DB $\bowtie$ HCI: Towards Bridging the Chasm Between Graph Data Management and HCI

Sourav S Bhowmick

School of Computer Engineering, Nanyang Technological University, Singapore
assourav@ntu.edu.sg

Abstract. Visual query interfaces enable users to construct queries without special training in the syntax and semantics of a query language. Traditionally, efforts toward such interface design and devising efficient query processing techniques are independent to each other. This is primarily due to the chasm between HCI and data management fields as since their inception, rarely any systematic effort is made to leverage techniques and principles from each other for superior user experience. In this paper, we lay down the vision of bridging this chasm in the context of visual graph query formulation and processing. Specifically, we present the architecture and novel research challenges of a framework for querying graph data visually where the visual interface management is *data-driven* and query processing and performance benchmarking are HCI-*driven*.

1 Introduction

"*Thirty years of research on query languages can be summarized by: we have moved from SQL to XQuery. At best we have moved from one declarative language to a second declarative language with roughly the same level of expressiveness. It has been well documented that end users will not learn SQL; rather SQL is notation for professional programmers.*"

Abiteboul et al. [1]

"*A picture is worth a thousand words. An interface is worth a thousand pictures.*"

B. Shneiderman, 2003

It is widely acknowledged that formulating a textual query using a database query language (*e.g.,* XQuery, SPARQL) often demands considerable cognitive effort from end users [1]. The traditional approach to alleviate this challenge is to improve the user-friendliness of the task by providing a visual querying scheme to replace data retrieval aspects of a query language. This has paved the way to a long stream of research in visual query languages in the context of relational databases [4, 30], object-oriented databases [10, 15], the Web [5, 13], semi-structured and XML databases [9, 14, 23], and graph databases [7, 8, 12]. These approaches leverage principles from the human-computer interaction (HCI) field to build visual query interfaces that involve a sequence

of tasks ranging from *primitive* operations such as pointing and clicking a mouse button, keyboard entry to higher-level tasks such as selection of menu items, selection and dragging of target objects, etc. In this paper, we leverage on this grand desire to provide user-friendly visual querying interface to lay down the vision of *integrating* DB and HCI (hereafter, referred to as *HCI-aware data management*) techniques and tools towards superior consumption and management of data.

Specifically, we discuss HCI-aware data management in the context of graph-structured data. This is because many modern applications (*e.g.,* drug discovery, social networks, semantic Web) are centered on such data as graphs provide a natural way of modeling data in a wide variety of domains. For example, searching for *similar* chemical compounds to aid drug design is essentially searching for graphs as it can be used to represent atoms and bonds in chemical compounds. Consequently, there is a pressing need to build user-friendly visual framework (*e.g.,* [12, 21]) on top of a state-of-the-art graph query system that can support user-friendly formulation of various types of graph queries such as subgraph search, reachability queries, homeomorphic queries, etc. Our vision can be easily extended to other types of data such as relational, XML, etc.

A visual interface for graph query formulation is typically composed of several panels such as a panel to display the set of labels of nodes or edges of the underlying data graph(s), a panel to construct a graph query graphically, a panel containing *canned patterns* to aid query formulation, and a panel to display query results in an intuitive manner. For example, Figure 1 depicts the screenshot of a real-world visual interface provided by `PubChem`[1] for substructure (subgraph) search for chemical compounds. Specifically, Panel 3 provides a list of chemical symbols that a user can choose from to assign labels to nodes of a query graph. Panel 2 lists a set of *canned patterns* (*e.g.,* benzene ring) which a user may drag and drop in Panel 4 during visual query construction. Note that the availability of such patterns greatly improves usability of the interface by enabling users to quickly construct a large query graph with fewer clicks compared to constructing it in an "edge-at-a-time" mode. For instance, the query graph in Panel 4 can be constructed by dragging and dropping two such canned patterns from Panel 2 instead of taking the tedious route of constructing 9 edges iteratively.

One may observe that the above efforts toward visual query interface design and devising efficient query processing techniques are traditionally independent to each other for decades. This is primarily due to the fact that the two key enablers of these efforts, namely HCI and database management, have evolved into two disparate and vibrant scientific fields, rarely making any systematic effort to leverage techniques and principles from each other towards superior realization of these efforts. Specifically, data management researchers have striven to improve the capability of a database in terms of both performance and functionality, often devoting very little attention to the HCI aspects of the solutions. On the other hand, the HCI community has focused on human factors, building sophisticated models of various types of visual tasks and menu design that are orthogonal to the underlying data management system.

