Incremental Graph Pattern Matching

WENFEI FAN, University of Edinburgh and Beihang University
XIN WANG, University of Edinburgh
YINGHUI WU, University of Edinburgh and University of California Santa Barbara

Graph pattern matching is commonly used in a variety of emerging applications such as social network analysis. These applications highlight the need for studying the following two issues. First, graph pattern matching is traditionally defined in terms of subgraph isomorphism or graph simulation. These notions, however, often impose too strong a topological constraint on graphs to identify meaningful matches. Second, in practice a graph is typically large, and is frequently updated with small changes. It is often prohibitively expensive to recompute matches starting from scratch via batch algorithms when the graph is updated.

This article studies these two issues. (1) We propose to define graph pattern matching based on a notion of *bounded simulation*, which extends graph simulation by specifying the connectivity of nodes in a graph within a predefined number of hops. We show that bounded simulation is able to find sensible matches that the traditional matching notions fail to catch. We also show that matching via bounded simulation is in cubic time, by giving such an algorithm. (2) We provide an account of results on incremental graph pattern matching, for matching defined with graph simulation, bounded simulation, and subgraph isomorphism. We show that the incremental matching problem is *unbounded*, that is, its cost is not determined alone by the size of *the changes* in the input and output, for all these matching notions. Nonetheless, when matching is defined in terms of simulation or bounded simulation, incremental matching is *semibounded*, that is, its worst-time complexity is bounded by a polynomial in the size of the changes in the input, output, and auxiliary information that is necessarily maintained to reuse previous computation, and the size of graph patterns. We also develop incremental matching algorithms for graph simulation and bounded simulation, by minimizing unnecessary recomputation. In contrast, matching based on subgraph isomorphism is neither bounded nor semibounded. (3) We experimentally verify the effectiveness and efficiency of these algorithms, and show that: (a) the revised notion of graph pattern matching allows us to identify communities commonly found in real-life networks, and (b) the incremental algorithms substantially outperform their batch counterparts in response to small changes. These suggest a promising framework for real-life graph pattern matching.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Graph pattern matching, graph simulation, subgraph isomorphism, incremental pattern matching

ACM Reference Format:
Fan, W., Wang, X., and Wu, Y. 2013. Incremental graph pattern matching. ACM Trans. Datab. Syst. 38, 3, Article 18 (August 2013), 47 pages.
DOI: http://dx.doi.org/10.1145/2489791

W. Fan and X. Wang are supported in part by the 973 Programs 2012CB316200 and 2014CB340302, Guangdong Innovative Research Team Program 2011D005 and the Shenzhen Peacock Program 1105100030834361 of China, as well as EPSRC EP/J015377/1, UK.

X. Wang is currently also affiliated with Southwest Jiaotong University.
Authors' addresses: W. Fan and X. Wang, School of Informatics, Laboratory for Foundations of Computer Science, Informatics Forum, 10 Crichton Street, Edinburgh EH8 9AB, Scotland, UK; Y. Wu (corresponding author), Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA 93106-5110; email: yinghuiwu.ed@gmail.com.

DOI: http://dx.doi.org/10.1145/2489791

1. INTRODUCTION

Graph pattern matching is to find all matches in a data graph G for a given pattern graph P. It has been increasingly used in computer vision, knowledge discovery, biology, cheminformatics, dynamic network traffic, intelligence analysis, and more recently, social network analysis, among other things (e.g., [Bruno et al. 2002; Chen et al. 2005; Cheng et al. 2008a; Tong et al. 2007; Zou et al. 2009]).

Graph pattern matching is typically defined in terms of the following.

—*Subgraph isomorphism*. This is to find all subgraphs of G that are isomorphic to P (see Gallagher [2006] for a survey); that is, a match of P is a subgraph G' of G such that there exists a *bijective function* f from the nodes of P to the nodes of G', and: (a) for each node v in G', v and $f(v)$ have the same label, and (b) there exists an edge from v to v' in P if and only if $(f(v), f(v'))$ is an edge in G'.

—*Graph simulation* [Milner 1989]. This is to find a binary *relation* $S \subseteq V_P \times V$, where V_P and V are the set of nodes in P and G, respectively, such that: (a) for each node u in V_P, there exists a node v in V such that $(u, v) \in S$, and u and v have the same label, and moreover, (b) for each $(u, v) \in S$ and each edge (u, u') in P, there is an edge (v, v') in G such that $(u', v') \in S$ [Brynielsson et al. 2010; Cho et al. 2000; Nardo et al. 2009].

Nevertheless, these traditional notions of graph pattern matching are often too restrictive to identify patterns in emerging applications such as social network analysis.

Example 1.1. Consider the structure of a drug trafficking organization [Natarajan 2000], depicted as a pattern graph P_0 in Figure 1. 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 (denoted by edge label 1). A drug ring G_0 is also shown in Figure 1 in which $\mathsf{A}_1, \ldots, \mathsf{A}_m$ are AMs, while A_m is both an AM and the secretary (S).

One wants to identify all suspects involved in the drug ring [Natarajan 2000], by finding matches for P_0 in G_0. However, graph pattern matching via subgraph isomorphism would not be able to find these, for the following reasons.

(1) Nodes AM and S in P_0 should be mapped to the same node A_m in G_0, which is not allowed by a bijection.
(2) The node AM in P_0 corresponds to multiple nodes $\mathsf{A}_1, \ldots, \mathsf{A}_m$ in G_0. This is not allowed by a function from the nodes of P_0 to the nodes of G_0. This suggests that we should use *relations* instead of *functions* when characterizing communities (matches).
(3) The edge from AM to FW in P_0 indicates that an AM supervises FWs within 3 hops. It should be mapped to a *path* of a bounded length in G_0 rather than to an *edge*.

For the same reason as (3) given before, graph pattern matching defined in terms of graph simulation is not capable of identifying the drug ring G_0 as a match of P_0 either.

As suggested by Example 1.1, we need to revise the traditional notions of graph pattern matching to efficiently identify sensible matches in emerging applications. In particular, in a variety of applications one wants to inspect the connectivity of a pair of nodes via a path of an arbitrary length [Cohen et al. 2003; Jin et al. 2009; Wang et al. 2006] or within a bound on the number of hops (e.g., 3, 1 in P_0) [Chan and Lim 2007; Cohen et al. 2003; Zou et al. 2009]. The need for this is also evident in, for example, activity planning [Li and Shan 2012] and team formation [San Martın et al. 2011], where close connections (e.g., collaboration, invitation) are specified as edges in the query patterns but need to be mapped to paths with bounded lengths in a data graph.

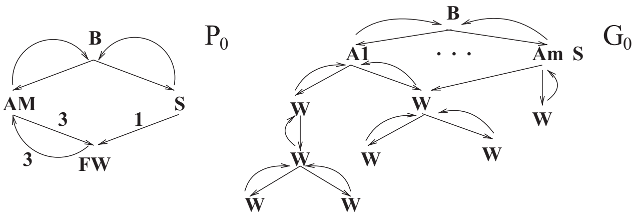


Fig. 1. Drug trafficking: Pattern and data graph.

Edge-to-edge mappings of subgraph isomorphism and graph simulation impose too strict a topological constraint to specify such connectivity in a data graph.

Another central issue of graph pattern matching in emerging applications concerns how to efficiently compute matches when graphs are updated. In practice a graph G is typically *large*. For instance, Facebook has more than 1.06 billion users (nodes) with 150 billion links (edges) [Smith 2013]. Moreover, it is frequently updated, for example, by insertions and deletions of edges in social networks (e.g., friendship, collaboration, citation) [Garg et al. 2009], Web graphs [Ntoulas et al. 2004], and traffic networks [Chen et al. 2009], when a user adjusts her friend cycle or edits her profile (see Section 4 for a real-life example). It is often cost prohibitive to recompute matches of a pattern P starting from scratch when G is updated. Indeed, it is NP-complete to determine whether G matches P via subgraph isomorphism (refer to Garey and Johnson [1979]), and it takes quadratic time to find the matches of P in G via simulation [Henzinger et al. 1995].

With the dynamic nature of social networks and Web graphs comes the need for *incremental matching algorithms*. As opposed to *batch algorithms* that recompute matches starting from scratch, an incremental matching algorithm aims to find changes ΔM to the matches in response to updates ΔG to G, by minimizing unnecessary recomputation. It is known that while real-life graphs are constantly updated, the changes are typically small. For example, only 5% to 10% of nodes are updated weekly in a Web graph [Ntoulas et al. 2004]). When ΔG is small, ΔM is often small as well, and is much less costly to compute than to recompute the entire set of matches. In other words, this suggests that we compute matches *once* on the entire graph via a batch matching algorithm, and then incrementally identify new matches in response to ΔG, without paying the price of the high complexity of graph pattern matching.

Contributions. This article investigates these two issues. (1) We propose a revision of the traditional notion of graph pattern matching, to find sensible matches in emerging applications. (2) We give a full treatment of incremental graph pattern matching, from the complexity bounds to effective algorithms, for matching defined in terms of graph simulation, bounded simulation, and subgraph isomorphism.

(1) We propose a notion of *bounded simulation*, an extension of simulation [Milner 1989]. We define pattern graphs in which a node specifies a search condition on the data content, and an edge is labeled with either a constant k or a $*$, denoting the connectivity of a pair of nodes in a data graph that is bounded within k hops or unbounded, respectively. In contrast to its traditional counterparts, matching based on bounded simulation is to find a maximum bounded simulation relation rather than functions (subgraph isomorphism), and it maps edges in a pattern to paths with various bounds in a graph, instead of edge-to-edge mappings (subgraph isomorphism and simulation).

(2) We show that graph pattern matching based on bounded simulation can be performed in *cubic time*, as opposed to the NP-completeness of the traditional notion via subgraph isomorphism. We provide an $O(|V||E| + |E_p||V|^2 + |V_p||V|)$-time algorithm for computing exact matches, for a pattern graph $P = (V_p, E_p)$ and a data

graph $G = (V, E)$. This is comparable to the complexity of graph simulation, which is in $O((|V| + |V_p|)(|E| + |E_p|))$ time [Henzinger et al. 1995]. Indeed, in practice pattern P is typically much smaller than data graph G, and $|E|$ could be $|V|^2$ in the worst case.

(3) To cope with the dynamic nature of data graphs in emerging applications, we give a full treatment of the *incremental graph pattern matching* problem. For changes to graphs, we consider *unit update*, that is, a single-edge deletion or insertion, and *batch update*, that is, a list of edge deletions and insertions mixed together.

We provide the first boundedness analysis of incremental matching, for matching defined in terms of simulation, bounded simulation and subgraph isomorphism. As argued in Ramalingam and Reps [1996b], the traditional complexity analysis for batch algorithms is no longer adequate for incremental algorithms. Instead, one should analyze the algorithms in terms of |CHANGED|, which indicates the size of the changes in the input and output. It 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 |CHANGED|, *instead of* the size of input. An incremental problem is said to be *bounded* if there exists a bounded incremental algorithm for it, and is *unbounded* otherwise.

Our first boundedness result is negative: we show that incremental pattern matching is *unbounded*, no matter whether it is defined in terms of simulation, bounded simulation, or subgraph isomorphism, even for unit updates and restricted patterns.

This motivates us to propose a notion of semiboundedness. We use |AFF| to denote the size of changes in the result and in auxiliary structures that are *necessarily maintained* for any incremental algorithms for the problem. We say that an incremental algorithm is *semibounded* if its worst-time complexity is bounded by a polynomial in $|\Delta G|$, |AFF| and the size $|P|$ of pattern. An incremental problem is *semibounded* if there exists a semibounded incremental algorithm for it. A semibounded algorithm is said to be *optimal* if it is in $O(|\Delta G| + |P| + |\mathsf{AFF}|)$ time, indicating the amount of work necessary to perform for any incremental algorithm for the problem.

We argue that the analysis of incremental algorithms should take |AFF| into account. The key idea of incremental algorithms is to speed up the process by maximally reusing previous computation. To keep track of the (partial) results of previous computation, auxiliary structures have to be maintained. Note that |CHANGED| may not include |AFF| when we consider, for example, partial matches found by previous process. That is, |CHANGED| may be too strict to capture the amount of necessary computation that an incremental algorithm has to conduct, and hence, an incremental algorithm is bounded only in ideal and special cases. While Ramalingam and Reps [1996b] actually advocated |AFF|, they stopped short of classifying semibounded incremental algorithms.

Based on this, the following boundedness results are established.

—For matching with graph simulation [Milner 1989], we show the following: (a) the incremental matching problem is semibounded, and has optimal incremental algorithms for: (a) (i) unit deletions and general patterns, and for (ii) unit insertions and DAG (directed acyclic graph) patterns, and (b) the problem is semibounded for batch updates and general patterns, by providing an effective incremental algorithm.

—For bounded simulation, we show that the problem is semibounded for batch updates and general patterns, by developing an efficient incremental matching algorithm. The algorithm employs landmark vectors and distance vectors, an extension of landmarks [Potamias et al. 2009], to help us find shortest paths in a data graph.

Moreover, we investigate incremental maintenance of the vectors. We show that: (a) it is bounded to maintain landmark vectors and (b) semibounded to maintain landmark vectors and distance vectors. We provide a semibounded incremental algorithm that updates the landmarks and necessary distance information.

—For subgraph isomorphism, we show that incremental matching is more intricate: it is (i) *unbounded* even for trees as patterns and forests as graphs, and (ii) NP-complete even for *fixed* data graphs and hence is not semibounded unless P = NP.

(4) Using two real-life datasets as well as synthetic data, we experimentally verify the effectiveness and scalability of our matching and incremental algorithms.

—We find that graph pattern matching via bounded simulation is able to accurately identify far more communities in, for instance, YouTube, than its traditional counterparts. We show that the matching algorithm is quite efficient and scales well with the sizes of data graphs and pattern graphs.

—We find that for batch updates and general (possibly cyclic) patterns, our incremental algorithms perform significantly better than their batch counterparts, when data graphs are changed up to 30% for graph simulation and 10% for bounded simulation. In addition, our algorithms consistently outperform previous incremental algorithms for (bounded) simulation [Shukla et al. 1997].

We contend that bounded simulation provides a useful alternative for graph pattern matching and allows us to catch sensible matches in, for example, social networks, while retaining low PTIME. In addition, the complexity (boundedness) results of the incremental matching problem disclose the inherent difficulty of the problem. The incremental algorithms yield a promising method for graph pattern matching in evolving real-life networks, to cope with the dynamic nature and the sheer size of those networks.

This article is an extension of earlier work [Fan et al. 2010a; Fan et al. 2011a], by including the following new contributions not found in Fan et al. [2010a, 2011a]: (1) semiboundedness results, extending previous complexity study in Fan et al. [2011a] (Sections 4, 5, and 6); (2) optimization techniques for incremental matching via simulation (Section 5); (3) new incremental algorithms and boundedness analysis of landmark maintenance (Section 6); (4) the complexity analysis of incremental subgraph isomorphism for fixed data graphs, and for tree patterns and data graphs (Section 7); and (5) enhanced experiments that verify the aforesaid techniques (Section 8). This article also includes detailed proofs for the results, which were not presented in Fan et al. [2010a; 2011a]. The complexity (boundedness) results and main algorithms of the article are summarized in the following table in which new contributions are marked with $*$.

Problem	Complexity	Algorithm
Matching	$O(\lvert V\rvert\lvert E\rvert + \lvert E_p\rvert\lvert V\rvert^2 + \lvert V_p\rvert\lvert V\rvert)$ time (Thm. 3.1)	Match (Section 3)
Incremental simulation	○ unbounded (unit updates, general patterns) ○ semi-bounded, optimal (unit deletion) ○ semi-bounded, optimal (unit insertion, DAG P) ○ semi-bounded (batch updates, general patterns) $*$ (Thm. 5.1)	○ IncMatch$^{+}_{\mathrm{dag}}$ (unit insertion, DAG P) ○ IncMatch^{-} (unit deletion) ○ IncMatch^{+} (unit insertion) ○ IncMatch (batch updates) $*$ (Section 5)
Incremental bounded simulation	○ unbounded (unit updates, path patterns) ○ semi-bounded (batch updates. general P) $*$ (Thm. 6.1)	○ IncBMatch^{+} (unit insertion) ○ IncBMatch (batch updates) (Section 6.3)
Incremental landmark maintenance	○ bounded for landmark vectors (Prop. 6.2) $*$ ○ semi-bounded for landmark and distance vectors (Prop. 6.3) $*$	○ InsLM (unit insertion) ○ DelLM (unit deletion) $*$ ○ IncLM (batch updates) $*$ (Section 6.4)
Incremental subgraph isomorphism	○ NP-complete for fixed data graphs $*$ ○ unbounded for unit updates $*$ (Thm. 7.1)	

Related Work. We next categorize the related work as follows.

Subgraph isomorphism. Graph pattern matching is typically defined in terms of subgraph isomorphism [Bruno et al. 2002; Chen et al. 2005; Cheng et al. 2008a; Tong et al. 2007; Zou et al. 2009]. In light of the intractability of the problem, approximate solutions have been studied to find inexact matches (see Gallagher [2006] and Shasha et al. [2002] for surveys). In contrast, this work revises graph pattern matching by introducing bounded simulation, to capture patterns commonly found in real-life networks, in polynomial time. We will further elaborate their differences in Section 2.

Graph simulation. Graph simulation has been used in, for example, process calculus [Nardo et al. 2009], social position detection [Brynielsson et al. 2010], and Web site classification [Cho et al. 2000]. An algorithm for computing graph simulation on a single graph was proposed in Henzinger et al. [1995]. Our matching algorithm (Section 3) is a nontrivial extension of Henzinger et al. [1995] to find matches in a graph for a pattern; it employs shortest path computation to handle bounded connectivity, among others.

Extensions of simulation and isomorphism. Several extensions of graph simulation and subgraph isomorphism have been studied for pattern matching. Among these are Nardo et al. [2009], Fan and Bohannon [2008], Fan et al. [2010b], and Zou et al. [2009]. A notion of weak similarity was proposed in Nardo et al. [2009], which extends simulation by mapping an edge to an unbounded path. It focuses on subgraph similarity, an NP-complete problem. Extensions of subgraph isomorphism were studied in Fan and Bohannon [2008] and Fan et al. [2010b] for XML schema mapping and for Web site matching, which also allow edge-to-path mappings, but are still NP-complete. None of these supports bounded connectivity or search conditions. Recently, bounded connectivity in graph patterns was considered in Zou et al. [2009]. It differs from this work in the following. (a) Patterns of Zou et al. [2009] impose the same bound on all edges. In contrast, we study patterns in which edges may carry various bounds or are unbounded at all, and moreover, nodes specify search conditions on data contents. (b) Matching in Zou et al. [2009] is based on an extension of subgraph isomorphism, which remains NP-complete, whereas our matching via bounded simulation is a cubic-time problem. (c) To find matches, Zou et al. [2009] explore joins and pruning, which are very different from our methods. (d) Zou et al. [2009] do not study incremental algorithms.

Distance, reachability and query languages. There has also been a host of work on reachability queries (e.g., [Cohen et al. 2003; Jin et al. 2009; Wang et al. 2006]), to decide whether there exists a path from a node to another in a graph, as well as work on distance queries (e.g., [Chan and Lim 2007; Cohen et al. 2003]), to compute the distance between a pair of nodes. In contrast, we study pattern graphs in which each edge denotes the connectivity of a pair of nodes and moreover, possibly carries a bound on the length of the paths. Query languages have also been developed for graphs (e.g., [He and Singh 2009; Ronen and Shmueli 2009]), which differ from this work in that the focus is on language constructs for expressing graph queries, rather than on the complexity and algorithms for (incrementally) finding matches in a data graph.

Incremental graph pattern matching. Incremental algorithms have been developed for various applications (see Ramalingam and Reps [1993] for a survey). As observed in Ramalingam and Reps [1996b], the complexity of an incremental algorithm is more accurately characterized in terms of the size of the area affected by updates, rather than the size of the entire input. We extend this complexity measure to incremental graph pattern matching, and propose the notion of semiboundedness. Incremental algorithms for the shortest path problem were provided in Ramalingam and Reps [1996a, 1996b]. We develop incremental algorithms for computing matches (Section 6), which make use of a procedure from Ramalingam and Reps [1996a, 1996b]. Incremental algorithms have also been developed for bisimulation [Saha 2007; Yi et al. 2004]. In contrast to our incremental methods: (a) those algorithms are based on an equivalence relation on a single graph, which does not exist for bounded simulation, and (b) they are unbounded, that is, they may conduct computation outside of the affected areas.

Inexact algorithms have been studied for incremental subgraph search [Wang and Chen 2009; Stotz et al. 2009]. An algorithm is developed in Wang and Chen [2009] to approximately determine whether a pattern is contained in graphs in a graph stream, based on an index of exponential size. An exponential-time incremental algorithm for inexact subgraph isomorphism is given in Stotz et al. [2009], which is claimed to be bounded. In contrast, we show that the incremental matching problem for subgraph isomorphism is unbounded even for unit updates and path patterns (Section 7).

About incremental simulation algorithms we are only aware of Saha [2007] and Shukla et al. [1997], mostly for verification and model checking. Incremental bisimulation is studied in Saha [2007]. In contrast to our work, it considers bisimulation on a single graph, which is quite different from incremental simulation across two graphs (a pattern and a data graph). Simulation is investigated in Shukla et al. [1997] based on HORN-SAT, which supports incremental updates on a single graph. However: (a) it does not consider whether the incremental simulation problem is bounded, and (b) it requires to update reflections and to construct an instance of size $O(|E|^2)$, where $|E|$ is the number of edges of the graph. In contrast, our algorithms for incremental simulation do not have to maintain large auxiliary structures (Section 6).

Incremental view maintenance has been studied for semistructured data modeled as a graph (e.g., [Abiteboul et al. 1998; Zhuge and Garcia-Molina 1998]). Assuming that data has a tree structure, Zhuge and Garcia-Molina [1998] maintain only the nodes of views. Incremental maintenance of graph views is studied in Abiteboul et al. [1998], which generates update statements in Lorel in response to updates. There has also been a large body of work on relational view maintenance (see Gupta and Mumick [2000] for a collection of readings). Unfortunately, as pointed out by Saha [2007], the incremental matching problem is nonmonotonic in nature for simulation (similarly for bounded simulation and subgraph isomorphism), and hence cannot be reduced to incremental evaluation of logic programs with stratified negation. As a result, these techniques cannot be directly used in incremental graph pattern matching.

Landmark vectors. Our incremental algorithms for bounded simulation employ landmark vectors and distance vectors, a revision of landmarks proposed in Potamias et al. [2009]. We also propose the incremental maintenance problem for landmark and distance vectors, and provide boundedness analysis as well as incremental maintenance algorithms, which are not addressed in Potamias et al. [2009].

Organization. The article consists of three parts. We first introduce bounded simulation in Section 2, and provide an algorithm for matching based on bounded simulation in Section 3. We then present the incremental graph pattern matching problem and its complexity metrics in Section 4. The boundedness analysis and incremental algorithms are given in Sections 5, 6, and 7 for matching defined in terms of simulation, bounded simulation, and subgraph isomorphism, respectively. Finally, an experimental study is presented in Section 8, followed by open issues for future work in Section 9.

2. GRAPH PATTERN MATCHING REVISED

In the following we first define data graphs and pattern graphs. We then introduce the notion of bounded simulation. Finally, we state the revised graph pattern matching problem.

2.1. Data Graphs and Pattern Graphs

A *data graph* is a directed graph $G = (V, E, f_A)$, where: (1) V is a finite set of nodes; (2) $E \subseteq V \times V$, in which (v, v') denotes an edge from node v to v'; and (3) $f_A(v)$ is a function such that for each node v in V, $f_A(v)$ is a tuple $(A_1 = a_1, \ldots, A_n = a_n)$, where a_i is a constant, and A_i is referred to as an *attribute* of v, written as $v.A_i = a_i$.

