ExpFinder: Finding Experts by Graph Pattern Matching

Wenfei Fan [1,3] Xin Wang [1] Yinghui Wu [2]

[1]*University of Edinburgh* [2]*UC Santa Barbara*

[3]*Big Data Research Center and State Key Laboratory of Software Development Environment, Beihang University*

{wenfei@inf, x.wang-36@sms, y.wu-18@sms}.ed.ac.uk

***Abstract*— We present ExpFinder, a system for finding experts in social networks based on graph pattern matching. We demonstrate (1) how ExpFinder identifies top-K experts in a social network by supporting bounded simulation of graph patterns, and by ranking the matches based on a metric for social impact; (2) how it copes with the sheer size of real-life social graphs by supporting incremental query evaluation and query preserving graph compression, and (3) how the GUI of ExpFinder interacts with users to help them construct queries and inspect matches.**

I. Introduction

Social networks provide a rich source of information for us to identify communities and social positions in a wide range of real-life applications, *e.g.,* social group identification [9], social position analysis [2] and recommendations [7], [11], [12]. These are typically conducted by using graph pattern matching: given a *pattern query* Q and a (social) *data graph* G, it is to find all the *matches* in G for Q, denoted as $M(Q, G)$, which satisfy the search conditions specified in Q.

It is, however, nontrivial to effectively conduct graph pattern matching in social networks modeled as graphs: (1) as shown in [4], the traditional notions of subgraph isomorphism and graph simulation are often too restrictive to match patterns in real-life social graphs; (2) real-life graphs are typically large, *e.g.,* Facebook has 901 million users and 125 billion friend connections [1]; it is often prohibitively expensive to query such graphs based on subgraph isomorphism (NP-complete) and graph simulation (quadratic time); and (3) the matches of a pattern Q in a social graph G are often excessively large: exponential in the size $|G|$ of G by subgraph isomorphism, and $O(|Q||G|)$ by graph simulation, where $|Q|$ (resp. $|G|$) denotes the total number of nodes and edges in Q (resp. G).

To effectively capture matches in real-life social graphs, we adopt *bounded simulation* [4], a revision of the traditional notion of graph simulation, and study its application in experts searches in *large* and *dynamic* real-life social networks.

Example 1: Consider a fraction of a collaboration network (excluding edge e_1) depicted as graph G in Fig. 1(b). Each node in G denotes a person, with attributes such as *name*, *field* (*e.g.,* system architect (SA), system developer (SD), business analyst (BA), system tester (ST)), *specialty* for the field (*e.g.,* programmer and database administrator for SD), and *experience* (number of years). Each edge indicates collaboration, *e.g.,* (Bob, Dan) indicates that Dan worked in a project led by Bob and collaborated well with Bob. Two people may also collaborate indirectly via a path of collaboration [8].

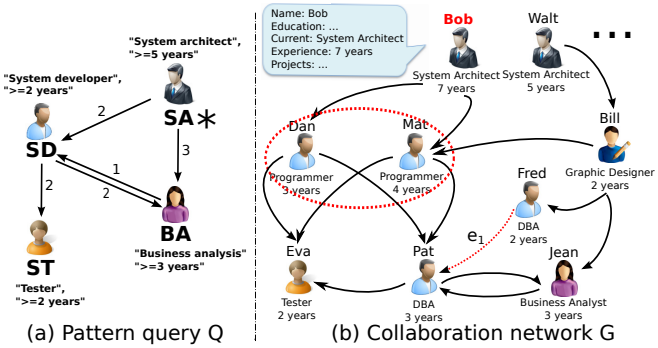


Fig. 1. Pattern query Q and collaboration network G

Suppose that a company wants to hire a system architecture designer (SA) and form a team to develop a medical record system [8]. The requirements are expressed as a *bounded simulation query* Q [4] (Fig. 1(a)) as follows: (1) the SA expert must have worked in a team consisting of three other types of experts SD, BA, ST, represented by the labeled nodes in Q; (2) the SA should have at least 5 years of working experience, shown as a search condition at node SA; (3) there are SD and BA experts who collaborated well with SA experts, via a collaboration chain no longer than 2 and 3, respectively, as indicated by labeled edges $(\mathsf{SA}, \mathsf{SD})$ and $(\mathsf{SA}, \mathsf{BA})$ in Q. Similarly, the other nodes and edges in Q depict the requirements of the team and SA experts. Here SA is marked as the "output node" with "$*$", *i.e.,* the users only require the matches of SA to be returned as the desired experts.