We believe that the chasm between these two vibrant fields sometimes create obstacles in providing superlative visual query formulation and data management services to end users. On the one hand, as visual query interface construction process is tradi-

[1] http://pubchem.ncbi.nlm.nih.gov/edit2/index.html?cnt=0

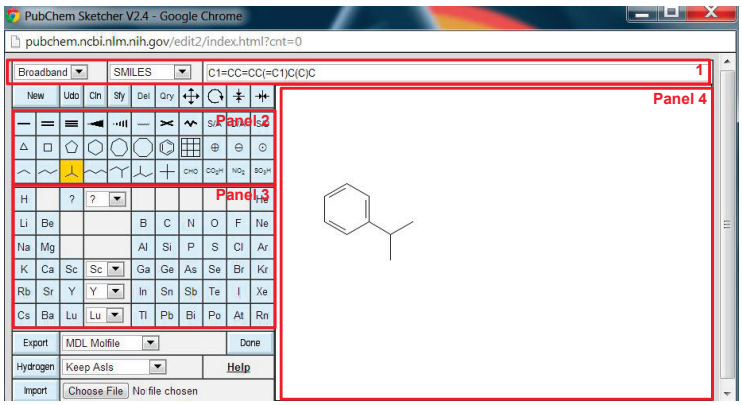

Fig. 1. GUI for substructure search in PubChem.

tionally *data-unaware*, it may fail to generate flexible, portable, and user-friendly query interface. For instance, reconsider the canned patterns in Panel 2 in Figure 1. Given the space constraint in the GUI, the selection of a limited set of patterns that are to be displayed on it are not "data-driven" but typically carried out manually by domain experts. An immediate aftermath of such manual selection is that the set of canned patterns may not be sufficiently diverse enough to support a wide range of graph queries as it is unrealistic to expect a domain expert to have comprehensive knowledge of the topology of the entire graph dataset. Consequently, an end user may not find the canned patterns in Panel 2 useful in formulating certain query graphs. Similar problem may also arise in Panel 3 where the labels of nodes may be manually added instead of automatically generated from the underlying data. Additionally, the visual interface is "static" in nature. That is, the content of Panels 2 and 3 remain static even when the underlying data evolves. As a result, some patterns (*resp.* labels) in Panel 2 (*resp.* Panel 3) may become obsolete as graphs containing such patterns (*resp.* labels) may not exist in the database anymore. Similarly, some new patterns (*resp.* labels), which are not in Panel 2 (*resp.* Panel 3), may emerge due to the addition of new data graphs. Furthermore, such visual query interface lacks of portability as the same interface cannot be seamlessly integrated on a graph database in a different domain (*e.g.,* computer vision, protein structure). As the contents of Panels 2 and 3 are domain-dependent and remain static, the GUI needs to be reconstructed from scratch when the domain changes in order to accommodate new domain-specific canned patterns and labels.

On the other hand, traditionally query processing techniques are only invoked once a user has completed her visual query formulation as the former is completely decoupled from the latter. For instance, during the formulation of a visual query in Panel 4, the underlying query processing engine remains idle and is only initiated after the Run icon is clicked. That is, although the final query that a user intends to pose is revealed gradually during visual query construction, it is not exploited by the query processor prior to clicking of the Run icon. Consequently, valuable opportunities to significantly

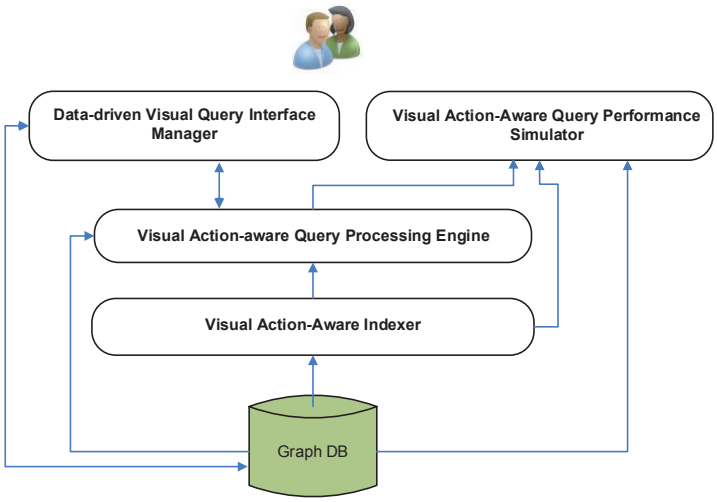

Fig. 2. The architecture.

