ABSTRACT

Composing queries is evidently a tedious task. This is particularly true of graph queries as they are typically complex and prone to errors, compounded by the fact that graph schemas can be missing or too loose to be helpful for query formulation. Automatic query completion, graph query autocompletion has received increasing research attention to alleviate users from the potentially painstaking task of graph query formulation. In this demonstration, we present a novel interactive user-focus based visual subgraph query autocompletion framework. Given a large collection of small or medium-sized graphs and a visual query fragment \( q \) constructed by a user, we return top-\( k \) query suggestions at the predicted user focus. We demonstrate that user focus exists in visual subgraph query formulation and is effective to enhance query autocompletion.

1. INTRODUCTION

Being a proficient query writer is essential for formulating a graph query. Learning a query language can be a difficult task for consumers of graph applications that may have different backgrounds. Therefore, visual tools are proposed to provide users a handy way to interactively construct queries. Nonetheless, query formulations under a visual environment yet still require lots of human efforts, which are verbose and tedious. As such, graph query autocompletion framework (e.g., AUTOG) is proposed to alleviate the burdens of structural query formulation. AUTOG takes an initial query as input and adds subgraph increments in arbitrary places of the query. Moreover, based on parameter settings provided by the user, it returns a ranked top-\( k \) query suggestion as output. One may iteratively invoke and choose a query suggestion generated by AUTOG to compose their queries. However, according to the research on human-computer interaction (HCI), humans can only focus on several recent software artifacts in hand. Users may only wish to extend and work on a small fraction of the query, which implies that many of the arbitrary suggestions could be irrelevant. Going through these suggestions may distract users. Consequently, we propose to identify user’s focus. The aim is to enhance the effectiveness of the current state-of-the-art of graph query autocompletion.

To address the aforementioned challenge, we determine user’s focus by proposing two novel frameworks — gFOCUS and MFOCUS. These two frameworks take different user’s implicit information as input and can be applied in different circumstances. gFocus derives a user’s focus by leveraging temporal locality principle and structural locality principle, inspired by HCI research. It takes user query and query formulation sequence as input, and determines the possible focus by applying the locality principles. User’s focus is automatically identified and maintained alongside with the query construction process of the current query. It requires no user explicit information and works regardless of input devices (e.g., touch screen or personal computers). On the other hand, MFOCUS determines user’s focus by capturing user’s mouse cursor position at the visual environment. Specifically, it exploits the real-time position of both the query graph and mouse cursor, then predicts a possible focused subgraph by calculating the distance from user’s cursor to each query node. Furthermore, through utilizing the mouse cursor coordinates, arbitrary shifts of user’s query focus can be detected. "Out of focus" status of user’s attention can also be well modeled when the user moves the cursor away from the visual editor.

We performed an experiment to verify the effectiveness of gFOCUS and MFOCUS. Our experimental results reflect that identifying user’s focus can boost up both the effectiveness and efficiency of the current state-of-the-art system. When compared to the state-of-the-art (e.g., AUTOG, which do not consider user focuses), gFOCUS and MFOCUS save more than 25% mouse clicks in query formulation. Our demonstration system’s architecture employs a client-server architecture and can be broken down into several modules, for example, client side modules, server side offline modules and server side online modules. We propose a ranking function to rank the query suggestions returned by MFOCUS to make the suggestions fit with the user’s preferences and reduce the number of irrelevant suggestions returned. According to the simulation and a user test, they reveal that user focus does exist in most cases since specifying user’s focus can bring a huge improvement in terms of mouse clicks needed in constructing a query graph. In addition, MFOCUS saves 59% of clicks on average. It indicates that our user-focus based methods are clearly effective for graph query autocompletion. Our demonstration focuses on illustrating how specifying user’s focus can enhance the performance on graph query autocompletion. Scenarios such as user’s focus shift, automatically determine user’s focus by previous formulation.

