Graph Pattern Matching Revised for Social Network Analysis

Wenfei Fan
University of Edinburgh & Harbin Institute of Technology
wenfei@inf.ed.ac.uk

Abstract

Graph pattern matching is fundamental to social network analysis. Traditional techniques are subgraph isomorphism and graph simulation. However, these notions often impose too strong a topological constraint on graphs to find meaningful matches. Worse still, graphs in the real world are typically large, with millions of nodes and billions of edges. It is often prohibitively expensive to compute matches in such graphs. With these comes the need for revising the notions of graph pattern matching and for developing techniques of querying large graphs, to effectively and efficiently identify social communities or groups.

This paper aims to provide an overview of recent advances in the study of graph pattern matching in social networks. (1) We present several revisions of the traditional notions of graph pattern matching to find sensible matches in social networks. (2) We provide boundedness analyses of incremental graph pattern matching, in response to frequent updates to social networks. (3) To cope with large real-life graphs, we propose a framework of query preserving graph compression, which retains only information necessary for answering a certain class of queries of users’ choice. (4) We also address pattern matching in distributed graphs, and in particular, advocate the use of partial evaluation techniques. Finally, we identify directions for future research.

Categories and Subject Descriptors: H.2.3 [Information Systems]: Database Management – Query languages; G.2.2 [Discrete mathematics]: Graph Theory – Graph algorithms

General Terms: Languages, Algorithms, Design.

1. Introduction

Graph pattern matching has been a longstanding issue for more than 30 years. Given a pattern graph \( Q \) and a data graph \( G \), it is to find all matches in \( G \) for \( Q \), denoted by \( M(Q,G) \). Here matching is typically defined in terms of:

- subgraph isomorphism [60]: \( M(Q,G) \) consists of all subgraphs \( G' \) of \( G \) to which \( Q \) is isomorphic, i.e., there exists a bijective function \( h \) from the nodes of \( Q \) to the nodes of \( G' \) such that \((u,u') \) is an edge in \( Q \) if and only if \((h(u),h(u')) \) is an edge in \( G' \); or
- graph simulation [42]: \( M(Q,G) \) is a binary relation \( S \subseteq V_Q \times V_G \), where \( V_Q \) and \( V_G \) are the set of nodes in \( Q \) and \( G \), respectively, such that:
  - for each node \( u \) in \( V_Q \), there exists a node \( v \) in \( V_G \) such that \((u,v) \in S \), and
  - for each \((u,v) \in S \) and each edge \((u,u') \) in \( Q \), there is an edge \((v,v') \) in \( G \) such that \((u',v') \in S \).

Graph pattern matching has been extensively studied for pattern recognition, knowledge discovery, biology, cheminformatics, dynamic network traffic and intelligence analysis, based on subgraph isomorphism (see [2, 16, 28, 57] for surveys). Graph pattern matching with graph simulation has been widely used in process calculus, Web site classification and social position detection (e.g., [6, 13, 45, 64]).

Recently there has been renewed interest in graph pattern matching for social network analysis. Social networks are often modeled as graphs in which a node denotes a person, and an edge indicates some relationship, e.g., in Facebook, Twitter and LinkedIn. To identify social communities and social positions, graph pattern matching is a routine process.

However, social networks introduce new challenges to graph pattern matching, from its definition to processing methods. (1) Real-life social graphs are typically large. For instance, Facebook has more than 500 million users (nodes) with 65 billion links (edges) [1]. It is often prohibitively expensive to query such large graphs. In particular, subgraph isomorphism is a \( \text{NP} \)-complete problem (cf. [29]), and worse still, there are possibly exponentially many subgraphs in \( G \) that match \( Q \). These hinder its applicability in social networks. (2) While graph simulation can be computed in quadratic time, this notion and subgraph isomorphism are \( \text{NP} \)-complete [2]. These hinder its applicability in social networks. (3) To cope with large real-life graphs, we propose a framework of query preserving graph compression, which retains only information necessary for answering a certain class of queries of users’ choice. (4) We also address pattern matching in distributed graphs, and in particular, advocate the use of partial evaluation techniques. Finally, we identify directions for future research.

Example 1: Consider the structure of a drug trafficking organization [46], depicted as a pattern graph \( Q_0 \) in Fig. 1. In such an organization, a “boss” (B) oversees the operations through a group of assistant managers (AM). An AM supervises a hierarchy of low-level field workers (FW), up to 3 levels as indicated by the edge label 3. The FWs deliver drugs, collect cash and run other errands. They report to AMs directly or indirectly, while the AMs report directly to the boss. The boss may also convey messages through a secretary (S) to the top-level FWs as denoted by the edge label 1.
A bounded simulation relation that represents exact matches in $G$ for $Q$. Contrast this with graph pattern matching via subgraph isomorphism, which is intractable to determine.

**Strong simulation.** There is a trade-off between the low complexity of bounded simulation and its ability to preserve the topology of data graphs in its match relation. As a consequence, a graph $G$ may turn out to match a pattern $Q$ that has a structure quite different from $G$. To circumvent this limitation, one may adopt the notion of strong simulation [38], which extends simulation by imposing two conditions: (a) the “duality” to preserve upward mappings, and (b) the locality to eliminate excessive matches.

Strong simulation captures the topology of patterns in its matches, such as “parents”, connectivity and cycles, while it takes cubic time to compute, as bounded simulation. Moreover, it excludes excessive matches and improves the quality of matches found. In addition, slight extensions to the notion make graph pattern matching intractable.

**Coping with large social graphs.** The cubic-time complexity of bounded (strong) simulation still makes graph pattern matching infeasible when conducted on social graphs with millions of nodes and billions of edges. This highlights the quest for effective techniques to query large graphs. One way around this is to develop inexact or approximate matching algorithms, a topic that has been well studied (see, e.g., [16, 28] for surveys). Alternatively, we advocate three approaches to computing exact matches, without compromising the accuracy of matches, outlined as follows.

**Incremental graph pattern matching.** Social networks are frequently updated. It is too costly to recompute all matches in $M(Q, G \oplus \Delta G)$ starting from scratch when changes $\Delta G$ are inflicted on $G$. This motivates us to adopt incremental algorithms for graph pattern matching. That is, we compute matches $M(Q, G)$ once on the entire graph via a batch algorithm, and then incrementally identify the changes $\Delta M$ to $M(Q, G)$ in response to $\Delta G$, by making maximal use of previous computation $M(Q, G)$, without paying the price of the high complexity of the batch algorithms. It is known that while real-life graphs are constantly updated, the changes are typically small [47]. When $\Delta G$ is small, $\Delta M$ is often small as well, and is much less costly to find than to recompute the entire $M(Q, G \oplus \Delta G)$. Hence the incremental approach is often more efficient than its batch counterpart.

As argued in [53], incremental algorithms should be analyzed in terms of $[\text{CHANGED}]$, the size of the changes in the input and output, which represents the updating costs that are inherent to the incremental matching problem itself. An incremental algorithm is said to be bounded if its cost can be expressed as a function of $[\text{CHANGED}]$, i.e., it depends only
on \([\text{CHANGED}]\), rather than on the size of the entire input (data graph \(G\) and pattern \(Q\)). An incremental matching problem is said to be \text{bounded} if there exists a bounded incremental algorithm for it, and is \text{unbounded} otherwise.

We present an account of results on the boundedness analysis of incremental matching defined in terms of graph simulation, bounded simulation or subgraph isomorphism. The main result is negative: the problem is unbounded even for patterns of a restrictive form and updates consisting of a single edge insertion or deletion. Nonetheless, there exists an incremental algorithm for matching via bounded simulation such that its cost is a \(\text{PTIME}\) function of \([\text{CHANGED}]\) and the size of pattern \(Q\); i.e., although it is unbounded, it is independent of the size of graph \(G\). This often suffices since in practice, \(Q\) is typically \text{much smaller} than \(G\).

\textit{Query preserving graph compression.} It is unlikely that we can lower the complexity of computing the set \(M(Q,G)\) of matches. Moreover, incremental pattern matching does not help us improve batch computation. To this end one may consider \textit{query preserving graph compression:} for a class \(Q\) of queries, we find a smaller graph \(G_c\), for a given graph \(G\) via an efficient compression function, such that for all queries \(Q \in \mathbb{Q}, Q(G_G)\) can be found by computing \(Q(G_c)\), the answer to \(Q\) in the smaller \(G_c\). In other words, while we may not change the complexity functions of graph queries, we reduce the size of their parameters, i.e., the data graphs.

This approach has been verified effective for graph pattern matching in a variety of real-life social graphs, reducing the graphs by 57% in average [24]. In contrast to lossless compression schemes (e.g., [5, 12, 25]), query preserving compression is \textit{relative} to a class \(Q\) of queries of users’ choice, i.e., it generates small \(G_c\) that preserves the information \textit{only relevant} to queries in \(Q\) rather than for the entire original graph \(G\). Hence, it achieves a better compression ratio. Moreover, any algorithm available for evaluating \(Q\) can be directly used to query \(G_c\), as is, without decompressing \(G_c\).

