions generated in Line 5. The algorithm takes $O(|M_q| \times |Q'_\mathcal{C}| \times T_{\mathsf{compose}})$ to generate all candidate suggestions. Denote the time to compute the util value in Line 7 as $O(T_{\mathsf{util}})$. The ranking time is then $O(k \times (|M_q| \times |Q'_\mathcal{C}|) \times T_{\mathsf{util}})$. The time complexity of Algo. 3 is $O((|M_q| \times |Q'_\mathcal{C}|) \times (T_{\mathsf{compose}} + k \times T_{\mathsf{util}}))$. We assume the size of the increment to the current query q is at most δ_{max}, which is small when compared to the query size $|q|$. Then, each candidate suggestion takes $O(|q|)$ space. The space complexity of Algo. 3 is $O(|M_q| \times |Q'_\mathcal{C}| \times |q|)$ for the candidate suggestions Q' where there are $O(|M_q| \times |Q'_\mathcal{C}|)$ candidates and each suggestion takes $O(|q|)$ space. The overall structural union U and the top-k suggestions Q'_k are negligible when compared to Q'. We observed from our experiments that the numbers of candidate suggestions typically ranged from hundreds to thousands only.

7 Indexed GQAC at User Focus

This section presents the Structural-Union-of-Suggestions DAG index (SUDAG), and how it optimizes Algo. 3. Due to space limitations, we highlight the novel parts of SUDAG that index the structural unions of the feature compositions of a database. The verbose definition and construction algorithm of SUDAG are provided in Appx. G of [18].

SUDAG adopts the index topology and relevant auxiliary information of features for GQAC from the state-of-the-art [3]. Each index node represents a feature and the index edge (f_i, f_j) indicates $f_i \subseteq_\lambda f_j$ and the embeddings of f_i in f_j are indexed. The left-hand side of Fig. 10 shows a sketch of the index topology of the PubChem dataset. The main novelty of SUDAG relies on, for each feature f, indexing the structural union (U_f) of all possible compositions (ζ_f) and their embeddings (η_f) in U_f, shown in the right-hand side of Fig. 10. We elaborate the main techniques below.

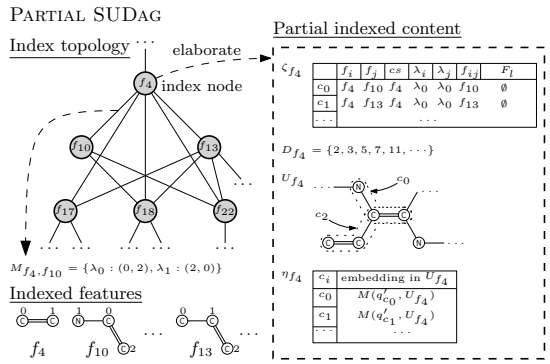


Fig. 10. The (partial) SUDAG of PUBCHEM

1) SUDAG is feature-based.[13] The indexed content of each feature includes all possible query compositions (*i.e.*, possible ways to combine graph features) for generating candidate suggestions (ζ_f) at f of q, and selectivity estimation (D_f). For example, the top right corner of Fig. 10 shows that f_4 can be composed together with f_{10} and f_{13}. A record c_1 indexes such a composition, which is used in Line 5 of Algo. 3, to generate the candidate suggestions. This eliminates enumerating the feature compositions online, whose runtime is exponential to the graph sizes.
2) SUDAG indexes (i) the structural union U_f of all query compositions of a feature (*i.e.*, U_{f_4} in Fig. 10) and (ii) the embedding of each feature composition in U_f (*i.e.*, η_{f_4} in Fig. 10) to accelerate the online ranking in Lines 1 and 7 of Algo. 3. Each U_f is precomputed, and retrieved (*e.g.*, the union graph in the right-hand side of Fig. 10) for online ranking. This eliminates computing the per-feature structural union online
3) Further, in Line 7, it computes the candidate suggestion that yields the largest util value. Hence, SUDAG indexes the IDs of the graphs that contain f (*i.e.*, D_f) for computing sel in util. Next, since the embedding of a feature composition in the structural union U_f has been indexed, the computation of coverage does not require subgraph matching but simple SUDAG lookups and counting.

8 EXPERIMENTAL EVALUATION

This section presents an experimental evaluation of GFOCUS. We first investigate the quality (or the effectiveness) of both user focus and suggestions via user studies. As user studies are hardly scalable, we make a connection of suggestion quality with a popular quantitative metric. We then conduct an extensive experimental evaluation on popular real datasets, to study the performance of GFOCUS.

Platforms. We implemented the GFOCUS prototype [10] on top of the state-of-the-art (AUTOG [3]). The prototype was mainly implemented in C++, using VF2 for subgraph queries and McGregor's algorithm (with minor adaptations) for mces_q. We used GSPAN [21] for frequent subgraph mining. We followed AUTOG to obtain a sufficient number of features offline for GFOCUS to generate compositions.

We conducted all the experiments on a machine with a 2.2GHz Xeon E5-2630 processor and 256GB memory, running Linux. All the indexes were built offline and loaded from hard disk and were then made fully memory-resident.

13. We assume that the graph features of the database have been mined using existing techniques (*e.g.*, GSPAN [21]) and the details are omitted since this work is orthogonal to the feature definitions.

TABLE 2
Some characteristics of the datasets

Dataset	$\|D\|$	avg($\|V\|$)	avg($\|E\|$)	$\|l(V)\|$	$\|l(E)\|$
PUBCHEM	1M	23.98	25.76	81	3
EMOL	9.14M	23.68	25.49	81	5
AIDS	10K	25.42	27.40	51	4

TABLE 3
Some statistics of the features of datasets

Dataset	minSup	$\|F\|$	avg($\|V\|$)	avg($\|E\|$)
PUBCHEM	0.10	1,206	7.44	6.47
EMOL	0.10	1,107	7.05	6.08
AIDS	0.10	460	6.15	5.20

Datasets. We used popular benchmarked datasets. (i) For ease of comparison with AUTOG, we used PUBCHEM, a real chemical compound dataset containing 1M graphs, unless otherwise specified. (ii) We used the eMolecules Plus database (denoted as EMOL) which consisted of 9.14M graphs; and (iii) AIDS (AIDS antiviral dataset), which consisted of 10k graphs. Experiments on EMOL and AIDS are presented in Appx. C and D of [18], due to space limitations. Tab. 2 and Tab. 3 report some characteristics of the datasets and features including the number of graphs ($|D|$), the average number of vertices and edges (avg($|V|$) and avg($|E|$)), the number of vertex/edge labels ($|l(V)|$ and $|l(E)|$) and $|F|$ is the number of features.

Query sets. We generated numerous sets of query graphs of different query sizes $|q|$ (the numbers of edges) and other characteristics. In particular, we generated queries that yield different minimum result set sizes $|D_q^{\min}|$ (*i.e.*, $|D_q| > |D_q^{\min}|$ for all query graphs) because the result set size is a part of the util function and queries with no answers are meaningless to users. Each query set contained 100 graphs.[14] Query sets of various query sizes (called Q4, Q8, ..., Q24) were generated with $|D_q^{\min}|$ set to 10. Query sets of the minimum result set sizes 1, 10, 100, and 1000 (if applicable) were generated with $|q|$ set to 20, respectively.

Default settings. We used the default values of AUTOG parameters when applicable. Regarding user focus, we set τ of Formulae 2 and 3 to 1.2. Users can use the GFOCUS prototype [15] to tune its values by observing the highlighted user focuses. We set the default maximum query increment size (*i.e.*, $\delta_{\max}$) to 5. To decouple the effect of different increment sizes from the experiment results, we introduced the fixed increment size δ_{fix}. Its default value is 2. We used δ_{fix} instead of $\delta_{\max}$ unless otherwise specified. For ranking function parameters of AUTOG and GFOCUS, we set α to 0.1, β to 0.1. k is set to 10. The default query size $|q|$ is 8. We set the default covering semantic of util to coverage. The default parameter values are summarized in Appx. H of [18]. We place some experiments on parameter tuning and other covering semantics in Appx. J of [18].

Index. We briefly summarize some characteristics of constructing SUDAG. The left-hand side of Tab. 4 shows the size and construction time of the feature DAG topology

14. A query was generated as follows. We randomly chose a graph g from the dataset and randomly selected $|q|$ edges from g to form a new graph q. We checked if q is connected and has a result count $|D_q|$ larger than $|D_q^{\min}|$.

