Demonstration

MovieFinder: A Movie Search System via Graph Pattern Matching

Xin Wang [1,2] Chengye Yu [2] Enyang Zhang [2] Tong Du [2]

[1] *Southwest Jiaotong University* [2] *ChangHong Inc.*

xinwang@swjtu.cn, {chengye.yu, enyang.zhang, tong.du}@changhong.com

ABSTRACT

In this demo, we present MovieFinder, a user-friendly movie search system with following characteristics: it (1) searches movies on social networks via the technique of top-k graph pattern matching; (2) supports distributive computation to handle sheer size of real-life social networks; (3) applies view-based technique to optimize local evaluation, and employs incremental computation to keep cached views up to date; and (4) provides graphical interface to help users construct queries, explore data and inspect results.

1. INTRODUCTION

In recent years, social networking sites have experienced fast development, and are endowed with enormous commercial value. One key issue to achieve commercial goals via social networks is how to help uses find their interested objects on big social data. In light of this, a host of techniques are developed, among which graph pattern matching defined in terms of subgraph isomorphism has been widely used and verified to be effective [5].

However, it is nontrivial to efficiently conduct graph pattern matching on social networks due to the following reasons: (1) graph pattern matching with subgraph isomorphism is computationally expensive as it is an NP-complete problem [3], and moreover, there may exist exponentially many matches of a pattern query Q in a data graph G; (2) real-life graphs are typically large, *e.g.,* Facebook has 1.18 billion daily active users, and the average number of friends is 155 [1], it is hence prohibitively expensive to query such big graphs; (3) social networks are often distributively stored, which makes graph pattern matching more challenging or even infeasible; (4) social networks evolve constantly, it is often expensive to recompute matches starting from scratch when social networks are updated with minor changes.

Example 1: Consider a fraction of IMDb [2] collaboration network depicted as graph G in Fig. 1(a). Each node in G either denotes a performer (p) (resp. director (d)), labeled by *id*, *name*; or a movie (m), with attributes *title*, *genres* (g), *rating* (r) and *release time* (t). Each directed edge from a performer (resp. director) to a movie indicates that the performer (resp. director) played in (resp. directed)

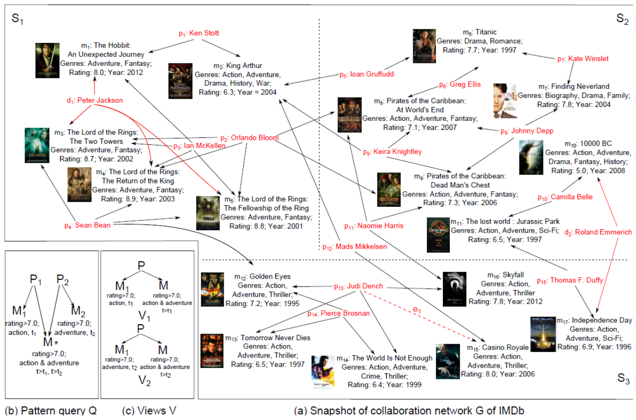

Figure 1: Pattern query Q, Views $\mathcal{V}$ and collaboration network G

the movie, where the edges connecting directors and movies are marked in red. The graph G is *geo-distributed* to three sites S_1, S_2 and S_3, each storing a fragment of G.

Suppose that one is looking for movies that he is interested in, then the search conditions can be expressed as a pattern query Q (Fig. 1(b)) as follows: (1) movies M should have high ratings, *e.g.,* $r > 7.0$, and are with genres "action" and "adventure"; (2) the M should be played by experienced performers P_1 and P_2. Specifically, P_1 (resp. P_2) played movie M_1 (resp. M_2) with $r > 7.0$, $g =$ "action" (resp. $g =$ "adventure") and $t_1 < t$ (resp. $t_2 < t$), where t (resp. t_1, t_2) is the release time of the M (resp. M_1, M_2).; and (3) the M is marked as "output node" with "*", *i.e.,* users only require the matches of M to be returned as search results.

