Graph Homomorphism Revisited for Graph Matching

Wenfei Fan[1,2] Jianzhong Li[2] Shuai Ma[1] Hongzhi Wang[2] Yinghui Wu[1]
[1]University of Edinburgh [2]Harbin Institute of Technology
{wenfei@inf.,shuai.ma@,y.wu-18@sms.}ed.ac.uk {lijzh, wangzh}@hit.edu.cn

Abstract

In a variety of emerging applications one needs to decide whether a graph G *matches* another G_p, *i.e.,* whether G has a topological structure similar to that of G_p. The traditional notions of graph homomorphism and isomorphism often fall short of capturing the structural similarity in these applications. This paper studies revisions of these notions, providing a full treatment from complexity to algorithms. (1) We propose p-*homomorphism* (p-hom) and 1-1 p-hom, which extend graph homomorphism and subgraph isomorphism, respectively, by mapping *edges* from one graph to *paths* in another, and by measuring *the similarity of nodes*. (2) We introduce metrics to measure graph similarity, and several optimization problems for p-hom and 1-1 p-hom. (3) We show that the decision problems for p-hom and 1-1 p-hom are NP-complete even for DAGs, and that the optimization problems are approximation-hard. (4) Nevertheless, we provide approximation algorithms with *provable guarantees* on match quality. We experimentally verify the effectiveness of the revised notions and the efficiency of our algorithms in Web site matching, using real-life and synthetic data.

1. Introduction

The notions of graph homomorphism and subgraph isomorphism [9] can be found in almost every graph theory textbook. Given two node-labeled graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the problem of graph homomorphism (resp. subgraph isomorphism) is to find a (resp. 1-1) mapping from V_1 to V_2 such that each node in V_1 is mapped to a (resp. distinct) node in V_2 with the same label, and each edge in E_1 is mapped to an edge in E_2.

These conventional notions are, however, often too restrictive for graph matching in emerging applications. In a nutshell, graph matching is to decide whether a graph G matches another graph G_p, *i.e.,* whether G has a structure similar to that of G_p, although not necessarily identical. The need for this is evident in, *e.g.,* Web anomaly detection [23], search result classification [25], plagiarism detection [20] and spam detection [3]. In these contexts, identical label matching is often an overkill, and edge-to-edge mappings only allow strikingly similar graphs to be matched.

Example 1.1: Consider two online stores depicted in Fig. 1 as graphs $G_p = (V_p, E_p)$ and $G = (V, E)$. In these graphs, each node denotes a Web page for sale of certain items, as indicated by its label; and the edges denote hyperlinks. One wants to know whether G matches G_p, *i.e.,* whether all the items specified by G_p are also carried by the store G, and G and G_p can be navigated similarly, *i.e.,* if a site for selling item a can be reached from a site for item b in G_p by following hyperlinks, then the site for item a can also be reached from the site for b in G.

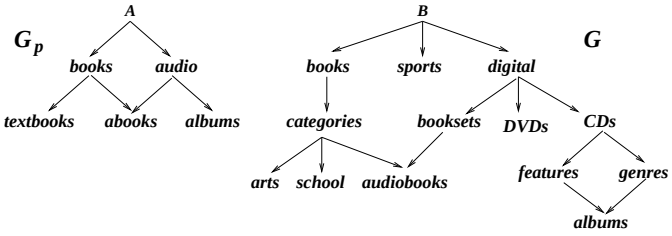


Figure 1: Graphs representing online stores

When graph homomorphism or subgraph isomorphism is used to measure graph similarity, G does *not* match G_p. Indeed, (a) nodes in G may not find a node in G with the same label, *e.g.,* audio; and worse still, (b) there exists no sensible mapping from V_p to V that maps edges in G_p to edges in G accordingly.

However, a page checker (*e.g.,* [8, 29]) may find connections between pages in G_p and those in G based on their functionality:

A $\mapsto$ B, books $\mapsto$ books, audio $\mapsto$ digital, textbooks $\mapsto$ school, abooks $\mapsto$ audiobooks, albums $\mapsto$ albums

That is, the store G indeed has the capability of G_p. While the edges in G_p are not preserved by the similarity relation, each edge in G_p is mapped to a *path* in G, *e.g.,* the edge (books, textbooks) in G_p is mapped to the path books/categories/school in G. This tells us that G preserves the navigational structure of G_p. Hence G should logically be considered as a match of G_p. □

These highlight the need for revising the conventional notions of graph matching. In response to these, several extensions of the conventional notions have been studied for graph matching [10, 11, 14, 24, 32]. However, a formal analysis of these extensions is not yet in place, from complexity bounds to approximation algorithms.