15. http://autog.comp.hkbu.edu.hk:8000/gfocus/

TABLE 4
Some statistics of SUDAG

	DAG topology			Structural unions (U_f)		
Dataset	$\|V\|$	$\|E\|$	time (s)	avg($\|V\|$)	avg($\|E\|$)	time (s)
PUBCHEM	1,206	31,610	2.6	173	374	65,512
EMOL	1,107	26,885	1.5	164	416	85,164
AIDS	460	6,965	0.4	131	279	13,447

of the SUDAG. The sizes (*i.e.*, number of edges) of the DAG topology are on average 22 times of the number of the features. The construction times of the DAG topology are within three seconds. The right-hand side of the Tab. 4 shows some statistics of the indexed structural unions of the SUDAG. The average sizes of structural unions are at hundreds of nodes and edges. The sizes of the structural union are on average 15, 16, 10 times of the average sizes of the graphs of PUBCHEM, EMOL and AIDS, respectively. Hence, the indexed structural unions are highly compact, considering that there are only thousands of structural unions for millions of data graphs. The times to index them are within a day when Prop. 6.2 for efficient structural union computation was used. With Prop. 6.2, 90% of the features finished the computation within one hour. The average time of computation was 1703 seconds. The 95% confidence interval of the time cost is [1561s, 1845s]. Without it, none of them finished within an hour. The distribution of the runtime of the structural union computation (for features with 7 edges) is reported in Fig. 16.

Quality metrics. We adopted several popular metrics for suggestion qualities [3], [22]. We report the number of suggestion adoptions (*i.e.*, #AUTO) and the *total profit metric* (*i.e.*, TPM). Specifically, the *total profit metric (*TPM*)* is adopted from [3], [22], which quantifies the % of mouse clicks saved by adopting suggestions in visual formulation:

$$\frac{no.\ of\ clicks\ saved\ by\ suggestions}{no.\ of\ clicks\ without\ suggestions} \times 100\%.$$

When appropriate, we report the increment size of the adopted suggestions (denoted as Δ) and the useful suggestion ratio U defined as $\frac{no.\ of\ useful\ suggestions}{no.\ of\ returned\ suggestions} \times 100\%$. Each reported number is the average of the 100 queries in each query set. The subscripts G and F indicate the results of AUTOG and GFOCUS, respectively. Note that even when the suggestions are correct, users still need mouse clicks to adopt them to obtain the target query. Hence, the optimal TPM of a GQAC system is *not* 100% (see Appx. I of [18]).

8.1 Suggestion quality via user studies

We conducted user studies to show (i) the effectiveness of user focus and using TPM as an indicator of suggestion qualities (User study 1), and (ii) the comparison of AUTOG by volunteers (User study 2) and chemists (User study 3).

We mimicked the GUI and followed the settings of AUTOG [3] for the user studies. All the users were not exposed to the user studies before. In each study, every user was given six 8-edge target queries with high, medium, and low TPM values obtained from the GFOCUS or AUTOG simulations. We randomly shuffled these queries and asked the users to formulate them. Users are reminded to wait for the suggestions from AUTOG before composing their queries further. To simplify the experiments, we disabled rollback as the users' rollback actions depend on their discretions

TABLE 5
Settings of user studies

User study	GQAC system	queries from	users
User study 1	GFOCUS	GFOCUS	21 volunteers
User study 2	GFOCUS and AUTOG [3]	AUTOG	10 volunteers
User study 3	GFOCUS and AUTOG [3]	AUTOG	3 chemists

but they affect our measurements in a non-trivial way. The computed user focuses were highlighted in red when users use GFOCUS. The users were not aware of their details. After formulating each query, they reported their agreement to following two statements:

1) "*The red colored edges capture the portion that I am working on.*"; and
2) "*The suggestions are useful when I draw my query.*"

We adopted a symmetric 5 level agree-disagree Likert scale: 1 means "*strongly disagree*" and 5 means "*strongly agree*").[16] We summarize the settings of the user studies in Tab. 5.

User study 1. Effectiveness of user focus and TPM**.** The objective of Statement 1) is to study the effectiveness of the determined user focus. We invited 21 volunteers using GFOCUS to formulate queries obtained from the GFOCUS simulations. The average user satisfaction to the determined user focus was consistently very high. In particular, users gave 4.45, 4.31, and 4.48 (between "strongly agree" and "agree") to the focuses of the queries having high, medium, and low TPM values, respectively. This verified the effectiveness of the focus.

Next, it is known that qualitative analysis is often hardly scalable. Hence, the objective of Statement 2) is to test the *effectiveness of using* TPM *as an indicator of suggestion qualities*. Consistent with the finding of AUTOG, the user's opinion on suggestion usefulness and the TPM values have a high correlation coefficient of 0.930 and a p-value of 0.007. Hence, TPM is a statistically significant indicator of suggestion usefulness. The average ratings of the queries with high, medium, and low TPM values was 4.14 (between "*strongly agree*" and "*agree*"), 3.07 (between "*agree*", and "*neither agree nor disagree*") and 2.45 (between "*neither agree nor disagree*", and "*disagree*"), respectively.

User study 2. Comparison of GFOCUS **and** AUTOG **from volunteers.** The objective of this study is to compare the suggestion qualities of GFOCUS and AUTOG using the same query set (used in the user study of AUTOG [3]). Similar to [3], we invited 10 volunteers using GFOCUS to formulate queries from the AUTOG simulations, and compared the results with [3].

There are totally 60 reported values of GFOCUS for each statement in this test. The average user satisfaction to the determined user focus from GFOCUS (response to Statement 1)) is generally high. Users gave 4.10, 4.00, and 3.80 (approximately "agree") to their satisfaction of the focuses of the queries having high, medium, and low TPM values.

Comparing the response to Statement 2) to the response of the user study of [3], the results show that GFOCUS

16. The questionnaire for the user studies can be found at https://goo.gl/sPNq4u. The agreements to the statements have several advantages over the query formulation time when investigating suggestion qualities. For example, query formulation times depend on the users' knowledge, users' concentration, and the physical environment during the studies, which complicate our study of the systems.

achieves higher user satisfaction for queries having medium and low TPM values (3.30 for GFOCUS and 2.95 for AUTOG on queries having medium TPM values; and 2.10 for GFOCUS and 1.65 for AUTOG on queries having low TPM values). This verifies that GFOCUS outperforms AUTOG in the cases where diversified suggestions from AUTOG are not desirable. For target queries having high TPM values, users gave higher ratings to AUTOG (3.85 for GFOCUS and 4.55 for AUTOG). It is not surprising since those queries are the best simulated queries of AUTOG.

The user's opinion on the suggestion usefulness and the TPM values had a relatively high correlation coefficient of 0.896 with a p-value of 0.016, which shows that TPM is a statistically significant indicator of the suggestion usefulness. The average ratings of the queries with high, medium, and low TPM values were 3.85, 3.30, and 2.10, respectively.

User study 3. Comparison of GFOCUS **and** AUTOG **from domain users (chemists).** We invited three chemists to do the same test as user study 2. For Statement 1), the average user satisfaction to the determined user focus is generally high. The chemists gave 5.0, 4.2 and 3.5 (approximately "agree") to their satisfaction of the focuses of the queries having high, medium, and low TPM values. For Statement 2), the average ratings of the queries with high, medium and low TPM values are 3.7, 3.5 and 3.2, respectively. Compared to the results of the user study reported in [3], the results show that GFOCUS generally achieves slightly higher user satisfaction (3.7 for GFOCUS and 3.5 for AUTOG on queries having high TPM values; 3.5 for GFOCUS and 3.5 for AUTOG on queries having medium TPM values; and 3.2 for GFOCUS and 1.2 for AUTOG on queries having low TPM values).

8.2 Suggestion quality via experiments

For large-scale experiments, we conducted simulations and hereafter mainly used TPM as an indicator of suggestion quality. Each simulation started with a 2-edge subgraph as the initial query. In each step, GFOCUS or AUTOG was invoked. Then, the simulation process simulates users adopting a suggestion at the user focus or adding a manual edge. If useful suggestions were present, the largest one was adopted. If there were ties, they are randomly broken since GFocus always attach subgraphs to the focus. If no useful suggestion was present, a "manual" edge (*e.g.*, the next edge in a canonical DFS order from the last formulated edge) was added.