Intuitively, the attributes of a node carry the content of the node, for example, label, keywords, blogs, comments, rating [Amer-Yahia et al. 2007].

We will also use the following notations. (1) A *path* ρ in graph G is a sequence of nodes $(v_1 \ldots, v_n)$ such that (v_i, v_{i+1}) is an edge in G for each $i \in [1, n-1]$. (2) The *length* of the path ρ, denoted by $\mathsf{len}(\rho)$, is $n-1$, *i.e.,* it is the number of edges in ρ. (3) The path ρ is *nonempty* if $\mathsf{len}(\rho) \geq 1$. Abusing terminologies for trees, we refer to v_2 as a *child* of v_1 (or v_1 as a *parent* of v_2), and v_i as a *descendant* of v_1 for $i \in [2, n]$. We denote the parents (respectively, children) set of a node u in G as $\mathsf{Pr}(u)$ (respectively, $\mathsf{Cr}(u)$).

Patterns. A *b-pattern* is defined as $P = (V_p, E_p, f_V, f_E)$, where: (1) V_p and E_p are the set of nodes and the set of directed edges, respectively, as defined for data graphs; (2) f_V is a function defined on V_p such that for each node u, $f_V(u)$ is the *predicate* of u, defined as a conjunction of atomic formulas of the form A op a; here A denotes an attribute, a is a constant, and op is a comparison operator $<, \leq, =, \neq, >, \geq$; and (3) f_E is a function on E_p such that for each edge (u, u'), $f_E(u, u')$ is either a positive integer k or symbol $*$.

Intuitively, the predicate $f_V(u)$ of a node u specifies a search condition on labels and data contents. We say that a node v in a data graph G *satisfies* the search condition of a pattern node u in P, denoted as $v \sim u$, if for each atomic formula 'A op a' in $f_V(u)$, there exists an attribute A in $f_A(v)$ such that $v.A$ op a.

As will be seen shortly, an edge (u, u') in a pattern P is mapped to a path ρ in a data graph G, and $f_E(u, u')$ is a bound on the length of ρ when it is not $*$.

When $f_V(u)$ is $A = l$, where A is the attribute denoting node label, we simply write $f_V(u)$ as A. We refer to P as a *normal pattern* if for each edge $(u, u') \in E_p$, $f_E(u, u') = 1$, and we omit $f_E(u, u')$ when it is 1. Intuitively, a normal pattern enforces edge-to-edge mappings, as found in graph simulation and subgraph isomorphism. Traditional graph pattern matching is defined on normal patterns [Gallagher 2006].

Example 2.1. Figure 1 depicts a b-pattern P_0, in which an edge is labeled with either 1 or 3. Each node denotes a suspect, with its predicate (omitted from the figure) defined in terms of characteristics discovered by law enforcement, such as criminal records and the density of contacts [Natarajan 2000].

As another example, P_1 in Figure 2 is a pattern taken from social matching [Terveen and McDonald 2005]. In P_1, each node denotes a person, with a predicate specifying her job title and hobby. To start up a company, user A wants to find in, for example, Facebook (depicted as G_1): (1) a software engineer (SE) and (2) a human-resource (HR) expert, both within 2 hops; and (3) sale managers (DM) who play golf and are connected to A through a chain of friends, and moreover, are within 1 hop of SE or 2 hops of HR.

Pattern P_2 in Figure 2 shows a pattern in, for instance, Twitter. Each node in P_2 denotes a person, with a predicate specifying her academic field, such as CS, Bio (Biology), Med (Medicine) and Soc (Sociology). Assume that nodes DB and AI have attribute dept = CS; Gen (genetics), and Eco (ecology) have attribute dept = Bio. A CS person B wants to find collaborators in biology (within 2 hops), sociology (3 hops), and in medicine who are in turn connected to CS people via chains of friends. In addition, the biology researchers should have connections to people in sociology (2 hops) and medicine (3 hops).

2.2. Bounded Graph Simulation

We now define bounded simulation.

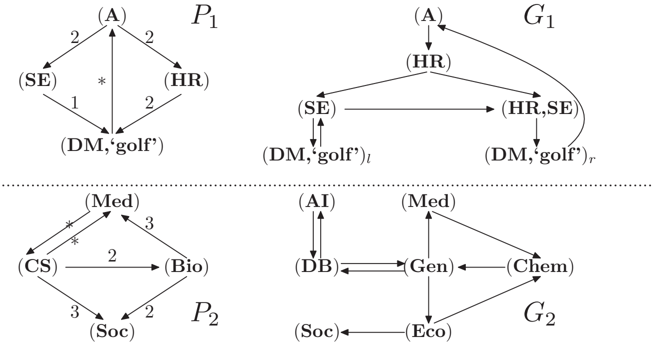


Fig. 2. Bounded simulation.

Bounded simulation. Consider a data graph $G = (V, E, f_A)$ and a b-pattern $P = (V_p, E_p, f_V, f_E)$. We say that graph G *matches* pattern P via *bounded simulation*, denoted by $P \trianglelefteq_{\mathsf{bsim}} G$, if and only if (iff) there exists a binary relation $S \subseteq V_p \times V$ such that:

(1) for each node u in V_p, there exists a node $v \in V$ such that: $(u, v) \in S$;
(2) for each pair $(u, v) \in S$, $v \sim u$; and
(3) for each edge (u, u') in E_p, there exists a nonempty *path* $\rho = (v, \ldots, v')$ from v to v' in G such that: (a) $(u', v') \in S$, and (b) $\mathsf{len}(\rho) \leq k$ if $f_E(u, u')$ is a constant k.

We refer to relation S as a *match* in G for P. To simplify the discussion, we also call $S = \emptyset$ a match for P, and write $P \ntrianglelefteq_{\mathsf{bsim}} G$ if there exists no nonempty match in G for P.

Intuitively, $(u, v) \in S$ if: (1) the node v in G satisfies the search condition specified by $f_V(u)$ in P, and (2) each edge (u, u') in P is mapped to a nonempty path $\rho = (v, \ldots, v')$ in G, such that the length of ρ is bounded by k if $f_E(u, u') = k$. If $f_E(u, u') = *$, $\mathsf{len}(\rho)$ is not bounded. Observe that the child u' of u is mapped to a descendant v' of v via S. Note that there exists a path ρ from v to v' with $\mathsf{len}(\rho) \leq k$ iff the shortest path from v to v' is no longer than k, that is, the distance from v to v' is no larger than k.

Example 2.2. In Figure 1, a match S_0 in G_0 for P_0 maps B to B, AM to $\mathsf{A}_1, \ldots, \mathsf{A}_m$, S to A_m, and FW to all the W nodes.

As another example, now consider graphs and patterns given in Figure 2.

(1) $P_1 \trianglelefteq_{\mathsf{bsim}} G_1$. A match S_1 in G_1 for P_1 is defined by mapping: (a) A to A, (b) SE to both (HR, SE) and SE, (c) HR to HR and (HR, SE), and (d) DM to both (DM,'golf') nodes in G_1. Here HR and SE in P_1 are mapped to the same node (HR, SE) in G_1, and DM is mapped to two nodes (DM, 'golf') in G_1. Edge (A, SE) in P_1 is mapped to paths in G_1. These are not allowed by bijective functions. Note that P_1 is *not isomorphic* to any subgraph of G_1.
(2) $P_2 \trianglelefteq_{\mathsf{bsim}} G_2$. Here a match S_2 in G_2 for P_2 can be defined by mapping CS to DB, Bio to Gen and Eco, Med to Med, and Soc to Soc. However, P_2 is *not isomorphic* to any subgraph of G_2. Here CS cannot be mapped to AI since there is no path within 3 hops from AI to Soc as required by the edge (CS, Soc) in P_2.
(3) $P_2 \ntrianglelefteq_{\mathsf{bsim}} G'_2$, where G'_2 (not shown) is the same as G_2 except that the edge (DB, Gen) is dropped. Indeed, CS can no longer find a match in G'_2 that is within 3 hops to Soc. In this case, the only match S for P_2 in G'_2 via bounded simulation is empty.

Remark.

(1) A match S is a relation rather than a function. Hence, for each u in V_p there may exist multiple nodes v in V such that (u, v) is in S, that is, each node in P is mapped to a nonempty set of nodes in G. As opposed to subgraph isomorphism, bounded simulation supports: (a) simulation relations rather than bijective functions, (b) predicates specifying search conditions 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 when only normal patterns are used, that is, when f_V consists of only $A = l$ and $f_E(u, u') = 1$ for all $(u, u') \in E_p$. It supports label equality testing and allows edges in P to be mapped to edges in G only.

(3) One can readily extend data graphs and patterns by incorporating edge colors to specify, for example, various relationships [Amer-Yahia et al. 2007]. We can extend bounded simulation by requiring match on edge colors, to enforce relationships in a pattern to be mapped to the same relationships in a data graph (see Fan et al. [2011b]).

Maximum match. There are possibly multiple matches in a graph G for a pattern P. Nonetheless, there exists a unique *maximum* match S_M in G for P. That is, for any match S in G for P, $S \subseteq S_M$. For instance, S_0, S_1, S_2 of Example 2.2 are maximum.

PROPOSITION 2.1. *For any graph G and pattern P, there exists a unique maximum match in G for P.*

PROOF. Observe that a match S for P in G always exists. Indeed, if $P \trianglelefteq_{\mathsf{bsim}} G$, then obviously there exists a match S for P in G that is total. When $P \not\trianglelefteq_{\mathsf{bsim}} G$, $S = \emptyset$ is such a match. Then it suffices to show the following.

(1) There exists a maximum match. If $P \not\trianglelefteq_{\mathsf{bsim}} G$, $S = \emptyset$ is the maximum match. If $P \trianglelefteq_{\mathsf{bsim}} G$, then for all matches S_1 and S_2, $S_3 = S_2 \cup S_1$ is also a match. Indeed, for each $(u, v) \in S_3$, (u, v) is either in S_1 or in S_2, and v is a match for u, that is, it satisfies the conditions of bounded simulation. Moreover, $S_1 \subseteq S_3$ and $S_2 \subseteq S_3$. From this it follows that the maximum match S_M exists, which is the union of all matches in G for P.

(2) There is a uniqueness of the maximum match. If $P \not\trianglelefteq_{\mathsf{bsim}} G$, the empty set S is the unique maximum match. If $P \trianglelefteq_{\mathsf{bsim}} G$, assume by contradiction that there exist two distinct maximum matches S_1 and S_2. Let $S_3 = S_2 \cup S_1$. Then S_3 is also a match, while $S_1 \subset S_3$ and $S_2 \subset S_3$. This contradicts the assumption that S_1 and S_2 are maximum. □

Intuitively, S_M captures all nodes of a community that match the pattern P in a network G. Note that the cardinality $|S_M|$ of S_M is bounded: $|S_M| \leq |V||V_p|$, where V (respectively, V_P) is the set of nodes in G (respectively, P).

2.3. The Graph Pattern Matching Problem

We revise graph pattern matching as follows. The *graph pattern matching problem* is to find, given any data graph G and pattern graph P, the *maximum match* in G for P, denoted by $M(P, G)$. We consider the following notions of graph pattern matching.

Bounded simulation. For a b-pattern P and a data graph G, the maximum match, denoted as $M_{\mathsf{ksim}}(P, G)$, is the unique maximum match for P in G as defined earlier. By Proposition 2.1, graph pattern matching via bounded simulation is well-defined.

Graph simulation. As remarked earlier, graph simulation is a special case of bounded simulation when P is a normal pattern. We use $P \trianglelefteq_{\mathsf{sim}} G$ to denote G matches P via graph simulation (see Section 1 for its definition). The maximum match, denoted by $M_{\mathsf{sim}}(P, G)$, is the unique maximum match for P in G (see Proposition 2.1).

Subgraph isomorphism. For a normal pattern P and a data graph G, the maximum match, denoted as $M_{\mathsf{iso}}(P, G)$, consists of all subgraphs $G' = (V', E')$ of G that are isomorphic to P (see Section 1 for subgraph isomorphism). If $M_{\mathsf{iso}}(P, G)$ is not empty, we say that G matches P, denoted as $P \trianglelefteq_{\mathsf{iso}} G$. Otherwise, $P \not\trianglelefteq_{\mathsf{iso}} G$.

We summarize the notions in the table that follows.

$\trianglelefteq_{\mathsf{iso}}$	subgraph isomorphism
$\trianglelefteq_{\mathsf{bsim}}$	bounded simulation
$\trianglelefteq_{\mathsf{sim}}$	graph simulation
$M_{\mathsf{sim}}(P, G)$	the unique maximum match in G for P via simulation, for a normal pattern P
$M_{\mathsf{ksim}}(P, G)$	the unique maximum match in G for P via bounded simulation, for a b-pattern P
$M_{\mathsf{iso}}(P, G)$	the set of all matches in G for P via subgraph isomorphism, for a normal P

Remark. Subgraph isomorphism is often used in, for example, chemical or bioinformatics, where matches with identical structure are preferred [Gallagher 2006]. However, its complexity makes it infeasible to find matches in “big” graphs such as social graphs. For Web and social networks analyses, it often suffices to find “inexact” matches [Gallagher 2006], which do not necessarily have identical structures of the pattern. Indeed, simulation and bounded simulation are used to detect social positions [Brynielsson et al. 2010] and classify Web sites [Cho et al. 2000]. Moreover, variants of simulation that preserve more topology, such as bisimulation [Dovier et al. 2001] or dual simulation [Ma et al. 2011], may induce results that “approximate” isomorphic subgraphs.

3. GRAPH PATTERN MATCHING VIA BOUNDED SIMULATION

In this section we investigate the graph pattern matching problem based on bounded simulation. The main result of this section is the following.

THEOREM 3.1. *For any pattern $P = (V_p, E_p, f_V, f_E)$ and graph $G = (V, E, f_A)$, it is in $O(|V||E| + |E_p||V|^2 + |V_p||V|)$ time to compute $M_{\mathsf{ksim}}(P, G)$ in G for P.*

As opposed to the NP-hardness of subgraph isomorphism, this revised notion of graph pattern matching allows us to find matches in polynomial time. As remarked earlier, it takes $O((|V| + |V_p|)(|E| + |E_p|))$ time to compute the maximum graph simulation relation from P to G [Henzinger et al. 1995]. This tells us that bounded simulation does not make our lives much harder since: (1) P is typically much smaller than G in practice, and (2) $|E|$ is in $O(|V|^2)$ in the worst case.

We next prove Theorem 3.1 by providing an algorithm with the desired properties.

Algorithm. The algorithm, referred to as Match, is shown in Figure 3. Given P and G, it returns the maximum match relation $M_{\mathsf{ksim}}(P, G)$ for P in G. In a nutshell, Match creates for each node in P a set of “potential” matches, which is a set of nodes in G. It iteratively refines the sets by removing from them those nodes that violate the connectivity and distance constraints posed by P, until no further change can be made.

To understand the algorithm, we first present notations it uses. We use u, u' to denote nodes in the pattern P, and v, v', v_1, v'_1 for nodes in the data graph G. In addition: (1) we use a *distance matrix* $\mathcal{X}$ to maintain the distances between all pairs of nodes in G. (2)

```
Input: A b-pattern $P = (V_p, E_p, f_V, f_E)$ and data graph $G = (V, E, f_A)$.
Output: The maximum match $M_{\mathsf{ksim}}(P, G)$ if $P \trianglelefteq_{\mathsf{bsim}} G$, and $\emptyset$ otherwise.
1.  set $S := \emptyset$; compute the distance matrix $\mathcal{X}$ of $G$;
2.  for each $(u', u) \in E_p$ and each $v \in V$ do
3.      if $f_A(v)$ satisfies $f_V(u)$ then compute $\mathsf{anc}(f_E(u', u), f_V(u'), v)$;
4.      if $f_A(v)$ satisfies $f_V(u')$ then compute $\mathsf{desc}(f_E(u', u), f_V(u), v)$;
5.  for each $u \in V_p$ do
6.     $\mathsf{mat}(u) := \{v \mid v \in V, f_A(v)$ satisfies $f_V(u)$,
                    and out-degree$(v) \neq 0$ if out-degree$(u) \neq 0\}$;
7.     $\mathsf{premv}(u) := \{v' \mid v' \in V,$ out-degree$(v') \neq 0$, and
                    $\nexists(u', u) \in E_p$ $(v \in \mathsf{mat}(u)$, $f_A(v')$ satisfies $f_V(u')$,
                    and $\mathsf{len}(v', \cdots, v) \leq f_E(u', u))\}$;
8.  while (there exists a node $u \in V_p$ with $\mathsf{premv}(u) \neq \emptyset$) do
9.     for each $(u', u) \in E_p$ and each $v_1 \in \mathsf{premv}(u)$ do
10.       if $v_1 \in \mathsf{mat}(u')$ then
11.          $\mathsf{mat}(u') := \mathsf{mat}(u') \setminus \{v_1\}$;
12.          if $\mathsf{mat}(u') = \emptyset$ then return $\emptyset$;
13.          for each $u''$ with $(u'', u') \in E_p$ do
14.             for each $v'_1 \in (\mathsf{anc}(f_E(u'', u'), f_V(u''), v_1) \setminus \mathsf{premv}(u'))$ do
15.                if $(\mathsf{desc}(f_E(u'', u'), f_V(u'), v'_1) \cap \mathsf{mat}(u') = \emptyset)$ then
16.                   $\mathsf{premv}(u') := \mathsf{premv}(u') \cup \{v'_1\}$;
17.    $\mathsf{premv}(u) := \emptyset$;
18. for each $u \in V_p$ and each $v \in \mathsf{mat}(u)$ do $S := S \cup \{(u, v)\}$;
19. return $S$ as $M_{\mathsf{ksim}}(P, G)$;
```

Fig. 3. Algorithm Match.

For each node u in P, we use a set $\mathsf{mat}(u)$ to record nodes v in G that may match u ($f_A(v)$ satisfies $f_V(u)$), and a set $\mathsf{premv}(u)$ for those nodes that cannot match any parent of u. (3) For each node $v \in V$ and edge $(u', u) \in E_p$, $\mathsf{anc}(f_E(u', u), f_V(u'), v)$ records nodes v' in the graph G such that: (i) the distance from v' to v is within the bound imposed by f_E, that is, $\mathsf{len}(v', \ldots, v) \leq f_E(u', u)$, and (ii) $f_A(v')$ (respectively, $f_A(v)$) satisfies the predicate $f_V(u')$ (respectively, $f_V(u)$) defined on u' (respectively, u); similarly for $\mathsf{desc}(f_E(u', u), f_V(u), v)$, for descendants of v. These notations are summarized next.

u, u' (resp. v, v', v_1, v'_1)	nodes in the pattern P (resp. G)
$\mathcal{X}$	distance matrix
$\mathsf{mat}(u)$	nodes in G that may match u in P
$\mathsf{premv}(u)$	nodes in G which cannot match any *parent* of u
$\mathsf{anc}(f_E(u', u), f_V(u'), v)$	nodes v' in G that may match a parent u' of u
$\mathsf{desc}(f_E(u', u), f_V(u), v)$	nodes v' in G that may match a child u of u'

Algorithm Match first computes the distance matrix $\mathcal{X}$ for G (line 1). Using $\mathcal{X}$, it then computes $\mathsf{anc}(\cdot)$ and $\mathsf{desc}(\cdot)$ by inspecting the predicates and bounds specified in P (lines 2–4). For each pattern node $u \in V_p$, Match also initializes $\mathsf{mat}(u)$ and $\mathsf{premv}(u)$ based on P and $\mathcal{X}$ (lines 5–7). The out-degrees of both u and v are inspected during the initialization; if v has zero out-degree and u has a child, v is identified as an invalid match of u, since v has no child that can match child of u. For each parent node u' of u (i.e., $(u', u) \in E_p$), Match then refines $\mathsf{mat}(u')$ by removing those nodes in G that cannot match u', namely, nodes $v_1 \in \mathsf{premv}(u)$ (lines 9–11). Moreover, it utilizes v_1 to identify nodes v'_1 that cannot match any parent u'' of u', and includes v'_1 in $\mathsf{premv}(u')$ (lines 13–16). More specifically, v'_1 is not a candidate match of u'' if v_1 is the only descendant of v'_1 that is within the bound $f_E(u'', u')$, satisfies the predicate $f_V(u')$, and is in $\mathsf{mat}(u')$.

The process (lines 8–17) iterates until no $\mathsf{mat}(\cdot)$ can be reduced, that is, if $\mathsf{premv}(u)$ is empty for all pattern node u (line 8). The nodes remaining in $\mathsf{mat}(u)$ are those that match u, and are collected in the match $M_{\mathsf{ksim}}(P, G)$ (lines 18–19). If $\mathsf{mat}(u)$ is empty for any $u \in V_p$ in the process, u cannot find a match in G, and Match returns $\emptyset$ (line 12).

Example 3.2. We show how Match computes the match in graph G_2 for pattern P_2 of Example 2.2. For each node in P_2, Match initializes $\mathsf{mat}(\cdot)$ and $\mathsf{premv}(\cdot)$ as follows.

P_2	$\mathsf{mat}(\cdot)$	$\mathsf{premv}(\cdot)$	P_2	$\mathsf{mat}(\cdot)$	$\mathsf{premv}(\cdot)$
CS	{DB, AI}	{DB,AI,Gen,Chem,Eco}	Bio	{Gen, Eco}	{Med,Gen,Eco,Chem}
Med	{Med}	{Med,Chem}	Soc	{Soc}	{AI,Med,Chem}

Algorithm Match then repeatedly removes from $\mathsf{mat}(\cdot)$ those nodes that do not make a match, by using $\mathsf{premv}(\cdot)$. For instance, AI is removed from mat(CS): while AI is a candidate match for CS, it cannot reach Soc within 3 hops, as indicated by AI $\in$ premv(Soc). Match terminates when all nodes in P_2 have an empty $\mathsf{premv}(\cdot)$ set, and it returns the match S_2 given in Example 2.2, which is maximum. Similarly, one can use Match to find the maximum match in G_0 for P_0 (Figure 1) and the match in G_1 for P_1 (Figure 2).

Now consider G'_2 described in Example 2.2. Then DB is in premv(Med) and premv(Soc), and all nodes in mat(CS) will be removed by Match. This is, for CS no match can be found, and Match returns $\emptyset$ to indicate that $P_2 \ntrianglelefteq G'_2$.

We show the correctness of algorithm Match as follows.

PROPOSITION 3.1. *Algorithm* Match *computes the maximum match in G for P.*

PROOF. It suffices to show the following: (I) if Match terminates, then Match returns the maximum match S in G for P, and (II) Match always terminates.

(I). To prove (I), we first show the following claim.

CLAIM 1. Given a pattern node u' and a node v' in G, for $(u', u) \in E_p$, v' is not a match of u' iff either: (i) $v' \notin \mathsf{mat}_0(u')$, or (ii) $v' \in \mathsf{premv}_i(u) \cap \mathsf{mat}_i(u')$ (lines 9–10 of Match).

Here $\mathsf{mat}_i(u')$ (respectively, $\mathsf{premv}_i(u)$) denotes $\mathsf{mat}(u')$ (respectively, $\mathsf{premv}(u)$ for a child u of u') at the i-th iteration of the **while** loop (lines 8–17 of Match).

If Claim 1 holds then so does (I). To see this, denote by S_r the match returned by Match. We show that if Claim 1 holds then (1) $S \subseteq S_r$ and (2) $S_r \subseteq S$, that is, $S_r = S$.