The matches of Q, denoted as $M(Q,G)$, consists of a set of subgraphs in G that are isomorphic to Q. For example, $M(Q,G)=$ $\{\{(\mathsf{P}_1,\mathsf{p}_{11})(\mathsf{P}_2,\mathsf{p}_2)(\mathsf{M}_1,\mathsf{m}_9)(\mathsf{M}_2,\mathsf{m}_i)(\mathsf{M},\mathsf{m}_8)|i \in [3,5]\}$, $\{(\mathsf{P}_1,\mathsf{p}_{13})(\mathsf{P}_2,\mathsf{p}_{11})(\mathsf{M}_1,\mathsf{m}_{12})(\mathsf{M}_2,\mathsf{m}_j)(\mathsf{M},\mathsf{m}_{16})|j \in [8,9]\}$, $\{(\mathsf{P}_1,\mathsf{p}_{11})(\mathsf{P}_2,\mathsf{p}_{13})(\mathsf{M}_1,\mathsf{m}_k)(\mathsf{M}_2,\mathsf{m}_{12})(\mathsf{M},\mathsf{m}_{16})|k \in [8,9]\}\}$. Observe that (1) it takes $O(|G|!|G|)$ time to compute $M(Q,G)$, where $|G|$ is the size of G [3]; due to high computational cost, optimization techniques, *e.g.,* view based evaluation, are needed to speed up query evaluation; (2) since the graph G is distributively stored, no match can be found in a single site, which indicates that data has to be shipped from one site to another to find matches. With this comes the need for distributive techniques for graph pattern matching; (3) as the "query focus" of Q is M, "At World's End" and "Skyfall" are returned as query results. While in practice, users may be interested in the best matches, rather than the whole set of matches of "query focus" M, then a metric is needed to rank matches. For example, compared with "At World's End", "Skyfall" and its corresponding isomorphic subgraph have higher comprehensive rating, which makes it a better match than "At World's End". □

In light of these, we present MovieFinder, a novel system to effectively identify movies in social networks via top-k graph pattern matching. In contrast to previous graph search systems (see [7] for a survey), MovieFinder (1) supports graph pattern matching with subgraph isomorphism [3], and combines graph pattern matching with result ranking, (2) evaluates top-k graph pattern matching in a parallel manner, and (3) optimizes local evaluation by using materialized views, and maintains views via incremental techniques [6].

To the best of our knowledge, MovieFinder is among the first efforts to search movies on large and distributed social networks via graph pattern matching. It should also be remarked that movie searching is just one application of the technique, one may apply the technique to find *e.g.,* people, hotels, restaurants and so on.

2. DISTRIBUTED TOP-K GRAPH PATTERN MATCHING

We first review the notion of subgraph isomorphism. We then introduce graph fragmentation, followed by the problem of distributed top-k graph pattern matching.

Subgraph isomorphism. Given a data graph $G = (V, E, f_A)$ and a pattern query $Q = (V_p, E_p, f_v)$, a match of Q in G via subgraph isomorphism is a subgraph G_s of G that is isomorphic to Q, *i.e.,* there is a bijective function h from V_p to the node set of G_s such that (1) for each node $u \in V_p$, $f_v(u) = f_A(h(u))$; (2) (u, u') is an edge in Q if and only if $(h(u), h(u'))$ is an edge in G_s. We denote by $G[M(Q, G)]$ to be the union of all the matches G_s in $M(Q, G)$.

To find matches of query focus, we extend Q by specifying one node in Q as output node, denoted as u_o. Then, the answer to Q in G, denoted by $M(Q, G, u_o)$, is the set of nodes $h(u_o)$, that match the output node u_o of Q in G_s, for all matches G_s of Q in G.

Distributed graphs. A fragmentation $\mathcal{F}$ of a graph $G = (V, E, f_A)$ is $(F_1, \cdots, F_n)$, where each fragment F_i is specified by $(V_i \cup F_i.O, E_i, f_{A_i})$ such that (1) $(V_1, \cdots, V_n)$ is a partition of V; (2) $F_i.O$ is the set of nodes v' such that there exists an edge $e = (v, v')$ in E, $v \in V_i$ and node v' is in another fragment; we refer to v' as a virtual node and e as a crossing edge; and (3) $(V_i \cup F_i.O, E_i, f_{A_i})$ is a subgraph of G induced by $V_i \cup F_i.O$. We assume *w.l.o.g.* that each F_i is stored at site S_i for $i \in [1, n]$.