\(^1\)Out of focus status: no focus node if the cursor is out of distance from the current query (e.g., not inside the visual editor). Users may sometimes shift their attention occasionally, i.e., starting a new subgraph (which is disconnected from the existing query) at some empty space in the visual editor and connect them later.
2. **SYSTEM ARCHITECTURE**

Figure 1 shows the major modules of the demonstration prototype, namely: 1. client side modules at the client side, 2. online modules and 3. offline modules at the server side.

### 1. User interface at the client side.

The client side modules mainly provide a user interface where one can interact with the system. Figure 2 depicts the screenshot of the mFOCUS visual interface. Panel 1 provides various target graph database for users to choose from. It lists a set of node labels and edge labels for users to construct their queries. In Panel 2, users may interact with the slides to tune the parameters of query suggestion ranking, depending on their own preferences. The Visual Graph Editor in Panel 3 is the area used to construct a query graph. Users may click on the empty space of the editor to add a new node, and drag from one node close to another to form a new edge. During the query graph construction process, one may obtain query suggestions by clicking Autocomplete button, or moving the cursor to the part that he/she wishes to extend and press ENTER to invoke mFOCUS.² Information such as User’s cursor coordinates, parameters setting

²For experiment purposes, the current prototype requires users to click the Autocomplete button for suggestions and press ENTER to capture the mouse location. Hence, users know exactly the output of a query suggestion step.

and query graph on Visual Graph Editor will then be sent to the server side and perform query autocompletion. After that, the Suggestion Visualizer located in panel 4 displays the relevant suggestions in real time. To show different components of the suggestions, for mFOCUS, we highlight the focused feature in red, and the incremented part in blue. Conversely, for query autocompletion, only the existing query is colored in light grey. If one of the suggestions returned is correct, users may accept it by simply clicking on it. The Subgraph Query Results of the current query graph are retrieved from the server and displayed on the visual interface when the Submit Query button is clicked.

### 2. Query analysis for focus.

The Autocomplete Query Processor module at the server side accepts user’s cursor coordinates, parameters setting and current query graph information (e.g., structure of q with labels, coordinates of nodes on the canvas) as input, and produces query suggestions as output.

In the first step, **User Focus Determination** determines the user’s query focus by analyzing the information mentioned above. The server calculates the Euclidean distance from user’s cursor to each node, and set the node with the smallest distance as focus node \( f_v \). This focus node \( f_v \) will be used as a factor for feature selection in later steps. The rationale of setting the closest node to cursor as focus node \( f_v \) is that mouse cursor position on the canvas directly reflects user’s attention on query graph, and the node that is closest to the mouse cursor has the highest chance to be the location where the user focuses their attention on. Further, the server decomposes the query into a feature set with respect to the features \( F \) computed offline. The output of the above process is a focus node \( f_v \) and a set of decomposed feature \( F \). In contrast, \( qFocus \) estimates user’s focus by automatically capturing the user query formulation sequence and using locality principles from HCI to determine and monitor user’s attention.

Next, in the **Feature Selection** step, we first use the focus node \( f_v \) and decompose the query into a feature set, where features are subgraphs that carry important characteristics of the data graphs. In this demonstration, we adopt the feature definition of the previous work [6]. We propose a two-level filtering algorithm to avoid determining a large amount of computationally intensive feature embeddings. For the first-level filtering, we filter out decomposed features that do not contain a node with \( I(f_v) \). Next, the server determines the embeddings (i.e., the location) of the extracted features remained after the filtering by invoking an efficient subgraph isomorphism algorithm called VF2. For the second-level filtering, we filter out the decomposed feature embedding that does not cover \( f_v \). Through skipping the unnecessary embeddings determination process for decomposed features that will not be user’s focus by implementing two-level filtering, we can speed up the overall Feature Selection performance. Finally, we choose the smallest decomposed embedding as output (user’s focus). If there are more than one valid feature embeddings have the smallest size, we randomly choose one between them.

### 3. Candidate generation and Suggestion ranking.

After feature selection, the query \( q \) is represented by a feature and its embedding. Then, in the **Candidate generation**
step, the server generates a set of candidate query suggestions. A candidate query suggestion is formed by adding a subgraph increment (which is a feature mined offline) into the focus feature of the current query \( q \). Different from the previous work, query increment is only added to the focus. We adopt an existing pruning technique to prune all empty suggestions with small subgraph increments, whereas the coverage of the suggestions set being composed can still be exponential, which is hard for the user to interpret.