Contributions. We propose several notions to capture graph structural similarity that encompass the previous extensions, and provide a full treatment of these notions for graph matching.

(1) We introduce p-*homomorphism* (p-hom) and *1-1 p-hom* in Section 3. These notions extend graph homomorphism and subgraph isomorphism, respectively, by (a) incorporates similarity metrics to measure *the similarity of nodes*, as opposed to node label equality; and (b) mapping edges in a graph to *paths* in another, rather than edge-to-edge mappings. In contrast to previous extensions, one can use node similarity to assure, *e.g.,* that two Web pages are matched only when they have similar contents [29] or play a similar role (as a hub or authority [6]). Edge-to-path mappings allow us to match graphs that have similar navigational structures but cannot be identified by the conventional notions of graph matching. In addition, these notions can be readily extended to deciding whether two graphs are similar to each other in a *symmetric* fashion.

(2) To provide a quantitative measure of graph similarity, we develop two metrics, also in Section 3, based on (a) the maximum

Proceedings of the VLDB Endowment, Vol. 3, No. 1

number of nodes matched, and (b) the maximum overall similarity when the weights of nodes are taken into account, respectively. These metrics give rise to two natural optimization problems, referred to as the *maximum cardinality* problem and the *maximum similarity* problem, respectively, for each of p-hom and 1-1 p-hom. In particular, the maximum common subgraph problem [19] is a special case of the maximum cardinality problem for 1-1 p-hom.

(3) We establish complexity bounds of the decision problems and optimization problems for p-hom and 1-1 p-hom, in Section 4. We show that the problems for determining p-hom and 1-1 p-hom, as well as the maximum cardinality problem and the maximum similarity problem, are all NP-complete, even for directed acyclic graphs (DAGs). Worse still, the optimization problems are hard to approximate: unless P = NP, it is beyond reach in practice to approximate the problems within $O(1/n^{1-\epsilon})$ of the optimal solutions for any *constant* ϵ. All proofs are given in the appendix.

(4) Nevertheless, we provide in Section 5 approximation algorithms for finding mappings with the maximum cardinality or the maximum similarity, for p-hom and 1-1 p-hom. These algorithms possess *performance guarantees* on match quality: for any graphs G_1 and G_2, the solutions found by the algorithms are provable to be within a *polynomial* $O(\log^2(n_1 n_2)/(n_1 n_2))$ of the optimal solutions, where n_1 (resp. n_2) is the number of nodes in G_1 (resp. G_2).

(5) Using Web site matching as a testbed, we experimentally evaluate our similarity measures in Section 6. We compare p-hom and 1-1 p-hom with three other methods: graph simulation [17], subgraph isomorphism [9] and vertex similarity matrix [21]. Using real-life Web sites and synthetic graphs, we show that our methods outperform those three methods in both match quality and efficiency.

We expect that p-hom and 1-1 p-hom will find applications in Web site classification [5, 12], complex object identification, plagiarism [20] and spam detection [3], among other things.

2. Related Work

There have been extensions of graph matching by allowing edges to map to paths, for trees [24], DAGs [10] or graphs [11, 14, 32]. An approximate retrieval method is proposed for matching trees [24], which identifies and merges regions of XML data that are similar to a given pattern, by using an inverted index. Stack-based algorithms are studied for matching DAGs [10], by leveraging filtering for early pruning. Exponential-time algorithms for matching general graphs are developed in [11], based on join operations over graphs encoded as tables. A notion of XML schema embedding is studied in [14], which is a special case of p-hom with two extra conditions. A form of graph pattern matching is considered in [32], in which edges denote paths with a fixed length. Algorithms for approximate graph matching can also be found in [27, 30]. Most prior work does not consider node similarity in pattern matching, such as all the work mentioned above except [24]. Further, except [14], the complexity of graph matching is not settled; indeed, some algorithms were claimed to be in polynomial time, whereas we show that the problem is NP-hard even for DAGs (Section 4). The complexity bounds of [14] are developed for a different problem, and do not carry over to (1-1) p-hom. In addition, none of the previous algorithms has provable guarantees on match quality, as opposed to the approximation algorithms of this paper.