8.2.1 Validation of the locality principles

To validate the locality principles, we have compared GFOCUS with a naïve user focus. A naïve user focus is computed from a constant attention weight for all query edges. Tab. 6 reports the quality metrics of these two settings. The results show that the user focuses computed from the locality principles clearly resulted in more useful suggestions (U_F by 25%) and saved more mouse clicks (TPM_F by 21%) when compared to the naïve user focus. The U value is around 10% and hence, on average, there was a useful suggestion among the top-10 suggestions.

We tested a large number of τ values and report some representative results in Tab. 7. The results show that there was an optimal range (*i.e.*, [1.01, 1.20]) for good quality suggestions. It validates that suggestions are useful when GFOCUS exploits a user focus. However, as the user focus becomes less localized (*i.e.*, τ is large), the suggestions are less useful.

TABLE 6
Quality metrics by varying user focus

user focus	$\#AUTO_F$	U_F	TPM_F
naïve user focus	4.5	8	43
localities based user focus	5.2	10	52

TABLE 7
Quality metrics by varying τ (PUBCHEM)

τ	1.00	1.01	1.10	1.20	2.00	10	100
$\#AUTO_F$	4.4	5.0	5.0	5.1	5.0	4.5	4.5
U_F	7	10	9	10	9	8	8
TPM_F	43	50	50	51	48	43	43

8.2.2 GFOCUS on query logs

It has been generally known that advanced query autocompletion (*e.g.*, personalized search) exploits query logs. While there have not been public real query logs for subgraph queries, we have voluminous simulated queries. We use them as query logs (*i.e.*, the dataset of GFOCUS) to investigate GFOCUS's performance.

More specifically, we randomly selected target queries of Q20 (*i.e.*, queries of 20 edges) and their intermediate queries in the simulation to form the query logs. The logs contained 1.36K queries, where the average number of vertices and edges were 11 and 10, respectively. The target queries were queries of various sizes from the query logs. We mined features from the logs (minSup = 5%). We used the default settings of GFOCUS. The TPMs under default $\delta_{\max}$ and δ_{fix} are 40 and 49, respectively. Those TPMs of PUBCHEM are 39 and 52, respectively. Hence, the suggestion qualities on the query logs and dataset are similar. Hence, GFOCUS can capture some characteristics of the logs to generate suggestions.

8.3 Online autocompletion efficiency

In this section, we vary some representative parameters and report the *Average Response Time* (ART) of both GFOCUS and AUTOG on PUBCHEM. More efficiency results on PUBCHEM and extensive experimental results on EMOL and AIDS can be found in Appx. B, C, and D of [18].

GFOCUS only computes suggestions at the focus as opposed to the whole query graph. Under the default setting, the ARTs of GFOCUS and AUTOG were 0.63s and 22s, respectively. GFOCUS was 35 times more efficient than the state-of-the-art on average. It can be considered *interactive*.

We further investigated the ART as a function of some important parameters. The ARTs of GFOCUS were always just a fraction of a second. Fig. 11 plots ART as a function of γ, where γ determines how existing queries were analyzed. This only affected the computation of the query decomposition but not the relatively costly candidate suggestion generation and ranking. The ART appeared independent to γ. Similar ARTs can be observed from Fig. 12, as we varied the parameters in the ranking function (β for GFOCUS and α for AUTOG).

The ARTs of suggestion ranking of GFOCUS were stable under various k and $|q|$ values (see Figs. 13 and 14) and GFOCUS is significantly faster. The reason for this is that

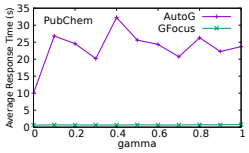


Fig. 11. ART by varying γ

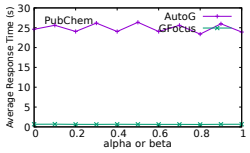


Fig. 12. ART by varying α/β

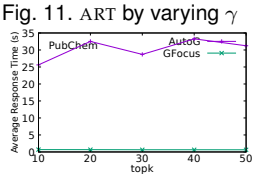


Fig. 13. ART by varying k

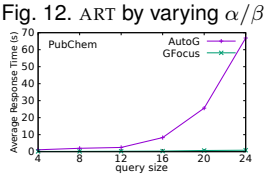


Fig. 14. ART by varying $|q|$

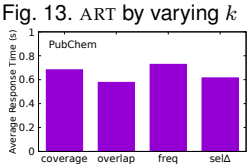


Fig. 15. ART by varying ranking functions

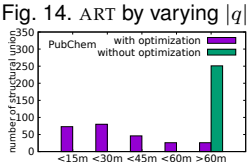


Fig. 16. Efficient Structural Union Computation (offline)

GFOCUS only ranks suggestions at the focus. Many irrelevant suggestions are not even computed. ARTs of GFOCUS were independent of the existing query size $|q|$.

Finally, we evaluate the additional ranking functions under the default settings. The ARTs of the baseline (*i.e.*, coverage), overlap, freq, and sel_Δ were 0.68s, 0.58s, 0.73s, and 0.62s, respectively. That is, all functions return suggestions within 1s, reported in Fig. 15.

9 Related Work

Query formulation aids have recently gained increasing research attention. Firstly, recent work has proposed a variety of innovative approaches to help query formulation. For example, GESTUREQUERY [23] proposes to use *gestures* for specifying SQL queries. *SnapToQuery* [24] guides users to explore query specification via *snapping* user's likely intended queries. [25] has proposed a data-driven approach for GUI construction. Exploratory search has been demonstrated as useful for enhancing interactions between users and search systems (*e.g.*, [26], [27], [28]). QUBLE [29] allows users to explore regions of a graph that contains at least a query answer. SEEDB [30] proposes *visualization* recommendations for supporting data analysis. [31] introduces *Meaningful Query Focus* (MQF) of given keywords to generate XQUERY. While keyword search (*e.g.*, [32]) has been proposed to query graphs, this approach does not allow users to precisely specify query structures. This paper contributes to the *query autocompletion* approach for query formulation.

Secondly, there is existing work on query autocompletion on various query types. For instance, there is work on query autocompletion for keyword search (*e.g.*, [22], [33], [34]) and structured queries (*e.g.*, [35]). Li et al. [36] extended keyword search autocompletion to XML queries. [31] associates structures to query keywords. LotusX provides position-aware autocompletion capability for XML [37]. An autocompletion learning editor for XML provides intelligence autocompletion [38]. [39] presents a conversational mechanism that accepts incomplete SQL queries, which then matches and replaces a part of the previously issued queries. There has been a stream of work on extending Query By Example to construct structural queries, *e.g.*, [40], [41], [42]. In contrast, this paper focuses on structural queries for graphs. Hence, we only include related work on *graphs*.

Regarding query autocompletion on graphs, Yi et al. [3] proposed AUTOG. As motivated, diversified suggestions over the *whole* existing query can be costly to compute yet irrelevant to users. In [43], Li et al. studied the why-not questions over a query autocompletion system. It outputs a small change of the parameters of the ranking function such that the system would return the desired suggestion. In [27], Mottin et al. proposed graph query reformulation, which determines a set of reformulated queries that maximally cover the *results* of the current query. When users start drawing a small query, its results can be many and not all of them are relevant. Pienta et al. [44] and Li et al. [2] demonstrated methods to produce *edge* or *node* suggestions for visual graph query construction. In contrast, this paper considers *subgraph* suggestions.

10 Conclusion

We have proposed GFOCUS that exploits the user focus on a user query being constructed to generate top-k query suggestions to help query formulation. Inspired by HCI research, we have proposed user focus and locality principles for query formulation. Possible query suggestions are represented by a notion of structural union and ranked online. We have proposed optimization and indexing techniques for suggestion ranking. Our user studies and experiments verified both the effectiveness and efficiency of GFOCUS.

This paper leads to a variety of interesting future work. We are extending the study of histories of users' activities [45] (*e.g.*, query logs) into the ranking. We are investigating techniques for GQAC for massive networks. We are studying the explanations of the few cases (*e.g.*, [46]) where GQAC returned incorrect suggestions. As this is the first work on user focus-aware GQAC, we plan to investigate other efficient notions of user focuses.

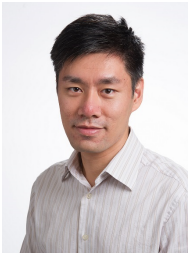
Acknowledgements. This work is partly supported by HKRGC GRF 12258116, 12201119, 12232716, 12201518, 12200817, and 12201018, and NSFC 61602395.