The matches of Q, denoted as $M(Q, G)$, is a relation between a query node and its valid matches [4] in G. More specifically, $M(Q, G) = \{(\mathsf{SA}, \mathsf{Bob}), (\mathsf{SA}, \mathsf{Walt}), (\mathsf{BA}, \mathsf{Jean}), (\mathsf{SD}, \mathsf{Mat}), (\mathsf{SD}, \mathsf{Dan}), (\mathsf{SD}, \mathsf{Pat}), (\mathsf{ST}, \mathsf{Eva})\}$. Observe the following: (1) the node SD in Q is mapped to both Mat (programmer) and Pat (DBA) in G, which is not allowed by a bijection in subgraph isomorphism; and (2) the edge from SA to BA in Q requires that the SA expert has supervised a BA within 3 hops; the edge is mapped to a path (*e.g.,* the path from Bob to Jean) of a bounded length in G; in contrast, graph simulation only allows edge to edge matching.

As SA may have multiple matches, a ranking metric should be provided to select the best experts with social impact. For example, both Bob and Walt are equally experienced matches of SA. Nevertheless, Bob has collaborated with all other team members more "closely" via shorter collaboration paths. Thus, Bob has a stronger social impact [8], [10], and makes a better expert for coordinating with team members. □

In this demo, we present ExpFinder, a system to effectively identify experts by graph pattern matching in social networks. It implements the techniques that we presented in [3]–[5]. In contrast to previous expert search systems (see [8] for a survey), (1) ExpFinder supports searches for experts in social networks via bounded simulation [4], (2) it provides a new facility to find top-K matches based on a metric for social impact, which was not studied in [3]–[5]; and (3) it supports efficient searches and provides graph storage for large and dynamic social data, by leveraging our incremental evaluation methods [3] and query preserving scheme of graph compression [5]. It should be remarked that expert search is just one of the applications of these techniques. The same methods can be used to, *e.g.,* recommend movies, find jobs, explore advertising strategies as well as to make travel plans.

II. The ExpFinder System

The architecture of the ExpFinder system is shown in Fig. 2. It consists four modules. (1) A *Graphical User Interface* (GUI) provides a graphical interface to help users formulate queries, manage data graphs and understand visualized query results. (2) A *Query Engine* evaluates pattern queries and ranks query results. (3) An *Incremental Computation Module* maintains the query results of a set of frequently issued queries (decided by the users) in response to updates incurred to data graphs. (4) A *Graph Compression Module* constructs and dynamically maintains compressed graphs, which can be directly queried by the query engine. In addition, all the graphs and query results are stored and managed as files. We next present the components of ExpFinder and their interactions.

Graphical User Interface. The GUI helps the users manage data graphs, construct queries and browse query results. (1) It provides a task-oriented panel, which facilitates the users to issue specific requests such as to view/select data graphs and construct queries. (2) The users can construct a (bounded) simulation query Q by drawing a set of query nodes and edges on a query panel of the GUI, specifying the search conditions (*e.g.,* expertise="system developer"; experience="3 years"), bounds on the edges, and indicating the particular "output" node for which users want to find matches (*e.g.,* SA in Fig. 1). They may also choose a data graph G to query. (3) The GUI visualizes the query results expressed as result graphs [4], in which each node is a match of a query node in Q, and each edge (marked with an integer d) represents a shortest path with length d corresponding to a query edge.