\textit{Distributed graph pattern matching.} Another approach for speeding up graph pattern matching in large social graphs is to employ distributed algorithms. One may partition a large graph \(G\) into fragments and distribute the fragments across different sites. Given a pattern graph \(Q\), we partially evaluate \(Q\) over these fragments in parallel, and assemble the partial results to get the set \(M(Q,G)\) of matches for \(Q\) in the entire graph \(G\). That is, we divide a large computational task into smaller ones of manageable sizes, and explore parallelism to conduct the computation. In fact many large real-life graphs are already fragmented and stored distributively in different sites, e.g., social networks [55], Web services networks [43] and RDF graphs [50].

It is natural to conduct distributed graph pattern matching by using partial evaluation (see [34] for a survey). Preliminary results [9, 15, 23] show that partial evaluation techniques yield distributed query evaluation algorithms with several performance guarantees: (a) each site is visited a \textit{fixed} number of times, (b) the communication cost (network traffic) is determined by the fragmentation of \(G\) and the size of \(Q\), \textit{independent} of the size of \(G\), and (c) the computational cost (response time) is determined by \(Q\) and the largest fragment of \(G\) in the partition, again \textit{independent} of the size of \(G\), by capitalizing on parallel computation.

\textit{Organization.} This paper aims to provide an informal overview of important issues in the area, to incite interest. A survey of graph pattern matching algorithms is beyond the scope of this paper. In the rest of the paper, Section 2 reviews the traditional notions of graph pattern matching, and Section 3 introduces their revisions. Section 4 presents techniques for querying large social graphs, including incremental graph pattern matching, query preserving graph compression and distributed graph pattern matching. Finally, Section 5 identifies open research issues.

\section{Traditional Graph Pattern Matching}

We begin with basic notations of data graphs and pattern graphs. We then review traditional graph pattern matching.

\textbf{Data graphs.} A \textit{data graph} is a directed graph \(G = (V,E,f)\), where

\begin{itemize}
  \item \(V\) is a finite set of nodes;
  \item \(E \subseteq V \times V\), in which \((v,v')\) denotes an edge from node \(v\) to \(v'\); and
  \item \(f_{1}(\cdot)\) is a function that associates each node \(v\) in \(V\) with a tuple \(f_{1}(v) = (A_{1} = a_{1}, \ldots, A_{n} = a_{n})\), where \(A_{1}\) is a constant, and \(A_{i}\) is referred to as an \textit{attribute} of \(v\), written as \(v.A_{i}\), carrying the contents of the node, e.g., label, keywords, blogs, rating.
\end{itemize}

\textbf{Pattern graphs.} A \textit{pattern graph} is defined as \(Q = (V_Q, E_Q, f_Q)\), where

\begin{itemize}
  \item \(V_Q\) is a finite set of nodes and \(E_Q\) is a set of directed edges, as defined for data graphs;
  \item \(f_{2}(\cdot)\) is a function defined on \(V_Q\) such that for each node \(u\), \(f_{2}(u)\) is the \textit{predicate} of \(u\), defined as a conjunction of atomic formulas of the form \(A \text{ op } a\); here \(A\) denotes an attribute, \(a\) is a constant, and \(\text{op}\) is one of the comparison operators \(<, \leq, =, \neq, >, \geq\); and
  \item \(f_{3}(\cdot)\) is a function defined on \(E_Q\) such that for each edge \((u,u')\) in \(E_Q\), \(f_{3}(u,u')\) is either a positive integer \(k\) or a symbol \(\ast\).
\end{itemize}

Intuitively, the predicate \(f_{2}(u)\) of a node \(u\) specifies a search condition. We say that a node \(v\) in a data graph \(G\) \textit{satisfies the condition} of a pattern node \(u\) in \(Q\), denoted as \(v \sim u\), if for each atomic formula \(A \text{ op } a\) in \(f_{2}(u)\), there exists an attribute \(A\) in \(f_{1}(v)\) such that \(v.A \text{ op } a\).

As will be seen in Section 3, an edge \((u,u')\) in \(Q\) may be mapped to a path \(\rho\) in a data graph \(G\), and \(f_{3}(u,u')\) imposes a bound on the length of \(\rho\).

We refer to \(Q\) as a \textit{simple pattern} if (a) for each node \(u\) in \(V_Q\), \(f_{2}(u)\) is a predicate \(A = a\) defined on a unique attribute, referred to as the \textit{label} of \(u\); and (b) for each edge \((u,u')\) in \(E_Q\), \(f_{3}(u,u') = 1\). Intuitively, a simple pattern inspects label equality and enforces edge to edge mappings, as in subgraph isomorphism and graph simulation.

\textbf{Graph pattern matching.} Consider a data graph \(G = (V,E,f_{1})\) and a simple pattern \(Q = (V_Q, E_Q, f_{2}, f_{3})\).

\textbf{Subgraph isomorphism.} A subgraph \(G' = (V',E',f'_{1})\) of \(G\) \textit{matches} \(Q\), denoted as \(Q \subseteq_{iso} G'\), if there exists a \textit{bijective} function \(h(\cdot) : V \rightarrow V'\) such that
we have to revise the traditional notions of graph pattern matching. In this section we present several revisions.

3.1 Matching with Bounded Simulation

The first revision is based on a notion of bounded simulation, which revises graph simulation by allowing edge-to-path mappings rather than edge-to-edge mappings, and by supporting search conditions beyond label equality.

Recall data graphs and pattern graphs defined in Section 2. Bounded simulation is defined for pattern graphs \( Q = (V_Q, E_Q, f_Q, f_A) \) in which the search condition \( f_A(u) \) for each node \( u \in V_Q \) is a conjunction of atomic formulas, and the edge label \( f_e(u, u') \) for each \( (u, u') \in E_Q \) is either a positive integer \( k \) or a symbol \( * \). In a data graph \( G = (V, E, f_A) \), a path \( \rho \) from node \( v \) to \( v' \) is a sequence \( (v = v_0, v_1, \ldots, v_n = v') \) such that \( (v_{i-1}, v_i) \in E \) for all \( i \in [1, n] \). The length of \( \rho \), denoted by \( \text{len}(\rho) \), is the number of edges on \( \rho \). A path \( \rho \) is said to be nonempty if \( \text{len}(\rho) \geq 1 \).

**Bounded simulation** [20]. A data graph \( G \) matches a pattern \( Q \) via bounded simulation, denoted by \( Q \subseteq_{\text{sim}}^B G \), if there exists a binary relation \( S \subseteq V_Q \times V \) such that

- for all \( u \in V_Q \), there is a node \( v \in V \) such that \( (u, v) \in S \); and
- for each pair \( (u, v) \in S \),
  - \( u \sim v \), and
  - for each edge \( (u, u') \in E_Q \), there exists an edge \( (v, v') \in E \) such that \( \langle u', v' \rangle \in S \).

We refer to \( S \) as a match in \( G \) for \( Q \).

It is known that if \( Q \subseteq_{\text{sim}} G \), then there exists a unique maximum match \( S_0 \) [32], i.e., for any match \( S \) in \( G \) for \( Q \), \( S \subseteq S_0 \). We define \( M_{\text{sim}}(Q, G) = S_0 \) if \( Q \subseteq_{\text{sim}} G \), and \( M_{\text{sim}}(Q, G) = \emptyset \) otherwise.

**Example 2**: Consider the simple pattern \( Q_1 \) and data graphs \( G_1, G_2 \) and \( G_3 \) shown in Fig. 2, where a node from a data graph satisfies the search condition of a pattern node if they have the same label. Observe the following.

1. \( Q_1 \subseteq_{\text{sim}} G_1 \). In contrast, no subgraph of \( G_2 \) or \( G_3 \) is isomorphic to \( Q_1 \), i.e., \( M_{\text{sim}}(Q_1, G_i) \) is empty for \( i \in \{2, 3\} \).

2. \( Q_1 \subseteq_{\text{sim}} G_1 \) and \( Q_1 \subseteq_{\text{sim}} G_2 \). A simulation match is a relation that maps a pattern node to multiple nodes in a graph, as opposed to functions in subgraph isomorphism. For example, node \( C \) in \( G_1 \) is mapped to two \( C \) nodes in \( G_2 \).

3. In contrast, \( Q_1 \nsubseteq_{\text{sim}} G_3 \), i.e., \( M_{\text{sim}}(Q_1, G_3) \) is empty since the node \( A \) is not adjacent to \( C \) in \( G_3 \).

The graph pattern matching problem. Given a simple pattern \( Q \) and a data graph \( G \), the graph pattern matching problem via subgraph isomorphism (resp. graph simulation) is to compute \( M_{\text{sim}}(Q, G) \) (resp. \( M_{\text{sim}}(Q, G) \)), i.e., it is to find all the subgraphs of \( G \) that are isomorphic to \( Q \) (resp. the unique maximum match in \( G \) for \( Q \)).