References

[1] S. S. Bhowmick, B. Choi, and C. Li, "Graph querying meets HCI: state of the art and future directions," in *SIGMOD*, 2017, pp. 1731–1736.
[2] N. Jayaram, S. Goyal, and C. Li, "VIIQ: Auto-suggestion enabled visual interface for interactive graph query formulation," *PVLDB*, pp. 1940–1951, 2015.
[3] P. Yi, B. Choi, S. S. Bhowmick, and J. Xu, "Autog: a visual query autocompletion framework for graph databases," *VLDB J.*, vol. 26, no. 3, pp. 347–372, 2017.
[4] M. M. Chun and J. M. Wolfe, "Visual attention," *Blackwell Handbook of Perception*, pp. 272–310, 1999.
[5] R. C. Atkinson and R. M. Shiffrin, "Human memory: A proposed system and its control processes," *The psychology of learning and motivation (Volume 2)*, pp. 89–195, 1968.
[6] A. Baddeley, M. Eysenck, and M. Anderson, *Memory*, ser. Cognitive Psychologie. Psychology Press, 2009. [Online]. Available: https://books.google.com.hk/books?id=3h-BPQAACAAJ
[7] A. Mack and I. Rock, "Inattentional blindness," *MIT Press*, 1998.
[8] W. J. M., "Inattentional amnesia," *In Fleeting Memories. In Cognition of Brief Visual Stimuli*, pp. 71–94, 1999.
[9] C. Roda and J. Thomas, "Attention aware systems: Theories, applications, and research agenda," *Computers in Human Behavior*, vol. 22, no. 4, pp. 557–587, 2006.

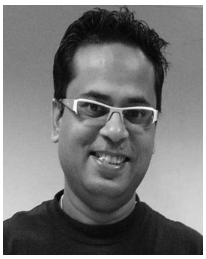
[10] N. Ng, P. Yi, Z. Zhang, B. Choi, S. S. Bhowmick, and J. Xu, "FGreat: Focused graph query autocompletion," in *ICDE*, 2019, pp. 1956–1959.
[11] M. G. Berman, J. Jonides, and R. L. Lewis, "In search of decay in verbal short-term memory." *J Exp Psychol Learn Mem Cogn*, p. 317, 2009.
[12] F. Katsarou, N. Ntarmos, and P. Triantafillou, "Performance and scalability of indexed subgraph query processing methods," *PVLDB*, vol. 8, pp. 1566–1577, 2015.
[13] A. Baddeley, "Recent developments in working memory," *Curr Opin Neurobiol*, pp. 234–238, 1998.
[14] E. Vergauwe and N. Cowan, "A common short-term memory retrieval rate may describe many cognitive procedures," *Front Hum Neurosci*, 2014.
[15] A. W. Madison and A. P. Batson, "Characteristics of program localities," *Commun. ACM*, pp. 285–294, 1976.
[16] P. J. Denning, "The locality principle," *Communications of the ACM*, pp. 19–24, 2005.
[17] A. Heathcote, S. Brown, and D. Mewhort, "The power law repealed: The case for an exponential law of practice," *Psychon Bull Rev*, pp. 185–207, 2000.
[18] P. Yi, B. Choi, Z. Zhang, S. S. Bhowmick, and J. Xu, "Gfocus: User focus-based graph query autocompletion," https://www.comp.hkbu.edu.hk/~bchoi/gfocus-tr-2020.pdf, 2020.
[19] D. S. Hochbaum and A. Pathria, "Node-optimal connected k-subgraphs," *manuscript, UC Berkeley*, 1994.
[20] N. Roussopoulos, "A max{m, n} algorithm for determining the graph h from its line graph c," *Inf. Process. Lett.*, vol. 2, pp. 108–112, 1973.
[21] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *ICDM*, 2002, pp. 721–724.
[22] A. Nandi and H. V. Jagadish, "Effective phrase prediction," in *VLDB*, 2007, pp. 219–230.
[23] A. Nandi, L. Jiang, and M. Mandel, "Gestural query specification," *PVLDB*, vol. 7, no. 4, pp. 289–300, 2013.
[24] L. Jiang and A. Nandi, "Snaptoquery: Providing interactive feedback during exploratory query specification," *PVLDB*, vol. 8, no. 11, pp. 1250–1261, 2015.
[25] S. S. Bhowmick, B. Choi, and C. E. Dyreson, "Data-driven visual graph query interface construction and maintenance: Challenges and opportunities," *PVLDB*, vol. 9, no. 12, pp. 984–992, 2016.
[26] D. Mottin and E. Müller, "Graph exploration: From users to large graphs," in *SIGMOD*, 2017, pp. 1737–1740.
[27] D. Mottin, F. Bonchi, and F. Gullo, "Graph query reformulation with diversity," in *KDD*, 2015, pp. 825–834.
[28] G. Marchionini, "Exploratory search: from finding to understanding," *Commun. ACM*, pp. 41–46, 2006.
[29] H. H. Hung, S. S. Bhowmick, B. Q. Truong, B. Choi, and S. Zhou, "QUBLE: blending visual subgraph query formulation with query processing on large networks," in *SIGMOD*, 2013, pp. 1097–1100.
[30] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis, "Seedb: Efficient data-driven visualization recommendations to support visual analytics," *PVLDB*, vol. 8, no. 13, pp. 2182–2193, 2015.
[31] Y. Li, C. Yu, and H. V. Jagadish, "Enabling schema-free xquery with meaningful query focus," *VLDB J.*, pp. 355–377, 2008.
[32] Y. Wu, S. Yang, M. Srivatsa, A. Iyengar, and X. Yan, "Summarizing answer graphs induced by keyword queries," *PVLDB*, vol. 6, no. 14, pp. 1774–1785, 2013.
[33] H. Bast and I. Weber, "Type less, find more: fast autocompletion search with a succinct index," in *SIGIR*, 2006, pp. 364–371.
[34] C. Xiao, J. Qin, W. Wang, Y. Ishikawa, K. Tsuda, and K. Sadakane, "Efficient error-tolerant query autocompletion," *PVLDB*, pp. 373–384, 2013.
[35] A. Nandi and H. V. Jagadish, "Assisted querying using instant-response interfaces," in *SIGMOD*, 2007, pp. 1156–1158.
[36] J. Feng and G. Li, "Efficient fuzzy type-ahead search in xml data," *TKDE*, pp. 882–895, 2012.
[37] C. Lin, J. Lu, T. W. Ling, and B. Cautis, "LotusX: A position-aware xml graphical search system with auto-completion," in *ICDE*, 2012, pp. 1265–1268.
[38] S. Abiteboul, Y. Amsterdamer, T. Milo, and P. Senellart, "Auto-completion learning for xml," in *SIGMOD*, 2012, pp. 669–672.
[39] Y. E. Ioannidis and S. Viglas, "Conversational querying," *Inf. Syst.*, pp. 33–56, 2006.
[40] S. Comai, E. Damiani, and P. Fraternali, "Computing graphical queries over xml data," *TOIS*, pp. 371–430, 2001.
[41] D. Braga, A. Campi, and S. Ceri, "XQBE (XQuery By Example): A visual interface to the standard xml query language," in *TODS*, 2005, pp. 398–443.
[42] N. Jayaram, M. Gupta, A. Khan, C. Li, X. Yan, and R. Elmasri, "GQBE: Querying knowledge graphs by example entity tuples," in *ICDE*, 2014, pp. 1250–1253.
[43] G. Li, N. Ng, P. Yi, Z. Zhang, and B. Choi, "Answering the why-not questions of graph query autocompletion," in *DASFAA*, 2018, pp. 332–341.
[44] R. Pienta, F. Hohman, A. Tamersoy, A. Endert, S. B. Navathe, H. Tong, and D. H. Chau, "Visual graph query construction and refinement," in *SIGMOD*, 2017, pp. 1587–1590.
[45] A. Zhang, A. Goyal, W. Kong, H. Deng, A. Dong, Y. Chang, C. A. Gunter, and J. Han, "adaqac: Adaptive query auto-completion via implicit negative feedback," in *SIGIR*, 2015, pp. 143–152.
[46] J. Li, Y. Cao, and S. Ma, "Relaxing graph pattern matching with explanations," in *CIKM*, 2017.

Peipei Yi is a Data Scientist at Lenovo Machine Intelligence Center, Hong Kong. He received the PhD and BEng degrees in computer science from Hong Kong Baptist University (HKBU) and University of Electronic Science and Technology of China (UESTC) in 2018 and 2013, respectively. His research interests include graph data processing and graph database usability.