Distributed Top-k Graph Pattern Matching. Given an integer k, a pattern query Q with output node u_o and a fragmentation $\mathcal{F}$ of a graph G, the distributed top-k graph pattern matching problem is to find the best k matches to u_o of Q in G.

We next show how MovieFinder supports distributed top-k graph pattern matching via parallel computation that integrates asynchronous message passing with optimized local evaluation.

3. THE SYSTEM OVERVIEW

The architecture of the MovieFinder, shown in Fig. 2, consists of the following three components. (1) A *Graphical User Interface* (GUI), which provides a graphical interface to help users formulate pattern queries, manage data graphs and understand visualized results. (2) A *coordinator* that communicates with GUI and *workers* (to be introduced shortly). Specifically, the *coordinator* (a) forwards various requests, received from GUI, to *workers* for their local processing; (b) assembles partial results from *workers*; (c) ranks matches and returns best k ones as search results. (3) Multiple worker machines (*a.k.a.workers* [4, 8]), which employ Query Executor (QE) to compute local matches, and Incremental Computation Module (ICM) to keep materialized views up to date. We next present the components of MovieFinder and their interactions.

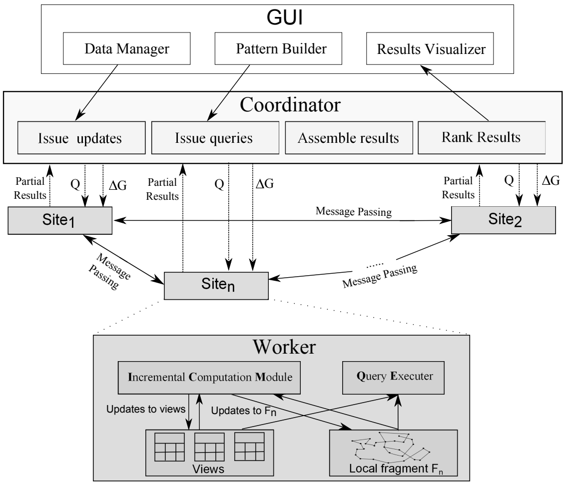

Figure 2: Architecture of MovieFinder

Graphical User Interface. The GUI helps to interact with users, *e.g.,* graph data manipulation, pattern query formulation, and result browse. Specifically, (1) It provides a task-oriented panel to facilitate users to manage graph data. (2) It is equipped with a query panel, which allows users to (a) manually construct a pattern query Q from scratch by drawing a set of query nodes and edges; (b) specify the search conditions of query nodes (*e.g.,* title="Skyfall", g="action & adventure"; $r \leq 7.0$; $t > t_1$); (c) indicate the particular "output" node for which users want to find matches (*e.g.,*M in Example. 1); (d) specify the number k of matches to the "output" node; and (e) designate query target from a list of data graphs. (3) The GUI visualizes query results by layout algorithm, hence the users can browse the matches with more intuition.

Coordinator. The *coordinator* interacts with GUI and *workers* as following. It (1) sends users' requests, received from GUI to *workers* for their local precessing, and returns query results to GUI for visualization; (2) collects partial results from *workers*, ranks matches based on the ranking metric, and identify best k matches.

Results Ranking. As there may exist a large set of matches of the output node u_o, and users may be only interested in the best k ones. The *coordinator* hence uses a ranking function to identify top-k matches. Intuitively, the ranking function follows one observation from social networks, that's the higher the rating of v and the total rating of G_s are, the better v is. To be more specific, given a pattern query Q with output node u_o, and a match G_s of Q with node v as the match of u_o, the rank of v is defined as:

$$f(v, u_o) = v.r * \Sigma_{v_i \in G_s} v_i.r$$

where $v.r$ (resp. $v_i.r$) indicates the rating of v (resp. v_i).