A variety of methods have been studied for measuring graph similarity, typically following one of three approaches. (a) *Feature-based*: it counts the number of common features in graphs, namely, domain-specific elementary structures, *e.g.,* root-leaf paths [18]. (b) *Structure-based*: it assesses the similarity of the topology of graphs based on simulation [17, 12], subgraph isomorphism (common maximum subgraph) [27, 30], or edit distance [31] (see [9, 26] for surveys). Graph simulation considers edge-preserving relations instead of functions from one graph to another. Graph edit distance is essentially based on subgraph isomorphism. (c) *Vertex similarity*: it builds a matrix of node similarity based on fixpoint computation [6, 21] and the roles of nodes in the structures of the graphs (*e.g.,* hubs or authorities [6]). As pointed out by [25, 30], the feature-based approach does not observe global structural connectivity, and is often less accurate than the structure-based measure. As observed by [4, 23], vertex similarity alone does *not* suffice to identify accurate matches since it ignores the topology of graphs by and large. For Web site matching in particular, it is essential to consider how pages are linked to each other. One cannot match two sites with different navigational structures even if most of their pages can be matched pairwise. Further, vertex similarity requires fixpoint operations and is often too expensive to compute on large graphs. As opposed to previous approaches, we introduce (1-1) p-hom to capture both structural similarity by enforcing edge-to-path mappings, and the contents of individual nodes by incorporating node similarity. In addition, we provide maximum cardinality and maximum overall similarity metrics to quantitatively measure graph similarity, which have not been studied by previous work.

A number of graph matching algorithms have been developed (see [9] for a survey). Our algorithms extend the algorithms of [7, 16] for computing maximum (weighted) independent sets.

3. Revisions of Graph Homomorphism

In this section we first introduce p-homomorphism and 1-1 p-homomorphism. We then present metrics to quantitatively measure graph similarity, and formulate related optimization problems.

3.1 Graphs and Node Similarity

A node-labeled, directed *graph* is defined as $G = (V, E, L)$, where (1) V is a set of nodes; (2) $E \subseteq V \times V$ is a set of edges, in which (v, v') denotes an edge from node v to v'; and (3) for each v in V, $L(v)$ is the *label* of v. The label $L(v)$ may indicate *e.g.,* the content or URL of a Web page [4, 5].

Consider graphs $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$.

We assume a *similarity matrix* $\mathsf{mat}()$. For each pair (v, u) of nodes in $V_1 \times V_2$, $\mathsf{mat}(v, u)$ is a number in $[0, 1]$, indicating how close the labels of v and u are.

The matrix $\mathsf{mat}()$ can be generated in a variety of ways. In Web site matching, for instance, $\mathsf{mat}(v, u)$ for each pair (u, v) of pages may be computed in terms of common *shingles* that u and v share. Here a shingle [8] is a meaningful region contained in a Web page, and $\mathsf{mat}(v, u)$ indicates the textual similarity of u and v. One may also treat vertex similarity matrix [6, 21] as $\mathsf{mat}()$, which measures the hub-authority structural similarity of two nodes [6] and incorporates certain topological structural properties of the graphs.

It may be too expensive to compute vertex similarity matrix on large graphs or to match those graphs. To cope with this we may use "skeletons" of the graphs instead, namely, subgraphs induced from "important" nodes such as hubs, authorities and nodes with a large degree. Indeed, approximate matching is commonly accepted in practice [6, 24, 27, 30]. We compute $\mathsf{mat}()$ for such nodes only.

We use a *similarity threshold* ξ to indicate the suitability of mapping v to u, such that v can be mapped to u only if $\mathsf{mat}(v, u) \geq \xi$.

3.2 P-Homomorphism and 1-1 P-Homomorphism

P-homomorphism. Graph G_1 is said to be *p-homomorphism* (p-hom) to G_2 *w.r.t.* a similarity matrix $\mathsf{mat}()$ and a similarity threshold ξ, denoted by $G_1 \precsim_{(e,p)} G_2$, if there exists a mapping σ from

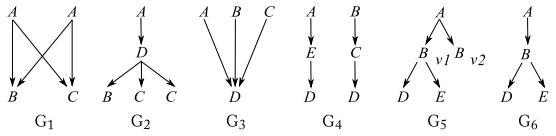


Figure 2: p-hom and 1-1 p-hom

V_1 to V_2 such that for each node $v \in V_1$,
(1) if $\sigma(v) = u$, then $\mathsf{mat}(v, u) \geq \xi$; and
(2) for each edge (v, v') in E_1, there exists a nonempty *path* $u/\ldots/u'$ in G_2 such that $\sigma(v') = u'$, *i.e.,* each edge from v is mapped to a *path* emanating from u.
We refer to σ as *a p-hom mapping* from G_1 to G_2.