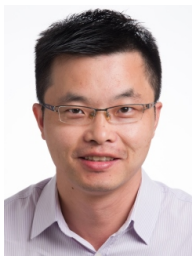


Byron Choi is an Associate Professor in the Department of Computer Science at the Hong Kong Baptist University. He received the bachelor of engineering degree in computer engineering from the Hong Kong University of Science and Technology (HKUST) in 1999 and the MSE and PhD degrees in computer and information science from the University of Pennsylvania in 2002 and 2006, respectively.

Zhiwei Zhang is a Research Assistant Professor in the Department of Computer Science at the Hong Kong Baptist University. He received the Bachelor degree in Computer Science & Technology from Renmin University of China in July 2010 and the PhD degree in Department of Systems Engineering and Engineering Management at the Chinese University of Hong Kong in December 2014.



Sourav S Bhowmick is an Associate Professor in the School of Computer Science and Engineering, Nanyang Technological University. Sourav's current research interests include data management, data analytics, computational social science, and computational systems biology. He has published many papers in major venues in these areas such as SIGMOD, VLDB, ICDE, SIGKDD, MM, TKDE, VLDB Journal, and Bioinformatics.



Jianliang Xu is a Professor in the Department of Computer Science, Hong Kong Baptist University (HKBU). He held visiting positions at Pennsylvania State University and Fudan University. He has published more than 150 technical papers in these areas, most of which appeared in leading journals and conferences including SIGMOD, VLDB, ICDE, TODS, TKDE, and VLDBJ.

Appendices of GFocus: User Focus-based Graph Query Autocompletion

APPENDIX A
PROPERTIES OF GREEDY AND STRUCTURAL UNION

In this appendix, we present the analysis of the properties of GREEDY and structural union.

Proposition 5.1. GREEDY determines the maximal user focus when $\omega(\tau) \geq 0$. □

(Proof sketch) From GREEDY, we can see that i) each expansion of the candidate focus increases its normalized weight; ii) the candidate focus is maximal when GREEDY terminates. In Lines 8-9 of Algo. 2, a focus is only expanded when the normalized weight increased. Lines 6-9 show that the iteration terminates when the focus is maximal, *i.e.*, cannot be expanded and meanwhile their weights are increased. □

Proposition 6.2. [Decomposition of structural union] Given a query q and its query suggestions Q': $\{q'_1, q'_2, \ldots, q'_n\}$ and a new query suggestion q'_{n+1}, $\mathsf{union}(Q' \cup \{q'_{n+1}\})$ is:

$$\mathsf{compose}(\mathsf{union}(Q'), q'_{n+1}, \mathsf{mces}_q(\mathsf{union}(Q'), q'_{n+1}), \lambda_i, \lambda_j), \quad (7)$$

where λ_i (resp.λ_j) is the embedding of the mces_q in $\mathsf{union}(Q')$ (resp. q'_{n+1}); and

$$\mathsf{mces}_q(\mathsf{union}(Q'), q'_{n+1}) = \mathsf{union}(\{\mathsf{mces}_q(q'_{n+1}, q') | q' \in Q'\}). \quad (8)$$

(Proof sketch) Formula 7 rewrites Def. 6.3. Formula 8 is established by arguing $\mathsf{mces}_q(\mathsf{union}(Q'), q'_{n+1})$ contains $\mathsf{mces}_q(q', q'_{n+1})$ for all q' in Q', via a proof by contradiction. □

Proposition 6.3. The RSQ problem is NP-hard. □

(Proof sketch) The maximization of this utility function is NP-hard, by a reduction from the *Set Cover* (SC) problem. Given an instance of SC problem, each subset S^i of elements $\{o^i_1, \ldots, o^i_m\}$ is converted to a candidate suggestion having the edge (r, o_i), where r is an artificial root node; the graph constructed is considered the structural union; and k remains the same. β of RSQ is set to 0; Finding the query suggestion set is then to find the i query suggestions, where i is smaller than or equal to k, that cover the union graph. It can be trivially mapped to the solution of SC, that covers all elements with the smallest number of subsets i. □

Proposition 6.4. util is monotone submodular. □

(Proof sketch) util is *monotone* since $\mathsf{util}(S) \leq \mathsf{util}(T)$ for any suggestion sets S, T such that $S \subseteq T$. A function f is submodular if the marginal gain from adding an element to a set S is at least as high as the marginal gain from adding it to a superset of S. Formally, f satisfies $f(S \cup \{o\}) - f(S) \geq f(T \cup \{o\}) - f(T)$ for all elements o and all pairs of sets $S \subseteq T$. Next, we proof util is *submodular* by contradiction. Assume

$$\mathsf{util}(T \cup \{q'\}) - \mathsf{util}(T) > \mathsf{util}(S \cup \{q'\}) - \mathsf{util}(S)$$

where S and T are sets of suggestions, such that $S \subseteq T$, and $q' \in Q' \setminus T$ is the suggestion being added. By substituting the definition of util to the inequality, we can eliminate the term sel. By simple arithmetic, we have got

$$\mathsf{coverage}(T \cup \{q'\}) - \mathsf{coverage}(T) > \mathsf{coverage}(S \cup \{q'\}) - \mathsf{coverage}(S)$$

By the definition of coverage, we have

$$|\mathsf{union}(T \cup \{q'\})| - |\mathsf{union}(T)| > |\mathsf{union}(S \cup \{q'\})| - |\mathsf{union}(S)|$$

Denote the left-hand and right-hand side of the inequality as Δ_T and Δ_S, *i.e.*, the number of new edges covered by the structural union when adding q' to T and S. We have $\Delta_T > \Delta_S$. By the definition of structural union (Def. 6.3), $\mathsf{union}(S) \subseteq_\lambda \mathsf{union}(T)$ given $S \subseteq T$. Hence we have $\Delta_S \geq \Delta_T$ since the structural union of q' and $\mathsf{union}(S)$ covers more new edges than that of $\mathsf{union}(T)$. This contradicts with $\Delta_T > \Delta_S$. Similar proofs can be established for $\mathsf{overlap}$, freq, and sel_Δ. □

APPENDIX B
ONLINE PERFORMANCE ON PUBCHEM (PUBCHEM)

In this section, we show the online performance of GFOCUS and AUTOG on the PUBCHEM dataset.

Foremost, GFOCUS only computes suggestions at the focus as opposed to the whole query graph. Under the default setting, the ARTs of GFOCUS and AUTOG were 0.63s and 22s, respectively. That is, GFOCUS could be considered *interactive* and was 35 times faster than AUTOG on average.

We further investigated the ART as a function of some important parameters. The ARTs of GFOCUS were always just a fraction of a second. Fig. 17 plots ART as a function of γ, where γ determines how existing queries were analyzed. This only affected the computation of the query decomposition but not the relatively costly candidate suggestion generation and ranking. The ART appeared independent to γ. Similar ARTs can be observed from Fig. 18, as we varied the parameters in the ranking function (β for GFOCUS and α for AUTOG).

The ARTs of suggestion ranking of GFOCUS were stable under various k and $|q|$ values (see Figs. 19 and 20). The reason for this is that GFOCUS only ranks suggestions at the focus. Many irrelevant suggestions are not even computed. Therefore, GFOCUS generated query suggestions significantly faster, and its ARTs were independent of the existing query size $|q|$.

Fig. 21 shows that the ART of GFOCUS gradually increased with the values of δ_{fix}. We note that AUTOG computed its structural diversity faster for large δ_{fix}s, due to its optimized implementation. We plotted the ARTs of GFOCUS as a function of τ in Fig. 22. It shows that the ART of GFOCUS remained roughly constant *w.r.t.* the values of τ.

Finally, we ran the additional ranking functions under the default settings. The ARTs of the baseline, $\mathsf{overlap}$, freq, and sel_Δ were 0.68s, 0.58s, 0.73s, and 0.61s, respectively. That is, all functions return suggestions well within 1s.

APPENDIX C
ONLINE PERFORMANCE ON EMOLECULES (EMOL)

To further demonstrate the online performance of GFOCUS with large datasets, we present detailed evaluations of both GFOCUS and AUTOG on the EMOL dataset under various parameter settings (reported in Figs. 23-28).