(1) $S \subseteq S_r$. If S is empty, then obviously (I) holds. If S is not empty, assume that there exists a match (u', v') in S but not in S_r, that is, $v' \notin \mathsf{mat}(u')$. Observe that $v' \in \mathsf{mat}(u')$ when $\mathsf{mat}(u)$ is initialized (line 6). As v' is not in $\mathsf{mat}(u')$ when S is returned, v' must be removed from $\mathsf{mat}(u')$ at some iteration i in the **while** loop. Hence v' must be in both $\mathsf{premv}_i(u)$ and $\mathsf{mat}_i(u')$ for some pattern edge (u', u) at the i-th iteration. By Claim 1, v' does not match u', contradicting the assumption that $(u', v') \in S$. Thus $S \subseteq S_r$.

(2) $S_r \subseteq S$. Then (I) already holds when S is empty. If S is not empty, and if there exists a match (u', v') in S_r, but v' cannot match u, then v' would be removed from $\mathsf{mat}(u')$ by Claim 1, contradicting the assumption that $v' \in \mathsf{mat}(u')$.

Putting (1) and (2) together, $S_r = S$. From this (I) follows.

PROOF. Claim 1.

(If) We show that node v' does not match u' if: (i) $v' \notin \mathsf{mat}_0(u')$, or (ii) $v' \in \mathsf{premv}_i(u) \cap \mathsf{mat}_i(u')$ at iteration i of the **while** loop for some i. Consider the following cases.

(1) If $v' \notin \mathsf{mat}_0(u')$, then v' does not satisfy $f_V(u')$. Thus v' cannot match u'.

(2) Suppose that $v' \in \mathsf{mat}_0(u')$. We prove it by induction on iteration i of the **while** loop.

If $i = 0$, then v' is added to $\mathsf{premv}_0(u)$ when $\mathsf{premv}(u)$ is initialized (line 7). Hence for any node v that can possibly match u, $\mathsf{len}(v', v) > f_E(u', u)$. Thus v' cannot match u'.

Assume the statement for $i \leq k$. We show that it also holds for $i = k + 1$. If $v' \in \mathsf{premv}_{k+1}(u) \cap \mathsf{mat}_{k+1}(u')$, then v' is added to $\mathsf{premv}_j(u)$ at some iteration j of the loop (line 16). By the induction hypothesis, there must exist a node $v_1 \in \mathsf{premv}_j(u) \cap \mathsf{mat}_j(u')$, that is, v_1 cannot match u, and v' cannot reach any node that is not v, within $f_E(u', u)$ hops, and it can match u. Thus v' is not a match of u'.

(Only-if). For a node u' in P and a node v' in G, if v' cannot match u' (i.e., a mismatch), then: (1) v' does not satisfy $f_V(u')$, or for a child u of u', either (2) no node v within $f_E(u', u)$ hops of v' satisfies $f_V(u)$, or (3) all nodes v that satisfy $f_V(u)$ and are within $f_E(u', u)$ hops of v' cannot match u. We show (Only-if) for all these cases.

For v' in case (1), it is not added to $\mathsf{mat}(u')$ at the initialization phase (line 6). Thus $v' \notin \mathsf{mat}_0(u')$. For case (2), v' will be added to $\mathsf{premv}_0(u)$ when $\mathsf{premv}_0(u)$ is initialized. Thus the test $v' \in \mathsf{premv}_0(u) \cap \mathsf{mat}_0(u')$ is true for v'.

If v' is in case (3), then there must exist a mismatch v_0 of u_0 in case (2) that makes v' a mismatch, directly (if u_0 is a child of u') or indirectly (if u_0 is a descendant of u'). Suppose that there is a sequence of mismatches $v_0, \ldots, v_k$ for $u_0, \ldots, u_k$, where for each (u_i, v_i) and (u_{i+1}, v_{i+1}), u_{i+1} is a parent of u_i, and v_{i+1} cannot match u_{i+1} because its only descendant v_i cannot match u_i. We show (Only-if) by induction on the index i of the sequence. (a) If $i = 1$, then: (i) v_0 is in $\mathsf{premv}(u_s)$ for some child u_s of u_0, and (ii) since v_1 has no descendant other than v_0 that can match u_0, v_1 will be added to $\mathsf{premv}(u_0)$, and hence, $v' \in \mathsf{premv}_1(u_0) \cap \mathsf{mat}_1(u_1)$. (b) Assume the statement for $i \leq k$. By the induction hypothesis, v_k must be in $\mathsf{premv}_k(u_{k-1}) \cap \mathsf{mat}_k(u_k)$ at iteration k. For mismatch v_{k+1} of u_{k+1}, v_{k+1} has no descendant other than v_k to match u_k. Thus v_{k+1} is moved to $\mathsf{premv}_k(u_k)$. Hence at iteration $k + 1$, v_{k+1} is in $\mathsf{premv}_{k+1}(u_k) \cap \mathsf{mat}_{k+1}(u_{k+1})$.

(*II*). We next prove (II). First, $\mathsf{mat}(u')$ is reduced monotonically by Match for each pattern node u'. Indeed, after the initialization of $\mathsf{mat}(\cdot)$, Match will only remove nodes from $\mathsf{mat}(\cdot)$, and never put nodes back. Second, Match reduces $\mathsf{mat}(u')$ for each pattern node u' if $\mathsf{premv}(u) \cap \mathsf{mat}(u')$ is nonempty for some pattern edge (u', u) (lines 9–11). If there is a pattern node u' such that $\mathsf{mat}(u')$ reduces to $\emptyset$, Match returns $\emptyset$ (line 12). On the other hand, if no $\mathsf{mat}(\cdot)$ can be further reduced, $\mathsf{premv}(u)$ will be set to $\emptyset$ (line 17) for each pattern node u, and the **while** loop terminates. In both cases, Match terminates.

From (I) and (II) the correctness of algorithm Match follows. □

Complexity. We next show that Match is in $O(|V||E| + |E_p||V|^2 + |V_p||V|)$ time. Algorithm Match consists of three phases: preprocessing (lines 1–7), match computation (lines 8–17), and match result collection (lines 18–19).

(i) For preprocessing: (a) it takes $O(|V|(|V|+|E|))$ time to compute the distance matrix $\mathcal{X}$ by using BFS search [Bang-Jensen and Gutin 2008] (line 1); (b) initializing $\mathsf{anc}(\cdot)$ and $\mathsf{desc}(\cdot)$ takes $O(|E_p||V|^2)$ time (lines 2–4); and (c) $\mathsf{mat}(\cdot)$ and $\mathsf{premv}(\cdot)$ are computed in $O(|V_p||V|)$ and $O(|E_p||V|^2)$ time, respectively (lines 5–7). The predicate of a node in P can be inspected at a node in G in linear time when the attributes in P and G are sorted in the same order. The total cost in this phase is thus $O(|E_p||V|^2 + |V_p||V| + |V||E|)$ time.

(ii) To compute the maximum match, Match maintains a matrix $\mathcal{X}'$ (omitted from Figure 3 to simplify the exposition). For each pattern edge (u', u) and each node $v' \in \mathsf{mat}(u')$, the result of $\mathsf{desc}(f_E(u', u), f_V(u), v') \cap \mathsf{mat}(u)$ is computed and stored in $\mathcal{X}'$. The matrix $\mathcal{X}'$ is computed in $O(|E_p||V|^2)$ time, and can be maintained incrementally when $\mathsf{mat}(u')$ is updated (line 11). With $\mathcal{X}'$, we can conduct the test of line 15 in constant time.

Utilizing $\mathcal{X}'$, observe the following about the complexity of the **while** loop (lines 8–17): given $u \in V_p$ and $v_1 \in \mathsf{premv}(u)$: (1) once v_1 is removed from $\mathsf{premv}(u)$, it will never be put back again; (2) for each pattern node u, $\mathsf{premv}(u)$ is bounded by $|V|$, and the test of $v_1 \in \mathsf{mat}(u')$ is at most once for a specified u' (here u' is a parent of u), thus the test of line 10 takes at most $O(|E_p||V|)$ times in the entire process; and (3) when $v_1 \in \mathsf{mat}(u')$ is true (only once), the inner **for** loop (lines 13–16) runs in $O(|\mathsf{out\_degree}(u'')||V|)$ time for a specified u, and hence takes $O(|E_p||V|)$ time in total, since (a) the **if** test (line 15) is conducted in $O(1)$ time with the matrix $\mathcal{X}'$; and (b) $\mathsf{anc}(f_E(u'', u'), f_V(u''), v_1)$ is bounded by $|V|$. Putting these together, the **while** loop takes $O(|E_p||V|)$ time.

(iii) The last phase (line 18) can be done in $O(|V_p||V|)$ time.

From these one can see that algorithm Match is in $O(|V||E| + |E_p||V|^2 + |V_p||V|)$ time. From Proposition 3.1 and the complexity analysis, Theorem 3.1 follows.

Remark. (1) Observe that (bounded) simulation is to compute a matching relation instead of an injective function in subgraph isomorphism, allowing a pattern node to be mapped to multiple nodes in a data graph. In this context, for each pattern node u, a set refinement process (lines 9–16) suffices to remove the nodes that cannot match u; in contrast, it may require exponential time to verify a single match for u in terms of isomorphism. Hence, (bounded) simulation leads to an exponential reduction in the size of the search space of subgraph isomorphism. (2) To demonstrate the worst-case time complexity of Match, one may verify that when P is a cycle consisting of two pattern nodes with the same node label a, and G is a path $(a_1, \ldots, a_k)$ (all with label a), Match takes $2 * \sum_{i=1}^{k} i = O(k^2)$ time to conclude that G does *not* match P.

Match can be readily extended to data graphs with weights on the edges following the same procedure. The only difference is that it computes the distance matrix with, for example, Floyd-Warshall algorithm [Floyd 1962] (in $O(|V|^3)$ time). This does not make our lives harder: its total time complexity is still in cubic time.

4. BATCH AND INCREMENTAL GRAPH PATTERN MATCHING

As remarked in Section 1, real-life graphs are typically large and are frequently updated. Although the cubic-time complexity of Match is better than intractable, it is still too costly to recompute matches every time when the graphs are updated.

This motivates us to study *incremental graph pattern matching*. In contrast to its batch counterpart, incremental matching takes as input a data graph G, a pattern (b-pattern) P, the matches $M(P, G)$ in G for P, and changes ΔG to G. It finds changes ΔM to the old matches such that $M(P, G \oplus \Delta G) = M(P, G) \oplus \Delta M$, where operator $\oplus$ applies changes to the original data. That is, when the data graph G is updated, it computes new matches by making maximal use of previous computation for $M(P, G)$ or in other words, by minimizing unnecessary recomputation. When ΔG is small, ΔM is often also small, and is much more efficient to find than to recompute $M(Q, G \oplus \Delta G)$.

In the rest of the article, we focus on incremental graph pattern matching, for matching defined in terms of bounded simulation, as well as for matching based on traditional graph simulation and subgraph isomorphism.

Example 4.1. Graph G_3 in Figure 4 (excluding edges e_1–e_5) depicts a fraction of FriendFeed (a social networking service http://friendfeed.com/). In G_3, each node has two attributes, name and job. The node (Ann, "CTO") denotes a person with (name = "Ann", job = "CTO"). Two pattern graphs P_3 and P_3' are also shown in Figure 4, where P_3 is a b-pattern in which an edge is labeled with either a bound or $*$, specifying connectivity, and P_3' is a normal pattern in which each edge is labeled with 1 (not shown).

(1) Pattern P_3 is to find the CTOs who are connected to a DB researcher within 2 hops and to a biologist within 1 hop; moreover, the DB researcher has to reach a biologist

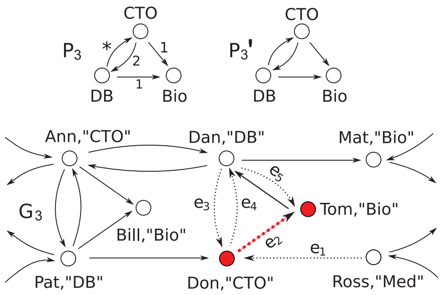


Fig. 4. Querying FriendFeed network.

within 1 hop and a CTO via a path of an arbitrary length. One can verify that $P_3 \trianglelefteq_{\mathsf{bsim}} G_3$ with $M_{\mathsf{ksim}}(P_3, G_3) = \{$(CTO, Ann), (DB, Pat), (DB, Dan), (Bio, Bill), (Bio, Mat)$\}$.

(2) Pattern P_3' is to find all subgraphs of G that are isomorphic to P_3'. Here $M_{\mathsf{iso}}(P_3', G_3)$ consists of a single subgraph of G_3 induced by nodes Ann, Pat, and Bill.

Suppose that graph G_3 is updated by inserting five edges e_1–e_5 (see Figure 4), denoted by ΔG_3. Then in the updated G_3, that is, $G_3 \oplus \Delta G_3$: (1) ΔG_3 incurs two new matches Don and Tom, for CTO and Bio in pattern P_3, respectively; and (2) ΔG_3 incurs a new subgraph induced by edges e_2–e_5 as a match for pattern P_3'. Observe that when ΔG_3 is small, the changes to the match result are also small. It is less costly to find the changes to the match result than to recompute all the matches starting from scratch.

As argued in Ramalingam and Reps [1996b], it is not very informative to define the cost of an incremental algorithm as a function of the size of the input, as found in traditional complexity analysis for batch algorithms. Instead, one should analyze the algorithms in terms of |CHANGED|, the size of the changes in the input and output of the incremental problem, which represents the updating costs that are *inherent to* the problem itself [Ramalingam and Reps 1996b]. Along the same line, we characterize the complexity of incremental matching algorithms in terms of |CHANGED|. Given P, G, $M(P, G)$, ΔG and ΔM, we define |CHANGED| as $|\Delta G| + |\Delta M|$, where $|\Delta G|$ (respectively, $|\Delta M|$) indicates the size of changes in the data graph (input) (respectively, match result (output)).

To visually depict match result and ΔM, we represent $M(P, G)$ as a graph, referred to as the *result graph* of P in M, and use it to intuitively illustrate |CHANGED|.

Result graphs. The result graph of a pattern P in a data graph G is a graph representation of the matches $M(P, G)$. It is a graph $G_r = (V_r, E_r)$ defined as follows.

—For subgraph isomorphism, G_r is the union of all the subgraphs G' of G in $M_{\mathsf{iso}}(P, G)$.
—For bounded simulation:
 —V_r consists of all the nodes v in G such that $(u, v) \in M_{\mathsf{ksim}}(P, G)$, that is, v is a match of some pattern node u in the maximum match; and
 —for each edge (u_1, u_2) in E_p, there is an edge $(v_1, v_2) \in E_r$ iff (u_1, v_1) and (u_2, v_2) are in $M_{\mathsf{ksim}}(P, G)$, and there exists a nonempty path ρ from v_1 to v_2 such that $\mathsf{len}(\rho) \leq k$ if $f_E(u_1, u_2) = k$, and $0 < \mathsf{len}(\rho)$ otherwise. That is, the edge (v_1, v_2) indicates the path in G to which the pattern edge (u_1, u_2) is mapped.
 Similarly result graphs are defined for graph simulation.

Observe that the edges in the result graphs indicate the connectivity among the matches, a projection from their counterpart in P.

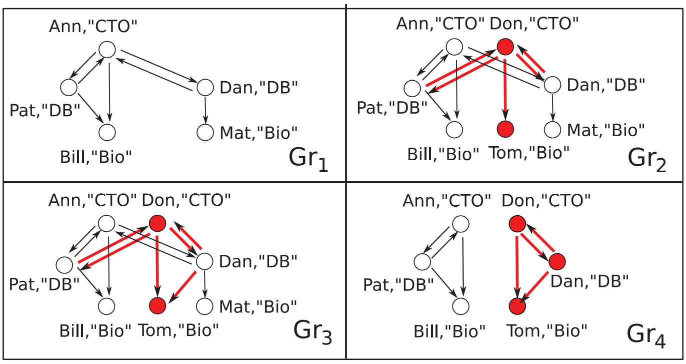


Fig. 5. Result graphs and ΔM.

The changes ΔM in the output can thus be characterized by the changes in the result graphs. Let G_r and G'_r be the result graphs of P in G and $G \oplus \Delta G$, respectively. Then ΔM is captured by the nodes and edges that are not shared by G'_r and G_r.

Example 4.2. Recall graph G_3 and b-pattern P_3 of Figure 4. The result graph of P_3 in G_3 is shown as G_{r1} in Figure 5, representing $M_{\mathsf{ksim}}(P_3, G_3)$.

When a new edge e_2 is inserted into G_3, that is, ΔG is the insertion of edge e_2, the new result graph G_{r2} of P_3 is shown in Figure 5. Then ΔM, reflected by the changes in the result graphs, includes two new nodes Don and Tom along with the new edges attached to them, that is, (Don, Pat), (Pat, Don), (Don, Tom) (Don, Dan), and (Dan, Don). That is, ΔM adds new pairs (CTO, Don) and (Bio, Tom) to $M_{\mathsf{ksim}}(P_3, G_3)$.

When $G_3 \oplus \Delta G$ is further changed by inserting edges e_1, e_3, e_4 and e_5, the new result graph is G_{r3}, also shown in Figure 5. Here the changes to G_{r1} contain nodes Don and Tom, along with all the new edges attached to them. Compared to G_{r2}, although four new edges are added to G_3, the match result ΔM contains only one new edge (Dan, Tom).

(Un)boundedness. An incremental algorithm is said to be *bounded* if its time complexity is bounded by a polynomial in |CHANGED|. An incremental graph pattern matching problem is said to be *bounded* if there exists a bounded incremental algorithm for it, and is said to be *unbounded* otherwise. A bounded problem can be solved by a PTIME algorithm with time complexity independent of $|G|$, the size of data graph.

Semiboundedness. Unfortunately, the boundedness of an incremental problem is often too strict to characterize the complexity of the problem in practice. For graph pattern matching, |CHANGED| is defined in terms of $|\Delta G|$ and $|\Delta M|$, where ΔM measures the changes in the complete matches. Consider, as an example, over a period of time, G is updated and yields a sequence of graphs $G_1, \ldots, G_{n+1}$. For $j \leq n$, G_j does not match P, that is, $M(P, G_j) = \emptyset$; but G_{n+1} matches P, that is, $M(P, G_{n+1})$ jumps to a nonempty set from $\emptyset$. While $M(P, G_{n+1})$ can be efficiently computed from partial matches in $G_1, \ldots, G_n$, the complexity measured in |CHANGED| does not reflect this. That is, complexity analysis in terms of |CHANGED| is *not* very informative: it does not capture the amount of information that an incremental algorithm necessarily maintains. As a result, an incremental problem is bounded only in special and ideal cases.

To rectify the weakness of |CHANGED|, a notion of *affected areas* AFF was proposed by Alpern et al. [1990] and Ramalingam and Reps [1996b]. Intuitively, AFF covers not only changes ΔM (complete matches), but also information that is necessarily needed to detect ΔM, including changes in partial matches, encoded in auxiliary data structures. As observed by Alpern et al. [1990], complexity analysis in terms of $|\Delta G|$

and |AFF| effectively demonstrates the advantage of incremental algorithms over their batch counterparts. We also adopt AFF to analyze our incremental matching algorithms since after all, incremental algorithms are to maximally reuse previous computation, including both prior complete matches and partial matches (in auxiliary structures).

Auxiliary information. We characterize the essential information needed to detect ΔM as "local information" for nodes v in G [Ramalingam and Reps 1996b].

(1) For graph simulation, the local information includes: (a) whether v is a match of some pattern node u, (b) whether v is a *candidate*, that is, it satisfies the predicate of u but does *not* yet match u, and (c) whether a child of v is a match or a candidate. For each node u in the pattern graph P, we use sets $\mathsf{match}(u)$ and $\mathsf{candt}(u)$ to store its matches and candidates in G, respectively. We also encode the connectivity relation on the matches and candidates in these two sets, which is part of the local information.

(2) For bounded simulation, in addition to $\mathsf{match}(\cdot)$ and $\mathsf{candt}(\cdot)$, the local information includes the distance between v and v' in G, where v and v' are matches or candidates of two adjacent nodes u and u' in P, respectively. We defer the definition of auxiliary structures for the distance information to Section 6.

The information is essential for detecting ΔM. If any of the information is missing, it can be verified that matches (respectively, candidates) that become invalid (respectively, new matches) cannot be detected by using prior computation without traversing irrelevant part of G (along the same lines as the proofs of Theorems 5.1(1) and 6.1(1)). In other words, the information is needed by *any* incremental algorithm for the problem, which is independent of algorithms but inherent to the incremental matching problem.

(3) For subgraph isomorphism, we show that incremental matching is inherently intricate no matter what auxiliary structures of polynomial size are used (Section 7).

Affected area. For P, ΔG, G and $M(P, G)$, the *affected area*, denoted as AFF, is the local information of the nodes in G given before that must be accessed to detect ΔM.

Intuitively, let G_r and G'_r be the result graphs of P in G and $G \oplus \Delta G$, respectively. Then AFF includes: (1) those nodes and edges that are not shared by G'_r and G_r (i.e., ΔM), (2) changes in auxiliary structures $\mathsf{match}(\cdot)$, $\mathsf{candt}(\cdot)$ (as well as those used to store the distance information for bounded simulation), especially changes to the children or parents of the nodes adjacent to those nodes identified in (1).

Semiboundedness. We now introduce the notion of semiboundedness. We say that an incremental matching algorithm is *semibounded* if: (1) its cost can be expressed as a *polynomial* function of |AFF|, $|P|$ and $|\Delta G|$, and (2) the size of the auxiliary structure is bounded by a polynomial in $|G|$. An incremental graph pattern matching problem is said to be *semibounded* if there exists a semibounded incremental algorithm for it.

That is, a semibounded incremental algorithm is in PTIME in the sizes of $|\Delta G|$, pattern P, and the amount |AFF| of information essential for identifying ΔM, by utilizing auxiliary structures of a small size. In other words, its cost depends *only* on $|\Delta G|$, $|P|$ and |AFF|, where AFF indicates the amount of work that must be done for computing ΔM by any incremental algorithms. Such algorithms often suffice in practice since the sizes of pattern P, ΔG and AFF are typically much smaller than big graph G.

In addition, we say an incremental matching algorithm is *optimal* if its cost is in $O(|\mathsf{AFF}| + |P| + |\Delta G|)$, indicating the necessary amount of work to perform for any incremental algorithm for the problem. An incremental matching problem is *optimal* if it has an optimal incremental algorithm. Note that an optimal problem is semibounded.

We summarize the notations in this section in Table I.

Table I. Notations: CHANGED, match(·), candt(·), AFF and G_r

|CHANGED|	size of total change in input and output, *i.e.,* $|\Delta G| + |\Delta M|$
match(u)	the matches of a node u in P
candt(u)	the nodes v in G that satisfy the predicate of u, but do not match u in P
AFF	affected area, ΔM and updates in auxiliary structures for essential local information
G_r	result graphs, a graph representation of $M(P, G)$

Example 4.3. The auxiliary structures store the essential information for identifying ΔM for any incremental algorithm, and improve query efficiency.

(1) Recall the sequence $G_1, \ldots, G_n$ resulted from updating a graph G mentioned earlier. With auxiliary structures match(·) and candt(·), one can efficiently compute $M(P, G_j)$ by leveraging partial matches computed earlier, no matter whether $P \trianglelefteq_{\mathsf{sim}} G_j$ (Section 5).
(2) As another example, consider a DAG pattern P and a graph G. Assume that G cannot match P via (bounded) simulation until n edges are inserted one by one into G. Without auxiliary structure the worst-case complexity is $O(n|P||G|)$. As will be seen in Section 5, with match(·) and candt(·), we can do it in $O(n|\mathsf{AFF}|)$ time, and $|\mathsf{AFF}| \ll |G|$. □

Based on this complexity model, next we first study incremental graph pattern matching defined in terms of graph simulation (Section 5). We then extend the study to bounded simulation (Section 6). We provide both complexity bounds and effective incremental algorithms in these settings. Finally, we provide complexity analysis for incremental matching via subgraph isomorphism (Section 7).