At last, in the Suggestion Ranking step, candidate suggestions are ranked by the processor. Since users may only be able to interpret a small subset of the candidate suggestions, MFOCUS returns only top-k suggestions w.r.t. user intent value \( \alpha \) and \( \beta \). It should be remarked that the ranking function \( \text{util} \) is designed to let the user quickly and efficiently find suggestions generated by \( \text{MFOCUS} \). The user can then select a set of suggestions to form the query.

Given a set of suggestions \( Q' = \{q'_1, q'_2, \ldots, q'_k\} \) and user preference \( \alpha \) and \( \beta \), the user intent value of \( Q' \) (\( \text{util} \)) is defined as follows:

\[
\text{util}(Q') = \alpha \sum_{q' \in Q'} \text{sel}(q') + \beta \sum_{q' \in Q'} \text{score}(q') + (1 - \alpha - \beta) \text{coverage}(Q'),
\]

where \( \alpha \in [0, 1], \beta \in [0, 1] \) and \( (1 - \alpha - \beta) \) is a parameter to control the slack between the \( \text{sel} \) and \( \text{score} \) contributions.

The sel function is the selectivity of \( q' \) on \( D \). The score function \( \text{score}(q') \) is defined as the \( \text{score}(q') \) of \( q' \) in the universe (union of all suggestions) that the suggestions cover. One should be reminded that the \( \alpha \) and \( \beta \) are different from that in \( \text{AUTOG} \) and \( \text{GFOCUS} \). Furthermore, \( \text{util} \) is a monotonic submodular function, which is often used to approximate the optimal solution.

### 3. EVALUATION

We have evaluated the quality of the suggestions via simulation using queries [2] with 20 edges. For each target query, we determine a location for each query node to simulate the visual graph editor environment using Fruchterman-Reingold force-directed algorithm [1]. During the simulation, the nodes of the current query will inherit the location of its corresponding node in the target query. Each simulation started with a 2-edge subgraph (e.g., the first 2 edges in the DFS traversal order from the first formulated vertex) as the initial query. In each iteration, the location of the mouse cursor is the same as the source node of the next manual edge. If there are useful suggestions returned, the one with the largest increment size is adopted. If no suggestion is useful, the query is augmented with a “manual” edge (e.g., the next edge in the DFS traversal order from the first formulated vertex) towards the target query. The simulation terminates when reaching the target query.

**Suggestion quality.**

For the simulation, we used PubChem [4] as our target graph database. We adopted several popular metrics for suggestion qualities [6, 3]. We report the number of suggestion adoptions (i.e., \#AUTO), the increment size of the adopted suggestions (i.e., \( \Delta \)) and the total profit metric (i.e., \( \text{TPM} \)). The difference of \( \text{TPM}_F \) and \( \text{TPM}_M \) (i.e., \( \text{TPM}_{\Delta M} \)) is also listed. Each reported number is the average of the 100 queries in each query set. Note that all target queries are 20-edge graphs, which are publicly available [5].

We report the suggestion quality of \( \text{AUTOG} \), \( \text{GFOCUS} \) and MFOCUS in Table 1 and Table 2, with varying \( \alpha \) and \( \beta \) settings to see the effects of \( \text{MFOCUS} \) and \( \text{GFOCUS} \) under a large variety of ranking settings. We remark that the optimal TPM with \( \delta_{\text{fix}} = 0 \) or \( \delta_{\text{max}} = 2 \) is 88%.

Table 1 shows the quality metrics with various \( \alpha \). From \( \text{TPM}_G \) to \( \text{TPM}_M \), we observe that the quality of query suggestions is significantly improved, which implies that user focus exists in graph query autocompletion, and specifying user focus is helpful to improve the effectiveness of the system. Moreover, \( \text{TPM}_{\Delta M} \) shows that MFOCUS can capture user’s focus more precisely than \( \text{GFOCUS} \). Given that MFOCUS captures user’s implicit expression of his/her attention on the visual graph interface, whereas \( \text{GFOCUS} \) determines user’s focus only by analysing previous query formulation sequence, occasional shifts of user’s attention on query graph can be better captured by using MFOCUS. The result also showed that \( \text{TPM} \) for both \( \text{GFOCUS} \) and MFOCUS the suggestion qualities increased as the value of \( \alpha \) increased and are stable when \( \alpha > 0.1 \).