Under the default setting, the ARTs of GFOCUS and AUTOG were 2.9s and 45.8s, respectively. This shows that

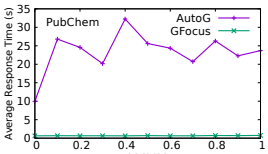


Fig. 17. ART by varying γ

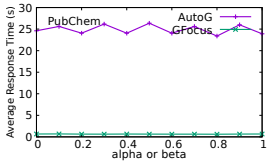


Fig. 18. ART by varying α/β

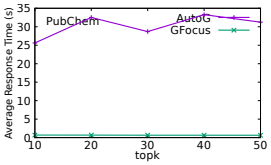


Fig. 19. ART by varying k

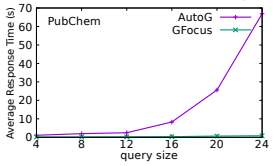


Fig. 20. ART by varying $|q|$

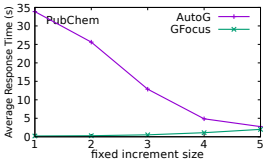


Fig. 21. ART by varying δ_{fix}

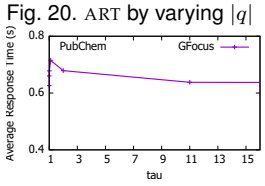


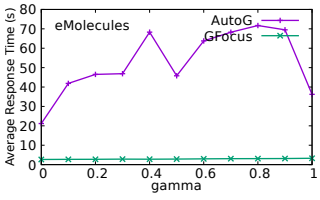
Fig. 22. ART by varying τ

GFOCUS outperformed AUTOG by a factor of 16. The ARTs of GFOCUS were less than 3s under most settings, while AUTOG returned suggestions after more than 20s in most cases. It is clear that GFOCUS is significantly more efficient than AUTOG. The ARTs of GFOCUS and AUTOG by varying some important parameters are reported.

Fig. 23 shows the ARTs of GFOCUS and AUTOG on EMOL by varying the parameter that controls the overlapping of the decomposed query features – namely γ – that is in the range [0,1]. Both GFOCUS and AUTOG tended to take relatively more time to return the suggestions for larger γs because the query (or focus for GFOCUS) may be decomposed into more feature embeddings. The ARTs of GFOCUS appeared to be a small constant, mainly because only the user focus is decomposed followed by the efficient ranking algorithm. Fig. 24 shows the ARTs of GFOCUS and AUTOG when varying the parameters in ranking (β for GFOCUS and α for AUTOG). The ARTs were independent of those parameters.

Fig. 25 and Fig. 26 show the ARTs of GFOCUS and AUTOG by using various k and $|q|$. Again, GFOCUS always returned the suggestions under 3s, while the ARTs of AUTOG increased to several minutes when the values of k or $|q|$ increased. One possible reason is that GFOCUS only ranks suggestions at the focus, which means that GFOCUS can handle large queries more efficiently.

Fig. 27 shows that GFOCUS had smaller ARTs than AUTOG for various suggestion increment sizes. The ARTs of GFOCUS increased with the value of δ_{fix} due to the computation of composing the candidate suggestions (Line 5 of Algo. 3). GFOCUS took a longer time when the increment size became larger. The short ARTs of AUTOG with large increment sizes were possibly due to its optimization when computing the structural diversity. Again, Fig. 28 shows that the ARTs of GFOCUS were constantly below 3s for a wide range of τ values.



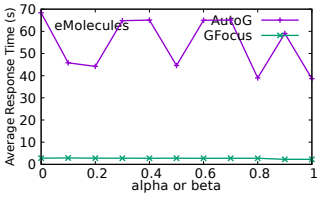
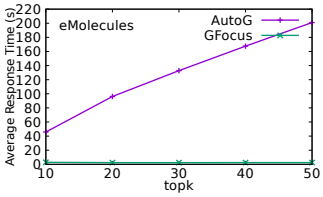


Fig. 23. ART by varying γ

Fig. 24. ART by varying α/β



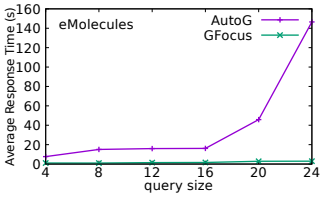
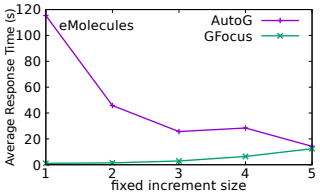


Fig. 25. ART by varying k

Fig. 26. ART by varying $|q|$



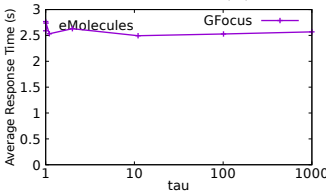
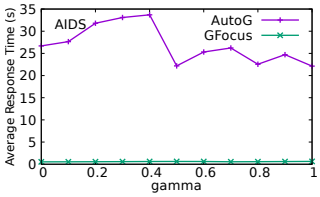


Fig. 27. ART by varying δ_{fix}

Fig. 28. ART by varying τ



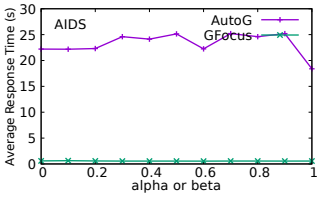
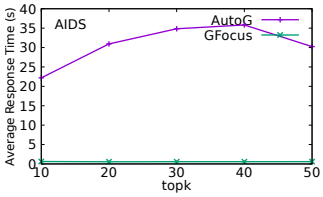


Fig. 29. ART by varying γ

Fig. 30. ART by varying α/β



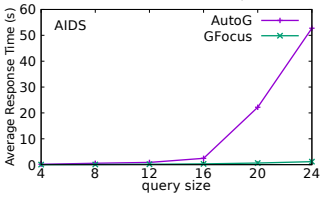
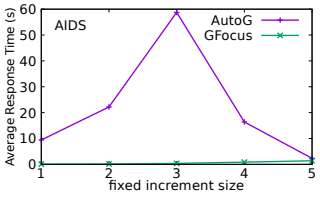


Fig. 31. ART by varying k

Fig. 32. ART by varying $|q|$



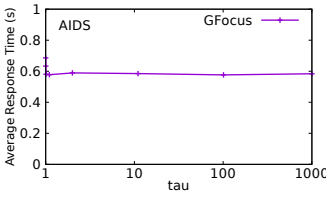


Fig. 33. ART by varying δ_{fix}

Fig. 34. ART by varying τ

APPENDIX D
ONLINE PERFORMANCE ON AIDS (AIDS)

In addition to the two large datasets (*i.e.*, PUBCHEM and EMOL), we report the evaluation of the online performance of both GFOCUS and AUTOG on AIDS when varying some important parameters, reported in Figs. 29-34.

Under the default setting, the ARTs of GFOCUS and AUTOG were 0.6s and 22.2s, respectively. GFOCUS was 37 times faster when compared to AUTOG.

Next, we present a detailed study of ARTs of both prototypes under various settings. In all, GFOCUS almost always returned suggestions within 1s under most settings, while AUTOG took more than 10s in most cases (except for small queries).

Fig. 29 shows the ARTs of GFOCUS and AUTOG on AIDS with different values for γ. The ARTs of GFOCUS and AUTOG both appeared to be independent of the γ setting, possibly because the query decomposition time was small when compared to the relatively costly candidate generation and ranking algorithms. Fig. 30 shows similar ARTs trends of GFOCUS and AUTOG when varying the weights of the ranking functions.

Fig. 31 and Fig. 32 show the ARTs of GFOCUS and AUTOG with various k and $|q|$. GFOCUS returned the suggestions in less than 1s, except that it took 1.2s for queries with 24 edges. We remark that both GFOCUS and AUTOG returned suggestions under 1s for small queries, such as $|q| = 4, 8$, and 12.

Fig. 33 and Fig. 34 show that GFOCUS returns suggestions within 1s for various δ_{fix} and τ settings, except 1.4s when $\delta_{\text{fix}} = 5$. While AUTOG generally returned suggestions after 10s, excepting 2.3s when $\delta_{\text{fix}} = 5$.

APPENDIX E
THE CHARACTERISTICS OF USER ATTENTION WEIGHTS

The user attention weights are defined and maintained by Formulae 2 and 3. The intuitive effects of the formulae are that the edges operated by the last query formulation operator have the highest attention weights.