Remark. In Ramalingam and Reps [1996b] and Fan et al. [2011a], a more general notion of |CHANGED| is adopted, which includes ΔG, ΔM and changes to certain auxiliary structure (AFF). We use a strict notion of |CHANGED| in this work, which only depends on $|\Delta G|$ and $|\Delta M|$, to distinguish boundedness and semiboundedness of incremental algorithms.

5. INCREMENTAL SIMULATION MATCHING

In this section we study the incremental simulation problem, referred to as IncSim. Given a *normal pattern P*, a data graph G, a result graph G_r (depicting the unique maximum simulation $M_{\mathsf{sim}}(P, G)$), and changes ΔG to G, IncSim is to compute *the changes* to result graph G_r, which represents ΔM such that $M_{\mathsf{sim}}(P, G \oplus \Delta G) = M_{\mathsf{sim}}(P, G) \oplus \Delta M$. The main results of this section are as follows.

THEOREM 5.1. *The incremental simulation problem is:*

(1) *unbounded even for single updates and general (possibly cyclic) patterns;*
(2) *semibounded and optimal for: (a) single-edge deletions and general patterns, and for (b) single-edge insertions and* DAG *patterns, both in linear time* $O(|\mathsf{AFF}|)$*; and*
(3) *semibounded in* $O(|\Delta G|(|P||\mathsf{AFF}| + |\mathsf{AFF}|^2))$ *time for batch updates and general patterns.*

To the best of our knowledge, Theorem 5.1 presents the first boundedness analysis for IncSim. While the problem is unbounded, it is semibounded: its cost depends only on the size of the changes in ΔG, P and necessary auxiliary information AFF, which are small in practice. Note that if |CHANGED| is defined to include $|\Delta G|$ and |AFF| [Ramalingam and Reps 1996b], the cases in Theorem 5.1(2) are bounded [Fan et al. 2011a].

PROOF OF THEOREM 5.1(1). We show that IncSim is unbounded for a single-edge *insertion* and a pattern with a single *cycle*. Consider an instance of IncSim shown in Figure 6,

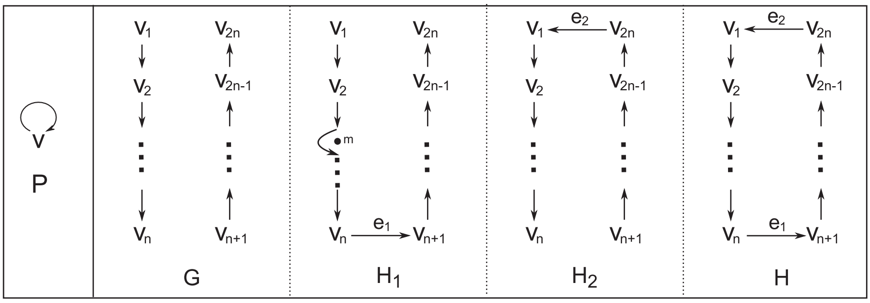


Fig. 6. Unboundedness of IncSim.

which consists of: (1) a pattern P with one cycle and a pattern node v, and (2) a data graph G consisting of two chains $(v_1, \ldots, v_n)$ and $(v_{n+1}, \ldots, v_{2n})$, where each node in G has the same label as v, and $n \geq 2$. Given unit update ΔG to G, one may verify that deciding whether $P \trianglelefteq_{\mathsf{sim}} G \oplus \Delta G$ is equivalent to checking whether there exists a cycle in the updated graph. Indeed, any node v_i in a cycle of G is a match of v.

Denote by Δ_1 the insertion of edge $e_1 = (v_n, v_{n+1})$ into G, and by Δ_2 the insertion of $e_2 = (v_{2n}, v_1)$. Figure 6 shows $H_1 = G \oplus \Delta_1$, $H_2 = G \oplus \Delta_2$ and $H = H_1 \oplus \Delta_2$. Observe that

—$M_{\mathsf{sim}}(P, G) = M_{\mathsf{sim}}(P, H_1) = M_{\mathsf{sim}}(P, H_2) = \emptyset$; and
—$M_{\mathsf{sim}}(P, H) = \{(v, v_i) \mid i \in [1, 2n]\}$.

We show that no bounded incremental algorithm $\mathcal{A}$ can carry out the updates, that is, IncSim is unbounded. Recall that $\mathcal{A}$ is bounded if given P, G, a unit update ΔG and the old output $M_{\mathsf{sim}}(P, G)$, $\mathcal{A}(P, G, \Delta G, M_{\mathsf{sim}}(P, G))$ computes ΔM and moreover, its cost is a function of |CHANGED| (Table I). Note that |CHANGED| depends on $|\Delta M|$ alone since $|\Delta G|$ and $|P|$ are constants. In terms of result graphs, $|\Delta M|$ after either Δ_1 or Δ_2 alone is 0, while $|\Delta M|$ after Δ_1 and Δ_2 is $4n$, which corresponds to a result graph as H itself.

Assume by contradiction that such algorithm $\mathcal{A}$ exists. Then $\mathcal{A}(P, G, \Delta_1, M_{\mathsf{sim}}(P, G))$ and $\mathcal{A}(P, G, \Delta_2, M_{\mathsf{sim}}(P, G))$ are both in $O(1)$ time, since Δ_1 and Δ_2 are unit updates, and the changes in the outputs (i.e., $M_{\mathsf{sim}}(P, H_1)$ and $M_{\mathsf{sim}}(P, H_2)$) are empty in both cases. Algorithm $\mathcal{A}$ conducts matching by traversing G and updating the status of those nodes visited. The status is associated with each node u, denoted by $s(u)$. To investigate the behavior of $\mathcal{A}$, we consider the sequence of nodes visited by $\mathcal{A}$ when computing ΔM, while $\mathcal{A}$ may update the status of those nodes. We refer to the sequence of nodes visited as the *trace* of $\mathcal{A}(P, G, \Delta G, M_{\mathsf{sim}}(P, G))$ and denote it by $T(G, \Delta G)$.

To see that such an algorithm $\mathcal{A}$ does not exist, observe the following.

(1) There exist nodes m in $T(G, \Delta_1)$ such that $\mathcal{A}(P, G, \Delta_1, M_{\mathsf{sim}}(P, G))$ updates their status. Indeed, $\mathcal{A}(P, G, \Delta_2, M_{\mathsf{sim}}(P, G))$ finds no match while $\mathcal{A}(P, H_1, \Delta_2, M_{\mathsf{sim}}(P, G))$ identifies all the nodes in G as matches. Since H_1 is the same as G except the insertion of edge e_1 and $\mathcal{A}(P, G, \Delta_2, M_{\mathsf{sim}}(P, G))$ is in $O(1)$ time, the different behaviors of $\mathcal{A}$ on G and H_1 when processing Δ_2 can only be triggered if $s(m)$ differs in G and H_1 for some nodes m, that is, $\mathcal{A}$ takes different actions based on $s(m)$. This could only happen if m is in $T(G, \Delta_1)$, and moreover, $\mathcal{A}(P, G, \Delta_1, M_{\mathsf{sim}}(P, G))$ updates $s(m)$ when processing Δ_1.

(2) Algorithm $\mathcal{A}$ is unbounded. Since $\mathcal{A}(P, G, \Delta_1, M_{\mathsf{sim}}(P, G))$ is in $O(1)$ time, $T(G, \Delta_1)$ consists of a *constant number* of nodes. Consider a graph H_1' that is obtained from H_1 by "bypassing" the nodes in $T(G, \Delta_1)$: for each node v_i in $T(G, \Delta_1)$, if $1 < i < 2n$, then add an edge (v_{i-1}, v_{i+1}) and leave out v_i along with edges adjacent to it, while updating only the "local" information in $s(v_{i-1})$ and $s(v_{i+1})$ such as parents and

Table II. Notations: cc, cs and ss Edges for IncSim

cc edges	edges (v', v) in ΔG such that $v' \in \mathsf{candt}(u')$ and $v \in \mathsf{candt}(u)$ for edge (u', u) in P
cs edges	edges (v', v) in ΔG such that $v' \in \mathsf{candt}(u')$ and $v \in \mathsf{match}(u)$ for edge (u', u) in P
ss edges	edges (v', v) in ΔG such that $v' \in \mathsf{match}(u')$ and $v \in \mathsf{match}(u)$ for edge (u', u) in P

children; and if i is 1 or $2n$, then remove v_i along with its edges while adjusting e_2 accordingly (e.g., e_2 is changed to (v_{2n}, v_2) if $i = 1$). Denote by H_1'' the graph $H_1' \oplus \Delta_2$. Note that H_1' is nonempty when n is sufficiently large. Then we have the following: (a) $\mathcal{A}(P, G, \Delta_2, M_{\mathsf{sim}}(P, G))$ and $\mathcal{A}(P, H_1', \Delta_2, M_{\mathsf{sim}}(P, G))$ should behave the same, since for all nodes v in H_1', $s(v)$ in G and $s(v)$ in H_1' are not different enough to trigger different actions of $\mathcal{A}$ on G and H_1'; but (b) $M_{\mathsf{sim}}(P, H_1'')$ is of size $O(n)$ while $M_{\mathsf{sim}}(P, H_2) = \emptyset$. Hence either $\mathcal{A}(P, H_1', \Delta_2, M_{\mathsf{sim}}(P, G))$ does not compute $M_{\mathsf{sim}}(P, H_1')$ correctly or the number of nodes in $T(G, \Delta_1)$ is not a constant. Both cases contradict the assumption that $\mathcal{A}$ is a bounded algorithm, that is, $\mathcal{A}$ is unbounded for unit updates. □

As an immediate result, IncSim is unbounded for batch updates and general patterns. In the rest of this section we show Theorem 5.1(2) for unit updates (Section 5.1) and Theorem 5.1(3) for batch updates (Section 5.2).

5.1. Incremental Simulation for Unit Updates

As optimal special cases of IncSim, we provide $O(|\mathsf{AFF}|)$-time incremental algorithms for: (a) unit deletions and general patterns, and (b) unit insertions and DAG patterns.

The algorithms in this section use $\mathsf{match}(\cdot)$ and $\mathsf{candt}(\cdot)$ as auxiliary structures, storing local information about whether a node and its child are a match or candidate of a pattern node (see Table I and Section 4). Note that the size of $\mathsf{match}(\cdot)$ and $\mathsf{candt}(\cdot)$ is bounded by $O(|P||G|)$, and P is typically much smaller than $|G|$. The affected area AFF is defined as the changed entries in these structures, which is not known in advance.

We keep track of three types of edges in ΔG, denoted by cc, cs, and ss edges (Table II). We will show that only certain type of these edges need to be processed by our incremental algorithms, that is, may change the match result. Note that these edges can be identified by looking up (a reversed index of) $\mathsf{match}(\cdot)$ and $\mathsf{candt}(\cdot)$ in constant time.

Unit deletions. The deletion of an edge from G may only *reduce* matches, that is, it leads to the removal of nodes and edges from the result graph G_r (Table I). Indeed, only deletion of ss edges can reduce G_r. Observe that the result graph G_r contains all the ss edges. It suffices to consider ss edges for edge deletions by the following result.

PROPOSITION 5.1. *Given a pattern P and a data graph G, only the deletions of* ss *edges for some pattern edge in G may reduce the matches of P in G.*

PROOF. Denote the result graph as G_r for a pattern P in G. The deletion of an edge $e = (v', v)$ in G has the following cases.

—Neither v' nor v is a match for any pattern node u'. Since v' is not a match of u', either v' does not satisfy $f_V(u')$ or there exists a child u of u', where none of the child of v' matches u. In either case, v' cannot match node u' after the removal of e.

—The node v' is in $\mathsf{match}(u')$, while the node v matches no pattern node. Since v' is a match for u', for any node u as a child of u', there exists a child $v_s \neq v$ of v' such that $v_s \in \mathsf{match}(u)$. Thus, the removal of e does not change G_r.

—The node $v \in \mathsf{match}(u)$ and the node v' matches no pattern node. Since graph simulation only considers the children of v, the removal of e does not change G_r.

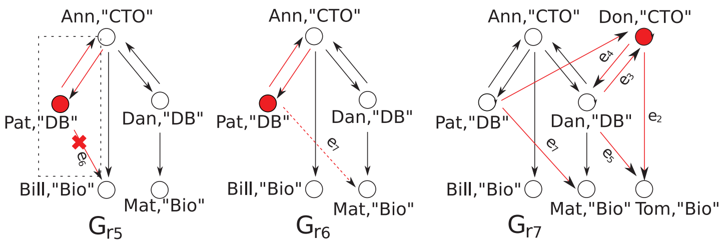


Fig. 7. IncSim in various updates.

—The edge e is an ss edge for a pattern edge (u', u). In this case, v' may no longer be a match for u' due to the removal of e. To see this, note that v may be the *only* child of v' that matches the child u of u'. If v' has another child v_s that matches u, the removal of e does not change G_r; otherwise, v' is no longer a match and should be removed (along with the attached edges) from G_r.

Thus, only the removal of ss edges may reduce G_r. □

Observe that when v' is no longer a match, the ancestors of v' that are matches appearing in G_r may also not remain matches, that is, we have to *propagate* the change in G_r given the single-edge deletion, as will be discussed later in this section.

Example 5.2. Consider the normal pattern P'_3 and data graph G_3 of Example 4.1. Observe that G_3 matches P'_3 via graph simulation, where the maximum match is {(CTO, Ann), (DB, Pat), (DB, Dan), (Bio, Bill), (Bio, Mat)}. The result graph G_{r5} is shown in Figure 7. Suppose that graph G_3 is updated by deleting e_6 = ((Pat, "DB"), (Bill, "Bio")), which is an ss edge for the pattern edge (DB, Bio) and is also in G_{r5}. When e_6 is removed, the node (Pat, "DB") is no longer a valid match for the pattern node DB, since there exists no edge from (Pat, "DB") to a node that can match the pattern node Bio.

Based on Proposition 5.1, we give an incremental algorithm for deleting an edge $e = (v', v)$, denoted by IncMatch$^-$ and shown in Figure 8. Intuitively, IncMatch$^-$ identifies the affected area caused by the deletion of ss edges, and "propagates" it by identifying and removing the nodes in G_r that are no longer matches due to the updates.

The algorithm first checks whether $e = (v', v)$ is an ss edge for a pattern edge. If not, the result graph G_r is unchanged (line 1). Otherwise IncMatch$^-$ propagates e to find all the matches v' that are no longer valid due to the removal of e, until the changed affected area AFF is identified, and G_r (Table I) is updated accordingly (lines 2–12).

More specifically, IncMatch$^-$ uses a stack eset (line 2) to store edges to be processed. For each pattern edge $e_p = (u', u)$ to which the ss edge e corresponds, it checks whether v' still has children to match u (lines 4–7). If not, then v' is removed from match(u') (Table I). If v' cannot match any pattern node, it is removed from G_r along with all the edges connected to it (lines 8–10). The removed edges (v'', v') are pushed into eset for further checking (line 9). If there is a pattern node that has no valid matches, then $G \setminus \{e\}$ no longer matches P, and G_r is empty (line 11). This process continues until all the "affected" nodes and edges are examined (lines 3–10).

Example 5.3. Recall P'_3 and G_{r5} from Example 5.2. When e_6 is removed, IncMatch$^-$ finds that no child of node Pat can match Bio. Thus Pat is no longer a match. The edge (Ann, Pat), an ss edge for (CTO,DB), is then checked. Since Ann has children Dan and Bill that match DB and Bio, respectively, IncMatch$^-$ updates G_{r5} by removing Pat and its three edges, which constitute AFF, as marked in Figure 7.

```
Input: A normal pattern $P$, the result graph $G_r = (V_r, E_r)$, match(·), candt(·),
       and an edge $e = (v', v)$ to be deleted from $G$.
Output: The updated result graph $G_r$.
1.  if $e = (v', v) \notin E_r$ then delete $e$ from $G$ and return $G_r$;
2.  stack eset := $\emptyset$;   eset.push($e$);
3.  while eset is not empty do
4.     edge $e$ := eset.pop();
5.     for each $e_p = (u', u)$ that $e = (v', v)$ can match do
6.        check if $v'$ can match $u'$;
7.        if $v'$ can not match $u'$ then
8.           for each $e' = (v'', v')$ in $E_r$ do
9.              $E_r := E_r \setminus \{e'\}$;   eset.push($e'$);
10.          $V_r := V_r \setminus \{v'\}$;   match($u'$) := match($u'$) $\setminus \{v'\}$;
11.          if match($u'$) = $\emptyset$ then return $\emptyset$;
12. return $G_r$.
```

Fig. 8. Algorithm IncMatch$^-$.

Correctness and complexity.

(1) Algorithm IncMatch$^-$ correctly updates the result graph G_r since it only removes nodes that are no longer valid matches and their edges from G_r. It terminates when all the invalid matches are removed.

(2) IncMatch$^-$ runs in $O(|\mathsf{AFF}|)$ time. Indeed, (a) IncMatch$^-$ only visits the nodes that can not match some pattern node in G_r and their parents at most *once*. If a node remains a valid match, IncMatch$^-$ stops the propagation (line 7); otherwise IncMatch$^-$ removes the node from match(·) and visits its 1-hop nodes in G_r, whose local information is changed. (b) IncMatch$^-$ determines if a match becomes invalid, by deriving and maintaining the number of the children of v' that are matches of u (not shown in Figure 8) in $O(|\mathsf{AFF}|)$ time. (i) The derivation of the local information is only conducted to the nodes with their local information changed (line 6), which takes in total $O(|\mathsf{AFF}|)$ time by using hashing techniques and linear time set operation. (ii) IncMatch$^-$ checks if the number becomes 0, and if so, v' is not a match of u'; it then propagates the changes of the local information, by reducing all such numbers associated to the parents of v' in G_r by 1, in $O(|\mathsf{AFF}|)$ time. (c) The number of updates to match(·) are bounded by the size of its change (which is in $O(|\mathsf{AFF}|)$), as match(·) is monotonically decreasing (line 10).

The algorithm and analysis given before complete the proof of Theorem 5.1 (2a).

Unit insertions. In contrast to edge deletions, inserting edges into s data graph G may only add new nodes and edges to the result graph G_r (Table I), but does not remove anything from it. One may verify that only cc edges and cs edges (Table II) may yield new matches when they are added to G. Indeed, one can verify the following.

PROPOSITION 5.2. *(1) For a* DAG *pattern P, only insertions of* cs *edges into a data graph G may increase matches of P. (2) For a general pattern P, only insertions of* cs *or* cc *edges into G may add new matches of P. (3) Moreover,* cc *edges alone only add new matches for pattern nodes in some strongly connected component (*SCC*) of P.*

PROOF. One may easily verify (1) by a case analysis on the types of the inserted edge, and prove by induction on the topological order of the DAG pattern P.

To see (2), one can verify that: (a) the insertion of an ss edge (v', v) (Table II) does not introduce new matches, and thus can be simply inserted into the result graph G_r;

(b) the insertion of an edge (v', v), where v' is a match and v is not, does not change G_r (Table I), since (i) v' is already a match, and (ii) whether v is a match only depends on whether its children become matches, and the insertion of (v', v) does not affect this; and (c) insertions of cs and cc edges (Table II) may result in new matches, for example, v' for pattern node u', since v (as a child of v') becomes a match of u (as a child of u').

To see (3), suppose that v' and v match u' and u, respectively, after the insertion of an cc edge $e = (v', v)$. Suppose that pattern edge (u', u) is not in any SCC of P. Then the insertion of e alone does not make v' a match for u', as v is a candidate, and the insertion of the cc edge does not make v a match because of (1) given earlier. Thus, the pattern edge (u', u) must already be in an SCC of P. □

Example 5.4. Consider again pattern P_3' and graph G_3 given in Example 5.3. Suppose that after the deletion of edge e_6, edge e_7 from Pat to Mat is inserted into G_3, which is a cs edge for the pattern edge (DB,Bio). This yields a new match Pat for the pattern node DB, and the new result graph G_{r6} is depicted in Figure 7.

Capitalizing on Proposition 5.2, shortly we propose incremental algorithms to process a single-edge insertion into general data graphs, denoted by $\mathsf{IncMatch}^+_{\mathsf{dag}}$ and $\mathsf{IncMatch}^+$, for DAG patterns and general patterns, respectively.

DAG *patterns.* $\mathsf{IncMatch}^+_{\mathsf{dag}}$ (not shown) identifies those nodes that yield a new match upon an edge insertion, and propagates the new matches until the entire AFF is found. When a cs edge (v', v) is inserted (Table II) , $\mathsf{IncMatch}^+_{\mathsf{dag}}$ checks whether each child of u' ($v' \in \mathsf{candt}(u')$) has a match as a child of v', and if so, v' becomes a match of u'. This may result in more new matches in the parents of v'. $\mathsf{IncMatch}^+_{\mathsf{dag}}$ propagates the new matches following a reversed depth-first strategy, until G_r can no longer be changed.

One can verify that $\mathsf{IncMatch}^+_{\mathsf{dag}}$ is correct and is in $O(|\mathsf{AFF}|)$ time, similar to its counterparts for $\mathsf{IncMatch}^-$. Indeed, $\mathsf{IncMatch}^+_{\mathsf{dag}}$ derives the following local information of v': does a child of a pattern node u' find no match in the children of v' ($v' \in \mathsf{candt}(u')$)? This can be derived from $\mathsf{match}(\cdot)$, $\mathsf{candt}(\cdot)$ and G_r in $O(|\mathsf{AFF}|)$ time. In addition, only the new matches and those nodes and edges within 1 hop of them in G_r are visited, at most once, which takes $O(|\mathsf{AFF}|)$ time. This completes the proof of Theorem 5.1 (2b).

General patterns. We present algorithm $\mathsf{IncMatch}^+$ in Figure 9. When it comes to cyclic graph patterns, it is more challenging to process edge insertions. Following Proposition 5.2, $\mathsf{IncMatch}^+$ first identifies $\mathsf{AFF}_{\mathsf{cs}}$ and $\mathsf{AFF}_{\mathsf{cc}}$, which include all the cc and cs edges (Table II) that may introduce new matches, respectively, when an edge e is inserted into G (lines 1–2). It then does the following.

(1) It invokes procedure propCS to find all new matches added by the insertion of cs edges (line 3). Note that new matches generated in this step reduce cc edges.
(2) It then uses procedure propCC to detect new matches formed in new SCCs in G consisting of all cc edges (line 4), which correspond to SCCs of P.
(3) Since new cs edges may be generated in step (2), $\mathsf{IncMatch}^+$ invokes propCS again to detect any new match (line 5). After these three phases no new match could be generated, and the updated result graph G_r (Table I) is returned (line 6).