improve the *system response time* (SRT)[2] by initiating query processing during visual query construction are wasted. Note that in contrast to the traditional paradigm where the SRT is the time taken to process the *entire* query, by bridging the chasm between visual query formulation and query processing activities, the SRT is reduced to processing a part of the query that is yet to be evaluated (if any). Additionally, due to this chasm, opportunities to enhance usability of graph databases by providing relevant *guidance* and *feedback* during query formulation are lost as efficient and timely realization of such functionalities may require prefetching of candidate data graphs during visual query construction. For instance, whenever a newly constructed edge makes a graph query fragment yield empty answer, it can be immediately detected by processing the prefetched data graphs. It is not efficient if it is only detected at the *end* of query formulation as a user may have wasted her time and effort in formulating additional constraints.

In this paper, we lay down the vision of bridging the long-standing chasm between traditional data management and HCI in the context of querying graph-structured data. Specifically, we propose an HCI-*aware visual graph querying* framework that aims to encapsulate several novel and intriguing research challenges toward the grand goal of bridging this chasm. Realization of these challenges entail significant rethinking of several long-standing strategies for visual interface construction and data management.

2 Novel Research Challenges

In this section, we first present the generic architecture of the HCI-aware visual graph querying framework to realize our vision. Next, we identify the key novel research challenges that need to be addressed to realize this framework. To facilitate exposition of

[2] The SRT is the duration between the time a user presses the `Run` icon to the time when the user gets the query results.

these challenges, we assume that the graph database consists of a large number of small or medium-sized graphs. However, these challenges are not limited to such collection of graphs as they are also pertinent to querying large networks such as social networks, biological networks, etc.

2.1 Architecture

Figure 2 depicts the generic architecture of the framework for realizing our vision of bridging the chasm between HCI and graph data management. The *Data-driven Visual Query Interface Manager* component provides a framework to construct various panels of the visual query interface in a *data-driven* manner. It also provides an interactive visual environment for query formulation without the knowledge of complex graph query languages as well as a framework for intelligent guidance and *interruption-sensitive* feedback to users to further ease the cognitive overhead associated with query formulation. The *Visual Action-aware Query Processing Engine* embodies our vision of blending query processing with visual query formulation by utilizing the latency offered by the GUI actions. The *Visual Action-aware Query Performance Simulator* module aims to provide a comprehensive framework for large-scale empirical study of the query processor by simulating the paradigm of blending query formulation and query processing. Finally, the *Visual Action-aware Indexer* component provides an array of *action-aware indexes* to support efficient query processing as well as query performance simulation in our proposed paradigm.

2.2 Data-driven Visual Interface Management

Data-driven visual query interface construction. Recall from Section 1 the limitations associated with the manual construction of panels of the visual query interface. There is one common theme that runs through these limitations: *the visual query interface construction is not data-driven*. Specifically, the GUI does not exploit the underlying graph data to automatically generate and maintain the content of various panels. Hence, it is necessary to rethink the traditional visual query interface construction strategy by taking a novel data-driven approach for visual interface construction. While the unique set of labels of nodes of the data graphs (Panel 3) can be easily generated by traversing them, automatically generating the set of canned patterns is computationally challenging. These patterns should not only be able to *maximally cover* the underlying graph data but should also minimize *topological similarity* (redundancy) among themselves so that a diverse set of canned patterns is available to the user. Note that there can be prohibitively large number of such patterns. Hence, the size of the pattern set should not be too large due to limited display space on the GUI as well as users' inability to absorb too many patterns for query formulation.

As some of the canned patterns may be frequent in the graph database, at first glance it may seem that they can be generated using any frequent subgraph mining algorithm (*e.g., gSpan* [27]). However, this is not the case as it is not necessary for *all* canned patterns to be frequent. It is indeed possible that some patterns are frequently used by end users to formulate visual queries but are infrequent in the database. Also, graph

summarization techniques [25], which focus on grouping nodes at different resolutions in a large network based on user-selected node attributes and relationships, cannot be deployed here as we generate concise canned pattern set by maximizing coverage while minimizing redundancy under the GUI constraint.