3. Graph Pattern Matching Revised

As remarked earlier, to effectively and efficiently identify matches in emerging applications such as social networks, we have to revise the traditional notions of graph pattern matching. In this section we present several revisions.

**Theorem 1** [20]: For any data graph \( G = (V, E, f_A) \) and pattern graph \( Q = (V_Q, E_Q, f_Q, f_A) \),

- there exists a unique maximum match \( M_{\text{sim}}^B(Q, G) \) in \( G \) for \( Q \), and
the relation $M^B_{sim}(Q, G)$ can be computed in $O(|V| |E| + |E_Q| |V|^2 + |V_Q||V|)$ time.

Observe that $|M^B_{sim}(Q, G)| \leq |V||V_Q|$, in contrast to possibly exponentially large $M_{bsim}(Q, G)$.

Given a pattern graph $Q$ and a data graph $G$, the graph pattern matching problem via bounded simulation is to compute the maximum match $M^B_{sim}(Q, G)$.

Remarks. To see the differences between various graph pattern matching metrics, observe the following.

(1) As opposed to subgraph isomorphism, bounded simulation supports (a) simulation relations rather than bijective functions, (b) predicates specifying search conditions based on the contents of nodes, and (c) edges to be mapped to (bounded) paths instead of edge-to-edge mappings.

(2) Graph simulation is a special case of bounded simulation, by only allowing simple patterns in which (a) all the nodes carry their labels as the only attributes, and (b) all the edges are labeled with 1, i.e., edge-to-edge mappings only.

(3) In contrast to the NP-hardness of subgraph isomorphism, graph pattern matching based on bounded simulation is in cubic-time. Compared to graph simulation, bounded simulation does not make our lives much harder. Indeed, it takes $O((|V| + |V_Q|)(|E| + |E_Q|))$ time to decide graph simulation from $Q$ to $G$ [32]. In practice, $Q$ is typically small.

(4) The notion of homeomorphism also allows edge-to-path mappings [27]. A graph $H = (V_H, E_H)$ is homeomorphic to a graph $G = (V, E)$ if there exists an injective function $h : V_H \rightarrow V$ such that $h$ maps edges in $E_H$ to pairwise node-disjoint simple paths of $G$. Bounded simulation differs from homeomorphism in that it is defined in terms of a relation rather than an injective function, and moreover, it does not require edges to be mapped to node-disjoint simple paths.

(5) Recall the notion of graph monomorphism: a graph $Q = (V_Q, E_Q)$ is monomorphic to a graph $G = (V, E)$ if there exists an injective function $h : V_Q \rightarrow V$ such that for each edge $(u, u') \in E_Q$, $h(u), h(u')$ is also an edge in $E$ (cf. [16]). One might be tempted to extend graph monomorphism by allowing edge-to-path mappings, and define graph pattern matching based on this revised notion. Unfortunately, this revision makes graph pattern matching intractable [21].

Theorem 2 [21]: It is NP-complete to decide whether there exists a subgraph of a graph $G$ that matches a pattern graph $Q$ via the revised monomorphism. It remains NP-hard even when $Q$ is a tree and $G$ is an acyclic directed graph (DAG).

(6) A notion of weak simulation was proposed in [45], which extends graph simulation by mapping an edge to an unbounded path, denoted by $\preceq_{wsim}$. It is a special case of bounded simulation, when all the edges in a pattern graph are labeled with $. A graph $H$ is said to be weakly similar to another graph $G$ if $H \preceq_{wsim} G$ and $G \preceq_{wsim} H$. It is shown [45] that the following problem is NP-complete: given two graphs $G$ and $H$, it is to decide whether there exists a subgraph of $G$ that is weakly similar to $H$.

(7) A revision of subgraph isomorphism was studied in [64], which allows edges in a pattern to be mapped to paths in a data graph with the same bound. Graph pattern matching defined in terms of this notion remains NP-complete.

(8) It is shown in [19, 20] experimentally that graph pattern matching based on bounded simulation is able to accurately identify a number of communities in real-life social networks that its traditional counterparts fail to find.

3.2 Incorporating Edge Relationships

Bounded simulation can be readily extended to incorporate edge relationships. To do this we first revise the specifications of data graphs and pattern graphs as follows. Let $\Sigma$ be a finite alphabet of colors denoting edge relationships, e.g., marriage, friendship, work, advice, support [40].

Data graphs and pattern graphs. A data graph is a directed graph $G = (V, E, f_A, f_C)$, where $V, E, f_A$ are as defined earlier in Section 2, and $f_C(\cdot)$ is a function defined on $E$ such that for each edge $e \in E$, $f_C(e)$ is a color in $\Sigma$.

To define pattern graphs, a fragment of regular expressions of the following form was adopted in [19]:

$$F ::= c | c^* | F F.$$  

Here (1) $c$ is either a color in $\Sigma$, or a wildcard _ that matches any color in $\Sigma$ (expressed as a regular expression $c_1 | \ldots | c_m$, when $\Sigma = \{c_i \mid i \in [1, m]\}$); (2) $k$ is a positive integer, and $c^{\leq k}$ denotes the regular expression $c^* \cup c^2 \cup \ldots \cup c^k$, where $c^j (j \in [1, k])$ denotes $j$ occurrences of $c$; and (3) $c^*|$ denotes one or more occurrences of $c$.

As argued in [19], these regular expressions suffice to express edge relationships commonly found in practice. Moreover, their containment and equivalence problems can be decided in linear-time as opposed to PSPACE-completeness for general regular expressions. This allows us to develop effective optimization strategies for such pattern queries.

A regular pattern is defined as $Q = (V_Q, E_Q, f_A, f_C)$, where $V_Q, E_Q$ and $f_A, f_C$ are as defined in Section 2, and for each edge $(u, u') \in E_Q$, $f_C(u, u')$ is a regular expression in $F$. The edge $(u, u')$ is to be mapped to a path of an unbounded length if $f_C(u, u')$ contains $c^*$ for some color $c$, and it is bounded otherwise (e.g., when $f_C(u, u')$ is $c^{\leq k}$).

Regular pattern matching [19]. A data graph $G$ matches a regular pattern $Q$, denoted by $Q \preceq_{\text{reg}} G$, if there exists a binary relation $S \subseteq V_Q \times V$ such that

- for each node $u \in V_Q$, there exists a node $v \in V$ such that $(u, v) \in S$;
- for each pair $(u, v) \in S$, $u \sim v$, and
  - for each edge $(u, u') \in E_Q$, there exists a path $p = (v = v_0, v_1, \ldots, v_n = v')$ in $G$ such that the sequence $f_C(v_0, v_1) \ldots f_C(v_{n-1}, v_n)$ of edge labels is a string in the language $L(f_C(u, u'))$ of the regular expression $f_C(u, u')$.

Intuitively, a pattern edge $(u, u')$ must be mapped to a path in the data graph $G$ such that the edge colors on the
path match the pattern specified by the regular expression
\( f_c(u, u') \). Note that bounded simulation is a special case of
regular pattern matching, when \( \Sigma \) consists of a single color.

**Example 4:** Consider an *Essemble* network service [7],
where users post and vote on controversial issues and topics.
Each person has attributes such as userid, job, as well
as a list of issues they support or disapprove, denoted by
“sp” and “dsp”, respectively. There are four types of
relationships between a pair of persons: (1) *friends-allies (fa)*,
connecting one user to a friend, if she shares the same views
on most (more than half) topics that her friend votes for;
(2) *friends-nemeses (fn)*, from one user to a friend if she
disagrees with her friend on most topics; and (3-4) similarly
*strangers-allies (sa) and strangers-nemeses (sn)* are defined,
relating a user to a stranger. Figure 3 depicts a part of the
network as a data graph \( G_4 \) that involves a debate on cloning
research. In \( G_4 \), each edge has a type in \{fa, fn, sa, sn\}.

Consider a regular pattern \( Q_3 \) posed on \( G_4 \), which is also
shown in Fig. 3. It is issued by a person \( D \) identified by
id “Alice001” who supports “cloning”. The person would
like to find all her friends-nemeses (via fn) who are doctors,
and are against “cloning” (node \( B \)). She also wants to know
if there are people such that (a) they are biologists (nodes
\( C \)), support “cloning research”, and are connected within
2 hops to someone via fa relationships, who is in turn within 2
hops to person \( D \) via sa (edge \( C, D \)); (b) they are in a scientist
in \( Q_3 \) is mapped to a path
\( C_3 \) → \( C_1 \) → \( D_1 \) in \( G \); similarly for other edges in \( Q_3 \). □

**Complexity.** One can define the maximum match in a data
graph \( G \) for a regular pattern \( Q \), denoted by \( M_{sim}^{R}(Q, G) \),
along the same lines as its counterpart for bounded simulation.
The result below tells us that adding edge relationships
does not incur extra complexity to graph pattern matching.
Note that the complexity might be higher if general regular
expressions were adopted in pattern graphs.