Example 2: Recall Example 1, the highest rating of the match in $M(Q, G)$ that contains m_8 (resp. m_{16}) is 8.9+7.3+7.1=23.3 (resp. 7.2+7.3+7.8=22.3). Then "Skyfall" makes the top-1 match since $f(\mathsf{m}_{16}, u_o) = 7.8 * 22.3 = 173.94$ is greater than $f(\mathsf{m}_8, \mathsf{u}_o) = 7.1 * 23.3 = 165.43$. □

Note that, though we used node attribute, *e.g.,*, movie rating, to define $f()$, while in general cases, other metrics which can be used to measure the "goodness" of matches can also be applied, and readily supported by the system.

Workers. Each *worker* has two modules: Query Executor (QE) and Incremental Computation Module (ICM).

Query Executor. The main task of the QE is query evaluation. As local information may not be sufficient to find matches, and query evaluation is computational expensive, the QE hence (1) applies multithreaded computation to collect necessary information from other sites, and integrates collected information with current fragment to conduct local evaluation; and (2) employs view-based technique to optimize evaluation of graph pattern matching.

(1) Local evaluation. Upon receiving pattern query Q from *coordinator*, the QE starts one thread to do the following. (a) It checks whether each virtual node v at current fragment F_i is a candidate match of some pattern node u, *i.e.,* v satisfies search conditions specified by u. (b) For each candidate match v, it then sends node pair $\langle u, v\rangle$ to the site S_j, where v accommodates; and requests the subgraph $G_j^N(u,v)$ of fragment F_j, where $G_j^N(u,v)$ contains neighborhood information of v in F_j, (see below for more details about computation of $G_j^N(u,v)$). (c) After all the $G_j^N(u,v)$ are received and merged with F_i, the QE computes matches with algorithm VF2 [3], and sends local results to the coordinator.

To response requests from other sites such that local evaluation can be processed in parallel at each site, the QE at site S_j constantly waits for messages from other sites, and initializes new threads to compute $G_j^N(u,v)$ when receiving messages $\langle u, v\rangle$ from other sites. Specifically, when message $\langle u, v\rangle$ sent from other site is received by site S_j, a new thread is started by the QE at S_j to conduct restricted breadth first search from v and u in F_j and Q, respectively. For any node v' (resp. u') encountered during the traversal in F_j (resp. Q), if v' is a candidate match of u', then v' is inserted in $G_j^N(u,v)$, and also connected to its neighbor nodes, which are already in $G_j^N(u,v)$.

Example 3: Recall pattern query Q in Example 1. Upon receiving Q, the QE at S_2 identifies m_{15} and m_{16} as the candidate match of the pattern nodes M_1, M_2 and M, and sends node pairs $\langle \mathsf{M}_1, \mathsf{m}_{15}\rangle$, $\langle \mathsf{M}_2, \mathsf{m}_{15}\rangle$, $\langle \mathsf{M}, \mathsf{m}_{15}\rangle$, $\langle \mathsf{M}_1, \mathsf{m}_{16}\rangle$, $\langle \mathsf{M}_2, \mathsf{m}_{16}\rangle$, $\langle \mathsf{M}, \mathsf{m}_{16}\rangle$ to S_3. Once receiving requests, S_3 computes $G_3^N(u,v)$ as response, *e.g.,* $G_3^N(\mathsf{M}, \mathsf{m}_{16})$, which includes two edges $(\mathsf{p}_{13}, \mathsf{m}_{12})$ and $(\mathsf{p}_{13}, \mathsf{m}_{16})$ are returned to S_2. After receiving the response, the QE at S_2 then merges $G_3^N(u,v)$ with F_2, invokes VF2 to compute $M(Q, F_2)$, and sends result $\{(\mathsf{P}_1, \mathsf{p}_{11})(\mathsf{P}_2, \mathsf{p}_{13})(\mathsf{M}_1, \mathsf{m}_i)(\mathsf{M}_2, \mathsf{m}_{12})(\mathsf{M}, \mathsf{m}_{16}) | i \in [8, 9]\}$ to the coordinator. □

(2) Optimization technique. As local evaluation involves subgraph isomorphism checking, which is an NP-complete problem and often computationally expensive, MovieFinder caches query results of commonly issued pattern queries at *workers* and adopts view-based technique to optimize local evaluation.