Data-driven visual query suggestion and feedback. Any visual querying interface should intelligently guide users in formulating graph queries by (a) helping them to formulate their desired query by making appropriate suggestions and (b) providing appropriate feedback whenever necessary in a timely manner. In order to realize the former, it is important to devise efficient techniques which can predict relevant node labels and canned patterns that may be of interest to a user during query formulation. Also, given a partial query already drawn by a user, it is important to provide suggestion of a concise set of fragments (canned patterns) that she is likely to draw in the next step. Successful realization of these tasks require analyzing topological features of the underlying data as well as query log (if any) w.r.t to the partially constructed query. On the other hand, query feedbacks are essential during query construction as a user may not know if the query she is trying to formulate will return any results. Hence it is important to observe user's actions during query formulation and notify her if a query fragment constructed at a particular step fails to return any results and possibly advise the user on which subgraph in the formulated query fragment is the "best" to remove in order to get non-empty results.

Interruption-sensitive notifications. Addressing aforementioned challenges improve the usability of visual querying systems by enabling users to construct graph queries without special training in the syntax and semantics of a query language. They also guide users into correct query construction by notifying them of a variety of problems with a query such as incorrect syntax, alerts for an empty result, etc. The state-of-the-art algorithms that notify users of such alerts and suggestions are, however, overly aggressive in their notifications as they focus on *immediate* notification regardless of its impact on the user. More specifically, they are insensitive to the cognitive impact of *interruptions* caused by notifications sent at *inopportune times* that disrupt the query construction process.

Many studies in the cognitive psychology and HCI communities have demonstrated that interrupting users engaged in tasks by delivering notifications inopportunely can negatively impact task completion time, lead to more errors, and increase user frustration [6, 16, 22]. For instance, suppose a user is notified (with a pop-up dialog box) of an empty result (due to previously formulated condition) when she is undertaking a task such as dragging a canned pattern from Panel 2 and preparing to drop it in Panel 4. This interruption may frustrate her as mental resources allocated for the current task are disrupted because she is forced to leave it to acknowledge and close the dialog box. Although, such inopportune interruption adversely affects the usability of visual querying systems, traditional data management techniques have devoted very little attention to the cognitive aspect of a solution. Here we need to ensure that a solution to the aforementioned issues is "cognitive-aware" by devising models and techniques to deliver a notification *quickly but at an appropriate moment* when the mental workload of the user is minimum. Detecting such an opportune moment should be transparent to the user and must not seek explicit input from her.

A promising direction in achieving this goal is to seamlessly integrate *defer-to-breakpoint* strategy [18, 19] with visual query formulation tasks when reasoning about when to notify the user. This will also entail to leverage work in HCI to build quantitative models for time available to complete various data management tasks (*e.g.,* detection of empty results, query suggestion) in order to ensure notification delivery at optimal breakpoints that lower the interruption cost. A keen reader may observe that making aforementioned solutions interruption-sensitive essentially questions the holy grail of database research over the past forty years: *whether faster evaluation is always better?* As remarked earlier, the HCI and cognitive psychology communities have observed that slowing or deferring some activities (*e.g.,* notification delivery) can increase usability. Hence, by making visual query formulation process cognitive-aware, we essentially aim to slow down a small part of the system in order to increase its usability.

2.3 Visual Action-aware Query Processing

Visual action-aware graph indexing. A host of state-of-the-art graph indexing strategies have been proposed since the last decade to facilitate efficient processing of a variety of graph queries (*e.g.,* [28]). While several of these techniques are certainly innovative and powerful, they cannot be directly adopted to support our vision. These techniques are based on the conventional paradigm that the *entire* query graph must be available *before* it can leverage the indexes for processing the query. However, in our proposed framework we aim to initiate query evaluation as soon as a fragment of the query graph is visually formulated. Hence, the indexes need to be aware of visual actions taken by a user and accordingly filter negative results after every action taken by her. Recall that a user may construct a single edge or a canned pattern at a particular step during query formulation. She may also modify it by deleting an existing canned pattern or an edge. Hence, the size of a partially-constructed query graph may grow or shrink by $k \geq 1$ at each step. Furthermore, a query fragment may evolve from a frequent pattern to infrequent one and vice versa. In this case, it is important to devise efficient indexing schemes to support identification of the data graphs that match (exact or approximate) the partially constructed query graph at each step. Note that since subgraph isomorphism testing is known to be NP-complete, the indexing scheme should minimize expensive candidate verification in order to retrieving these partial results.