**Theorem 3** [19]: For any data graph \( G = (V, E, f_A, f_C) \)
and regular pattern \( Q = (V_Q, E_Q, f_e, f_c) \),

\[ M_{sim}^{R}(Q, G) = \sum_{i=1}^{k} \lambda_i M_{sim}^{R}(Q_i, G) \]

where \( \lambda_i \) is the weight assigned to the \( i \)-th component of \( Q \).

The low complexity of graph pattern matching based on
(bounded) simulation comes at a price: (bounded) simulation
may match a graph \( G \) and a pattern \( Q \) with radically
different structures, as illustrated by the following example.

**Example 5:** Consider a real-life example taken from [59].
A headhunter wants to find a biologist (Bio) to help a group
of software engineers (SE’s) analyze genetic data. To do
this, she uses an expertise recommendation network \( G_5 \),
as depicted in Fig. 4. In \( G_5 \), a node denotes a person labeled
with expertise, and an edge indicates recommendation, e.g.,
HR recommends Bio1 in the figure there is an edge from
each DMi (data mining specialist) to Bioi, for \( i \in [1, k] \).

The biologist Bio needed is specified with a pattern graph
\( Q_4 \), also shown in Fig. 4. Intuitively, the Bio has to be
recommended by: (a) an HR person; (b) an SE, i.e., the Bio
has experience working with SE’s; and (c) a DM, as data
mining techniques are needed for the job. Moreover, (d)
the SE is recommended by the same HR who recommends
the Bio, and (e) there is an artificial intelligence expert (AI)
who recommends the DM and is recommended by a DM.
The edges in \( Q_4 \) are labeled 1 (omitted).

When subgraph isomorphism is used, no match can be
found, i.e., no subgraph of \( G_5 \) is isomorphic to \( Q_4 \).

When (bounded) graph simulation is adopted, *all* four bi-
ologists in \( G_5 \) are matches of Bio in \( Q_4 \). However, Bio1
and Bio2 are recommended by either HR only or by SE only
in \( G_5 \), and Bio3 by neither an HR nor an SE. Hence they are
not the ones that the headhunter really wants. Only Bio4
satisfies all these conditions and makes a good candidate.

This tells us that (bounded) simulation does not preserve
the topology well, and may return excessive “matches” that
one does not want. Indeed, observe the following. (a) While
\( Q_4 \) is a connected graph (via undirected paths), \( G_5 \) is
disconnected, but \( G_5 \) matches \( Q_4 \) via simulation. (b) Node Bio
in \( Q_4 \) has three “parents”, but it matches nodes Bio3 and
Bio2 in \( G_5 \) that have a single “parent” each. Here abusing
notations of trees, for a pair \( u, u' \) of nodes, we refer to \( u' \)
as a *child* of \( u \) (or \( u \) as a *parent* of \( u' \)) if there exists an edge
\((u, u')\). (c) The directed cycle with two nodes Al and DM
in \( Q_4 \) matches the long cycle consisting of Al1, DM1, \ldots, Alk,
DM₄, AI₁ in G₅, and the undirected cycle with nodes HR, SE and Bio in Q₄ matches the tree rooted at HR₁ in G₅.

Strong simulation [38]. To circumvent the limitations of (bounded) simulation, one can use a notion of strong simulation by imposing two conditions on simulation [42]: duality and locality. These conditions aim to capture the topology of graphs and eliminate excessive matches, while retaining a low PTIME computational complexity. To simplify the discussion we define strong simulation as a revision of graph simulation. Nevertheless, this notion can be readily defined for bounded simulation with regular graph patterns.

Dual simulation. A data graph G matches a simple pattern Q based on dual simulation, denoted by \( Q \preceq_D G \), if:

- \( Q \preceq_D G \) with a binary match relation \( S \subseteq Q \times V \), and
- for each pair \( (u, v) \in S \) and each edge \( (u₂, u) \in E_q \), there exists an edge \( (v₁, v) \in E \) such that \( (u₂, v₁) \in S \).

Intuitively, dual simulation enhances graph simulation by preserving both the child and parent relationships.

One can verify that for any simple pattern Q and data graph G, there is a unique maximum match relation based on dual simulation, along the same lines as Theorem 1.

Locality. We enforce the locality by requiring matches to be contained in a subgraph of a certain radius. Indeed, as observed in [8], when social distance increases, the closeness of relationships decreases and the relationships may become irrelevant. Therefore, it often suffices in practice to consider only those matches of a pattern that fall in a small subgraph. To specify the locality, we need the following notions.

(1) A graph \( (V_s, E_s, f_s) \) is a subgraph of data graph \( G = (V, E, f_A) \), denoted as \( G[V_s, E_s] \), if (a) for each \( v \in V_s \), \( v \in V \) and \( f_s(v) = f_A(v) \), and (b) for each edge \( e \in E_s \), \( e \in E \).

(2) An undirected path \( p \) in a data graph G is a sequence of nodes \( (v₁, \ldots, v_n) \) such that either \( (v_i, v_{i+1}) \) or \( (v_{i+1}, v_i) \) is an edge in G for all \( i \in [1, n-1] \).

(3) Given two nodes \( v, v' \) in a graph G, the distance between \( v \) and \( v' \), denoted by \( \text{dist}(v, v') \), is the length of the shortest undirected path between \( v \) and \( v' \) in G.

The diameter of a graph G, denoted by \( d_G \), is the longest shortest distance between all pairs of nodes in G, i.e., \( d_G = \max(\text{dist}(v, v')) \) for all nodes \( v, v' \) in G.

(4) For a node \( v \) in a graph G and a non-negative integer \( r \), the \( r \)-radius subgraph centered at v is a subgraph of G, denoted by \( G[v, r] \), such that (1) it contains all and only those nodes \( v' \) in G with \( \text{dist}(v, v') \leq r \), and (2) it has exactly the edges that appear in G over the same node set.

We will enforce the locality by using the following notion. Consider a subgraph \( G[v, r] \) such that \( Q \preceq_D G[v, r] \) with the maximum match relation \( S \). The match graph \( w.r.t. S \) is a graph \( G_s = (V_s, E_s) \) in which (1) a node \( v \in V_s \) if f is in \( S \), and (2) an edge \( (u, v') \in E_s \) if there is an edge \( (u, u') \) in G with \( (u, v) \in S \) and \( (u', v') \in S \).

Strong simulation. A data graph G matches a simple pattern Q via strong simulation, denoted by \( Q \preceq_S G \), if:

- there is a node \( v \) in G such that \( Q \preceq_D G[v, d_q] \), and
- the match graph \( G_s \) \( w.r.t. S \) is a subgraph of \( G[v, d_q] \),

where \( d_q \) is the diameter of \( Q \), and \( S \) is the maximum match for \( Q \preceq_D G[v, d_q] \). We refer to \( G_s \) as a match in G for Q.

Intuitively, a match \( G_s \) of Q satisfies the following conditions: (1) it preserves both the child and parent relationships of Q; and (2) \( G_s \) is contained in a subgraph \( G[v, d_q] \) of G with a radius bounded by the diameter of Q; and (3) all the nodes and edges needed to match Q in \( G[v, d_q] \) are contained in \( G_s \); these rules out excessively large matches.

Example 6: Consider pattern graph Q₄ and data graph G₅ of Fig. 4. Observe the following. (1) No subgraph of G₅ is isomorphic to Q₄. Indeed, there exists no directed cycle in G₅ that matches the direct cycle DM₁, DM₂ in Q₄.

(2) When simulation is adopted, the entire data graph G₅ is included in the match relation, which maps HR, SE, Bio, DM and AI in Q₄ to \{HR₁, HR₂\}, \{SE₁, SE₂\}, \{Bio₁, Bio₂, Bio₃, Bio₄\}, \{DM¹, DM₂, DM₃, ..., DM₅\} and \{AI₁, AI₂, AI₃, ..., AI₉\} in G₅, respectively.

(3) When it comes to strong simulation, the connected component Gₛ of G₅ that contains Bio₄ is the only match, which maps HR, SE, Bio, DM and AI in Q₄ to \{HR₁\}, \{SE₂\}, \{Bio₄\}, \{DM₁, DM₂\} and \{AI₁, AI₂\} in G₅, respectively. Indeed, one can verify the following: (1) Q₄ \( \preceq_D G_s \), mapping Bio in Q₄ only to Bio₄ in G₅; and (b) the subgraph centered at Bio₄ with radius 3 (the diameter of Q₄) is exactly Gₛ. As opposed to simulation, the cycle AI₁, DM₁, ..., AI₉, DM₉, AI₁ in G₅ is rightfully excluded from the match.

Graph pattern matching. Given a graph G and a simple pattern Q, matching via strong simulation is to find the set \( M_s(Q, G) \) of all matches \( G_s \) in G for Q, such that \( Q \preceq_D G_s \), for some node \( v \) in G, and \( G_s \) is the match graph with the maximum match relation \( S \) for \( Q \preceq_D G[v, d_q] \).