We next present the procedures used by $\mathsf{IncMatch}^+$. (1) Similar to $\mathsf{IncMatch}^+_{\mathsf{dag}}$, procedure propCS (not shown) first identifies new matches added by $\mathsf{AFF}_{\mathsf{cs}}$, and then inductively checks their parents for propagation of the new matches. (2) Procedure propCC is given in Figure 9. It detects those new matches added only by cc edges, corresponding to SCCs in P. It first constructs a graph G_s for P, in which each node is an SCC (line 1). For each SCC node in G_s that contains at least a pattern edge, propCC checks whether

```
Input: A normal pattern $P$, $\mathsf{match}(\cdot)$, $\mathsf{candt}(\cdot)$, the result graph
        $G_r = (V_r, E_r)$, and an edge $e = (v', v)$ to be added to $G$.
Output: The updated result graph $G_r$.
1. $\mathsf{AFF}_{cs} := \{(v', v) \mid (v', v)$ is a $cs$ edge for a $(u', u) \in E_p\}$;
2. $\mathsf{AFF}_{cc} := \{(v', v) \mid (v', v)$ is a $cc$ edge for a $(u', u) \in E_p\}$;
3. $\mathsf{propCS}(\mathsf{AFF}_{cs}, \mathsf{AFF}_{cc}, P, G_r)$;
4. $\mathsf{propCC}(\mathsf{AFF}_{cs}, \mathsf{AFF}_{cc}, P, G_r)$;
5. $\mathsf{propCS}(\mathsf{AFF}_{cs}, \mathsf{AFF}_{cc}, P, G_r)$;
6. return $G_r$.

Procedure propCC
Input: Sets $\mathsf{AFF}_{cs}$, $\mathsf{AFF}_{cc}$, pattern $P$, and the result graph $G_r$.
Output: The updated result graph $G_r$, $\mathsf{AFF}_{cs}$ and $\mathsf{AFF}_{cc}$.
1. construct the SCC graph $G_s$ of $P$;
2. for each SCC $scc_i$ of $G_s$ do
3.     $\mathsf{AFF}_{cc_i} := \{(w', w) \mid (w', w)$ is a $cc$ edge for $(u', u)$ in $scc_i\}$;
4.     if $\mathsf{AFF}_{cc_i} \neq \emptyset$ then
5.         for each node $u \in scc_i$ do $\mathsf{match}'(u) := \mathsf{candt}(u)$;
6.         compute the matches for subgraph $scc_i$ in $\mathsf{AFF}_{cc_i}$;
7.         if $\mathsf{match}'(u) \neq \emptyset$ then update $G_r$, $\mathsf{AFF}_{cs}$ and $\mathsf{AFF}_{cc}$;
8. return $G_r$, $\mathsf{AFF}_{cs}$ and $\mathsf{AFF}_{cc}$;
```

Fig. 9. Algorithm IncMatch$^+$.

there exists a new match formed by the cc edges (lines 3–6). If new matches are found, G_r is updated by including the new nodes and edges (line 7). After each SCC in P is examined (lines 2–7), the updated G_r, AFF_{cs}, and AFF_{cc} are returned (line 8).

Correctness and complexity. IncMatch$^+$ correctly updates G_r because all the matches found are valid, since IncMatch$^+$ adds a new match v' to pattern node u' only if each child of u' has a match as a child of v'. For the complexity, note that: (1) procedure propCS is in $O(|\mathsf{AFF}|)$ time, similar to IncMatch$^+_{\mathsf{dag}}$; and (2) propCC is in $O(|P||\mathsf{AFF}| + |\mathsf{AFF}|^2)$ time, where for all SCC of P, it takes in total $O(|P||\mathsf{AFF}_{\mathsf{cc}}|)$ time to identify AFF_{cc_i} (lines 2–3 of propCC), $O(|\mathsf{AFF}_{\mathsf{cc}}|^2)$ time to find new matches (lines 5–7), and $|\mathsf{AFF}_{cc}| \leq |\mathsf{AFF}|$.

5.2. Incremental Simulation for Batch Updates

We next prove Theorem 5.1(3) by presenting IncMatch, an incremental simulation algorithm for general patterns and a list ΔG of edge deletions and insertions (batch updates). Its main idea is to: (1) remove redundant updates as much as possible, and (2) handle multiple updates simultaneously rather than one by one.

Algorithm IncMatch is shown in Figure 10, using the same auxiliary structures as remarked earlier. It first invokes procedure minDelta to reduce the list ΔG of updates (line 1). It then collects for each pattern edge e all its ss edges, and handles deletions first to identify invalid matches (lines 2–3). After the invalid matches are removed from G_r, it checks new matches formed in all the cs and cc edges, for each SCC of P (lines 4–5).

Reducing redundant updates. Procedure minDelta reduces ΔG. It first removes all updates e that do not inflict changes to the result, that is, those that are not an ss, cs or cc edge (Table II) for *any* pattern edge e_p (lines 1–6), by leveraging $\mathsf{match}(\cdot)$ and $\mathsf{candt}(\cdot)$. It then identifies and combines updates that "cancel" each other. These include, for each pattern edge $e_p = (u', u)$: (a) insertions and deletions of ss edges for $v' \in \mathsf{match}(u')$, and (b) insertions and deletions of cs edges for $v' \in \mathsf{candt}(u')$ (Table I) . Indeed, for the same pattern edge e_p, if ss edges (v', v_1) and (v', v_2) are inserted into G and deleted from G in (a), then v' remains a valid match of u'; similarly for (b). Such updates are removed from

```
Input: A normal pattern $P$, the result graph $G_r$, match$(\cdot)$, candt$(\cdot)$, and
       batch updates $\Delta G$.
Output: The updated result graph $G_r$.
1. minDelta$(\Delta G, P, \mathsf{match}(\cdot), \mathsf{candt}(\cdot))$;
2. for each pattern edge $e_p$ and its ss edges do
3.     iteratively identify and remove invalid matches; update $G_r$;
4. for each SCC in $P$ and related cc and cs edges do
5.     iteratively identify and add new matches; update $G_r$;
6. return $G_r$;

Procedure minDelta
Input: A normal pattern $P$, match$(\cdot)$, candt$(\cdot)$, updates $\Delta G$.
Output: The reduced $\Delta G$.
1. for each edge $e$ to be inserted do
2.     if there is no edge $e_p \in E_p$ for which $e$ is a cs or cc edge then
3.     $\Delta G := \Delta G \setminus \{e\}$;
4. for each edge $e$ to be deleted do
5.     if there is no edge $e_p \in E_p$ for which $e$ is an ss edge then
6.     $\Delta G := \Delta G \setminus \{e\}$;
7. for each $e_p \in E_p$ and its cs and ss edges do
8.     reduce $\Delta G$ via combination and cancellation;
9. return $\Delta G$;
```

Fig. 10. Algorithm IncMatch.

ΔG, including but not limited to those that insert and delete the same edge. Finally, it further combines the updates and induces a processing order, using a topological rank (see the following). Updates that involve the same data node are combined such that they are processed only once in minDelta and IncMatch (lines 7–8).

To improve the efficiency, minDelta employs a strategy to reduce redundant updates based on the notion of *topological ranks*, an extension of simulation ranks of Gentilini et al. [2003]. We consider a graph G_I induced by the matches, candidates, and the edges among them. The strongly connected component graph $\mathsf{G_{SCC}}$ of G_I is obtained by collapsing each strongly connected component SCC of G_I into a single node. Each node v of G_I is in an SCC node $[v]$ in $\mathsf{G_{SCC}}$. The rank $r(v)$ of a node v in G_I is computed as follows: (a) $r(v) = 0$ if $[v]$ contains a single node, and is a leaf node in $\mathsf{G_{SCC}}$; (b) $r(v) = \infty$ if $[v]$ reaches a nontrivial SCC (i.e., with at least 2 nodes); and (c) $r(v) = \max\{1 + r(v') \mid$ edge $([v], [v'])$ in $\mathsf{G_{SCC}}\}$ otherwise. We also define $r(e) = r(v)$ for an edge update $e = (v, v')$. Note that G_I and the topological ranks over G_I can be derived from match$(\cdot)$ and candt$(\cdot)$. Similarly, we define a topological rank over P with the SCC graph of P. The next lemma connects topological ranks and the simulation relation.

LEMMA 5.1. *In any pattern P and graph G, $r(u)$ in P is no greater than $r(v)$ in G_I if $(u, v) \in M_{\mathsf{sim}}(P, G)$.*

PROOF. We prove this by contradiction. Suppose that there exists a node pair $(u, v) \in M_{\mathsf{sim}}(P, G)$, where $r(u)$ in P is greater than $r(v)$ in G_I. Consider the following cases. (a) If $r(u) = 1$, then $r(v) = 0$, and hence u has at least one child as pattern node, while v has no child in G_I as either match or candidate. Thus, $(u, v) \notin M_{\mathsf{sim}}(P, G)$, contradicting our assumption that $(u, v) \in M_{\mathsf{sim}}(P, G)$. (b) Suppose that $r(u) = k$ and $r(v) = k - i$, where k and i are integers. By induction, one may verify that there is a descendant u' of u in P such that $r(u') = i$, where there is a descendant v' of v in G_I with $r(v') = 0$, and that $(u', v') \in M_{\mathsf{sim}}(P, G)$. This leads to a contradiction as in (a). (c) If $r(u)$ is ∞ and $r(v)$ is an integer, then u must reach a nontrivial SCC in P with at least a cycle, while v does not reach any SCC in G_I. This leads to a contradiction as in (a). □

By Lemma 5.1, minDelta uses topological ranks to remove redundant updates. Given a set of updates ΔG (after the initial process of minDelta, lines 1–6), it does the following:

(1) dynamically maintain the topological ranks for each edge $e = (v, v') \in \Delta G$ in G_I, and sort ΔG based on the updated rank $r(e)$;
(2) group the updates $e \in \Delta G$ with the same source node v together into a set, and identify redundant $e = (v, v')$ by checking if any of the following holds for e:
  (a) e is an insertion, and there exists no pattern edge e_p with a lower rank than e;
  (b) e is an insertion, and for each pattern edge $(u, u') \in P$ such that $v \sim u$, v has no edge to nodes in match(u');
  (c) e is a deletion, and there exists no pattern edge e_p with a higher rank than e; or
  (d) e is a deletion, and there exists an edge (v, v_1) in G such that for any pattern edge (u, u') with $v \sim u$, if $v' \sim u'$ then $v_1 \sim u'$ and v_1 does not appear in ΔG;
(3) return ΔG after removing all the redundant updates identified in (2).

Example 5.5. Consider again P_3' and G_3 of Figure 4. Consider batch updates ΔG, which insert edges $e_1, e_2, e_3, e_4, e_5, e_7$ and delete e_6, where e_6 and e_7 are given in Examples 5.2 and 5.4, respectively. The result graph is depicted as G_{r7} in Figure 7. Given these, algorithm IncMatch first invokes minDelta to reduce ΔG: (1) the insertions of e_1 and e_5 are removed from ΔG or simply conducted to G_{r7} as they do not yield increment to matches; and (2) the deletion of e_6 and the insertion of e_7 cancel each other as they are both ss edges of the pattern edge (DB, Bio) for node Pat, which remains an unaffected match. After minDelta, ΔG contains the insertion of edges e_2, e_3, e_4.

IncMatch then identifies the new match (Don, "CTO") generated by the insertion of cs edges e_2, e_3, and e_4, and includes it in G_{r7}. Observe that: (1) the affected area AFF in G_{r7} includes the new node (Don, "CTO"), the newly inserted and deleted edges, and the edges attached to (Don, "CTO") from other matches in G_{r7}, and (2) node (Pat, "DB") remains a match, although it is affected twice by the deletion of e_6 and the insertion of e_7 (as discussed in Examples 5.2 and 5.4). IncMatch avoids the unnecessary recomputation by canceling these updates via minDelta, rather than processing them one by one.

Correctness and complexity. To see that IncMatch is correct, note the following: (1) minDelta removes only those updates that have no impact on the final match; and (2) IncMatch handles updates along the same line as in IncMatch$^-$ and IncMatch$^+$, which are shown correct earlier. The overall complexity of minDelta is in $O(|\mathsf{AFF}| + |P||\Delta G|)$. To see this: (a) it takes |AFF| time to update the topological rank, via a topological sort as in Kahn [1962], (b) it takes $O(|P||\Delta G|)$ time to identify redundant updates in ΔG. From the complexity of minDelta, IncMatch$^-$, and IncMatch$^+$, it follows that IncMatch is in $O(|\Delta G|(|P||\mathsf{AFF}| + |\mathsf{AFF}|^2))$ time. Hence IncSim is semibounded for batch updates and general patterns. Again ΔG and P are typically small in practice.

This completes the proof of Theorem 5.1.

6. INCREMENTAL BOUNDED SIMULATION MATCHING

In this section we study the incremental bounded simulation problem, referred to as IncBSim. We first present the complexity (boundedness) result (Section 6.1). To show the result, we employ landmark and distance vectors as auxiliary structures, to efficiently compute and maintain the local distance information (Section 6.2). Based on the notion, we develop an incremental algorithm for graph pattern matching defined in terms of bounded simulation (Section 6.3). Finally, we provide complexity bounds and algorithms for incrementally maintaining those vectors (Section 6.4).

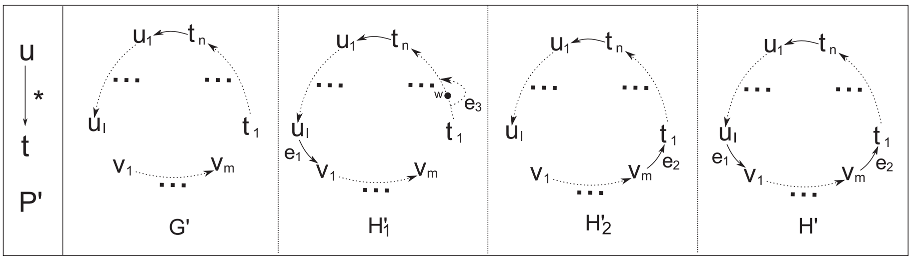


Fig. 11. Unboundedness of IncBSim.

6.1. Incremental Bounded Simulation Problem

The *incremental bounded simulation problem* IncBSim is a generalization of the incremental simulation problem IncSim. It takes as input a b-pattern P, a data graph G, changes ΔG to G, and a result graph G_r that depicts the unique maximum bounded simulation $M_{\mathsf{ksim}}(P, G)$. It computes *the changes* to G_r, which represents ΔM such that $M_{\mathsf{ksim}}(P, G \oplus \Delta G) = M_{\mathsf{ksim}}(P, G) \oplus \Delta M$. The main results of this section are as follows.

THEOREM 6.1. *The incremental bounded simulation problem is:*

(1) *unbounded even for unit updates and path patterns; and*

(2) *semibounded, in* $O(|\Delta G|(|P||\mathsf{AFF}| + |\mathsf{AFF}|^2))$ *time for batch updates and general patterns.*

We first prove Theorem 6.1(1), and then show Theorem 6.1(2) in Section 6.3.

PROOF OF THEOREM 6.1(1). We show that IncBSim is unbounded even for a single-edge insertion and a pattern with a single edge. Consider an instance of IncBSim shown in Figure 11, consisting of a pattern P' and a data graph G'. The pattern P' has a single edge connecting pattern nodes u and t, labeled with $*$. The graph G' consists of paths $(u_1, \ldots, u_l)$, $(v_1, \ldots, v_m)$ and $(t_1, \ldots, t_n)$, as well as edge (t_n, u_1). The node labels are shown in Figure 11, for example, u in P' and u_i in G' ($i \in [1, l]$) are labeled with u. One may verify that there exists a match for P' in G' based on bounded simulation if and only if there exists a path in G' from a node labeled with u to a node labeled with t.

Denote by Δ_1 the insertion of edge $e_1 = (u_l, v_1)$, and by Δ_2 the insertion of $e_2 = (v_m, t_1)$. The updated graphs $H'_1 = G' \oplus \Delta_1$, $H'_2 = G' \oplus \Delta_2$, and $H' = H'_1 \oplus \Delta_2$ are shown in Figure 11. Observe the following.

—$M_{\mathsf{ksim}}(P', H'_1) = M_{\mathsf{ksim}}(P', H'_2) = M_{\mathsf{ksim}}(P', G') = \emptyset$; but
—$M_{\mathsf{ksim}}(P', H') = \{(u, u_i), (t, t_j) \mid i \in [1, l], j \in [1, n]\}$.

We next show that no bounded incremental algorithm $\mathcal{A}$ can compute the updates, that is, IncBSim is unbounded. Recall that $\mathcal{A}$ is bounded if it computes ΔM with time cost as a function of |CHANGED| (Table I). Here |CHANGED| depends on $|\Delta M|$ alone, as $|\Delta G|$ is 1. In terms of result graphs, $|\Delta M|$ after either Δ_1 or Δ_2 alone is 0, while $|\Delta M|$ after Δ_1 and Δ_2 is $n * l + n + l$, which corresponds to a result graph that contains all the nodes labeled with u (i.e., u nodes) and t in H', and edges from each u node to t node.

Assume by contradiction that there exists such a bounded incremental algorithm $\mathcal{A}$ that given P', graph G', a unit update $\Delta G'$ and the old output $M_{\mathsf{ksim}}(P', G')$, $\mathcal{A}(P', G', \Delta G', M_{\mathsf{ksim}}(P', G'))$ computes ΔM. Thus, $\mathcal{A}(P', G', \Delta_1, M_{\mathsf{ksim}}(P', G'))$ and $\mathcal{A}(P', G', \Delta_2, M_{\mathsf{ksim}}(P', G'))$ are both in $O(1)$ time, since for unit updates Δ_1 and Δ_2, the changes from $M_{\mathsf{ksim}}(P', G')$ to the new results $M_{\mathsf{ksim}}(P', H'_1)$ and $M_{\mathsf{ksim}}(P', H'_2)$

are empty in both cases. As in the proof of Theorem 5.1, we introduce the status $s(v)$ associated with each node v in a graph G', as well as the *trace* $T(G', \Delta G')$ of $\mathcal{A}(P', G', \Delta G', M_{\mathsf{ksim}}(P', G'))$. We next show that such an algorithm $\mathcal{A}$ does not exist.

(1) There exist nodes w in $T(G', \Delta_1)$ such that $\mathcal{A}(P', G', \Delta_1, M_{\mathsf{ksim}}(P', G'))$ changes their status. Observe that $T(G', \Delta_2)$ and $T(H_1', \Delta_2)$ generate different outputs. The different behaviors of $\mathcal{A}$ on G' and H_1' when processing Δ_2 can only be triggered if $s(w)$ differs in G' and H_1' for some nodes w. Since graph H_1', as an input of $\mathcal{A}$, differs from G' only due to Δ_1 that inserts edge e_1, $\mathcal{A}(P', G', \Delta_1, M_{\mathsf{ksim}}(P', G'))$ must visit nodes w and change their status $s(w)$ during its update process $T(G', \Delta_1)$.

(2) Algorithm $\mathcal{A}$ is unbounded. Since $\mathcal{A}(P', G', \Delta_1, M_{\mathsf{ksim}}(P', G'))$ is in $O(1)$ time, $T(G', \Delta_1)$ consists of a constant number of nodes. Consider a graph H_1'' constructed from H_1' by "bypassing" those nodes in $T(G', \Delta_1)$: for each node x_v in $T(G', \Delta_1)$: (a) if x_v is (i) the node u_i for some $i \in [1, l]$, (ii) the node v_j for $j \in [2, m-1]$, or (iii) the node t_k for $k \in [2, n]$, then we add an edge (v_p, v_c), where v_p and v_c are the parent and child of x_v in G', respectively, and leave out x_v along with edges adjacent to it, while updating only the "local" information in $s(v_p)$ and $s(v_c)$ such as parents and children; and (b) if x_v is v_1, v_m or t_1, then remove x_v along with its edges while adjusting the edge e_1 and e_2 accordingly (e.g., e_1 is changed to (u_l, v_2), and e_2 is changed to (v_{m-1}, t_1)). Denote by H_s'' the graph $H_1'' \oplus \Delta_2$. Observe the following: (a) $\mathcal{A}(P', G', \Delta_2, M_{\mathsf{ksim}}(P', G'))$ and $\mathcal{A}(P', H_1'', \Delta_2, M_{\mathsf{ksim}}(P', G'))$ should behave the same, since for all nodes x_v in H_1'', $s(x_v)$ in G and $s(x_v)$ in H_1'' are the same; but (b) $M_{\mathsf{ksim}}(P', H_s'')$ is of size $O(l * n)$, while $M_{\mathsf{ksim}}(P', H_2')$ is $\emptyset$, the same as $M_{\mathsf{ksim}}(P', G')$ as remarked earlier. Hence either $\mathcal{A}(P', H_1'', \Delta_2, M_{\mathsf{ksim}}(P', G'))$ does not compute $M_{\mathsf{ksim}}(P', H_s'')$ correctly or the number of nodes in $T(G', \Delta_1)$ is not a constant. Both cases contradict that $\mathcal{A}$ is bounded. □

We next prove Theorem 6.1(2) by developing an incremental algorithm for IncBSim. In contrast to the incremental algorithms of Fan et al. [2010a] that only work on DAG *patterns*, our algorithm is able to handle possibly *cyclic* patterns, and is semibounded in $|\mathsf{AFF}|$, $|P|$ and $|\Delta G|$. The algorithm employs landmark vectors [Potamias et al. 2009] and distance vectors as auxiliary structures, which are presented next.

6.2. Landmark Vectors

As remarked in Section 4, in addition to match(·) and candt(·), the local information for bounded simulation includes distance information about the matches and candidates of pattern nodes in P, to cope with the length constraints posed on the pattern edges. One way to encode the distance information is to maintain an all-pair distance matrix as the auxiliary structure [Fan et al. 2010a], which always takes $O(|V|^2)$ space. In this work we introduce landmark and distance vectors as more "compact" auxiliary structures.

(1) A *landmark vector* $\mathsf{lm} = <v_1, \ldots, v_{|\mathsf{lm}|}>$ for a data graph G is a list of nodes in G such that for each pair (v'', v') of nodes in G, there exists a node in lm that is on a shortest path from v'' to v', that is, lm "covers" all-pair shortest distances [Potamias et al. 2009].

(2) In addition, with each node v in G we associate two *distance vectors*, each of size $|\mathsf{lm}|$: $\mathsf{distv}_f = <\mathsf{dis}(v, v_1), \ldots, \mathsf{dis}(v, v_{|\mathsf{lm}|})>$, and $\mathsf{distv}_t = <\mathsf{dis}(v_1, v), \ldots, \mathsf{dis}(v_{|\mathsf{lm}|}, v)>$.

As observed in Potamias et al. [2009], we can use a landmark vector and distance vectors to find the distance between any pair of nodes in G as follows. The distance $\mathsf{dis}(v'', v')$ from node v'' to v' in G is the minimum of the sums of $\mathsf{distv}_f[i]$ of v'' and $\mathsf{distv}_t[i]$ of v' for all $i \in [1, |\mathsf{lm}|]$. It can be found by a *distance query*, denoted as $\mathsf{dist}(v'', v', \mathsf{lm})$, with at most $|\mathsf{lm}|$ operations. In practice $|\mathsf{lm}|$ is typically small and can even be taken as a constant [Potamias et al. 2009].

Table III. Notations: cc, cs, ss Pairs for IncBSim

cc pair	node pairs (v', v), where $v' \in \mathsf{candt}(u')$, $v \in \mathsf{candt}(u)$ for edge (u', u) in P
cs pair	node pairs (v', v), where $v' \in \mathsf{candt}(u')$, $v \in \mathsf{match}(u)$ for edge (u', u) in P
ss pair	node pairs (v', v) where (1) $v' \in \mathsf{match}(u')$, $v \in \mathsf{match}(u)$ for edge (u', u) in P; and (2) $\mathsf{dis}(v', v) \leq k$ if $f_E(u', u) = k$, and $0 < \mathsf{dis}(v', v)$ otherwise

Selection of landmarks. There are multiple landmark vectors for a graph G. For example, any *vertex cover* V_c of G can be considered as a landmark vector. Indeed, since V_c is a vertex cover, for any edge $e = (v_1, v_2)$ in G, v_1 or v_2 is in V_c. Thus, for any two nodes v' and v and any shortest path ρ from v' to v, there is a node $v'' \in V_c$ that is on some edge $e \in \rho$. In our experimental study, we compute a minimum vertex cover as a landmark vector using heuristic algorithm (see Section 8).