Visual action-aware query matching. Our goal is to utilize the latency offered by the GUI actions to retrieve partial candidate data graph. Specifically, when a user draws a new edge or canned pattern on the query canvas, candidate data graphs containing the current query fragment need to be efficiently retrieved and monitored by leveraging the indexes. If the candidate set is non-empty at a specific step then high-quality suggestions to complete the construction of the subsequent steps may be provided at an opportune time. On the other hand, if the candidate set is empty then there does not exist any data graphs that match the query fragment at a specific step. In this scenario, the user needs to be notified appropriately in an interruption-sensitive manner and guided to modify the query appropriately. The above process is repeated until the user clicks on the `Run` icon to signify the end of the query formulation step. Consequently, the final query results are generated from the prefetched candidate data graphs by performing verification

test whenever necessary. Note that as this step invokes the subroutine for subgraph isomorphism test, the verification process needs to be minimized by judiciously filtering as many false candidates as possible without any verification test.

Non-traditional design issues. Observe that in order to realize the aforementioned visual query processing paradigm, we need a query processor that incorporates the following three non-traditional design issues.

- First, is the need for materialization of intermediate information related to all partial candidate graphs matching the query at each step during query construction. While this has always been considered as an unreasonable assumption in traditional databases, materialization of all intermediate results is recently supported to enhance database usability [11]. Note that this issue is particularly challenging here due to computational hardness of subgraph isomorphism test. Hence judicious strategy to minimize candidate verification while retrieving partial candidates is required.
- Second, materialization of partial candidates needs to be performed efficiently within the available GUI latency. This is pivotal as inefficient materialization can slow down generation of candidate graphs at each query construction step eventually adversely affecting the SRT. Ideally, we should be able to materialize candidate graphs of a query fragment *before* the construction of the succeeding edge (or canned pattern). Consequently, it is paramount to accurately and systematically estimate the time taken by a user for constructing a query fragment (edge or pattern) as this latency is exploited by the query processing paradigm to prefetch candidate matches. Here it is important to drew upon the literature in HCI (*e.g.,* [2, 3]) to quantitatively model the time available to a user to perform different visual tasks such as selection of canned patterns. This will enable us to quantify the "upper bound" of materialization time and seek efficient solution accordingly.
- Third, the query processor needs to support *selectivity-free* query processing. Selectivity-based query processing, that exploits estimation of predicate selectivities to optimize query processing, has been a longstanding approach in classical databases. Unfortunately, this strategy is ineffective in our proposed framework as users can formulate low and high selective fragments in any arbitrary sequence of actions. As query processing is interleaved with the construction (modification) of each fragment, it is also not possible to "push-down" highly selective fragments. Similarly, query feedbacks such as detection and notification of empty result are intertwined with the order of constructed query conditions as it must be delivered at an opportune time. Hence, it needs to operate in a selectivity-free environment as well. The only possible way to bypass this stumbling block in this environment is to ensure that the sequence of visual actions formulated by a user is ordered by their selectivities. However, users cannot be expected to be aware of such knowledge and it is unrealistic to expect them to formulate a query in a "selectivity-aware" order.

2.4 HCI-driven Performance Simulation

The aforementioned framework must have adequate support to evaluate visual query performance at a large scale. In contrast to traditional paradigm where the runtime per-

formance of a large number of graph queries can be easily measured by automatically extracting a random collection of query graphs from the underlying data and executing them, each query in the proposed framework must be formulated by a set of real users. Furthermore, each query can follow many different query formulation sequences. Consequently, it is prohibitively expensive to find and engage a large number of users who are willing to formulate a large number of visual queries. Hence, there is a need for a performance measurement framework that can simulate formulation of a large number of visual graph queries without requiring a large number of users.