Suppose a set of view definitions $\mathcal{V} = \{\mathsf{V}_1, \cdots, \mathsf{V}_n\}$ have their extensions $M(\mathcal{V}, F_i) = \{M(\mathsf{V}_1, F_i), \cdots, M(\mathsf{V}_n, F_i)\}$ cached at site S_i. Given pattern query Q, the QE at site S_i computes matches of Q using $\mathcal{V}$ and $M(\mathcal{V}, F_i)$ as following. It first verifies whether Q can be answered using $\mathcal{V}$ by checking whether Q is the same as the union of $Q[M(\mathsf{V}_k, Q)]$ $(k \in [1, n])$. If Q can be answered by using $\mathcal{V}$, the algorithm Match, which takes Q, $\mathcal{V}$ and $M(\mathcal{V}, F_i)$ as input is then invoked to compute matches. Specifically, Match first initializes an empty pattern query $\mathsf{Q_s}$ and an empty set S as the match set of $\mathsf{Q_s}$. It then iteratively invokes Procedure Merge to "merge" $\mathsf{Q_s}$ with V_k, and matches in S with matches in $M(\mathsf{V}_k, F_i)$. In particular, Merge checks whether matches m_1 of $\mathsf{Q_s}$ can be merged with matches m_2 of V_k following the mapping λ that guides the "merge" of $\mathsf{Q_s}$ and V_k. If so, a new match m_0 of the newly formed pattern query $\mathsf{Q_s}$ (merged with V_k) is formed by merging m_1 with m_2, and the set S is updated by replacing m_1 with m_0. When the termination condition, *i.e.,* $\mathsf{Q_s} = Q$ is met, the set S is returned as the match set of Q at S_i.

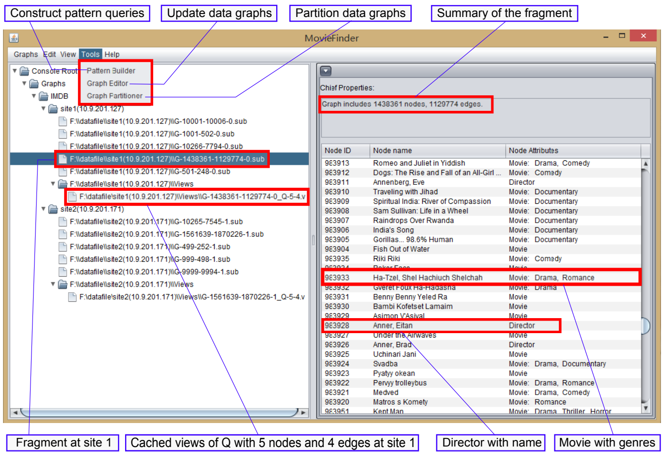

Figure 3: Visual interface: MovieFinder **Manager**

Example 4: Recall view definitions $\mathcal{V} = \{\mathsf{V}_1, \mathsf{V}_2\}$, shown in Fig. 1(c), their extensions $M(\mathcal{V}, F_3)$ at S_3 are listed in table below.

View definitions	Extensions
V_1	$\{(\mathsf{P}_1, \mathsf{p}_{13})(\mathsf{M}_1, \mathsf{m}_{12})(\mathsf{M}, \mathsf{m}_{16})\}$
V_2	$\{(\mathsf{P}_1, \mathsf{p}_{13})(\mathsf{M}_1, \mathsf{m}_{12})(\mathsf{M}, \mathsf{m}_{16})\}$

At site S_3, the QE computes matches of Q (see Fig. 1(b)) using $\mathcal{V}$ and $M(\mathcal{V}, F_3)$, as following. (1) It first determines that Q can be answered using $\mathcal{V}$ since Q is the same as $\bigcup_{i \in [1,2]} Q[M(\mathsf{V}_i, Q)]$. (2) It then invokes Match to compute matches. Since no match of V_1 and V_2 can be merged, following the mapping which guides the merge of V_1 and V_2, then no match of Q exists at site S_3. □

Incremental Computation Module. Real-life social networks change constantly, hence the cached views $M(\mathcal{V}, F_i)$ at site S_i need to be updated, in response to the changes to F_i. However, due to that subgraph isomorphism is computationally expensive and the input, *i.e.,* F_i, is often large, it is costly to recompute $M(\mathsf{V}, F_i \oplus \Delta F_i)$ for each $\mathsf{V} \in \mathcal{V}$, where $F_i \oplus \Delta F_i$ denotes F_i updated by ΔF_i. Instead of recomputation, the ICM incrementally identifies changes to $M(\mathcal{V}, F_i)$, in response to ΔF_i. As ΔF_i is often small in practice, the incremental computation hence is far more efficient than batch computation. The ICM applies the incremental subgraph isomorphism algorithm of [6] to update cached views, for both unit and batch updates.