Matching via strong simulation is also in cubic time.

Theorem 4 [38]: For any data graph \( G = (V, E, f_A) \) and simple pattern \( Q = (V_q, E_q, f_q, f_s) \), \( M_s(Q, G) \) is computable in \( O(|V|(|V| + |E_q|) + |V| + |E|) \) time. □

It has been shown [38] that strong simulation preserves the following topological structures in graph pattern matching.

- Child relationship. If a node \( u \) in the pattern graph Q matches node \( v \) in the data graph G, then each child of \( u \) in Q must match a child of \( v \) in G.

- Parents. If a node \( u \) in Q matches node \( v \) in G, then each parent of \( u \) also matches a parent of \( v \).

- Connectivity. If Q is connected, then so are matches of Q in G. Here a graph is connected if for each pair of nodes in the graph, there exists an undirected path connecting them.

- Cycles. An undirected (resp. directed) cycle in Q must match an undirected (resp. directed) cycle in G.

- Bounded matches. For any match Gₛ of Q in G, its diameter is no larger than \( 2 \times d_q \). Moreover, there exist at most \( |V| \) matches in \( M_s(Q, G) \).
Table 1: Graph pattern matching

<table>
<thead>
<tr>
<th>matching</th>
<th>complexity</th>
<th>cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \leq_{\text{iso}} ) (subgraph isomorphism)</td>
<td>NP-complete</td>
<td>( O(</td>
</tr>
<tr>
<td>( \leq_{\text{sim}} ) (graph simulation)</td>
<td>quadratic time</td>
<td>( O(</td>
</tr>
<tr>
<td>( \leq_{\text{sim}}^\text{b} ) (bounded simulation)</td>
<td>cubic time</td>
<td>( O(</td>
</tr>
<tr>
<td>( \leq_{\text{sim}}^\text{r} ) (with regular patterns)</td>
<td>cubic time</td>
<td>( O(</td>
</tr>
<tr>
<td>( \leq_{\text{sim}}^S ) (strong simulation)</td>
<td>cubic time</td>
<td>( O(</td>
</tr>
</tbody>
</table>

Proposition 5 [38]: For any simple pattern \( Q \) and any data graph \( G \),
- if \( Q \leq_{\text{iso}} G \), then \( Q \leq_{\text{sim}}^S G \);
- if \( Q \leq_{\text{sim}}^S G \), then \( Q \leq_D G \); and
- if \( Q \leq_D G \), then \( Q \leq_{\text{sim}} G \).

It is shown [38] analytically and experimentally that graph pattern matching based on strong simulation further improves the quality of matches found by bounded simulation.

Proposition 5 [38]: The bounded cycle problem is coNP-hard even when pattern graphs contain a single cycle.

Bisimulation. One might be tempted to use graph bisimulation [42] rather than graph simulation in graph pattern matching. A graph \( G_s \) matches a pattern graph \( Q \) via bisimulation, denoted by \( Q \sim G_s \), if \( Q \leq_{\text{sim}} G_s \) with the maximum match relation \( S \) and \( G_s \leq_{\text{sim}} Q \) with the inverse \( S^- \) of \( S \) as its maximum match relation. Pattern matching via bisimulation is to find all subgraphs \( G_s \) of a graph \( G \) such that \( Q \sim G_s \). Graph bisimulation is a notion stronger than simulation but weaker than isomorphism.

In contrast, (1) graph simulation does not preserve the parent relationships, connectivity and undirected cycles as we have seen in Example 5, while it preserves the child relationships and directed cycles. (2) Dual simulation does not have bounded cycles, but it preserves the child relationships, parents, connectivity, directed and indirectly cycles. Graph pattern matching based on subgraph isomorphism preserves all the structural properties given above.

The connections between graph simulation, dual simulation, strong simulation and subgraph isomorphism are:

Below we present two negative results: extending strong simulation makes its computation from PTIME to NP-hard.

Bounded cycles. Given a pattern \( Q \) and a graph \( G \) such that \( Q \leq_D G \) with the maximum match relation \( S \), the bounded cycle problem is to decide whether the longest simple cycle in \( Q \) is no longer than the longest simple cycle in \( G \). Obviously bounded cycle is a desirable locality property that one would have wanted to further impose on strong simulation. Unfortunately, this additional condition would make graph pattern matching intractable.

Theorem 6 [38]: The bounded cycle problem is coNP-hard even when pattern graphs contain a single cycle.

However, pattern matching via bisimulation becomes intractable. Indeed, subgraph bisimulation is NP-hard [18], although graph bisimulation is solvable in PTIME [42]. In contrast, subgraph simulation is equivalent to graph simulation, i.e., checking whether there exists a subgraph \( G_s \) of \( G \) such that \( Q \leq_{\text{sim}} G_s \) is the same as checking whether \( Q \leq_{\text{sim}} G \).

Summary. Table 1 summarize various notions of graph pattern matching, along with the complexity of computing \( M(Q, G) \) and the cardinality of \( M(Q, G) \), for data graph \( G = (V, E, f_A, f_C) \) and pattern graph \( Q = (V_Q, E_Q, f_v, f_e) \).

4. Querying Large Social Graphs

Graph pattern matching is costly: NP-complete for subgraph isomorphism [29], quadratic-time for simulation [32], and cubic-time for bounded (strong) simulation [20]. As remarked earlier, real-life social graphs are typically large. These highlight the need for techniques to cope with large social graphs. In this section we present three such approaches.

4.1 Incremental Graph Pattern Matching

The first approach is incremental graph pattern matching, to accommodate the dynamic nature of social networks.

Incremental approach. Given a pattern graph \( Q \), a data graph \( G \), the matches \( M(Q, G) \) in \( G \) for \( Q \) and changes \( \Delta G \) to \( G \), incremental graph pattern matching is to find changes \( \Delta M \) to the matches such that \( M(Q, G \oplus \Delta G) = M(Q, G) \oplus \Delta M \), where (1) \( \Delta G \) consists of a set of edges to be inserted into or deleted from \( G \), and (2) operator \( \oplus \) applies changes \( \Delta S \) to \( S \), where \( S \) is a data graph \( G \) or the match result \( M \).

As opposed to batch algorithms that recompute the new output starting from scratch, an incremental matching algorithm aims to minimize unnecessary recomputation and improve response time. Indeed, in real life \( \Delta G \) is typically small. For example, only 5% to 10% of nodes are updated weekly [47]. When \( \Delta G \) is small, \( \Delta M \) is often small as well, and is much less costly to compute than \( M(Q, G \oplus \Delta G) \).

Example 7: Figure 5 depicts graph \( G_6 \) (excluding edges \( e_1, e_5 \)), a fraction of FriendFeed (a social networking service http://friendfeed.com/). Also shown in Fig. 5 are patterns \( Q_5 \) and \( Q_6 \) (all the edges in \( Q_6 \) are labeled with 1 and are omitted). With bounded simulation, \( M_6^B(Q_5, G_6) \) is the relation \( \{(\text{CTO}, \text{Ann}), (\text{DB}, \text{Pat}), (\text{DB}, \text{Dan}), (\text{Bio}, \text{Bill}), (\text{Bio}, \text{Mat})\} \). With subgraph isomorphism, the set \( M_6(Q_6, G_6) \) of
The changes $\Delta$ in the input and output, i.e., $G(Bio, Tom)$. This yields the new output $M_{iso}^{B}(Q_6, G_6 \oplus \Delta G) = M_{iso}^{B}(Q_6, G_6) \cup \Delta M_2$, where $\Delta M_2$ consists of the subgraph of $G_6 \oplus \Delta G$ induced by edges $e_2-e_5$.

When $\Delta G$ is small, the change $\Delta M_1$ (resp. $\Delta M_2$) to the old output $M_{iso}^{B}(Q_5, G_6)$ (resp. $M_{iso}(Q_6, G_6)$) is also small. When $G$ is large as commonly found in practice, it is less costly to find $\Delta M_1$ (resp. $\Delta M_2$) than to recompute the entire $M_{iso}^{B}(Q_5, G_6 \oplus \Delta G)$ (resp. $M_{iso}(Q_6, G_6 \oplus \Delta G)$).

Boundedness analyses. As pointed out in [53], the traditional complexity analysis for batch algorithms is no longer adequate for incremental algorithms. Indeed, it is not very informative to define the cost of an incremental algorithm as a function of the size of the input. Instead, one should analyze the algorithms in terms of $|CHANGED|$, the size of the changes in the input and output, i.e., the updating costs that are inherent to the incremental problem itself. Below we analyze graph pattern matching in terms of $|CHANGED|$, with (bounded) simulation and subgraph isomorphism.

To characterize $|CHANGED|$, we define the following.

Result graphs. The result graph $G_r = (V_r, E_r)$ of a pattern $Q$ in a graph $G$ is a graph representing the matches $M(Q, G)$.