The performance benchmarking framework needs to address two key challenges. First, is the automated generation of a large number of queries of different types (frequent and infrequent), topological characteristics, and result size. The action-aware graph indexing framework can be leveraged here to extract frequent and infrequent subgraphs from the underlying data graphs that satisfy different constraints. Second, is to simulate the formulation of each generated query following different query formulation sequences. Note that different users may take different time to complete each visual action during query formulation. For instance, time taken to move the mouse to Panel 2 (Figure 1) and select a canned pattern may be different for different users. It is important to accurately and systematically estimate the time taken by a user for each of these actions as this latency is exploited by the query processing paradigm to prefetch candidate matches. Similar to the materialization of candidate graphs, here it is important to drew upon the literature in HCI (*e.g.,* [2, 3]) to quantitatively model the time available to a user to perform different visual tasks. Then for each step in the query construction process, the query simulation algorithm *waits* for appropriate amount of time to simulate the execution of the task by a user before moving to the next step.

2.5 Extension to Massive Graphs

Recall that the aforementioned research challenges to realize our vision assume that the database contains a large collection of small or medium-sized graphs. However, in recent times graphs with millions or even billions of nodes and edges are commonplace. As a result, there is increasing efforts in querying massive graphs by exploiting distributed computing [24]. We believe that the aforementioned challenges need to be addressed as well to realize our vision on such massive graph framework. However, the solution to these challenges in this framework differ. For example, the data-driven visual interface construction needs to generate the canned patterns and labels in a distributed manner as graph data is stored in multiple machines. Similarly, each visual action during query processing needs to be judiciously processed in a distributed environment by selecting relevant slaves that may contain the candidate matches as well as minimizing communication costs among the machines.

3 Early Efforts

In [7, 7, 17, 20, 21], we took the first step to implement the visual action-aware query processing module for subgraph containment and subgraph similarity search queries for

a set of small or medium-sized graphs as well as for large networks. Our study demonstrates that the paradigm of blending visual query formulation and query processing significantly reduces the SRT as well as number of candidate data graphs compared to state-of-the-art techniques based on traditional paradigm. However, in these efforts we assume that the visual query is formulated using an "edge-at-a-time" approach (canned patterns are not used), making it tedious to formulate large queries. Consequently, the underlying indexing schemes and query processing strategies need to be adapted to support such query construction. Furthermore, it is interesting to investigate whether the proposed paradigm can efficiently support a wider variety of graph queries such as reachability queries, supergraph containment queries, homeomorphic graph queries, etc. Lastly, the data-driven visual interface management and the HCI-aware performance benchmarking framework are open research problems that are yet to be addressed.

Visual action-aware query processing is also studied in the context of XML query processing [26, 29]. Similar to visual graph querying, here we proposed a visual XML query system called XBLEND which blends visual XML query formulation and query processing in a novel way. It is built on top of a relational framework and exploits the latency offered by the GUI-based query formulation to prefetch portions of the query results by issuing a set of SQL queries.

4 Conclusions

This paper contributes a novel vision of HCI-aware graph data management to bridge the chasm between HCI and data management fields. We present a framework for querying graph data visually where the visual interface management is data-driven and query processing and performance benchmarking are HCI-driven. Addressing the research challenges associated with this vision entail a multi-disciplinary effort drawing upon the literature in HCI, cognitive psychology, and data management. Although, in this paper we focused on graph querying, it is easy to see that our vision can be extended to other visual querying environments.

Acknowledgement: Sourav S Bhowmick was supported by the Singapore-MOE AcRF Tier-1 Grant RG24/12. His travel expenses were supported by the Marsden fund council from Government funding, administered by the Royal Society of New Zealand. The author would also like to thank Changjiu Jin, H. Hung, and B. Q. Trung for implementing several features of the vision.

References

1. S. Abiteboul, R. Agrawal, P. Bernstein et al. The Lowell Database Research Self-Assessment. *In Communication of the ACM*, 2005.
2. J. Accot, S. Zhai. Refining Fitts' Law Models for Bivariate Pointing. *In ACM SIGCHI*, 2003.
3. D. Ahlstrom. Modeling and Improving Selection in Cascading Pull-Down Menus Using Fitt's Law, the Steering Law, and Force Fields. *In CHI*, 2005.
4. M. Angelaccio, T. Catarci, G. Santucci. QBD*: A Graphical Query Language with Recursion. *In IEEE Trans. Soft. Engg.* , 16(10):1150–1163, 1990.