Query Engine. The query engine performs (a) query evaluation, and (b) top-K result selection for the output node. It finds a *unique, maximum* match graph for the (bounded) simulation query [4]. The query result is then visualized by the GUI.

Bounded simulation. Given a graph G and a pattern query Q, $M(Q,G)$ is the maximum relation such that for each node $u \in Q$, there is a node $v \in G$ such that (1) $(u,v) \in M(Q,G)$, and (2) for each $(u,v) \in M(Q,G)$, (a) the content of v satisfies the search condition specified by the pattern node u, and (b)

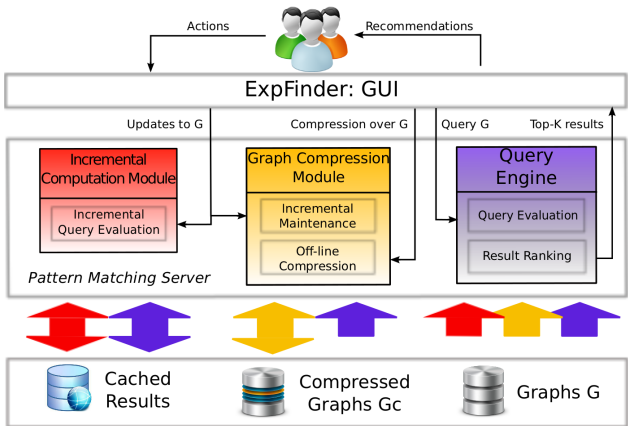


Fig. 2. Architecture of ExpFinder

for each edge (u,u') in Q, there exists a nonempty path ρ from v to v' in G such that $(u',v') \in M(Q,G)$, and the length of ρ does not exceed the bound on (u,u'). Example 1 illustrates $M(Q,G)$ for the query Q and graph G given in Fig. 1.

As shown by [4], (1) $M(Q,G)$ is unique for each G and Q, (2) graph simulation is a special case when the bound on each pattern edge (u,u') is 1, and (3) bounded simulation is able to catch sensible matches that subgraph isomorphism and simulation fail to identify, as we have seen in Example 1.

Query evaluation. To efficiently find $M(Q,G)$ in a large graph G, the query engine coordinates with the incremental computation and graph compression modules as follows. Upon receiving a pattern query Q, (1) the query engine directly returns $M(Q,G)$ if it is already cached; (2) otherwise, if a compressed graph G_c for G is already computed by the compression module, Q is evaluated on G_c directly [5] (as will be discussed); and (3) if the users opt not to compress G at this stage, the query engine finds $M(Q,G)$, by employing a quadratic-time algorithm [6] to evaluate simulation queries, and a cubic-time algorithm [4] for bounded simulation queries. After $M(Q,G)$ is computed, the query engine computes a result graph to represent the result [4]. The users may decide whether the query and its result need to be cached at this stage.

Results Ranking. As remarked earlier, the query result is typically large, while the users may only be interested in the best K experts that match the designated output node in Q, *e.g.,* SA in Example 1. To this end, the query engine identifies top-K matches by using a ranking function. The intuition of the ranking function comes from the following observation about social networks: *two nodes that are closer to each other often have more social impact to each other* [8], [10]. Given an edge (u',u) in pattern query Q, and two matches v_1 and v_2 of u in a social network, where u is the output node, for a match v' of u', v_1 is preferred to v_2 if v' is closer to v_1 than to v_2. Indeed, in practice v_1 may represent an expert who collaborates with expert v' more closely than the other expert v_2. In light of this, given an output node u_o and its match v in the result graph $G_r = (V_r, E_r)$, the rank $f(u_o, v)$ is defined as:

$$f(u_o,v) = \frac{\Sigma_{u \in V_r} \mathsf{dist}(u,v) + \Sigma_{u' \in V_r} \mathsf{dist}(v,u')}{|V_r'|}$$

where (a) $\mathsf{dist}(u, v)$ (resp. $\mathsf{dist}(v, u')$) represents the distance (as the sum of the edge weight in a shortest path) from an ancestor u to v (resp. from v to its descendant u') in G_r, and (b) V_r' is the set of nodes in G_r that can reach v or can be reached from v. Intuitively, $f(u_o, v)$ computes the *average* distance of v from (to) other nodes in G_r. The top-K matches of u_o is the set of K matches with the *minimum* ranks.