- For bounded simulation,
  - $V_r$ consists of all the nodes $v$ in $G$ such that $(u, v) \in M_{iso}^{B}(Q, G)$, i.e., $v$ is a match of some pattern node $u$ in the maximum match;
  - for each edge $(u_1, u_2)$ in $Q$, there exists an edge $(v_1, v_2)$ in $E_r$ iff $(u_1, v_1)$ and $(u_2, v_2)$ are in $M_{iso}^{B}(Q, G)$ and moreover, there exists a nonempty path $\rho$ from $v_1$ to $v_2$ that satisfies the bound specified for $(u_1, u_2)$ in $Q$. That is, the edge $(v_1, v_2)$ indicates a path in $G$ to which the pattern edge $(u_1, u_2)$ is mapped.

Similarly result graphs are defined for simulation.

- For subgraph isomorphism, $G_r$ is the disjoint union of all the subgraphs of $G$ in $M_{iso}(Q, G)$.

Affected areas. The changes $\Delta M$ in the matches are characterized in terms of the affected area in the result graph. Let $G_r$ and $G_r'$ be the result graphs of $Q$ in $G$ and $G \oplus \Delta G$, respectively. Then the affected area $\text{AFF}$ of $G_r'$ by $\Delta G$ is defined to be the difference between $G_r$ and $G_r'$, i.e., the changes in both nodes and edges inflicted by $\Delta G$.

Bounded incremental algorithms. We define $|CHANGED| = |\Delta G| + |\text{AFF}|$, which indicates the size of changes in the data graph (input) and match results (output). An incremental algorithm is bounded if its complexity is determined only by $|CHANGED|$, independent of $G$. It is said to be optimal if it is in $O(|CHANGED|)$ time, which characterizes the amount of work that is absolutely necessary to perform for any incremental algorithm. An incremental matching problem is said to be bounded if there exists a bounded incremental algorithm for it, and is unbounded otherwise.

We say that an incremental graph pattern matching problem is semi-bounded if there exists an algorithm for it such that its cost is a PTIME function of $|CHANGED|$ and $|Q|$. That is, its cost depends only on the size of the changes and the size of pattern $Q$, independent of the size of data graph $G$. A semi-bounded incremental algorithm often suffices in practice since the size of pattern $Q$ is typically small.

Boundedness results. Consider a pattern $Q$, a data graph $G$ and changes $\Delta G$. We call $\Delta G$ a unit update if it consists of a single edge insertion or deletion, and a batch update if it is a sequence of edge insertions and deletions. We say that $Q$ is a path pattern if it consists of a single path.

The results below tell us that pattern matching based on (bounded) simulation is unbounded even for unit updates and path patterns. Nevertheless, it is semi-bounded [22].

Theorem 7 [22]: The incremental simulation problem is
1. unbounded even for unit updates and general patterns;
2. bounded for (a) single-edge deletions and general patterns, and for (b) single-edge insertions and DAG patterns, in optimal time $O(|AFF|)$; and
3. semi-bounded, in $O(|\Delta G|(|Q||AFF| + |AFF|^2))$ time for batch updates and general patterns.

Theorem 8 [22]: Incremental bounded simulation is
1. unbounded even for unit updates and path patterns;
2. semi-bounded, in $O(|\Delta G|(|Q||AFF| + |AFF|^2))$ time for batch updates and general patterns.

When it comes to subgraph isomorphism, incremental graph pattern matching is no longer semi-bounded. To see this, consider the following problem, denoted by Inclso, which is to determine, given $Q$, $G$, $M_{iso}(Q, G)$ and $\Delta G$, whether $Q \subseteq G \oplus \Delta G$, i.e., whether there exists a subgraph in the updated graph $G \oplus \Delta G$ that is isomorphic to $Q$. This problem is intractable, even when the data graph $G$ is fixed. Hence incremental pattern matching via subgraph isomorphism is not semi-bounded unless $P = NP$.

Theorem 9 [22]: For subgraph isomorphism,
1. the Inclso problem is NP-complete even when $G$ is a fixed graph; and
2. incremental matching is unbounded for unit updates, even when $Q$ is a path pattern and $G$ is a DAG.

The main boundedness results are summarized in Table 2. It is shown [22] that even for batch updates and general (possibly cyclic) patterns, incremental algorithms perform

<table>
<thead>
<tr>
<th>matching</th>
<th>bounded</th>
<th>semi-bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leq_{iso}$</td>
<td>$\times$ (NP-complete)</td>
<td></td>
</tr>
<tr>
<td>$\leq_{sim}$</td>
<td>$\checkmark$ (unit deletions)</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>$\leq_{r}$</td>
<td>$\times$ (unit updates)</td>
<td>$\checkmark$</td>
</tr>
</tbody>
</table>

Table 2: Incremental graph pattern matching
significantly better than their batch counterparts. Indeed, when 10% of data in graphs are changed, the improvement is from 40% to 50% for simulation, and from 30% to 40% for bounded simulation. When it comes to subgraph isomorphism, incremental matching outperforms its batch counterpart when data graphs are changed up to 20%.

Incremental algorithms have also been developed for simulation in [58] and for bisimulation in [56], which, however, did not consider whether incremental matching is bounded.

4.2 Query Preserving Graph Compression

The second approach is based on query preserving graph compression, relative to a class \( Q \) of queries of users’ choice.

**Compression scheme.** A query preserving graph compression for \( Q \) is a pair \( < R, P > \), where \( R(\cdot) \) is a compression function, and \( P(\cdot) \) is a post-processing function. For any graph \( G, G_c = R(G) \) is a graph computed from \( G \) by \( R(\cdot) \), referred to as the compressed graph of \( G \) via \( R(\cdot) \), such that

- \( |G_c| \leq |G| \), and
- for all queries \( Q \in Q \), \( Q(G) = P(Q(G_c)) \),

where \( Q(G) \) is the answer to \( Q \) in \( G \), and \( P(Q(G_c)) \) is the result of post-processing the answer \( Q(G_c) \) in \( G_c \).

As indicated in Fig. 6, (1) for any query \( Q \in Q \), the answer \( Q(G) \) to \( Q \) in \( G \) can be computed by evaluating the same query \( Q \) on the (smaller) compressed graph \( G_c \) of \( G \); (2) the compression is generic: any algorithm for evaluating queries in \( G \) can be directly used to compute \( Q(G_c) \); (3) in contrast to generic lossless compression schemes (e.g., [25]), we do not need to restore the original graph \( G \) from \( G_c \). That is, \( G \) only needs to retain the information necessary for answering queries in \( Q \). Moreover, the compressed graph \( G_c \) is not necessarily a subgraph of \( G \).

**Graph pattern preserving compression.** As an example, we next focus on query preserving graph compression for graph pattern queries based on bounded simulation.

**Theorem 10 [24]:** There exists a graph pattern preserving compression \( < R, P > \) for bounded simulation, in which for any graph \( G = (V, E, f_A) \), \( R(\cdot) \) is in \( O(|E| \log |V|) \) time, and \( P(\cdot) \) is in linear time in the size of the query answer. \( \square \)

One way to construct such a graph pattern preserving compression scheme is by using bisimulation relations [42] (see Section 3.3). One can verify that for any graph \( G = (V, E, f_A) \), (1) there is a unique maximum bisimulation relation \( R_b \subset V \times V \) on \( G \), and (2) \( R_b \) is an equivalence relation, i.e., it is reflexive, symmetric, and transitive.

We define the bisimulation equivalence relation of \( G \) to be the maximum bisimulation relation on \( G \), denoted by \( R_b \). We denote by \( [v]_{R_b} \) the equivalence class containing node \( v \). We say that nodes \( v \) and \( v' \) are bisimilar if \( (v, v') \in R_b \). Since for any nodes \( v \) and \( v' \) in \( [v]_{R_b}, f_A(v) = f_A(v') \), we simply call \( f_A(v) \) the label of \( [v]_{R_b} \).

Based on the equivalence relations, we define \( < R, P > \).

1. **Compression function** \( R(\cdot) \). For \( G = (V, E, f_A) \), its compressed graph \( R(G) = G_c = (V_c, E_c, f'_A) \), where
   - \( V_c = \{ [v]_{R_b} \mid v \in V \} \);
   - an edge \( ([v]_{R_b}, [w]_{R_b}) \) is in \( E_c \) if there exist nodes \( v' \in [v]_{R_b} \) and \( w' \in [w]_{R_b} \) such that \( (v', w') \in E \), and
   - for each \( [v]_{R_b} \in V_c \), \( f'_A([v]_{R_b}) = f_A(v) \).

   Intuitively, (a) for each node \( v \in V \), there exists a node \( [v]_{R_b} \in V_c \); abusing \( R(\cdot) \), we use \( [v]_{R_b} \) to denote \([v]_{R_b} \); (b) for each edge \( (v, w) \in E \), \([v]_{R_b}, [w]_{R_b} \in V_c \) is an edge in \( E_c \); and (c) each \([v]_{R_b} \) has the same attributes as \( v \).