5. P. Atzeni, G. Mecca, P. Merialdo. To Weave the Web. *In VLDB*, 1997.
6. B. P. Bailey, J. A. Konstan. On the Need for Attention Aware Systems: Measuring Effects of Interruption on Task Performance, Error Rate, and Affective State. *In Journal of Computers in Human Behavior*, 22(4), 2006.
7. S. S. Bhowmick, B. Choi, S. Zhou. VOGUE: Towards a Visual Interaction-aware Graph Query Processing Framework. In *CIDR*, 2013.
8. H. Blau , N. Immerman , D. Jensen. A Visual Language for Querying and Updating Graphs. *Tech. Report 2002-037*, Univ. of Mass., Amherst, 2002.
9. D. Braga, A. Campi, S. Ceri. XQBE (XQuery By Example): A Visual Interface to the Standard XML Query Language. In *TODS*, 30(2), 2005.
10. M. Carey, L. Haas, V. Maganty, J. Williams. PESTO: An Integrated Query/Browser for Object Databases.*In VLDB*, 1996.
11. A. Chapman, H. V. Jagadish. Why Not? *In SIGMOD*, 2009.
12. D. H. Chau , C. Faloutsos, H. Tong, et al. GRAPHITE: A Visual Query System for Large Graphs. *ICDM Workshop* , 2008.
13. S. Comai, E. Damiani, R. Posenato, L. Tanca. A Schema Based Approach for Modeling and Querying WWW data. *In FQAS*, 1998.
14. S. Comai, E. Damiani, P. Fraternali. Computing Graphical Queries Over XML Data. *In ACM TOIS*, 19(4): 371–430, 2001.
15. I. F. Cruz, A. O. Mendelzon, P. T. Wood. A Graphical Query Language Supporting Recursion. *In ACM SIGMOD*, 1987.
16. E. Cutrell, et al. Notification, Disruption, and Memory: Effects of Messaging Interruptions on Memory and Performance. *In Proc. of the IFIP TC.13 Int Conf on HCI*, Tokyo, Japan, 2001.
17. H. Hung, S. S. Bhowmick et al. QUBLE: Towards Blending Interactive Visual Subgraph Search Queries on Large Networks. *The VLDB Journal*, 23(3), Springer-Verlag, May 2014.
18. S. T. Iqbal, B. P. Bailey. Understanding and Developing Models for Detecting and Differentiating Breakpoints during Interactive Tasks. *In CHI*, 2007.
19. S. T. Iqbal, B. P. Bailey. Effects of Intelligent Notification Management on Users and Their Tasks. *In CHI*, 2008.
20. C. Jin, et al. GBLENDER: Towards Blending Visual Query Formulation and Query Processing in Graph Databases. *In ACM SIGMOD*, 2010.
21. C. Jin, et al. PRAGUE: A Practical Framework for Blending Visual Subgraph Query Formulation and Query Processing. *In ICDE*, 2012.
22. C. A. Monk, D. A. Boehm-Davis, J. G. Trafton. The Attentional Costs of Interrupting Task Performance at Various Stages. *In Proc of the Human Factors and Ergonomics Society*, 2002.
23. Y. Papakonstantinou, et al. QURSED: Querying and Reporting Semistructured Data. *In ACM SIGMOD*, 2002.
24. Z. Sun, et al. Efficient Subgraph Matching on Billion Nodes Graphs. *In VLDB*, 2013.
25. Y. Tian, et al. Efficient Aggregation for Graph Summarization. *In SIGMOD*, 2008.
26. B. Q. Truong, S. S. Bhowmick, et al. MUSTBLEND: Blending Visual Multi-Source Twig Query Formulation and Query Processing RDBMS. *In DASFAA*, 2013.
27. X. Yan, et al. gSpan: Graph-based Substructure Pattern Mining. *In ICDM*, 2002.
28. X. Zhao, et al. A Partition-Based Approach to Structure Similarity Search. *In VLDB*, 2013.
29. Y. Zhou, S. S. Bhowmick, et al. XBLEND: Visual XML Query Formulation Meets Query Processing. *In ICDE*, 2009.
30. M. M. Zloof. Query-By-Example: A Data Base Language. *IBM Syst. J.*, 16(4):324–343, 1977.