One may also want to use a "high-quality" landmark vector lm, with a small number of nodes that are not changed frequently when G is updated. In this context, a set of landmarks can also be selected as the nodes with, for example, larger degrees, attached edges that are less frequently updated [Kumar et al. 2006], or larger betweenness centrality [White and Smyth 2003], a normalized measurement for the number of shortest paths in G that go through the node v. Intuitively, the selection favors the smaller and more stable lm. We illustrate this using an example, but defer a full treatment of such landmark vectors to a future publication, due to the space constraint.

Example 6.2. Recall graph G_3 of Example 4.1. A landmark vector lm for G_3 is <(Ann, "CTO"), (Dan, "DB"), (Pat, "DB"), (Ross, "Med")>. Here distv_f of Dan is $<1, 0, 2, \infty>$, and distv_t of Bill is $<1, 2, 1, \infty>$. From these we find 2 as the distance from Dan to Bill.

Suppose that Ann frequently updates her contacts, that is, frq(Ann) is high, while Bill seldom updates his contacts. Although the degree and betweenness of Ann) are large, Bill is a better choice for a landmark, since he is more stable than Ann. Thus a better landmark vector is <(Bill, "Bio"), (Dan, "DB"), (Pat, "DB"), (Ross, "Med")>.

We will study how to incrementally maintain the landmark and distance vectors to Section 6.4. Next we assume that the landmark and distance vectors are available, and develop incremental algorithms for IncBSim by using the vectors.

6.3. Incremental Matching for Bounded Simulation

Based on landmark vectors, we develop incremental algorithms for IncBSim. The algorithm uses match(·) and candt(·), as for incremental simulation (Section 5). In addition, it uses landmark and distance vectors as auxiliary structures to encode the local distances (Section 4) between node pairs that are matches or candidates of pattern nodes in P. The affected area AFF (Table I) is defined as the changed entries in these structures. For example, it includes changes to the connectivity and distance information, represented by the updated entries in the landmark and distance vectors. We defer the details of AFF to Section 6.4, where changes to the vectors are elaborated.

The change of the local distance information may affect a pair of nodes from match(·) or candt(·) that are not connected. Thus, instead of cc, cs, and ss edges (as for IncSim), we keep track of three types of node pairs, given in Table III. A pair (v', v) of nodes from match(·) or candt(·) is: (a) a cs (respectively, cc) pair if $v' \in \mathsf{candt}(u')$ and $v \in \mathsf{match}(u)$ (respectively, $v \in \mathsf{candt}(u)$) for edge (u', u) in P; and (b) an ss pair if $v' \in \mathsf{match}(u')$ and $v \in \mathsf{match}(u)$ for edge (u', u) in P, and $\mathsf{dis}(v', v) \leq k$ if $f_E(u', u) = k$, and $0 < \mathsf{dis}(v', v)$ otherwise.

One may verify the following result for incremental bounded simulation.

```
Input: A $b$-pattern $P$, landmark vector $\mathsf{lm}$,
       the result graph $G_r$, and single insertion $e$.
Output: The updated result graph $G_r$.
1. $\mathsf{lm}' := \mathsf{InsLM}(P, e, \mathsf{lm})$;
2. identify all cc and cs pairs for each $e_p$ of $P$;
3. for each SCC in $P$ and related cc and cs pairs do
4.   iteratively identify and add new matches; update $G_r$;
5. return $G_r$;
```

Fig. 12. Algorithm IncBMatch$^+$.

PROPOSITION 6.1. *Given a b-pattern P and a graph G, (1) $P \trianglelefteq_{\mathsf{bsim}} G$ if and only if $P \trianglelefteq_{\mathsf{sim}} G_r$ (P is treated as a normal pattern), where G_r is the result graph for $M_{\mathsf{ksim}}(P, G)$; and (2) $M_{\mathsf{ksim}}(P, G)$ can be increased (respectively, reduced) only by those* cs *and* cc *(respectively,* ss*) pairs with updated distance satisfying (respectively, not satisfying) the bound for a pattern edge.*

PROOF. (1) First assume that $P \trianglelefteq_{\mathsf{bsim}} G$ (bounded simulation). Then for any $(u, v) \in M_{\mathsf{ksim}}(P, G)$ and any child u' of u in P, there exist a node v' in G such that $(u', v') \in G$, and a path from v to v' that satisfies the bound on the pattern edge (u, u'). Observe that for all such v and v', (v, v') is also an edge in G_r (Table I). Thus, $M_{\mathsf{ksim}}(P, G)$ is a match for P in G_r based on simulation. Conversely, if G_r matches P based on graph simulation, one can verify that $P \trianglelefteq_{\mathsf{bsim}} G$ similarly, since G_r is the result graph of $M_{\mathsf{ksim}}(P, G)$.

(2) This can be verified by a case study on the updates of the node pairs, along the same lines as the proof of Proposition 5.1 (see Section 5). □

Proposition 6.1 reduces bounded simulation in a data graph to simulation in the result graph G_r. It suggests a two-step strategy for IncBSim: (1) identify all the cc, cs, and ss pairs (Table III) via auxiliary structures; and (2) find changes ΔM, by treating cc and cs pairs (respectively, ss pairs) as insertions of the edges to G_r (respectively, deletions from G_r).

Next we start with unit updates. We will then study batch updates.

Singl-edge insertions. An algorithm to handle a single-edge insertion is given in Figure 12, denoted as IncBMatch$^+$. Intuitively, it determines the affected areas for simulation in G_r, and propagates the changes. It first invokes procedure InsLM to identify all the cc and cs pairs (lines 1–2), to check simulation in G_r. By Proposition 6.1, these pairs are insertions to the result graph G_r. Hence the algorithm finds new matches by updating G_r (lines 3–4), along the same lines as algorithm IncMatch$^+$ (see Section 5.1).

Procedure InsLM updates landmarks as well as the distance vectors of the nodes in G, when an edge $e = (v', v)$ is inserted. The details will be discussed in Section 6.4.

Example 6.3. Consider pattern P_3 and graph G_3 of Example 4.1. Given a landmark vector for G_3 <(Ann, "CTO"), (Dan, "DB"), (Pat, "DB"), (Ross, "Med")>. The table that follows shows the distance vectors of the nodes Don, Dan, Pat, and Tom.

V **in** G_3	distv$_f$	distv$_t$	V **in** G_3	distv$_f$	distv$_t$
(Don, "CTO")	$<\infty, \infty, \infty, \infty>$	$<2, 3, 1, \infty>$	(Pat, "DB")	$<1, 2, 0, \infty>$	$<1, 2, 0, \infty>$
(Dan, "DB")	$<1, 0, 2, \infty>$	$<1, 0, 2, \infty>$	(Tom, "Bio")	$<2, 1, 3, \infty>$	$<\infty, \infty, \infty, \infty>$

When edge e_2 is inserted into G_3, the process of InsLM is illustrated in Figure 13. It first identifies nodes Don, Pat, Ann, and Dan, from which the distances *to* Tom are changed. It inserts Don into lm as a new landmark, and updates distance vectors distv$_f$ accordingly. Similarly, it finds nodes whose distances *from* Don are changed, and

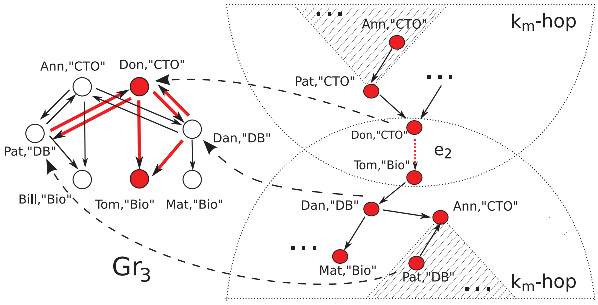


Fig. 13. Incremental bounded simulation.

updates the distance vectors distv$_t$. The new distv$_f$ of (Don, "CTO") is $<3, 2, 4, \infty, 0>$, and distv$_t$ of (Dan, "DB") is $<1, 0, 2, \infty, 2>$. The new distance from Don to Dan is 2. After this, we have the following.

V **in** G_3	distv$_f$	distv$_t$	V **in** G_3	distv$_f$	distv$_t$
(Don, "CTO")	$<3, 2, 4, \infty, 0>$	$<2, 3, 1, \infty, 0>$	(Pat, "DB")	$<1, 2, 0, \infty, 1>$	$<1, 2, 0, \infty, 4>$
(Dan, "DB")	$<1, 0, 2, \infty, 3>$	$<1, 0, 2, \infty, 2>$	(Tom, "Bio")	$<2, 1, 3, \infty, 4>$	$<3, 4, 4, \infty, 1>$

IncBMatch$^+$ then incrementally finds new matches by operating on result graph G_{r1} of Figure 5, via simulation. It finds new cc and cs pairs, such as (Don, Tom), (Don, Dan), and (Don, Pat), and inserts them as edges into G_{r1}. This yields new result G_{r3} of Figure 13.

Single-edge deletions. Similarly, when an edge $e = (v', v)$ is deleted, we first identify node pairs (v_1, v_2) for which: (1) dis(v_1, v) *or* dis(v', v_2) is changed, and (2) v_1 and v_2 are within k_m hops of v and v', respectively, where k_m is the maximum (finite) bound on a pattern edge in P. For each such pair (v_1, v_2), we: (1) compute the distance from v_1 to v_2 following a new shortest path between them, (2) select and add a new landmark on a shortest path from v_1 to v_2 to lm, and (3) extend the distance vectors distv$_f$ of v_1 and distv$_t$ of v_2 with the new distances from and to the landmark, respectively. We finally collect ss pairs following Proposition 6.1, and treat these node pairs as edges to be deleted from the result graph G_r. The invalid matches are removed as in IncMatch$^-$ (see Section 5.1), and changes to the match result ΔM are identified. The landmark and distance vectors are maintained by a procedure DelLM to be given in Section 6.4.

Batch updates. For batch updates ΔG, we do the following. (1) We adopt a variant of a dynamic fixed point algorithm [Ramalingam and Reps 1996a], denoted as IncLM (see Section 6.4), to identify all the node pairs (v_1, v_2) for which: (a) dis(v_1, v_2) is changed, and (b) v_1 and v_2 are within k_m hops of the nodes in the edge of ΔG inserted or deleted; here k_m is given as earlier. (2) We collect all ss, cs, and cc pairs (Table III) from those pairs examined in (1) that have new distances satisfying the condition specified in Proposition 6.1. We then find changes ΔM to the matches by removing redundant updates as for simulation (Section 5.2), and incrementally compute simulation of P in G_r, as in algorithm IncMatch that handles batch updates for simulation.

Correctness and complexity. The correctness of the incremental algorithms for IncBSim is assured by Proposition 6.1. For the complexity, it takes, for a single update: (a) $O(|P| + |\mathsf{AFF}| \log |\mathsf{AFF}| + |\mathsf{AFF}|^2)$ time to maintain the landmark and distance vectors (see Section 6.4), and (b) $O(|P||\mathsf{AFF}| + |\mathsf{AFF}|^2)$ time to update the matches as incremental simulation in G_r, following an analysis as in Section 5. Thus, the total time complexity of IncBSim is $O(|\Delta G|(|\mathsf{AFF}|\log|\mathsf{AFF}| + |P||\mathsf{AFF}| + |\mathsf{AFF}|^2))$. This verifies Theorem 6.1(2).

6.4. Incremental Maintenance of Landmarks

For IncBSim, we need to incrementally maintain landmark vectors when the data graph is updated, to keep track of the distance changes in the data graph. Shortly we study incremental techniques to maintain landmark and distance vectors. More specifically, for a data graph G, we study the following: *the incremental landmark problem*, to maintain a landmark vector and *the incremental landmark and distance problem*, to maintain a landmark vector as well as the distance vectors for IncBSim.

Maintaining landmarks. The incremental landmark problem, denoted as IncLMK, takes as input a graph G, a landmark vector lm, and batch updates ΔG. It is to find an updated landmark vector lm' for $G \oplus \Delta G$. Here $|\mathsf{CHANGED}| = |\Delta G| + |\Delta \mathsf{lm}|$, where $|\Delta \mathsf{lm}|$ is the size of different entries between the original and updated lm. We show that IncLMK is bounded and can be solved in linear time of $|\mathsf{CHANGED}|$.

Proposition 6.2. IncLMK *is bounded for batch updates in* $O(|\mathsf{CHANGED}|)$ *time.*

Proof. We first show that for single-edge insertions, the problem is bounded, by providing a bounded algorithm as follows. Given an edge (v', v) to be inserted into G, the algorithm checks whether v' or v is already in the landmark vector lm. If none of them is in lm, it simply inserts either v' or v into lm; otherwise lm remains unchanged.

The algorithm correctly maintains lm, because: (a) edge insertions only cause new nodes to be added into lm, (b) adding v' or v to lm covers *all* the node pairs with their distance changed, and (c) if lm is a landmark vector, then $\mathsf{lm} \cup \{v'\}$ is a landmark vector, for any node v' of G. The algorithm can be implemented in $O(1)$ time (via, e.g., hashing).

For single-edge deletions, one can verify that if lm is already a landmark vector of G, then it remains a landmark vector for $G \setminus \{(v', v)\}$, where (v', v) is the edge to be deleted. Thus, there is no need to change lm in response to the deletion, and the algorithm simply removes the edge from G, which is in $O(1)$ time.

For batch updates ΔG, one can invoke the two algorithms as given before, one for each update in ΔG. The algorithm is in time $O(|\mathsf{CHANGED}|)$. Hence the problem is bounded. □

Incremental landmark and distance problem. Given P, G, a landmark vector lm, and batch updates ΔG, the incremental problem, denoted as IncLMDK, is to maintain a landmark vector as well as the distance vectors in response to ΔG.

Next we develop techniques for IncLMDK, to incrementally compute bounded simulation (IncBSim). IncLMDK maintains both a landmark vector and distance vectors as auxiliary structures for IncBSim. It needs to change those landmarks that affect matches, while leaving the rest to be adapted offline based on a “lazy” strategy. Here $|\mathsf{CHANGED}|$ is $|\Delta G| + |\Delta \mathsf{lm}| + |\Delta \mathsf{distv}|$, where $|\Delta G|$ and $|\Delta \mathsf{lm}|$ are the same as for IncLMK, and $|\Delta \mathsf{distv}|$ is the size of the changed entries in the distance vectors.

To identify $|\mathsf{CHANGED}|$, we employ the necessary local information (see Section 4) of a node u as: (a) the distances from u to the landmarks, and the distances from the landmarks to u, and (b) the distances of (a) for the neighborhood of u. One may verify that the changes in lm and distv cannot be incrementally detected without traversing irrelevant part of G if any information in (a) or (b) is missing for a node u. Auxiliary structures lm and distv encode the distance information in (a) and (b). In addition, the neighborhood of u can be derived from its distance vectors, by extracting the nodes with distance 1 in $\mathsf{distv}_f(u)$ (respectively, $\mathsf{distv}_t(\cdot)$) as its parents (respectively, children). The

```
Procedure DelLM
Input: A $b$-pattern $P = (V_p, E_p, f_V, f_E)$, edge $e = (v', v)$ deleted,
        landmark vector lm.
Output: Landmark vector lm$'$ as the updated lm.
1.  $k_m := \max(f_E(e_p))$ for all $e_p \in E_p$;
    stack vset := $\{v'\}$; lm$'$ := lm;
2.  if $v'$ has no child then lm$'$ := lm$'$ $\cup \{v'\}$;
3.  while vset $\neq \emptyset$ do
4.    Boolean flag := false;
5.    node $u$:=vset.pop(); affUP := affUP $\cup \{u\}$;
6.    for each node $u'$ as parent of $u$ with dist$(u', v, \mathsf{lm})$ = 1+ dist$(u, v, \mathsf{lm})$ do
7.       for each node $u''$ as child of $u'$ with dist$(u', v, \mathsf{lm})$ = 1+ dist$(u'', v, \mathsf{lm})$ do
8.          if $u'' \notin$ affUP then flag := true; break ;
9.       if flag = false and $u'$ is within $k_m$ hops of $v'$ then vset.push $(u')$;
10. compute affDW similarly;
11. for each node $v_{\mathsf{AFF}} \in$ affUP do
12.   for each node $v_{\mathsf{lm}} \in$ lm$'$ do $v_{\mathsf{AFF}}$.distv$_f[v_{\mathsf{lm}}]$ := dis$(v_{\mathsf{AFF}}, v_{\mathsf{lm}})$;
13. update $v_{\mathsf{AFF}}$.distv$_t[v_{\mathsf{lm}}]$ similarly for $v_{\mathsf{AFF}} \in$ affDW and $v_{\mathsf{lm}} \in$ lm$'$;
14. return lm$'$;
```

Fig. 14. Procedure DelLM.

affected area AFF for IncLMDK consists of nodes and edges with their local information changed.

Note that AFF for the incremental matching algorithm IncBSim (Section 6.3) is the same as AFF in IncLMDK. The total size of landmark and distance vectors is bounded by $O(|G||\mathsf{lm}|)$, which is much smaller than an all-pair distance matrix adopted by Fan et al. [2010a], in particular when |lm| is treated as a constant in practice [Potamias et al. 2009]. Moreover, $|\mathsf{AFF}| = |\Delta\mathsf{lm}| + |\Delta\mathsf{distv}|$ is much smaller than lm and distv.

The distance vectors are updated once lm is updated, using a *lazy strategy* as follows. (a) We maintain lm in response to ΔG, by keeping track of node pairs that lm covers. We add a landmark only when necessary, and only extend the distance vectors of those node pairs with changed distances; and (b) we rebuild space-efficient landmark vectors periodically via an offline process when, for example, |lm| approaches the number of nodes in G.

PROPOSITION 6.3. IncLMDK *is in* $O(|P| + |\mathsf{AFF}| \log |\mathsf{AFF}| + |\mathsf{AFF}|^2)$ *time, that is, semibounded, for batch updates.*

We prove this by presenting semibounded algorithms to maintain landmark vectors and distance vectors, for single-edge deletions (Procedure DelLM), single-edge insertions (Procedure InsLM), and batch updates (Procedure IncLM).

Single-edge deletions. Procedure DelLM is given in Figure 14. It updates lm in response to a single-edge deletion $e = (v', v)$. Given e, DelLM first initializes two sets affUP and affDW, to store the nodes with distance to v and from v' changed, respectively; it also initializes vector lm$'$ as lm, and a stack vset with v' (line 1). DelLM also updates lm by adding those nodes v' without any child (line 2). It then computes affUP (lines 3–9). More specifically, it first initializes a Boolean flag to be false, and selects a node u from the stack vset and adds it to affUP (line 5). It identifies the parents u' of u (by checking distv), where their *old* distance to v may be affected by the removal of e (line 6). For each such parent u', it then checks if there is a child u'' of u' that is: (a) not in the set affUP, and (b) the original distance from u to v is not changed. If there is no such u'', u is inserted into affUP, and is pushed to the stack vset. The process stops when vset is empty. The set affDW is similarly computed (line 10). Note that DelLM only inspects those nodes that have changed entries and are within k_m hops of the deleted edge.

After the sets affUP and affDW are computed, procedure DelLM updates the distance vectors for the affected nodes (lines 11–13). For each affected node $v_{\mathsf{AFF}} \in$ affUP and each landmark v_{lm}, it updates the distance vector distv_f of v_{AFF} with the new distance (lines 11–12). Similarly, it updates the distance vectors of the nodes in affDW (line 13). It then returns the updated landmark vector lm′ (line 14).

Correctness and complexity. Procedure DelLM correctly maintains the landmark vector and updates distance vectors for each affected node (with local information changed). Indeed: (1) the loop (lines 3–10) correctly finds affected node sets affUP and affDW. (2) After affUP and affDW are computed, procedure DelLM iteratively updates the distance vectors for the affected nodes, by updating their distance from or to the new landmark vectors, respectively (lines 11–13). For the complexity, observe the following. (1) It takes $O(|P|)$ to find k_m (line 1). (2) It takes $O(|\mathsf{AFF}|^2)$ time to find affUP and affDW (lines 3–10), as: (a) DelLM only visits the nodes with local information changed once, and (b) each time it identifies the distance with linear time in $|\mathsf{AFF}|$. (3) It takes in total $O(|\mathsf{AFF}| \log |\mathsf{AFF}|)$ time to update the distance vectors, by implementing distv as priority queues (lines 11–13). To see this, note that: (a) DelLM visits each node in affUP as an ancestor of at least a landmark in lm′, in $O(|\mathsf{AFF}|)$ time, and (b) DelLM updates distv of a node v_{AFF} in affUP, by: (i) updating distv of the children of v_{AFF}, and (ii) computing distv of v_{AFF} directly with distv of its children, via priority queue insertion in $O(\log |\mathsf{AFF}|)$ time. Thus procedure DelLM is in $O(|P| + |\mathsf{AFF}| \log |\mathsf{AFF}| + |\mathsf{AFF}|^2)$ time. As verified by our experimental study, $|\mathsf{AFF}|$ is typically small in practice.

Single-edge insertions. Procedure InsLM incrementally updates lm in response to a single-edge insertion (v', v), similarly as DelLM. It finds those nodes v_1 such that: (1) $\mathsf{dis}(v_1, v)$ is changed, and (2) v_1 is within k_m hops of v, where k_m is the maximum bound in P. It updates the old landmark vector and distv_f for these nodes, and propagates the changes. Similarly it processes v'. The complexity of InsLM is in $O(|P|+|\mathsf{AFF}| \log |\mathsf{AFF}|+|\mathsf{AFF}|^2)$ time, the same as DelLM. Observe that InsLM is “lazy”: (a) the distance vectors of the nodes are updated only if they are within k_m hops of the edge e *and* if their distances are changed; and (b) at most one new landmark is added, while the other landmarks are updated later by an offline process in the background.

Batch updates. We next present IncLM to incrementally maintain landmark vectors and distance vectors in response to batch updates ΔG. Instead of dealing with updates one by one, it handles multiple updates simultaneously.

Given ΔG, algorithm IncLM first initializes two sets affUP and affDW. It uses affUP to store all those nodes u for which there exists an update $(v', v) \in \Delta G$ such that $\mathsf{dis}(u, v)$ is changed in $G \oplus \Delta G$. Similarly, affDW stores those nodes u with changed distance $\mathsf{dis}(v', u)$, for some $(v', v) \in \Delta G$. After these, it updates G with ΔG, and updates lm based on ΔG following procedures InsLM and DelLM. For each update $e \in \Delta G$, it then computes affUP and affDW, by identifying the affected nodes along the same lines as in procedures DelLM and InsLM. After all the affected nodes are identified, IncLM updates their distance vectors, and returns the updated vectors.

Correctness and complexity. The correctness of IncLM follows from that of InsLM and DelLM. For the complexity, observe the following: the number k_m is computed in $O(|P|)$ time once and can be reused, and lm can be updated in $O(|\Delta G|)$ time (Proposition 6.2). Affected node pairs can be found in $O(|\mathsf{AFF}|^2)$ time. Note that $|\Delta G|$ is subsumed by $|\mathsf{AFF}|^2$ in this phase, as IncLM handles multiple updates simultaneously instead of one

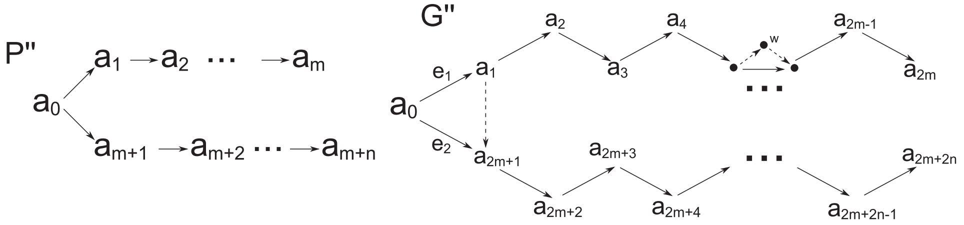


Fig. 15. Unboundedness of IncIsoMat.

by one. The distance vectors can be updated in $O(|\mathsf{AFF}| \log |\mathsf{AFF}|)$ time. Thus IncLM is in $O(|P| + |\mathsf{AFF}| \log |\mathsf{AFF}| + |\mathsf{AFF}|^2)$ time. This completes the proof of Proposition 6.3.