2. **Post processing function** \( P(\cdot) \). Recall that \( Q(G) = M_{\text{sim}}^\beta(Q, G) \) is the maximum match in \( G \) for pattern \( Q \). We define \( P(\cdot) \) such that \( P(Q(G_c)) = Q(G) \) as follows. For each \( (u, [v]_{R_b}) \in Q(G_c) \) and each \( v' \in [v]_{R_b} \), \( (u, v') \in Q(G) \). Intuitively, if \([v]_{R_b} \) simulates \( u \) in \( G \), then so does each node \( v' \in [v]_{R_b} \) in \( G \). Hence, \( P(\cdot) \) expands \( Q(G_c) \) via the inverse of \( R(\cdot) \) (omitted from Fig. 6), in \( O(|Q(G)|) \) time, a cost necessary for any pattern matching algorithm.

**Example 8:** Graph \( G_7 \) in Fig. 7 is a fraction of a multi-agent recommendation network. Each node denotes a customer (C), a book server agent (BSA), a music shop agent (MSA), or a facilitator agent (FA) assisting customers to find BSA’s and MSAs. Each edge indicates a recommendation.

To locate potential buyers, a bookstore owner issues a pattern query \( Q_7 \) depicted in Fig. 7. One may verify that \( M_{\text{sim}}^\beta(Q_7, G_7) = \{ (X, X_i) \} \) for \( X \in \{ \text{BSA, FA}, C \} \) and \( i \in [1, 2] \). It is costly to compute the matches when \( G_7 \) is large.

Using the graph pattern preserving compression \( < R, P > \), one can get the compressed graph \( G_c \) of \( G_7 \) shown in Fig. 7, in which e.g., \( R(\text{FA}_1) = R(\text{FA}_2) = \text{FA}_r \), where \( \text{FA}_r \) is the equivalent class containing \( \text{FA}_1 \) and \( \text{FA}_2 \).

Observe that (1) \( Q_7 \) can be directly evaluated on \( G_c \); its result \( \{ (X, X_i) \} \) can be converted to \( M_{\text{sim}}^\beta(Q_7, G_7) \) by simply replacing \( X_i \) with the set of nodes represented by \( X_i \); (2) evaluating \( Q_7 \) in \( G_c \) is more efficient than in \( G \); and (3) for all pattern queries \( Q \) posed on \( G_7 \), not limited to \( Q_7 \), we can directly evaluate \( Q \) on the much smaller \( G_c \) instead. \( \square \)
Figure 8: Limitations of graph indexing structures

It is shown [24] that for matching with bounded simulation, the compression function \( R(\cdot) \) given above reduces the size of \( G \) by 57% in average, for a variety of real-life social graphs. The reduction by query preserving graph compression is more significant for reachability queries (to determine whether a node can reach another), about 95% in average.

Remarks. Query preserving graph compression differs from generic lossless graph compression and indexing as follows.

1. Lossless graph compression schemes (e.g., [5, 12, 25, 52, 54]) require to restore original graphs from compressed graphs even to answer simple queries, as observed in [5]. In contrast, query preserving compressed graphs can be directly queried without decompression, and moreover, achieves a better compression ratio [24] since they do not need to retain all the information of the original graphs.

2. For neighborhood queries [39] (to find nodes connected to a designated node in a graph), a notion of query-able compression has been studied. The idea is similar to query preserving compression. However, to answer those queries, the compact structures of [39] have to be (partially) decompressed [5], and query evaluation algorithms on original graphs cannot be directly applied to the compact structures.

3. A variety of indexing structures have been developed for graphs, notably 1-index [44], \( A(k) \)-index [35] and their generalization \( D(k) \)-index [51] based on (parameterized) graph bisimulation. These indices, however, do not preserve query results for graph pattern queries, as illustrated below.

Example 9: Consider graph \( G_8 \) and its index graph \( G_A \) of \( A(k) \)-index when \( k = 1 \), shown in Fig. 8. Although nodes \( A_1, A_2 \) and \( A_3 \) are not bisimilar, they all have only have \( B \) children; as such, they are 1-bisimilar [51], and are merged into a single node in \( G_A \). However, \( G_A \) cannot be directly queried by \( e.g., \) a pattern \( Q \) consisting of two query edges \( \{(B, C), (B, D)\} \), both with bound 1. Indeed, for \( Q, G_A \) returns all the \( B \) nodes in \( G \) as matches of query node \( B \) in \( Q \), but only \( B_1 \) and \( B_3 \) are the true matches in \( G_8 \).

Incremental graph compression. For each graph \( G \), we need to compute its compressed graph \( G_c \), once for all queries in \( Q \), and \( G_c \) is incrementally maintained in response to updates to \( G \). More specifically, given a graph \( G \), a compressed graph \( G_c = R(G) \) of \( G \), and batch updates \( \Delta G \) to \( G \), incremental graph compression for graph patterns is to compute changes \( \Delta G_c \) to \( G_c \) such that \( G_c \ominus \Delta G_c = R(G \oplus \Delta G) \).

Theorem 11 [24]: Incremental graph compression for graph patterns is unbounded for unit updates. However, it is in \( O(|A^I F|^2 + |G_c|) \) time, i.e., compressed graphs \( G_c \) can be incrementally maintained without decompressing \( G_c \).

Figure 9: Querying distributed social networks

4.3 Distributed Graph Pattern Matching

The third approach is distributed graph pattern matching, based on partial evaluation, described as follows.

Partial evaluation. Partial evaluation (a.k.a. program specialization) has been proved useful in a variety of areas including compiler generation, code optimization and dataflow evaluation (see [34] for a survey). Intuitively, given a function \( f(s, d) \) and part of its input \( s \), partial evaluation is to specialize \( f(s, d) \) with respect to the known input \( s \). That is, it conducts the part of \( f \)'s computation that depends only on \( s \), and generates a partial answer, i.e., a residual function \( f' \) that depends on the as yet unavailable input \( d \).

This idea can be naturally applied to distributed graph pattern matching. Consider a pattern graph posed on a graph \( G \) that is partitioned into fragments \( (F_1, \ldots, F_n) \) of manageable sizes, where \( F_i \) is stored in site \( S_i \). To compute \( M(Q, G) \), each site \( S_i \) can find the partial answer to pattern query \( Q \) in fragment \( F_i \) in parallel, by taking \( F_i \) as the known input \( s \) while treating the fragments in the other sites as yet unavailable input \( d \). These partial answers (matches) are collected and combined by a coordinator site, to derive \( M(Q, G) \), the matches for \( Q \) in the entire \( G \).

Example 10: Figure 9 depicts a fraction \( G_9 \) of a recommendation network, where each node denotes a person with name and job titles (e.g., database researcher (DB), human resource (HR)), and each directed edge indicates a recommendation. The graph \( G_9 \) is geo-distributed to three data centers \( DC_1, DC_2 \) and \( DC_3 \), each storing a fragment of \( G_9 \).

Consider a pattern query \( Q_9 \) given in Fig. 9 posed on \( DC_1 \). It is to find whether there exists a chain of recommendations from a CTO Ann to her finance analyst (FA) Mark, through either a list of DB people or a list of HR people. Observe that such a path exists: (Ann, CTO) \( \rightarrow \) (Walt, HR) \( \rightarrow \) (Mat, HR) \( \rightarrow \) (Fred, HR) \( \rightarrow \) (Emmy, HR) \( \rightarrow \) (Ross, HR) \( \rightarrow \) (Mark, FA). However, it is nontrivial to verify this in the distributed setting. A naive method is to first ship data from \( DC_1, DC_2 \) and \( DC_3 \) to a single site, and then evaluate the query using an algorithm developed for centralized data (i.e., graphs stored in a single site). This is infeasible because its data shipment may be prohibitively expensive and worse still, may not even be allowed for data privacy. Another way is to use a distributed graph traversal algorithm, by sending messages between different sites. This, however, requires messages to be sent along \( DC_1 \rightarrow DC_2 \rightarrow DC_1 \rightarrow DC_2 \rightarrow DC_3 \rightarrow DC_1 \), incurring unbounded number of visits to each site, excessive communication cost, and unnecessary delay in response.

We can do better by using partial evaluation. We send
the query $Q_a$ to DC1, DC2 and DC3, as is. We compute
the partial answers to $Q_a$ at each site, in parallel, by taking
the fragment residing in the site as known input and intro-
ducing Boolean variables to indicate unknown input (i.e.,
fragments in the other sites). The partial answers are a set
of Boolean equations defined with Boolean variables, one
associated with each node that has an edge to a fragment
stored at another site. These equations indicate (1) at DC1,
from Ann there exist an HR path to Walt and a BB path to
Bill, while there are edges from Walt to Mat. Bill to Pat
and from Fred to Emmy; (2) at DC2, from Emmy there exist an
HR paths to Mat and an edge to Ross, and there is an edge
from Mat to Fred; and (3) at DC3, there exists an HR path
from Ross to Mark. These partial answers are collected by
a coordinator site (DC1), which solves the system of (recur-
sively defined) Boolean equations, to find the truth values
of those Boolean variables. It yields answer true to query
$Q_b$, i.e., there exists an HR path from Ann to Mark.