Suppose the users' last operator is op_t and O_t is the set of edges operated by op_t, where t is the t-th step of the query formulation. An edge $e \in O_t$ got its initial weight w_0. Then, according to the definition of Formula 2, the weight of e at the t-th step is $w_e^t = w_0 \mathsf{e}^{-1/\tau}$. Next, consider an existing edge e, *i.e.*, $e \in q.E$ and $e \notin O_t$. During the decay step (temporal locality), we have the following:

$$\begin{aligned} w_e^{temp} &= w_e^{t-1} \mathsf{e}^{-1/\tau} \\ &\leq w_0 \mathsf{e}^{-2/\tau} \end{aligned}$$

During the propagation step (structural locality), we have the following:

$$\begin{aligned} w_e^t &= w_e^{temp} + \Delta w \mathsf{e}^{-h/\tau} \\ &\leq w_0 \mathsf{e}^{-2/\tau} + (w_0 - w_0 \mathsf{e}^{-1/\tau}) \mathsf{e}^{-h/\tau} \\ &\leq w_0 \mathsf{e}^{-2/\tau} + (w_0 - w_0 \mathsf{e}^{-1/\tau}) \mathsf{e}^{-1/\tau} \\ &\leq w_0 \mathsf{e}^{-1/\tau} \end{aligned}$$

Hence, despite the non-trivial effects of Formulae 2 and 3 on the user attention weights, any edge that is not operated by op_t have a smaller weight than the latest operated edges.

APPENDIX F
THE PSEUDOCODE FOR DETERMINING THE USER FOCUS

This appendix elaborates the pseudocode of GREEDY for determining the user focus presented in Sec. 5. The pseudocode is provided in Algo. 2. Algo. 2 employs a bottom-up strategy to enumerate the candidate focuses. Initially, each edge in the query is considered as a candidate focus, and the user focus is set to one edge with the largest normalized weight (Lines 1-2). Then, in each iteration, GREEDY expands each candidate focus with an edge that makes the normalized weight the largest (Lines 5-7). The expanded candidates with an increased normalized weight are kept for the next iteration (Lines 8-9). The user focus is updated to the expanded candidate if the candidate has a higher normalized weight (Lines 10-11). The next iteration starts after all candidates have been expanded. The iterations terminate when there is no candidate focuses to be expanded (Lines 12-13). At last, the algorithm returns the user focus (Line 14).

APPENDIX G
THE DEFINITION OF SUDAG AND ITS CONSTRUCTION

To be self-contained, this appendix presents the definition of the SUDAG index below.

Definition G.1. SUDAG is a DAG: (V, E, M, A, ζ, D, freq, U, η), where

1) V is a set of index nodes. Each node v represents a feature, denoted as f_v. For clarity of presentation, we may use f_v to refer to the index node;
2) $E \subseteq V \times V$ is a set of edges, $(v_i, v_j) \in E$ *iff* $f_{v_i} \subseteq_\lambda f_{v_j}$. Further, M is a function that takes an edge (v_i, v_j) as input and returns the subgraph isomorphism *embedding*s of f_{v_i} in f_{v_j}, denoted as M_{f_i,f_j};
3) A takes a feature f_v as input and returns the automorphism embeddings of f_v, denoted as A_{f_v}, where A is used to prune structurally identical composed graphs;
4) ζ is a function that takes a feature f_v as input and returns a set of composition records $\mathcal{C}$ as output, where each record in $\mathcal{C}$ is a 6-ary tuple (f_v,f_{v_j},cs, λ_v,λ_{v_j},F_l), where cs is a common subgraph of f_v and f_{v_j},λ_v (resp. λ_{v_j}) is the embedding of cs in f_v (resp. f_{v_j}) and F_l is the set of features embedded in the composed graph;
5) D takes an index node f_v as input and outputs the IDs of the graphs that contain f_v. We denote the graph IDs of f_v as D_{f_v};
6) freq takes an edge as input and returns the number of graphs contain it in the database; and
7) Each index node f_v also stores the structural union U_{f_v} of all possible compositions with it. η takes a composition record c =(f_v, f_{v_j}, cs, λ_v, λ_{v_j}, F_l) as input, and returns the embedding $M(q'_c, U_{f_v})$ of the composed graph in the structural union U_{f_v}. □

Next, we present the construction of SUDAG (shown in Algo. 4). The SUDAG construction exploits existing algorithms (Lines 1-2) of AUTOG to construct the feature DAG topology and enumerate the query compositions (the valid combinations of two feature graphs). However, it does not require the costly computation of maximum common subgraph (mces_aux) anymore after the offline enumeration. What is unique in Algo. 4 is that it precomputes the structural union of the possible query compositions of each feature (Lines 3-4) and indexes the structural union graph

Algorithm 4 Index Construction

Input: graph database D, feature set F, max. increment size δ
Output: SUDAG I

1: Construct the feature DAG topology (Algo. 3 in AUTOG [3])
2: Enumerate feature-pair compositions (Algo. 4 in AUTOG [3]) without computing mces_aux
3: **for all** $f \in F$ **do**
4: $f.U_f \leftarrow$ UNION({compose(c) for $c \in \zeta(f)$})
5: **return** I
6: **function** UNION($Q'_\mathcal{C}$)
7: Denote $Q'_\mathcal{C}$ as the composed graphs $\{q'_1, q'_2, \ldots, q'_n\}$
8: Let q'_i be the smallest composed graph in $Q'_\mathcal{C}$ // for efficiency
9: Let $U_f \leftarrow q'_i$
10: **for all** $q'_c \in Q'_\mathcal{C}$ and $q'_c \neq q'_i$ **do**
11: $U_f \leftarrow$ union(U_f, q'_c) // optimized using Prop. 6.2
12: **return** U_f

TABLE 8
Default parameters

Parameter	Range	Default	Meaning
τ	[1,1000]	1.2	user's relative memory strength
γ	[0,1]	0.5	degree of overlapping features in q
$\delta_{\max}$	[1,5]	5	maximum increment size
δ_{fix}	[1,5]	2	fixed increment size
α	[0,1]	0.1	weighting factor for AUTOG
β	[0,1]	0.1	weighting factor for GFOCUS
k	[4,50]	10	number of suggestions
m	[1,16]	4	selectivity sampling interval
$\|q\|$	[4,24]	20	target query size
$\|D_q^{\min}\|$	[1,1000]	10	minimum result set size

U_f and the embedding of the composed graph for each composition (in U_f) in SUDAG I.

More specifically, the UNION function exploits the decomposition property of the structural union (Prop. 6.2) and therefore, avoids the costly mces_q computation between large graphs. In Line 8, the algorithm selects the smallest composition q'_i from the possible query compositions for efficiency. Then, in Lines 10-11, it iteratively unions one composed graph with the union graph U_f to yield the final structural union.

The runtime of Algo. 4 is mainly determined by (i) the number of compositions based on the feature and (ii) the time to compute the structural union of composition pairs. Denote the time to compute the mces of two compositions as $O(T_{\text{mces}})$. Denote $O(|Q'_\mathcal{C}|)$ to be the number of possible query compositions of each feature. The algorithm takes $O(|F| \times |Q'_\mathcal{C}|^2 \times T_{\text{mces}})$ to compute the structural union for each feature. The worst-case space complexity is $O(|F| \times |Q'_\mathcal{C})| \times |f|)$ for the structural union graphs. We observed from our experiments that the size of the structural union graphs typically ranged from hundreds of edges.

APPENDIX H
DEFAULT PARAMETERS

The optimal parameters for GFOCUS are dataset-specific. We ran extensive simulation tests on the parameters and determined their default values. Tab. 8 shows a summary of the default values for the parameters. These values are obtained from large-scale performance simulations of the datasets (Tab. 2) and selected the robust ones. We stick to these default settings throughout the experimental evaluation unless otherwise specified.

TABLE 9
Comparison of the TPM of GFOCUS with optimal TPM

δ_{fix}	TPM_F	TPM_{OPT}
1	58	75
2	52	88
3	44	92
4	37	93
5	35	94

We remark that existing techniques could help to tune the parameters, *e.g.*, applying machine learning techniques [38] to tune α and β automatically and conducting pretest of user's memory strength to set τ properly.

APPENDIX I
OPTIMAL TPM

In this appendix, we illustrate the optimal values of TPM, which is extensively used in the experiments (in Sec. 8) using Formula 9.

$$\text{TPM}_{\text{OPT}}(q, \delta_{\text{fix}}) = \frac{(|q.E| + |q.V| - 5) \times 2 - \lceil \frac{|q.E|-2}{\delta_{\text{fix}}} \rceil}{(|q.E| + |q.V| - 5) \times 2}, \quad (9)$$

where the denominator is the number of clicks without autocompletion (5 is the number of vertices and edges from the *initial query* used in the experiments, 2 clicks are needed for adding each vertex and edge, since their labels are considered), and the numerator is the number of clicks saved by autocompletion ($\lceil \frac{|q.E|-2}{\delta_{\text{fix}}} \rceil$ is the number of adoptions of δ_{fix} sized increments needed to correctly formulate the target query from the initial query).