Table 2 shows the quality metrics under various settings of \( \beta \). Different from Table 1, in here, we use \( \delta_{\text{fix}} \) instead of \( \delta_{\text{max}} \) to show the effects of \( \text{score}(q') \) when we vary \( \beta \). We observe that when we increase \( \beta \), the number of adoption \#AUTO also increased. This is because increasing \( \beta \) will return more small suggestions, and small suggestions have higher chance to be a part of the target query and hence be adopted by users. Similarly, the suggestion qualities reach the peak and remain identical when \( \beta > 0.1 \).

### 4. DEMONSTRATION DESCRIPTION

The key objective of the demonstration is to let the attendees interactively experience the following features through the MFOCUS GUI. A video of MFOCUS can be found at https://goo.gl/R9xzXZ.

**Interactive experience of autocompletion with query focus during query formulation.** In our demonstration, one will be able to formulate their query by adopting query suggestions generated by MFOCUS/GFOCUS during the query graph construction process. Through utilizing both cursor coordinates and query position on the canvas, user’s focus
can be successfully captured using MFOCUS. Even without cursor coordinates as input (e.g., using our system on a touch screen device), user focus can still be estimated using gFocus. Hence, with the advantages brought by specifying user’s focus, one may construct a subgraph query with significantly fewer clicks when comparing to the existing work (e.g., AUTOG, graph query autocompletion without concerning about user focus). An attendee will also be able to experience the faster query suggestion time and the improved quality of top-k suggestions generated with the assist of query focus identification.

Interactive experience of the effect of different parameters. With different parameters settings, one may tune their preference on selectivity, subgraph increment size and structural coverage of top-k suggestions. For instance, by increasing alpha, users can prioritize query suggestions with higher selectivity, meanwhile decreasing the importance of subgraph increment size and coverage. During the demonstration, an attendee will also be able to use various settings of parameters associated with the query suggestion generation process through the sliding bar at the left bottom corner of the GUI and interactively experience their impact on the query suggestions. The detailed changes (e.g., average subgraph increment size) led by altering parameters’ values will be shown in the demonstration as well.

5. CONCLUSION

In this paper, we present a novel technique to determine user’s focus in graph query autocompletion, namely MFOCUS. User’s cursor position is utilized as the main factor for determining user’s focus. We calculate the distance from user’s cursor to each node in the visual graph editor. We set the nearest node as focus node $v_f$, and only choose the smallest decomposed feature that covers $v_f$. Features (subgraph increment) is only added to the focused feature. We propose a ranking function to let user reflects his/her intent in query suggestions. We conducted an experiment to evaluate the effectiveness of MFOCUS. The existence of user focus in graph query autocompletion can be affirmed based on the clear improvement in terms of TPM when comparing to graph query autocompletion without focus. The results also show that MFOCUS can save 59% of clicks on average, which means our user-focus based method is effective in graph query autocompletion. For the demonstration, we aim to let users have interactive experience of autocompletion with query focus. We showed different user-focus based scenarios such as shifting of user’s attention on different parts of the query graph, determining user’s focus automatically according to previous query formulation sequence, and displaying the effects of changing parameters setting in suggestions ranking function.

6. FUTURE WORK

In practice, users may have preferences on query suggestions when using our proposed graph query autocompletion techniques. However, due to the improper parameters setting, the ranking of expected suggestions may be too low and thus not included in the returned suggestions set. In the future, we will explore Why-not queries on the existing graph autocompletion platform (i.e., MFOCUS or gFOCUS) to improve database usability and user experience by applying an optimal adjustment of the weights (e.g., $\alpha$ and $\beta$) in the suggestion ranking function.

7. REFERENCES