This method guarantees the following: (1) each site is
visited once; (2) the total amount of data shipped (network
traffic) is independent of the size of $G_a$; and (3) the com-
putation is conducted in parallel at each site, without waiting
for the outcome or messages from any other site.

Distributed matching of simple patterns. Partial eval-
uation has been studied for evaluating XPath queries [3, 
9, 15] on distributed XML documents, SPARQL queries on
distributed RDF graphs [26], and simple pattern queries on
distributed social graphs [23]. Below we present preliminary
results on matching in distributed social graphs [23], based
on simple Boolean patterns; distributed pattern matching
based on (bounded) simulation remains to be studied.

Consider Boolean patterns of the form $Q(s, t, U)$, where
$s, t$ are nodes in a graph $G$, and $U$ is a regular expres-
sion:

$$U ::= a | \overline{U} | U \cup U | U^*,$$

where $a$ is a label in an alphabet $\Sigma$, $\overline{U}$, $U \cup U$ and $U^*$
denote alternation, concatenation and the Kleene closure,
respectively. We say that a path $\rho$ satisfies $U$ if the label
of $\rho$ is a string in the regular language defined by $U$. The
Boolean query is to determine whether there exists a path
$\rho$ from $s$ to $t$ such that $\rho$ satisfies $U$, i.e., the reachability.

On a graph $G$ partitioned into $\mathcal{F} = \{F_i | i \in [1, n]\}$ such
that $F_i$ resides at site $S_i$, such a query $Q(s, t, U)$ can be
evaluated with the following performance guarantees.

Theorem 12 [23]: On a fragmentation $\mathcal{F}$ of graph $G$, 
Boolean reachability queries $Q(s, t, U)$ can be evaluated

- by visiting each site once,
- in $O(|F_m||U|^2 + |U|^2|V_j|^2)$ time, and
- with the communication cost in $O(|U|^2|V_j|^2)$,

where $F_m$ is the largest fragment in $\mathcal{F}$ and $V_j$ is the set of
nodes in $G$ that have edges across different fragments.

That is, (1) each site is visited a fixed number of times; (2)
the response time is dominated by the largest fragment in
$\mathcal{F}$, independent of the size $|G|$ of $G$; (3) the total amount of
data shipped is determined by the size of the query and how
$G$ is fragmented, again independent of $|G|$, and (4) the per-
formance guarantees remain intact no matter how $G$ is frag-
mented and distributed. Distributed evaluation of XPath
queries possesses similar performance guarantees [9, 15].

More specifically, this is conducted as follows [23].

1. We first construct an automaton $G_q(U)$ representing $Q(s, t, U)$, and post the same $G_q$ to each fragment in $\mathcal{F}$.

2. Upon receiving $G_q(U)$, each site computes a partial an-
swer of $Q$ using $G_q$, in parallel. The partial answer at each
fragment $F_i$ is a set of Boolean equations composed of dis-
juncts, each indicating whether a node in $V_j$ matches a state
of $G_q$. The equations are sent to a coordinator site $S_c$.

3. The site $S_c$ collects the equations from each site. It
then solves the system of the Boolean equations and finds
the final answer to $Q(s, t, U)$ in the entire graph $G$.

Example 11: Consider pattern $Q_b$ and graph $G_9$ given in
Fig. 9. The partial answer at fragment DC1 includes:

$$Y(\text{Ann, Mark}) = X(\text{Pat, DB}) \lor X(\text{Mat, HR}),$$

$$X(\text{Fred, HR}) = X(\text{Emmy, HR}).$$

Here $Y(\text{Ann, Mark})$ is a Boolean variable indicating whether
there exists a path from Ann to Mark that satisfies the regular
expression of $Q_b$, and variable $X(\text{Pat, DB})$ indicates whether
or not there exists a DB path from Pat to Mark; similarly
for variables $X(\text{Mat, HR}), X(\text{Emmy, HR})$ and $X(\text{Fred, HR})$.

These equations can be constructed using local information
at DC1, while the site DC1 also keeps track of outgoing edges
from nodes in DC1 to another fragment, such as (Bill, Pat),
(Walt, Mat) and (Fred, Emmy). Note that the truth value of
$Y(\text{Ann, Mark})$ is defined as the disjunction of two Boolean
variables $X(\text{Pat, DB})$ and $X(\text{Emmy, HR})$.

Similarly, the equations at DC2 and DC3 include:

$$X(\text{Emmy, HR}) = X(\text{Ross, HR}) \lor X(\text{Fred, HR}), \quad /* \text{DC2 } */$$

$$X(\text{Mat, HR}) = X(\text{Fred, HR}); \quad /* \text{DC2 } */$$

$$X(\text{Fred, HR}) = \text{true.} \quad /* \text{DC3 } */$$

Note that the truth value of $X(\text{Ross, HR})$ is determined true
locally at DC3 since there exists an HR path from Ross to
Mark. Similarly, $X(\text{Pat, DB})$ is evaluated to be false locally.

These equations are constructed in parallel at each site.

The equations are collected by site DC1, and form a sys-
tem of Boolean equations. Solving these equations yields
$Y(\text{Ann, Mark}) = \text{true}$, the answer to $Q_b$ in $G_9$.

It has also been shown [23] that partial evaluation can also
be readily implemented in the MapReduce framework [17].

5. Open Research Issues

We have presented an informal overview of recent work on
graph pattern matching for social network analysis, empha-
sizing (a) revisions of traditional notions of graph pattern
matching to improve the quality of matches, and (b) tech-
techniques to cope with the sheer size of real-life social graphs.
The study has raised as many questions as it has answered.

1. In practice one may want to query both data and topo-
logy in social network analysis, e.g., paths and subgraphs sat-
sifying constraints defined on both their topological struc-
tures and data contents. Bounded simulation (with edge
relationships specified by regular expressions) is just a first step towards developing a practical query language for social network analysis. There is much more to be done, to identify primitives that are necessarily supported by such a language, characterize the expressive power of the language, and establish the complexity of fundamental problems associated with queries in such a language. There has been initial work in this direction, e.g., [4, 37].

(2) There is still room to improve the lower bounds of graph pattern matching based on, e.g., bounded and strong simulation. These suggest a full treatment of graph indexing, summarization and compression methods, to develop more efficient batch and incremental matching algorithms. Moreover, query preserving compression remains to be studied for matching based on subgraph isomorphism.

(3) The study of distributed graph pattern matching is still in its infancy. Distributed matching algorithms based on, e.g., (bounded) simulation, are not yet in place. It remains to be investigated whether graph pattern matching still retains the same performance guarantees on the number of visits, data shipment and response time as given above when general graph patterns are adopted. Moreover, it is nontrivial to partition graphs such that distributed graph pattern matching can be conducted with minimum network traffic and response time, a possibly intractable problem [49].

(4) To cope with large social networks, one may want to use inexact or approximate matching algorithms [16, 28]. These algorithms should be extended to support graph pattern matching defined in terms of, e.g., bounded simulation. In addition, such algorithms are needed both for graphs stored at a single site and for graphs partitioned and distributed.

(5) Another approach to querying large social networks is based on query rewriting using views, to capitalize on previous computation (cached views). This highlights the need for studying graph pattern query rewriting using views, a nontrivial extension to prior work on path query rewriting [10] and relational query rewriting using views [31, 36].

(6) It has been recognized that social data analysis should be incorporated into search engines. The principal goal of search engines has been to help people find what they are looking for. Social networks produce an immense amount of data about what people like and what they want to share with their friends. It is hence natural to improve searches by capitalizing on social data. Google, Bing and newly launched search engines such as Blekko and DuckDuckGo are already exploring this. To approach this, one may want to integrate graph pattern matching and keyword search [61, 63]. It should be remarked that graph pattern matching is a “stronger form” of keyword search, by specifying keywords with search conditions in patterns, and imposing topological constraints on how keywords are related. It remains to investigate what topological constraints are needed in keyword search (e.g., [41]), and how one can extract top-k matches from the result returned in searches, when matching is defined in terms of, e.g., bounded or strong simulation.

Acknowledgments. Most of the results presented here are based on joint work with Shuai Ma, Yinghui Wu and Xin Wang [19, 20, 21, 22, 23, 24, 38]. In particular, the idea of strong simulation came from Shuai. This work is supported in part by a RSE-NSFC Joint Project grant, an IBM scalable data analytics for a smarter planet innovation award, NSFC 61133002, and the 973 Program 2012CB316200 of China.

6. References

regular expressions to graph reachability and pattern queries. In ICDE, 2011.