7. INCREMENTAL SUBGRAPH ISOMORPHISM

We next study incremental matching for subgraph isomorphism, denoted as IncIsoMat. Given a normal pattern P, a data graph G, matches $M_{\mathsf{iso}}(P, G)$, and changes ΔG to G, IncIsoMat is to find ΔM_{iso}, the set of subgraphs of G that are to be added to (or deleted from) $M_{\mathsf{iso}}(P, G)$, such that $M_{\mathsf{iso}}(P, G \oplus \Delta G) = M_{\mathsf{iso}}(P, G) \oplus \Delta M_{\mathsf{iso}}$.

We also study the problem for deciding whether there exists a subgraph in the updated graph $G \oplus \Delta G$ that is isomorphic to P, that is, $P \trianglelefteq_{\mathsf{iso}} G \oplus \Delta G$, referred to as IncIso.

The main results of this section are negative: these problems are hard even when data graphs are fixed, and for pattern and data graphs that have a tree structure.

THEOREM 7.1. *For subgraph isomorphism:*

(1) IncIso *is NP-complete even when data graphs are fixed, and*
(2) IncIsoMat *is unbounded for unit updates, even when patterns are trees and data graphs are forests.*

PROOF.

(1) We show that IncIso is NP-complete when G is fixed. The problem is in NP. Indeed, there exists an NP algorithm that: (a) computes $G \oplus \Delta G$ by applying ΔG to G, (b) guesses a subgraph G_s of $G \oplus \Delta G$, and (c) checks if G_s is isomorphic to P in PTIME. We show that IncIso is NP-hard by reduction from the Maximum Clique Problem (MCP), which is NP-complete (refer to Garey and Johnson [1979]). An instance of MCP $I = (G_0, k)$ consists of a graph $G_0 = (V_0, E_0)$ and an integer k. It is to determine whether there exists a clique of at least k nodes in G_0. Given I, we construct an instance of IncIso as follows. (a) We define a normal pattern P as a k-clique. (b) We fix graph G to be empty, that is, the node and edge sets of G are empty. (c) Fixing G, we define ΔG such that $G_0 = G \oplus \Delta G$. Observe that for any graph G_0 and the fixed G, ΔG always exists. It is easy to verify that there exists a k-clique in G_0 if and only if there exists a match for P in $G \oplus \Delta G$. Since MCP is NP-hard, IncIso is NP-complete even for a fixed graph G.
(2) We next show that IncIsoMat is unbounded for unit updates, and when P is a tree and G is a forest. We construct a normal pattern P'' and a graph G'' as shown in Figure 15 (ignore edges e_1 and e_2 for the moment). (a) The pattern P'' consists of a tree rooted as node a_0, which consists of $m+n+1$ nodes labeled with a. (b) The graph G'' consists of a single node a_0, and two paths $(a_1, \ldots, a_{2m})$ and $(a_{2m+1}, \ldots, a_{2m+2n})$. We define Δ_1 as the insertion of edge $e_1 = (a_0, a_1)$, and Δ_2 as the insertion of edge $e_2 = (a_0, a_{2m+1})$. Let $G''_1 = G'' \oplus \Delta_1$, $G''_2 = G'' \oplus \Delta_2$, and $G''_3 = G''_1 \oplus \Delta_2$. As in the proof of Theorem 5.1, we use $s(u)$ to denote the status of a node u in G''. Observe

that $M_{\mathsf{iso}}(P'', G'') = M_{\mathsf{iso}}(P'', G''_1) = M_{\mathsf{iso}}(P'', G''_2) = \emptyset$, while $M_{\mathsf{iso}}(P'', G''_3)$ is a tree of $m+n+1$ nodes.

Assume by contradiction that IncIsoMat is bounded. Then there exists an incremental algorithm $\mathcal{A}$ such that given a pattern P, graph G, a unit update ΔG, and the old output $M_{\mathsf{iso}}(P, G)$, $\mathcal{A}(P, G, \Delta G, M_{\mathsf{iso}}(P, G))$ computes ΔM_{iso} with its cost as a function of |CHANGED|. Thus, $\mathcal{A}(P'', G'', \Delta_1, M_{\mathsf{iso}}(P'', G''))$ and $\mathcal{A}(P'', G'', \Delta_2, M_{\mathsf{iso}}(P'', G''))$ are both in $O(1)$ time, since for unit updates Δ_1 and Δ_2, the changes from $M_{\mathsf{iso}}(P'', G'')$ to the new results $M_{\mathsf{iso}}(P'', G''_1)$ and $M_{\mathsf{iso}}(P'', G''_2)$ are empty in both cases. Along the same line as in the proof of Theorem 5.1, we consider the *trace* $T(G'', \Delta G'')$ of $\mathcal{A}(P'', G'', \Delta G'', M_{\mathsf{iso}}(P'', G''))$. We show that such an algorithm $\mathcal{A}$ does not exist.

(A) There exist nodes v in $T(G'', \Delta_1)$ such that $\mathcal{A}(P'', G'', \Delta_1, M_{\mathsf{iso}}(P'', G''))$ changes their status, because $T(G'', \Delta_2)$ and $T(G''_1, \Delta_2)$ generate different outputs. This can only happen if $s(v)$ differs in G'' and G''_1 for some nodes v, since $\mathcal{A}(P'', G'', \Delta_2, M_{\mathsf{iso}}(P'', G''))$ is in $O(1)$ time. Since graph G''_1 as an input of $\mathcal{A}$ differs from G' only due to Δ_1, $\mathcal{A}(P'', G'', \Delta_1, M_{\mathsf{iso}}(P'', G''))$ must visit nodes v in $T(G'', \Delta_1)$ and change their status.

(B) Algorithm $\mathcal{A}$ is unbounded. Consider a graph G''_v constructed from G''_1 by "shortcutting" those nodes v in $T(G'', \Delta_1)$: (1) if such a node v is a_0, then remove a_0 along with its edges while adjusting e_2 accordingly (e.g., e_2 is changed to (a_1, a_{2m+1})); (2) if v is a node a_i ($i \neq 0$ and $i \neq (2m+1)$), and if it is not a leaf, an edge is added from its parent a_{i-1} to a_{i+1}; (3) if v is the node a_{2m+1}, edge e_2 is adjusted to (a_0, a_{2m+2}); and (4) if v is one of the two leaf nodes a_{2m} and a_{2m+2n}, then remove the edges attached to v. Let $G''_{v_2} = G''_v \oplus \Delta_2$. Observe the following: (a) $\mathcal{A}(P'', G'', \Delta_2, M_{\mathsf{iso}}(P'', G''))$ and $\mathcal{A}(P'', G''_v, \Delta_2, M_{\mathsf{iso}}(P'', G''))$ should behave the same, since for all nodes u in G''_v, $s(u)$ in G and $s(u)$ in G''_v are the same; however, (b) $M_{\mathsf{iso}}(P'', G''_{v_2})$ is of size $O(m+n+1)$ while $M_{\mathsf{iso}}(P, G''_2) = \emptyset$. To see (b), note that for any node set with status changed, the corresponding G''_v with edges "bypassing" these nodes always contains a tree isomorphic to P'' after the insertion Δ_2. Hence either $\mathcal{A}(P'', G''_v, \Delta_2, M_{\mathsf{iso}}(P'', G''))$ is not correct or $T(G'', \Delta_1)$ is not of a constant size (while $\mathcal{A}(P'', G'', \Delta_2, M_{\mathsf{iso}}(P'', G''))$ is in $O(1)$ time). Both cases contradict the assumption that $\mathcal{A}$ is a bounded algorithm. □

Theorem 7.1(1) shows that the intractability of IncIso is introduced by pattern and updates, that is, the "dynamic" nature of the incremental problem. From this it also follows that IncIso is not semibounded unless P = NP, since there exists no algorithm with a polynomial cost in the size of $|\Delta G|$ and |AFF| otherwise, no matter what auxiliary structure is used as long as it is bounded by a polynomial in the size of G. Theorem 7.1(2) shows that IncIsoMat is unbounded although its batch counterpart is in PTIME (when P'' and G'' are trees [Garey and Johnson 1979]). In contrast, the incremental (bounded) simulation problems are semibounded and their batch counterparts are in PTIME.

8. EXPERIMENTAL EVALUATION

We next present an experimental study of our matching methods (Section 8.1) and incremental methods (Section 8.2), using real-life and synthetic data. The experiments were conducted on a machine with an Intel Core(TM)2 Dual Core 3.00 GHz CPU and 4GB of RAM. Each experiment was run at least 5 times, and the average is reported.

8.1. Experiments for Graph Pattern Matching

We first conducted two sets of experiments to evaluate: (1) the effectiveness of graph pattern matching based on bounded simulation (Section 2), and (2) the efficiency and scalability of algorithm Match (Figure 3) for graph pattern matching.

Experimental setting. We used real-life data and synthetic data in our experiments.

(1) *Real-life data.* We used: (a) a crawled YouTube graph [YouTube 2012] with 14829 nodes and 58901 edges, where each node denotes a video with attributes (e.g., length, category, age), and edges indicate recommendations; and (b) a citation network [Tang et al. 2008] with 17292 nodes and 61351 edges, where each node represents a paper with attributes (e.g., title, author, the year of publication), and edges denote citations.

(2) *Synthetic data.* We used the Java boost graph generator to produce graphs with 3 parameters: the number of nodes, the number of edges, and a set of node attributes. We generated sequences of data graphs following the densification law [Leskovec et al. 2007] and linkage generation models [Garg et al. 2009].

(3) *Pattern generator.* We designed a generator to produce meaningful pattern graphs for both real-life and synthetic data, controlled by 4 parameters: the number of nodes $|V_p|$, the number of edges $|E_p|$, the average number $|\mathsf{pred}|$ of predicates carried by each node, and an upper bound k such that each pattern edge has a bound k' with $k - c \leq k' \leq k$, for a small constant c. We will use $(|V_p|, |E_p|, |\mathsf{pred}|, k)$ to characterize a pattern.

(4) *Implementation.* We implemented the following algorithms in Java: (1) Match; (2) two variants of Match, Match with BFS and Match with 2-hop, which use Breadth-First Search (BFS) to compute node distances and leverage 2-hop labeling [Cheng et al. 2008b] to prune disconnected nodes, respectively; these were to explore whether the existing techniques could help bounded simulation; and (3) VF2, a matching algorithm based on subgraph isomorphism [Cordella et al. 2004]. Observe that given a normal pattern in which all the edges are labeled 1, Match conducts matching via simulation.

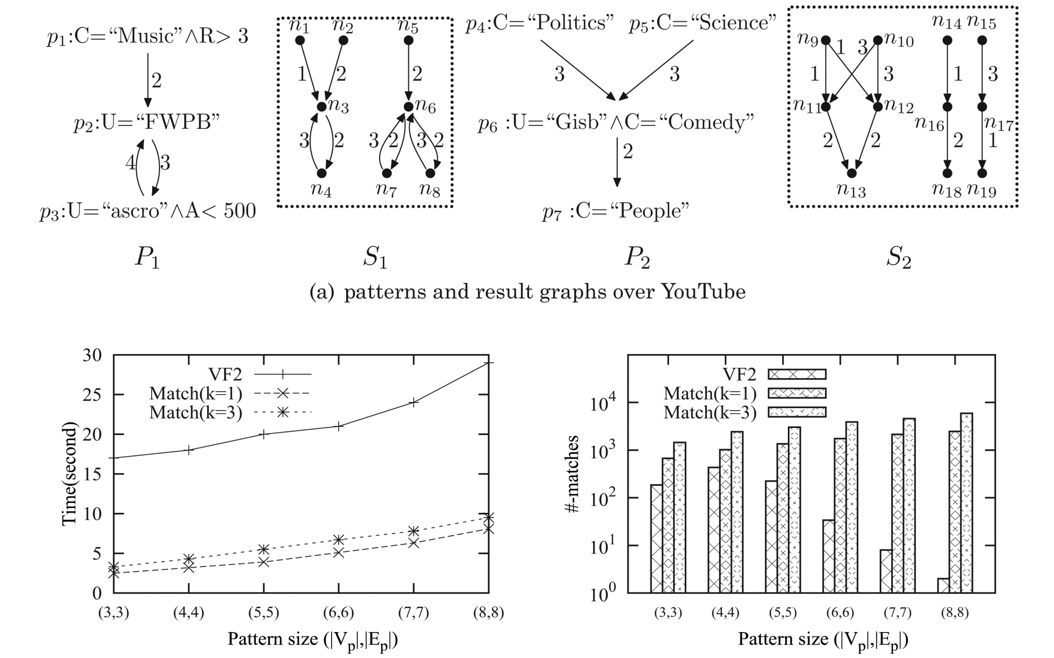
Experimental results. We next present our findings.

Exp-1: Effectiveness and efficiency. In this set of experiments, we first evaluated the effectiveness of Match versus VF2 in identifying sensible matches in YouTube. We then studied the efficiency of different matching methods, using large synthetic datasets.

Effectiveness. We manually constructed 20 patterns for YouTube, to find popular videos. Two sample patterns and their result graphs are shown in Figure 16(a). Pattern P_1 is to find "music" videos with rating (p_1), which are linked to videos of user "FWPB" within 2 hops (p_2); node p_2 is within 3 hops to videos uploaded by "Ascrodin" (p_3), which are less than 500 days old and are in turn connected to p_2 in 4 hops. Pattern P_2 is to find all "comedy" videos from "Gisburgh" (p_6), which are referenced by both "politics" (p_4) and "science" videos (p_5) in 3 hops, and have links to "people" videos within 2 hops (p_7).

We ran Match and VF2 on YouTube for each pattern. We then manually inspected the result graphs found by Match and the subgraphs found by VF2 that are isomorphic to the pattern, to check the accuracy of the matches. We find the following. (1) For 2 out of 20 patterns, VF2 could not find any match, while Match returned meaningful results with 9 matches in average per pattern node. These happened even when the bound k was set to 1 to favor VF2. (2) When VF2 did not fail, Match consistently identified more meaningful matches than VF2. Indeed, while VF2 found only 1 match for each pattern node, Match found in average 5 matches per pattern node. For instance, partial matches found by Match were shown as S_1 and S_2 in Figure 16(a), which were missed by VF2.

Efficiency. We evaluated the efficiency of Match versus VF2 using YouTube. Figure 16(b) shows the results, where x-axis represents $(|V_p|, |E_p|)$ in a pattern $P(V_p, E_p)$.



(a) patterns and result graphs over YouTube

(b) Match vs VF2 efficiency

(c) Match vs VF2 effectiveness

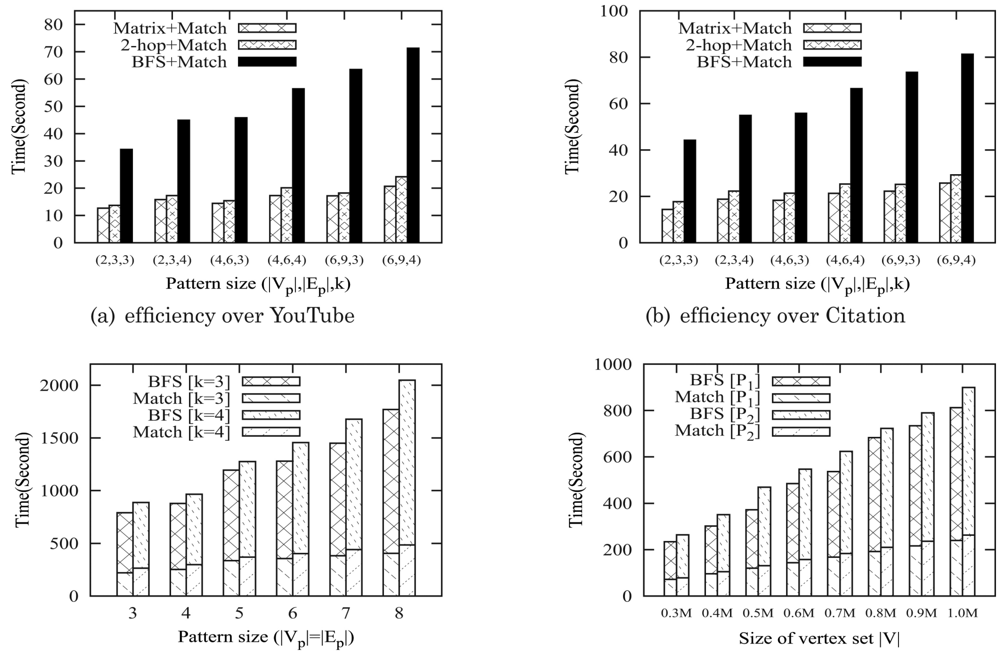
Fig. 16. Exp-1: Effectiveness and efficiency of bounded simulation.

We used $k = 1$ to favor VF2 and Match based on graph simulation. The curves Match ($k = 1$) and Match ($k = 3$) reflect the elapsed time for matching (excluding the time for computing a distance matrix, since it was computed only once and shared by all patterns). The results tell us that Match is much faster than VF2, no matter whether $k = 1$ or $k = 3$. Moreover, when $k = 3$, Match ran slower than when $k = 1$ (but not much), because bounded simulation is more costly than simulation, as expected.

Figure 16(c) reports the number of distinct matches found by VF2, Match ($k = 1$) and Match ($k = 3$). It tells us the following: (1) Match consistently finds much more meaningful matches than VF2, since matching via subgraph isomorphism imposes too strong a topological constraint; and (2) Match captures more sensible matches when $k = 3$ (i.e., based on bounded simulation) than Match when $k = 1$ (simulation), since pattern matching via bounded simulation allows edge-to-path mappings, as expected.

Exp-2: Efficiency and scalability. The second set of experiments evaluated: (1) the efficiency of various implementations of Match by using distance matrices, BFS and 2-hop, respectively, to identify ancestors or descendants of a node within a distance bound k; and (2) the scalability of Match with the size of data graphs and patterns.

Efficiency. Figures 17(a) and 17(b) show the results on real-life datasets YouTube and Citation, respectively. The x-axis represents ($|V_p|$, $|E_p|$, k) for a pattern. The results tell us the following: (1) Match with a distance matrix outperforms the other approaches; (2) the more complex the patterns are, the more costly for all the three methods, as expected; and (3) when the pattern size is fixed, varying the distance bound (from $k = 3$ to $k = 4$) increases the computational time of all these methods.



(a) efficiency over YouTube

(b) efficiency over Citation

(c) scalability to pattern graph

(d) scalability to data graph

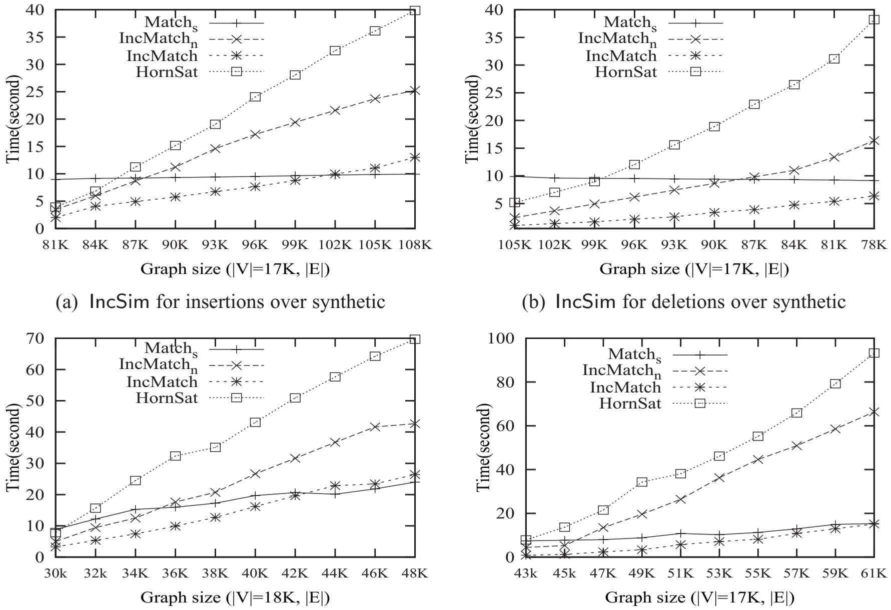
Fig. 17. Exp-2: Efficiency and scalability of bounded simulation.

Scalability. When data graphs are large, it is not feasible to build a distance matrix and 2-hop. This highlights the need for Match with BFS. Hence we focused on the scalability of Match via BFS, first with the complexity of patterns. When $k = 3$ or $k = 4$, we varied $|V_p| = |E_p|$ from 3 to 8, on a graph G with $|V| = 1M$, $|E| = 2M$. The results in Figure 17(c) tell us the following. (1) Match via BFS scales well with the size of patterns. (2) BFS accounts for 80% of the total computational time of Match. (3) The larger the bound k or the pattern, the more costly BFS. Indeed, when k or patterns get larger, BFS visits more nodes to identify ancestors or descendants of a node within the distance bound k.

We also evaluated the scalability of Match via BFS with the size of data graph $|G|$. We varied $|V|$ from $0.3M$ to $1M$, in $0.1M$ increments, while letting $|E| = 2|V|$. We used two patterns P_1 and P_2, with $(V_p, E_p, k) = (3, 3, 3)$ and $(4, 4, 3)$, respectively. As shown in Figure 17(d): (1) Match via BFS scales with $|G|$; this verifies the complexity analysis of Match via BFS (Section 3); and (2) it is more costly for Match to find matches of P_2 than the smaller P_1. This is consistent with the observation of Figure 17(c).

8.2. Experiments for Incremental Graph Pattern Matching

We conducted three sets of experiments to evaluate: (1) the performance of IncMatch for incremental simulation, compared with: (a) its batch counterpart Match$_\mathrm{s}$ [Henzinger et al. 1995], (b) IncMatch$_\mathrm{n}$, a naive algorithm that processes unit updates one by one by invoking IncMatch$^+$ and IncMatch$^-$, and (c) HORNSAT, the incremental simulation algorithm of Shukla et al. [1997]; (2) the efficiency of IncBMatch for incremental bounded simulation (see Section 6), compared with: (a) its batch counterpart Match$_\mathrm{bs}$ [Fan et al. 2010a], and (b) the incremental algorithm IncBMatch$_\mathrm{m}$ of Fan et al. [2010a] on DAG patterns, using a distance matrix; (3) the effectiveness of the optimization techniques,



(a) IncSim for insertions over synthetic

(b) IncSim for deletions over synthetic

(c) IncSim over YouTube

(d) IncSim over Citation

Fig. 18. Exp-1: Incremental graph simulation.

that is, (a) landmark and distance vectors, and (b) procedures minDelta and InsLM, DelLM, IncLM. All the algorithms (summarized next) are implemented in Java.