The ranking function $f()$ assesses the social impact in terms of node distance, as one of the commonly used metrics in social network analysis [8], [10]. Note that other metrics can be readily supported by ExpFinder. We remark that top-K matches were not studied in the previous work [3]–[5].

Example 2: Recall the match result $M(Q, G)$ described in Example 1. Its result graph G_r is a weighted graph with a set of nodes $\{\mathsf{Bob}, \mathsf{Walt}, \mathsf{Jean}, \mathsf{Mat}, \mathsf{Dan}, \mathsf{Pat}, \mathsf{Eva}\}$. One may verify that the rank $f(\mathsf{SA}, \mathsf{Bob}) = (1+1+2+3+2)/5 = \frac{9}{5}$, and $f(\mathsf{SA}, \mathsf{Walt}) = (2+2+3)/3 = \frac{7}{3}$. Therefore, Bob is the top-1 match for SA, since compared to Walt, he has shorter social distance to other collaborators, and hence has stronger social impact on the group members. □

Incremental Computation Module. Real-life social graphs are typically large and are constantly changed. Given a graph G, a query Q and updates ΔG to G, it is costly to recompute $M(Q, G \oplus \Delta G)$ starting from scratch each time G is updated, where $G \oplus \Delta G$ denotes G updated by ΔG. The incremental module copes with the dynamic nature of social networks by incrementally identifying changes to $M(Q, G)$ in response to ΔG, *without* accessing G. When ΔG is small as commonly found in practice, it is far more efficient to incrementally compute $M(Q, G \oplus \Delta G)$ than to recompute it starting from scratch. The module supports the incremental evaluation algorithms of [3] for simulation and bounded simulation queries.

Example 3: Recall the query Q and graph G of Fig. 1, and the matches $M(Q, G)$ from Example 1. Suppose that G is updated by inserting the edge e_1 (see Fig. 1(b)), denoted by ΔG. Then ΔG incurs increment ΔM to $M(Q, G)$ as a new pair $(\mathsf{SD}, \mathsf{Fred})$. Instead of recomputing $M(Q, G \oplus \Delta G)$, the incremental module finds the change $\{(\mathsf{SD}, \mathsf{Fred})\}$ to $M(Q, G)$ by only accessing $M(Q, G)$ and e_1. □

Graph Compression Module. The incremental module efficiently maintains matches in dynamic networks. Nevertheless, it is costly to compute $M(Q, G)$ and it is unlikely to lower the computational complexity of query evaluation. The graph compression module aims to reduce the size of the input for query evaluation, by constructing smaller *compressed* graphs G_c for a data graph G. The compressed graph G_c (1) has less nodes and edges than G, and (2) can be directly queried by the query engine and incremental module, such that for *any* (bounded) simulation query Q, $M(Q, G)$ can be obtained by a linear time post-processing from $M(Q, G_c)$ [5]. Moreover, G_c is incrementally maintained in response to changes to G.

The graph compression module is developed to (1) compute the compressed graphs G_c of G, and (2) dynamically maintain

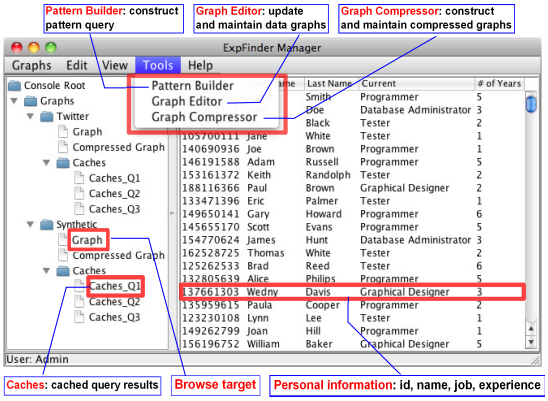


Fig. 3. Visual interface: ExpFinder Manager

G_c when G is updated, by implementing the techniques of [5]. The module works seamlessly with the other modules: it invokes the compression algorithm to construct G_c for data graph G upon receiving requests from GUI, and dynamically maintains G_c in response to changes to G issued through GUI. The compressed graphs are then stored, and are accessed by the query engine when processing query, as remarked earlier.