Example 5: Recall Q, G in Example 1. Suppose that an edge e_1 (marked in red in Fig 1(a)) is inserted into G , then the change to G incurs four new matches: $\{(\mathsf{P}_1, \mathsf{p}_{13})(\mathsf{P}_2, \mathsf{p}_{11})(\mathsf{M}_1, \mathsf{m}_{15})(\mathsf{M}_2, \mathsf{m}_i)(\mathsf{M}, \mathsf{m}_{16}) | i \in [8, 9]\}$, and $\{(\mathsf{P}_1, \mathsf{p}_{11})(\mathsf{P}_2, \mathsf{p}_{13})(\mathsf{M}_1, \mathsf{m}_j)(\mathsf{M}_2, \mathsf{m}_{15})(\mathsf{M}, \mathsf{m}_{16}) | j \in [8, 9]\}$. Instead of recomputing $M(Q, G \oplus \Delta G)$ from scratch, the ICM only visits nodes that are 3 hops away from p_{13}, and identifies the new matches. □

Remark. The MovieFinder identifies all the matches of Q by exact algorithms, *i.e.,* VF2 or our view-based technique, at all *workers*, hence can find top-k matches of u_o with 100% accuracy.

4. DEMONSTRATION OVERVIEW

The demonstration is to show the following: (1) the use of GUI to formulate pattern queries and browse query results; (2) the efficiency of computation of $M(Q, G)$ and top-k matches of u_o when G is distributively stored; (3) effectiveness of view-based optimization technique employed by the QE; and (4) efficiency of the incremental technique applied by the ICM.

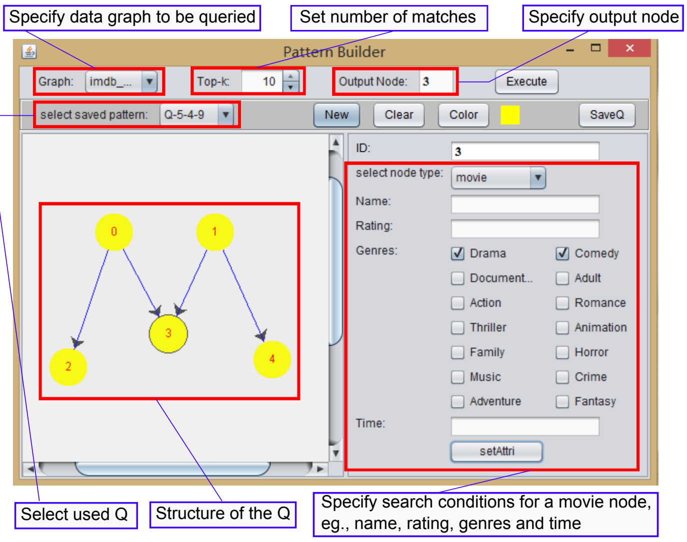

Figure 4: Visual interface: Pattern Builder

Setup. To show the performance of MovieFinder, we used a fraction of IMDb [2] with $|V|$=1.1M, $|E|$=1.7M, randomly partitioned it into a set of fragments controlled by the number of fragments $|\mathcal{F}|$. The system is implemented in Java and deployed with fragments on a cluster of 8 machines with 2.9GHz CPU, 8GB Memory.