Tab. 9 shows the TPMs of the optimal autocompletion determined by Formula 9 and the TPMs of GFOCUS on the default query size ($|q|$ = 20). These numbers indicate that GFOCUS saved the majority manual clicks even when compared to an optimal autocompletion system (*e.g.*, when $\delta_{\text{fix}} = 1$ or 2). It is expected that the TPMs of GFOCUS will drop as δ_{fix} increases since it is much harder to provide accurate suggestions with large increments.

APPENDIX J
FURTHER EXPERIMENTS ON PARAMETERS

We studied the effects of the major parameters of GFOCUS and compared the results of GFOCUS with AUTOG. We reported the representative simulation results in Tabs. 10-15. The performance characteristics presented here can be helpful for users to set their default parameter values, which are dataset-specific. For ease of comparison, we used PUBCHEM. We note that GFOCUS can run simulations on the larger dataset EMOL, but AUTOG cannot finish some of them within 5s.

Varying the maximum increment sizes ($\delta_{\max}$). Tab. 10 shows the quality metrics of Q20 with various $\delta_{\max}$. The results show the qualities decrease as $\delta_{\max}$ increases. (The same trend can be observed from other query sets.) (i) From $\#\text{AUTO}_G$ and $\#\text{AUTO}_F$, we note that GFOCUS achieved more adoptions than AUTOG when the $\delta_{\max}$ value was 1 or 2. (ii) From Δ_G and Δ_F, we observe that the average increment sizes of the adopted suggestions of GFOCUS were roughly 13% larger than those of AUTOG. (iii) The difference

TABLE 10
Quality metrics by varying δ_{max} (PUBCHEM)

δ_{max}	#AUTO$_G$	#AUTO$_F$	Δ_G	Δ_F	TPM$_G$	TPM$_F$	TPM$_{\Delta\%}$
1	12.1	13.6	1.0	1.0	50	58	16
2	5.6	5.9	1.7	1.9	45	54	19
3	3.4	3.4	2.2	2.7	36	47	28
4	2.5	2.6	2.7	3.2	32	43	32
5	2.2	2.2	3.2	3.7	32	39	22

TABLE 11
Quality metrics by varying δ_{fix} (PUBCHEM)

δ_{fix}	#AUTO$_G$	#AUTO$_F$	U$_G$	U$_F$	TPM$_G$	TPM$_F$	TPM$_{\Delta\%}$
1	12.1	13.6	19	18	50	58	16
2	4.6	5.2	8	10	44	52	16
3	2.2	2.8	3	5	34	44	27
4	1.5	1.8	2	3	32	37	15
5	1.0	1.3	1	2	28	35	25

TABLE 12
Quality metrics by varying $|q|$ (PUBCHEM)

$\|q\|$	#AUTO$_G$	#AUTO$_F$	U$_G$	U$_F$	TPM$_G$	TPM$_F$	TPM$_{\Delta\%}$
8	1.2	1.4	3	5	36	41	13
12	2.5	2.8	6	9	44	50	12
16	3.6	4.0	7	9	45	51	13
20	4.6	5.2	8	10	44	52	16
24	5.3	6.4	7	10	43	53	23

in TPM$_G$ and TPM$_F$ (*i.e.*, TPM$_{\Delta\%}$) show that GFOCUS offered a 16%-32% improvement in TPM.

We remark that even #AUTO$_G$ and #AUTO$_F$ were roughly the same when δ_{max} was 3 to 5, GFOCUS exhibited better performance. The reason is that its suggestions were generally with larger increment size which led to more mouse clicks being saved.

Another remark on Tab. 10 is that δ_{max} has a significant impact on the qualities in a non-trivial way, *e.g.*, small suggestions were more likely to be useful but large ones saved more mouse clicks. To illustrate the suggestion qualities more clearly, we used suggestions of the fixed increment size (δ_{fix}) in subsequent experiments.

Varying the fixed increment sizes (δ_{fix}). Tab. 11 shows the quality metrics of Q20 with various δ_{fix}. They verify that the quality decreased as δ_{fix} increased. #AUTO$_G$ and #AUTO$_F$ show that GFOCUS on average resulted in 19% more suggestion adoptions. U$_G$ and U$_F$ show GFOCUS generally produced more useful suggestions. TPM$_G$ and TPM$_F$ show that GFOCUS outperformed AUTOG by 20% on average.

Varying the target query sizes ($|q|$). Tab. 12 shows the quality metrics of various $|q|$. It is not surprising that both GFOCUS and AUTOG achieved more suggestion adoptions as $|q|$ increased. Importantly, GFOCUS resulted in 14% more adoptions on average than AUTOG. GFOCUS returned 35% more useful suggestions on average when compared to AUTOG. GFOCUS outperformed AUTOG in TPM by 16% on average.

Varying the user-specified constraint k. Tab. 13 shows the quality metrics with various k values (*i.e.*, the number of suggestions returned in each iteration). The results show the qualities increased with k. The relative performances of GFOCUS were better when k was small. This suggests that user focuses were particularly vital when only a few suggestions were allowed. GFOCUS resulted in 15% more adoptions on average than AUTOG. GFOCUS returned 28% more useful suggestions on average when compared to AUTOG. GFOCUS outperformed AUTOG in TPM by 21% on average.

TABLE 13
Quality metrics by varying k (PUBCHEM)

k	#AUTO$_G$	#AUTO$_F$	U$_G$	U$_F$	TPM$_G$	TPM$_F$	TPM$_{\Delta\%}$
4	3.2	3.8	9	11	30	38	25
6	3.8	4.4	8	11	36	44	21
8	4.2	4.9	8	10	40	49	20
10	4.6	5.2	8	10	44	52	16

TABLE 14
Quality metrics by varying β or α (PUBCHEM)

β	#AUTO$_F$	U$_F$	TPM$_F$
0.00	3.3	3	32
0.02	5.0	8	50
0.04	5.1	8	51
0.06	5.1	9	51
0.08	5.1	9	51
0.10	5.2	10	52
0.20	5.1	10	51
0.40	5.1	10	51
0.60	5.1	10	51
0.80	5.1	10	51
1.00	4.9	10	49

α	#AUTO$_G$	U$_G$	TPM$_G$
0.00	4.0	4	39
0.02	4.7	7	45
0.04	4.8	7	46
0.06	4.7	7	45
0.08	4.5	7	44
0.10	4.6	8	44
0.20	4.4	8	43
0.40	4.4	9	43
0.60	4.3	9	41
0.80	4.4	9	42
1.00	4.4	9	42

TABLE 15
Quality metrics by varying $|Dq|$ (PUBCHEM)

$\|D_q^{\min}\|$	#AUTO$_G$	#AUTO$_F$	U$_G$	U$_F$	TPM$_G$	TPM$_F$	TPM$_{\Delta\%}$
1	4.2	5.0	7	10	41	50	22
10	4.6	5.2	8	10	44	52	16
100	5.0	5.5	8	11	48	55	14
1000	4.9	5.8	8	11	48	58	20

Varying ranking functions. In the absence of user provided application-specific information, we ran the proposed additional ranking functions under the same default simulation setting. All three additional functions achieved consistently high quality, *i.e.*, approximately 5.2, 10%, and 52% for the quality metrics #AUTO$_F$, U$_F$, and TPM$_F$.

Varying the parameter β and α in the ranking functions. Tab. 14 shows the quality metrics of GFOCUS under various β values. We ran a similar experiment on various α values with AUTOG and reported the results in Tab. 14. The qualities of GFOCUS are consistently high and stable. On average, GFOCUS returned 12% more useful suggestions and saved 14% more mouse clicks when compared to AUTOG.

Varying the minimum result set size of the target queries ($|D_q^{\min}|$). Tab. 15 shows the quality metrics of GFOCUS and AUTOG on query sets of different minimum result set sizes. The results show that both GFOCUS and AUTOG resulted in higher quality suggestions when the target queries yielded more answers (larger $|D_q^{\min}|$). These results are consistent to query autocompletion in the context of web search, suggesting that queries that yield many web pages are suggested, whereas those with fewer web pages are often omitted.