Example 4: Recall pattern query Q and data graph G from Fig. 1. Observe that both Fred and Pat (DBA) collaborated with ST and BA people. Since they "simulate" the behavior of each other in the collaboration network G, they could be considered *equivalent* when computing $M(Q, G)$. Similarly, pairs (Emmy, Eva) and (Dan, Mat) can also be considered equivalent. The nodes that are pairwise equivalent form an equivalence class, and the compressed graph G_c is constructed by merging the nodes in the same equivalence class. □

III. Demonstration Overview

The demonstration is to show the following: (1) how the GUI of ExpFinder handles users' requests and displays query results; (2) how efficient the query engine evaluates queries and identifies top-K experts; (3) how the incremental module manages batch updates to data graphs, and (4) how the compression module computes and maintains compressed graphs.

DataSet. ExpFinder loads both synthetic and real-life datasets. (1) We design a synthetic graph generator to generate arbitrarily large graphs and show the efficiency of ExpFinder; and (2) we use a fraction of Twitter to show the performance of each module of ExpFinder, and interpret query results in details.

Interacting with the GUI. We invite users to use the GUI, from query design to intuitive illustration of query results.

(1) Users may operate on ExpFinder Manager as the main control panel. As shown in Fig. 3, users can select, view and modify the detailed information of data graphs, and may access the modules of ExpFinder as listed in the tools.

(2) Users can define their own queries through our Pattern Builder (PB) panel as shown in Fig. 4. PB provides the users with a canvas to create a new pattern query. For example, Figure 4 shows three pattern queries Q_1, Q_2 and Q_3 constructed via PB, with different search conditions and topology.

(3) The GUI provides various ways to help users understand

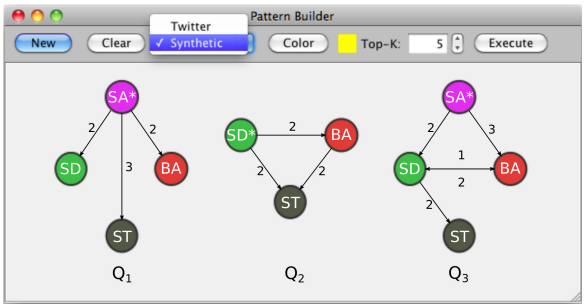


Fig. 4. Visual interface: Pattern Builder

query results. We show how the users can browse (a) result graphs relevant to matches, and (b) top-K matches, by using the GUI. As an example, the result graphs and the top 1 (best) SA expert (marked in red) are shown in Fig. 5 for queries Q_1, Q_2 and Q_3 (in Fig. 4), respectively. Besides visualizing result graphs, ExpFinder also supports *Drill Down* and *Roll Up* analysis. That is, the users can drill down to see profiles of the nodes, edge weights and other detailed information in a result graph, and can roll up to view the global structure of the result graph. Hence the GUI enables ExpFinder to display the result at different granularity.