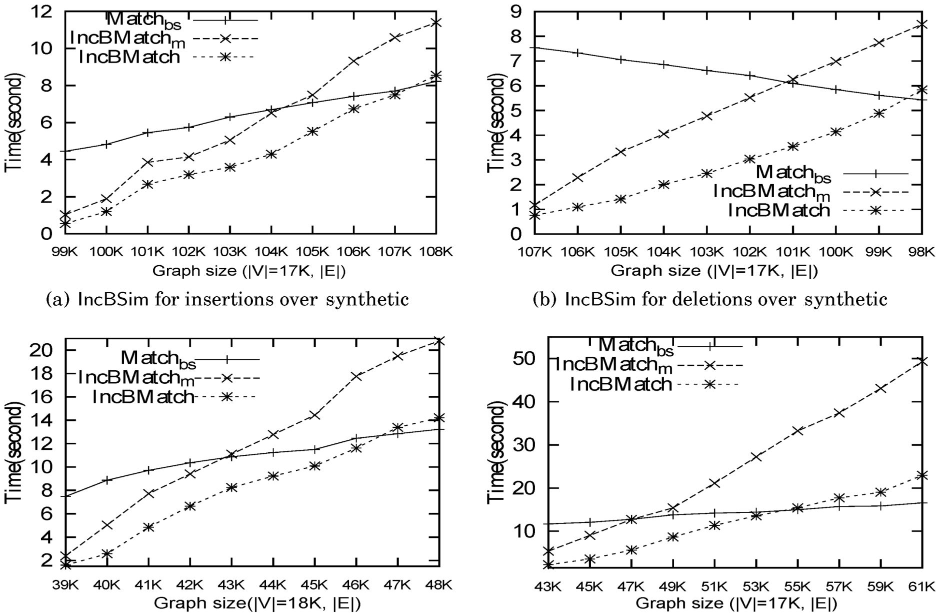
Problem	Batch	Incremental
IncSim	Match$_s$	IncMatch, IncMatch$_n$, HORNSAT
IncBSim	Match$_{bs}$	IncBMatch, IncBMatch$_m$
Optimizations	BatchLM, minDelta	InsLM, DelLM, IncLM

Experimental setting. We used the real-life datasets, synthetic graph generator, and pattern generator given in Section 8.1. Updates were selected following the densification law [Leskovec et al. 2007]: we selected nodes with larger degree with higher probability for edge deletion (respectively, insertion) if they are (respectively, not) connected. We used a greedy algorithm [Vazirani 2003] to approximately compute a minimum vertex cover for each data graph as a landmark vector, as well as corresponding distance vectors.

Experimental results. We next report our findings.

Exp-1: Incremental graph simulation. We first evaluated the efficiency of IncMatch. We generated 30 *normal patterns* for each of YouTube, Citation, and synthetic data, with parameters (4, 5, 3, 1) for synthetic data and (6, 8, 3, 1) for real-life data.

Fixing $|V| = 17K$ on synthetic data, we varied $|E|$ from $78K$ to $108K$ (respectively, from $108K$ to $78K$) in $3K$ increments (respectively, decrements). The results in Figures 18(a) and 18(b) tell us the following. (a) IncMatch outperforms Match$_s$ when insertions are no more than 30% (respectively, 30% for deletions; not shown). When the changes are 11% for insertions (respectively, 18% for deletions), IncMatch improves Match$_s$ over by 40% (respectively, 50%). (b) IncMatch and IncMatch$_n$ consistently do better than HORNSAT. HORNSAT does not scale well with $|\Delta G|$, due to its additional costs for



(a) IncBSim for insertions over synthetic

(b) IncBSim for deletions over synthetic

(c) IncBSim over YouTube

(d) IncBSim over Citation

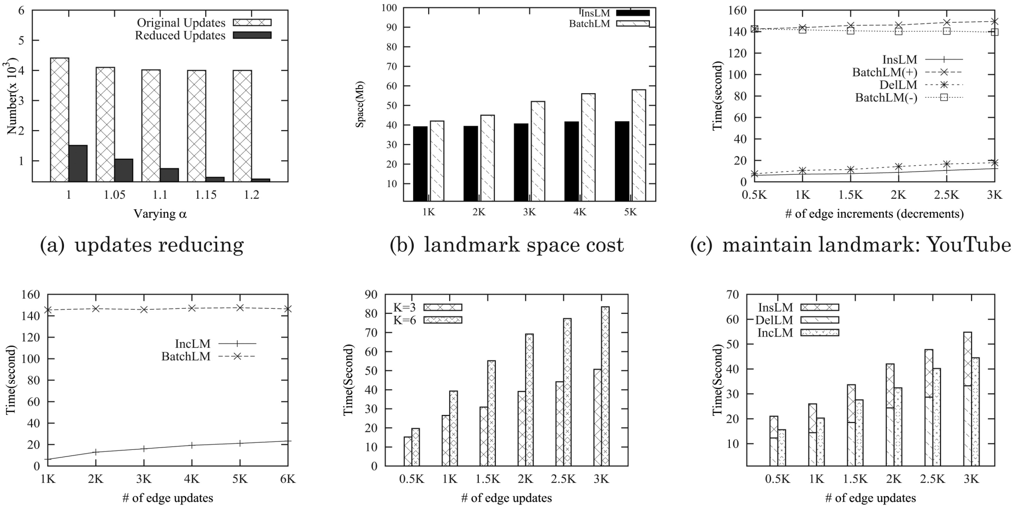
Fig. 19. Exp-2: Incremental bounded simulation.

updating reflections and maintaining its auxiliary structures. (c) IncMatch does better than IncMatch$_n$. This verifies the effectiveness of minDelta, which reduces $|\Delta G|$. (d) As opposed to Match$_s$, IncMatch and IncMatch$_n$ are sensitive to $|\Delta G|$, as expected. This is because the larger $|\Delta G|$, the larger the affected area, so is the computation cost. This justifies the complexity measure of incremental algorithms in terms of the size of $|\Delta G|$ and |AFF|.

On real-life data, Figures 18(c) and 18(d) show the results for edges inserted into YouTube and Citation datasets, respectively. Each dataset has $|V| = 18K$ (respectively, $17K$), and $|E|$ as shown in the x-axis. Here the updates are the differences between snapshots with respect to the age (respectively, year) attribute of YouTube (respectively, Citation), reflecting their real-life evolution. The results confirm our observations on synthetic data. For instance, IncMatch outperforms Match$_s$ on YouTube even when the changes are up to 50%.

Exp-2: Incremental bounded simulation. We evaluated the efficiency of IncBMatch versus Match$_{bs}$ and IncBMatch$_m$. We produced 30 b-patterns with parameters $(4, 5, 3, 3)$ for synthetic data, and $(6, 8, 3, 3)$ for real-life data. To favor IncBMatch$_m$ that only works on DAGS, we used DAG b-patterns.

Fixing $|V| = 17K$ on synthetic data, we varied $|E|$ from $98K$ to $108K$ (respectively, from $108K$ to $98K$) in $1K$ increments (respectively, decrements). The results shown in Figures 19(a) and 19(b) tell us the following. (a) IncBMatch outperforms Match$_{bs}$ when both edge insertions and deletions are no more than 10%. (b) IncBMatch consistently does better than IncBMatch$_m$, by about 30% (respectively, 40%) for insertions (respectively, deletions) when $|\Delta G| = 10K$. Note that IncBMatch$_m$ emforploys a distance matrix to compute the distance between two nodes, and does not scale well with large graphs. In contrast, IncBMatch uses landmark vectors to improve the scalability. (c) For the same $|\Delta G|$, IncBMatch takes longer to process insertions than deletions. As



(a) updates reducing (b) landmark space cost (c) maintain landmark: YouTube

(d) IncLM over YouTube (e) IncLM over Citation (f) Maintain landmark: synthetic

Fig. 20. Exp-3: Optimization techniques.

indicated by Theorem 5.1(2), edge insertions introduce more complications than deletions to (bounded) simulation. For deletions (Figure 19(b)), the larger ΔG, the smaller $G \oplus \Delta G$, and hence so is the cost for handling ΔG. Note that $\mathsf{Match_{bs}}$ is less sensitive to $|\Delta G|$ than IncBMatch, as expected, although it takes less time for graphs with less edges. This further justifies the complexity analysis of incremental algorithms in terms of ΔG as opposed to $|G|$.

Figures 19(c) and 19(d) show the performance of the algorithms for edge insertions to YouTube and Citation datasets, respectively, in the same setting as before. The results show that IncBMatch does even better on real-life data than on synthetic data; for example, IncBMatch outperforms $\mathsf{Match_{bs}}$ on YouTube when changes are no more than 20%.

Exp-3: Optimization techniques. In this set of experiments we evaluated: (1) the effectiveness of minDelta, (2) the space cost of landmark and distance vectors, and (3) the efficiency of InsLM, DelLM, and IncLM for updating those vectors. In these experiments, we generated synthetic graphs following the densification law [Leskovec et al. 2007], by using one more parameter α for our generator such that $|E| = |V|^{\alpha}$.

Effectiveness. To analyze the effectiveness of minDelta, we fixed $|V| = 20K$, varied parameter α, and randomly inserted and deleted 4000 edges. The results are shown in Figure 20(a). We find that minDelta significantly reduces redundant updates. This becomes more evident when α is increased, that is, if the graphs have more edges. In this case, more nodes are in the result graphs, and those updated edges are less likely to affect the match results. The results also demonstrate the potential benefits of minDelta in real-life applications, where insertions are more common (e.g., [Garg et al. 2009]).

Space cost. Fixing $|V| = 10K$ and $\alpha = 1.1$, Figure 20(b) shows the space cost of landmark and distance vectors, incrementally maintained and recomputed from scratch, respectively. Observe that: (a) landmark and distance vectors take much less space than a $(10K)^2$ distance matrix [Fan et al. 2010a]; and (b) compared to recomputation, InsLM updates the landmark and distance vectors with extra space cost of at most

2%. After $5K$ edges are inserted, the recomputed landmark and distance vectors takes $56M$, while the total extra space added by InsLM is only $674K$.

Efficiency. We evaluated the efficiency of InsLM versus BatchLM$^+$ (respectively, DelLM versus BatchLM$^-$) over YouTube. Here BatchLM$^+$ (respectively, BatchLM$^-$) denotes a batch algorithm for edge insertion (respectively, deletion). Fixing $|V| = 18K$ and $k = 5$, we varied $|E|$ from $59K$ to $62K$ (respectively, from $59K$ to $56K$). The results are reported in Figure 20(c), which tell us the following. (1) InsLM (respectively, DelLM) is much more efficient than BatchLM$^+$ (respectively, BatchLM$^-$): InsLM (respectively, DelLM) takes only 8% (respectively, 13%) of the time of BatchLM$^+$ (respectively, BatchLM$^-$) when $3K$ edges are inserted (respectively, removed). (2) InsLM is more efficient than DelLM; this is because edge deletions tend to affect more nodes with changed distance from (respectively, to) the nodes in landmark vector, and BatchLM$^+$ outperforms BatchLM$^-$ for the same reason. This is more evident when $|\Delta G|$ gets larger.

We also evaluated the efficiency of IncLM versus the batch algorithm BatchLM, using YouTube. Fixing $k = 5$, we varied updates from $1K$ to $6K$, with 50% of edge insertions and 50% of edge deletion. As shown in Figure 20(d), IncLM is much more efficient than BatchLM, taking only 15% of the time used by BatchLM for updates of $6K$.

Moreover, we evaluated the impact of the maximum bound k on IncLM, using Citation. Fixing $|V| = 17K$ and $|E| = 62K$, we varied k from 3 to 6, and generated batch updates (edge insertions and deletions). As shown in Figure 20(e), it is more costly for IncLM to maintain landmark vectors for larger k. Indeed, the larger k, the more node pairs IncLM has to inspect, to find out whether these nodes are affected by the updates.

Finally, we evaluated the efficiency of IncLM versus a naive incremental algorithm, denoted by InsLM+DelLM, which invokes InsLM and DelLM one by one for each update. We used synthetic graphs in this experiment. Fixing $|V| = 15K$, $|E| = 40K$ and $k = 5$, we generated ΔG with both edge insertions and deletions. The results shown in Figure 20(f) tell us that IncLM consistently outperforms InsLM + DelLM, by 20% on average. These verified the effectiveness of the optimization strategies used by IncLM, which, among other things, substantially eliminated redundant updates from ΔG.

Summary. We summarize our findings in the following table.

Problems	Algorithm comparisons		Results
	Traditional ones	Our algorithms	
Revised graph pattern matching	VF2 (subgraph isomorphism)	Match (bounded simulation)	○ Match identifies far more sensible matches ○ Match is more efficient than VF2 ○ Match scales well with data graphs
Incremental simulation IncSim	Match$_{s}$, HORNSAT	IncMatch, IncMatch$_{n}$	○ IncMatch is much more efficient than batch Match$_{s}$ and naive process IncMatch$_{n}$ ○ IncMatch does much better than HORNSAT
Incremental bounded simulation IncBSim	Match$_{bs}$	IncBMatch, IncBMatch$_{m}$	○ IncBMatch is far better than batch Match$_{bs}$ and naive IncBMatch$_{m}$ on DAG patterns
Optimizations	BatchLM	minDelta, InsLM, DelLM, IncLM	○ minDelta significantly reduces updates ○ InsLM takes less space than BatchLM ○ IncLM is more efficient than BatchLM ○ InsLM (resp. DelLM) is far better than BatchLM for insertions (resp. deletions)

9. CONCLUSION

We have proposed a revision of graph pattern matching, based on a notion of bounded simulation. This yields a cubic-time method for finding matches, as opposed to the intractability of its counterpart via subgraph isomorphism. Moreover, it is able to capture more sensible matches in emerging applications. We have also investigated the incremental pattern matching problem for matching defined in terms of subgraph isomorphism, graph simulation, and bounded simulation, from complexity (boundedness) analysis to incremental algorithms. We have shown that the incremental matching problem is unbounded for matching based on all the three notions. Nonetheless, for simulation and bounded simulation, we have shown that their incremental matching problems are semibounded, and developed efficient incremental algorithms for (possibly *cyclic*) patterns and batch updates. We have also developed incremental algorithms for maintaining auxiliary data structures, that is, landmark and distance vectors. These allow us to efficiently find matches when data graphs are updated, minimizing unnecessary recomputation. Our experimental results have verified the scalability and effectiveness of our batch and incremental methods, using real-life and synthetic data.

We are experimenting with real-life datasets in various domains, to identify areas in which the revised matching is most effective. We are also investigating optimization techniques, as well as lower bounds for incremental matching by exploring usage patterns of real-life networks [Kumar et al. 2006; Ntoulas et al. 2004; White and Smyth 2003]. Another topic is to develop bounded incremental heuristic algorithms for subgraph isomorphism with performance guarantees. Finally, we are extending our incremental matching methods to querying distributed graphs, using MapReduce.

REFERENCES

Abiteboul, S., Mchugh, J., Rys, M., Vassalos, V., and Wiener, J. L. 1998. Incremental maintenance for materialized views over semistructured data. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98).* 38–49.

Alpern, B., Hoover, R., Rosen, B. K., Sweeney, P. F., and Zadeck, F. K. 1990. Incremental evaluation of computational circuits. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'90).* 32–42.

Amer-Yahia, S., Benedikt, M., and Bohannon, P. 2007. Challenges in searching online communities. *IEEE Data Engin. Bull. 30,* 2, 23–31.

Bang-Jensen, J. and Gutin, G. Z. 2008. *Digraphs: Theory, Algorithms and Applications*. Springer.

Bruno, N., Koudas, N., and Srivastava, D. 2002. Holistic twig joins: Optimal XML pattern matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'02).* 310–321

Brynielsson, J., Hogberg, J., Kaati, L., Martenson, C., and Svenson, P. 2010. Detecting social positions using simulation. In *Proceedings of the International Conference on Advances in Social Networks Analysis and Mining (ASONAM'10).* 48–55.

Chan, E. P. and Lim, H. 2007. Optimization and evaluation of shortest path queries. *The Int. J. Very Large Data Bases 16,* 3, 343–369.

Chen, L., Gupta, A., and Kurul, M. E. 2005. Stack-based algorithms for pattern matching on dags. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05).* 493–504.

Chen, Z., Shen, H. T., Zhou, X., and Yu, J. X. 2009. Monitoring path nearest neighbor in road networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'09).*

Cheng, J., Yu, J. X., Ding, B., Yu, P. S., and Wang, H. 2008a. Fast graph pattern matching. In *Proceedings of the 24th International Conference on Data Engineering (ICDE'08).*

Cheng, J., Yu, J. X., Lin, X., Wang, H., and Yu, P. S. 2008b. Fast computing reachability labelings for large graphs with high compression rate. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'08).*

Cho, J., Shivakumar, N., and Garcia-Molina, H. 2000. Finding replicated web collections. *SIGMOD Rec. 29,* 2.

Cohen, E., Halperin, E., Kaplan, H., and Zwick, U. 2003. Reachability and distance queries via 2-hop labels. *SIAM J. Comput. 32,* 5, 1338–1355.

Cordella, L. P., Foggia, P., Sansone, C., and Vento, M. 2004. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell. 26,* 10, 1367–1372.

Dovier, A., Piazza, C., and Policriti, A. 2001. A fast bisimulation algorithm. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01).* 79–90.

FAN, W. AND BOHANNON, P. 2008. Information preserving xml schema embedding. *ACM Trans. Datab. Syst. 33,* 1.

FAN, W., LI, J., LUO, J., TAN, Z., WANG, X., AND WU, Y. 2011a. Incremental graph pattern matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'11).* 925–936.

FAN, W., LI, J., MA, S., TANG, N., AND WU, Y. 2011b. Adding regular expressions to graph reachability and pattern queries. In *Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE'11).* 39–50.

FAN, W., LI, J., MA, S., TANG, N., WU, Y., AND WU, Y. 2010a. Graph pattern matching: From intractable to polynomial time. *Proc. VLDB Endow. 3,* 1–2, 264–275.

FAN, W., LI, J., MA, S., WANG, H., AND WU, Y. 2010b. Graph homomorphism revisited for graph matching. *Proc. VLDB Endow. 3,* 1–2, 1161–1172.

FLOYD, R. W. 1962. Algorithm 97: Shortest path. *Comm. ACM 5,*6, 345.

GALLAGHER, B. 2006. Matching structure and semantics: A survey on graph-based pattern matching. In *Proceedings of the Conference of the American Association for Artificial Intelligence (AAAI'06).*

GAREY, M. AND JOHNSON, D. 1979. *Computers and Intractability: A Guide to the Theory of NP- Completeness.* W. H. Freeman and Company.

GARG, S., GUPTA, T., CARLSSON, N., AND MAHANTI, A. 2009. Evolution of an online social aggregation network: An empirical study. In *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference (IMC'09).* 315–321.

GENTILINI, R., PIAZZA, C., AND POLICRITI, A. 2003. From bisimulation to simulation: Coarsest partition problems. *J. Autom. Reason. 31,* 1, 73–103.

GUPTA, A. AND MUMICK, I. 2000. *Materialized Views.* MIT Press.

HE, H. AND SINGH, A. K. 2009. Graphs-at-a-time: Query language and access methods for graph databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'08).* 405–418.

HENZINGER, M. R., HENZINGER, T., AND KOPKE, P. 1995. Computing simulations on finite and infinite graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95).* 453.

JIN, R., XIANG, Y., RUAN, N., AND FUHRY, D. 2009. 3-hop: A high-compression indexing scheme for reachability query. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'09).* 813–826.

KAHN, A. B. 1962. Topological sorting of large networks. *Comm. ACM 5,* 11, 558–562.

KUMAR, R., NOVAK, J., AND TOMKINS, A. 2006. Structure and evolution of online social networks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06).* 611–617.

LESKOVEC, J., KLEINBERG, J., AND FALOUTSOS, C. 2007. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Disc. Data 1,* 1, 2.

LI, C.-T. AND SHAN, M.-K. 2012. Composing activity groups in social networks. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM'12).*

MA, S., CAO, Y., FAN, W., HUAI, J., AND WO, T. 2011. Capturing topology in graph pattern matching. *Proc. VLDB Endow. 5,* 4, 310–321.

MILNER, R. 1989. *Communication and Concurrency.* Prentice Hall.

NARDO, L. D., RANZATO, F., AND TAPPARO, F. 2009. The subgraph similarity problem. *IEEE Trans. Knowl. Data Engin. 21,* 5, 748–749.

NATARAJAN, M. 2000. Understanding the structure of a drug trafficking organization: A conversational analysis. *Crime Prevent. Stud. 11*, 273–298.

NTOULAS, A., CHO, J., AND OLSTON, C. 2004. What's new on the web? The evolution of the web from a search engine perspective. In *Proceedings of the 13th International Conference on World Wide Web (WWW'04).* 1–12.

POTAMIAS, M., BONCHI, F., CASTILLO, C., AND GIONIS, A. 2009. Fast shortest path distance estimation in large networks. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM'09).*

RAMALINGAM, G. AND REPS, T. 1993. A categorized bibliography on incremental computation. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'93).* 502–510.

RAMALINGAM, G. AND REPS, T. 1996a. An incremental algorithm for a generalization of the shortest-path problem. *J. Algor. 21,* 2, 267–305.

RAMALINGAM, G. AND REPS, T. 1996b. On the computational complexity of dynamic graph problems. *Theor. Comput. Sci. 158,* 1–2, 233–277.

RONEN, R. AND SHMUELI, O. 2009. SoQL: A language for querying and creating data in social networks. In *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE'09).* 1595–1602.

SAHA, D. 2007. An incremental bisimulation algorithm. In *Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07).* 204–215.

SAN MARTIN, M., GUTIERREZ, C., AND WOOD, P. T. 2011. SNQL: A social networks query and transformation language. In *Proceedings of the 5th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW'11).*

SHASHA, D., WANG, J. T. L., AND GIUGNO, R. 2002. Algorithmics and applications of tree and graph searching. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'02)*. 39–52.

SHUKLA, S. K., SHUKLA, E. K., ROSENKRANTZ, D. J., III, HUNT, H. B., AND STEARNS, R. E. 1997. The polynomial time decidability of simulation relations for finite state processes: A hornsat based approach. In *Proceedings of the DIMACS Conference Series in Discrete Mathematics and Computer Science*.

SMITH, C. 2013. By the numbers: 25 amazing facebook stats. http://expandedramblings.com/index.php/by-the-numbers-17-amazing-facebook-stats/

STOTZ, A., NAGI, R., AND SUDIT, M. 2009. Incremental graph matching for situation awareness. In *Proceedings of the 12th International Conference on Information Fusion (FUSION'09).*

TANG, J., ZHANG, J., YAO, L., LI, J., ZHANG, L., AND SU, Z. 2008. ArnetMiner: Extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08).* 990–998.

TERVEEN, L. AND MCDONALD, D. W. 2005. Social matching: A framework and research agenda. *ACM Trans. Comput.-Hum. Interact. 12,* 3.

TONG, H., FALOUTSOS, C., GALLAGHER, B., AND ELIASSI-RAD, T. 2007. Fast best-effort pattern matching in large attributed graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07).* 737–746.

VAZIRANI, V. V. 2003. *Approximation Algorithms*. Springer.

WANG, C. AND CHEN, L. 2009. Continuous subgraph pattern search over graph streams. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'09).* 393–404.

WANG, H., HE, H., YANG, J., YU, P. S., AND YU, J. X. 2006. Dual labeling: Answering graph reachability queries in constant time. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06).* 75.

WHITE, S. AND SMYTH, P. 2003. Algorithms for estimating relative importance in networks. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03).* 266–275.

YI, K., HE, H., STANOI, I., AND YANG, J. 2004. Incremental maintenance of xml structural indexes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'04).* 491–502.

YOUTUBE. 2012. Youtube dataset. http://netsg.cs.sfu.ca/youtubedata/

ZHUGE, Y. AND GARCIA-MOLINA, H. 1998. Graph structured views and their incremental maintenance. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98).* 116–125.

ZOU, L., CHEN, L., AND OZSU, M. T. 2009. Distance-join: Pattern match query in a large graph database. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB'09).* 864–875.

Received December 2012; revised April 2013; accepted May 2013