Interacting with the GUI. We invite users to use the GUI, from pattern query construction to intuitive illustration of query results.
(1) The Manager panel, which is the main control panel of MovieFinder, is used to manipulate the system. As shown in Fig. 3, users can access each module of the MovieFinder as listed in the Tools menu, view both summarized and detailed information, *e.g.,* fragment summary, node attributes, of the selected site.
(2) The Pattern Builder (PB) panel, shown in Fig. 4, facilitates users' construction of pattern queries. Specifically, the PB (a) provides users with a canvas to create new query nodes (resp. edges), (b) allows users to specify search conditions on the query nodes, set output node u_o and the number k of its matches, and (c) supports users to save pattern queries, and reuse them afterwards. For example, a pattern query Q, shown in Fig. 4, is constructed to find movies that are (a) with genres "Drama" and "Comedy", (b) played by people (marked by node "0") who had performed "Romance" movies (marked by node "2"), and (c) directed by people (marked by node "1") who had directed "Action" movies (marked by node "4"). The query focus is marked as "output" node with dark border (node "3"). The pattern query Q can be saved for future use if it is frequently issued.
(3) The GUI provides intuitive ways to help users interpret query results. In particular, the GUI allows users to browse (a) all the matches *w.r.t.* Q, and (b) top-k matches *w.r.t.* u_o. As an example, the query results of Q, given in Fig. 4, are shown in Fig. 5, and the top-2 movies, *i.e.,* "White Collar" and "Our Footloose Remake" are marked with thickened border.

Performance of query evaluation. We also aim to show (a) the performance of the parallel computation supported by the MovieFinder, and (b) the performance of Query Executor (QE) and Incremental Computation Module (ICM) supported by workers.

Performance of parallel computation. We will show efficiency and scalability of parallel computation supported by MovieFinder. As will be seen, when the number $|\mathcal{F}|$ of sites increases from 4 to 8, the query time is reduced by 35%, in average.

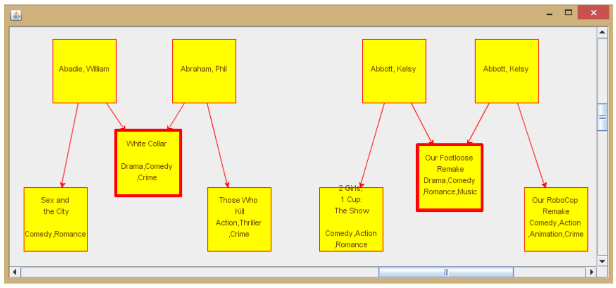

Figure 5: Visual interface: Query results

Performance of QE. We will show (a) the efficiency of QE by reporting its performance on IMDb; and (b) how substantial the performance is improved when view-based technique is applied. We show that in average the query time can be reduced by 70% with optimization technique.

Performance of ICM. We will also show the improvement of the ICM compared to batch computation that recomputes the materialized views in response to updates. In particular, we will report the performance of incremental computation by varying data graphs with unit update (single edge insertion/deletion) as well as batch updates (a list of edge insertions/deletions). As will be seen, the ICM performs significantly better than its batch counterparts, when data graphs are changed up to 30%.

Summary. This demonstration aims to show the key ideas and performance of the movie search system MovieFinder, based on the technique of distributed top-k graph pattern matching. The MovieFinder is able to (1) evaluate pattern queries defined in terms of subgraph isomorphism in parallel and identify top-k movies on large, distributively stored social networks; (2) efficiently compute matches with view-based technique; (3) incrementally maintain materialized views for dynamic social graphs; and (4) facilite users' use and understaing with intuitive graphical interface. These together convince us that the MovieFinder can serve as a promising tool for movie search on real-life social networks.

Acknowledgments. Wang is supported in part by NSFC 61402383 and 71490722, Sichuan Provincial Science and Technology Project 2014JY0207, and Fundamental Research Funds for the Central Universities, China.

5. REFERENCES

[1] Facebook statistics; visited december 2016. *http://expandedramblings.com/index.php/by-the-numbers-17-amazing-facebook-stats/.*
[2] Imdb dataset. *http://www.imdb.com/interfaces.*
[3] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. *TPAMI*, 26(10):1367–1372, 2004.
[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150, 2004.
[5] W. Fan. Graph pattern matching revised for social network analysis. In *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 8–21, 2012.
[6] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu. Incremental graph pattern matching. In *SIGMOD*, 2011.
[7] 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.
[8] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, 2010.