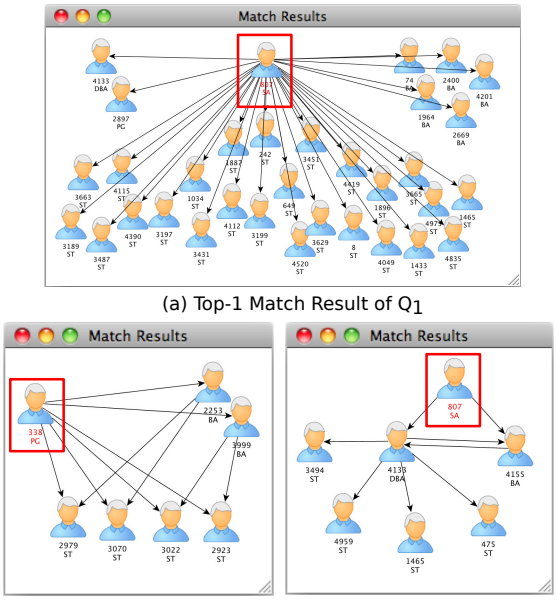
Performance of query evaluation. This demonstration also aims to show the performance of the query engine, the incremental module and the graph compression module.

Performance of the query engine. We will show (a) how (bounded) simulation queries are processed on large graphs by generating optimized query plans, and (b) how top-K matches are selected based on the ranking function. We will use real-life datasets and queries to provide intuitive illustrations.

Coping with the dynamic world. We will also show the performance gains of incremental computation compared to batch computation that recomputes the matches in response to updates. We show the improvement by varying the size of the data graphs with unit update (single edge insertion/deletion) as well as batch updates (a list of edge insertions/deletions). We show that for batch updates and general (possibly cyclic) patterns, our incremental module performs significantly better than their batch counterparts, when data graphs are changed up to 30% for simulation, and 10% for bounded simulation.

Querying compressed graphs. In addition, we will show (1) how graph compression module effectively compresses a data graph, (2) how substantial the performance is improved when evaluating (bounded) simulation queries by using compressed graphs instead of the original graphs, and (3) how the compressed graphs are dynamically maintained. We show that in average, the graphs can be reduced by 57%, which in turn reduces query evaluation time by 70%. Moreover, the compression module efficiently maintains the compressed graphs, and outperforms the method that recomputes compressed graphs, even when large batch updates are incurred.

Summary. This demonstration aims to show the key ideas and performance of our expert search system ExpFinder based on graph pattern matching. ExpFinder is able to (1) effectively identify top-K experts in social networks by using pattern queries specified with search conditions and bounded connectivity constraints, (2) efficiently evaluate the queries on large real-life social graphs, (3) incrementally answer queries on dynamic graphs in response to batch updates, (4) support graph compression for efficient graph storage and query evaluation, and (5) provide intuitive graphical interface to facilitate the users to construct queries and interpret query results. We contend that ExpFinder can serve as a promising tool for expert finding in large and dynamic real-life social networks.



(a) Top-1 Match Result of Q_1

(b) Top-1 Match Result of Q_2 (c) Top-1 Match Result of Q_3

Fig. 5. Match Results relevant to output node of Q_1, Q_2 and Q_3

Acknowledgement. Fan and Wang are supported in part by the RSE-NSFC Joint Project Scheme and EPSRC EP/J015377/1, UK, and the 973 Program 2012CB316200 and NSFC 61133002, China.

References

[1] Facebook statistics; visited july 2012. *http://www.facebook.com/press/info.php?statistics.*

[2] J. Brynielsson, J. Högberg, L. Kaati, C. Martenson, and P. Svenson. Detecting social positions using simulation. In *ASONAM*, 2010.

[3] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu. Incremental graph pattern matching. In *SIGMOD*, 2011.

[4] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. In *PVLDB*, 2010.

[5] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *SIGMOD*, 2012.

[6] M. R. Henzinger, T. Henzinger, and P. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.

[7] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, 2009.

[8] T. Lappas, K. Liu, and E. Terzi. A survey of algorithms and systems for expert location in social networks. In *Social Network Data Analytics*. 2011.

[9] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27:415–444, 2001.

[10] M. E. Newman. Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. *Phys Rev E Stat Nonlin Soft Matter Phys*, 64(1 Pt 2), 2001.

[11] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *KDD*, 2008.

[12] L. Terveen and D. W. McDonald. Social matching: A framework and research agenda. *ACM Trans. Comput.-Hum. Interact.*, 12(3), 2005.