Example 3.1: Recall G_p and G of Fig. 1. As shown in Example 1.1, G_p is neither homomorphic nor isomorphic to a subgraph of G. In contrast, suppose that a page checker [8, 29] yields $\mathsf{mat_e}()$:

$\mathsf{mat_e}(A, B) = \mathsf{mat_e}(\mathsf{audio}, \mathsf{digital}) = 0.7$
$\mathsf{mat_e}(\mathsf{books}, \mathsf{books}) = 1.0$
$\mathsf{mat_e}(\mathsf{abooks}, \mathsf{audiobooks}) = 0.8$
$\mathsf{mat_e}(\mathsf{books}, \mathsf{booksets}) = \mathsf{mat_e}(\mathsf{textbooks}, \mathsf{school}) = 0.6$
$\mathsf{mat_e}(\mathsf{albums}, \mathsf{albums}) = 0.85$
$\mathsf{mat_e}(\mathsf{v}, \mathsf{u}) = 0$, for all other node pairs

Then $G_p \precsim_{(e,p)} G$ *w.r.t.* $\mathsf{mat_e}()$ and any threshold $\xi \leq 0.6$. Indeed, the mapping given in Example 1.1 is a p-hom mapping.

To further illustrate p-hom, let us consider the graphs of Fig. 2. In each pair of the graphs, assume that $\mathsf{mat}(v, u) = 1$ if u and v have the same label, and $\mathsf{mat}(v, u) = 0$ otherwise, for nodes v in one graph and u in another. Fix $\xi = 0.5$. One can see the following.
(1) $G_1 \precsim_{(e,p)} G_2$. A p-hom mapping is defined by mapping *both* A nodes in G_1 to the A node in G_2, the node B in G_1 to the B node in G_2, and the node C in G_1 to *any* of the two C nodes in G_2.
(2) $G_3 \not\precsim_{(e,p)} G_4$. Mapping the D node in G_3 to *only one* of the D nodes in G_4 does not make a p-hom mapping, because either the edge (A, D) or (B, D) in G_3 cannot be mapped to a path in G_4.
(3) $G_5 \precsim_{(e,p)} G_6$, for the same reason as (1). □

1-1 p-homomorphism. A graph G_1 is *1-1 p-hom to* G_2, denoted by $G_1 \precsim^{1-1}_{(e,p)} G_2$, if there exists a 1-1 (injective) p-hom mapping σ from G_1 to G_2, *i.e.,* for any distinct nodes v_1, v_2 in G_1, $\sigma(v_1) \neq \sigma(v_2)$. We refer to σ as a *1-1 p-hom mapping* from G_1 to G_2.

Example 3.2: For G_p and G of Fig. 1, the p-hom mapping given in Example 3.1 is also a 1-1 p-hom mapping, *i.e.,* $G_p \precsim^{1-1}_{(e,p)} G$.

As another example, consider G_1 and G_2 of Fig. 2. While $G_1 \precsim_{(e,p)} G_2$, $G_1 \not\precsim^{1-1}_{(e,p)} G_2$. In particular, the p-hom mapping given in Example 3.1 is not injective, since it maps both A nodes in G_1 to the same A node in G_2. Similarly, while $G_5 \precsim_{(e,p)} G_6$, $G_5 \not\precsim^{1-1}_{(e,p)} G_6$ as a p-hom mapping has to map both B nodes in G_5 to the B node in G_6, which is not allowed by a 1-1 mapping. □

Note that subgraph isomorphism is a special case of 1-1 p-hom: G_1 is isomorphic to a subgraph of G_2 iff there exists a 1-1 p-hom mapping σ from G_1 to G_2 that (a) maps each edge (v, v') in G_1 to an edge $(\sigma(v), \sigma(v'))$ in G_2, (b) adopts node label equality, and moreover, (c) if $(\sigma(v), \sigma(v'))$ is an edge in G_2, then (v, v') *must be* an edge in G_1; in contrast, 1-1 p-hom only requires edges from G_1 to find a match in G_2, but not the other way around. Similarly, graph homomorphism is a special case of p-hom.

Remark. For $G_1 \precsim_{(e,p)} G_2$ ($G_1 \precsim^{1-1}_{(e,p)} G_2$) we require an edge-to-path mapping from G_1 to G_2 when G_1 is a *pattern* for a data graph G_2 to match. Nevertheless, (1-1) p-hom can be readily made *symmetric* that maps paths between G_1 and G_2. Indeed, one only need to compute G_1^+, the *transitive closure* of G_1 (in $O(|G_1|^2)$-time [22]), and check whether $G_1^+ \precsim_{(e,p)} G_2$ ($G_1^+ \precsim^{1-1}_{(e,p)} G_2$).

CPH	maximum cardinality for p-hom
CPH^{1-1}	maximum cardinality for 1-1 p-hom
SPH	maximum overall similarity for p-hom
SPH^{1-1}	maximum overall similarity for 1-1 p-hom

Table 1: Notations: Optimization problems

3.3 Metrics for Measuring Graph Similarity

In practice one often wants to measure the similarity of graphs G_1 and G_2 although G_1 may *not* be (1-1) p-hom to G_2. We next provide two metrics that give a quantitative measure of the similarity of two graphs in the range of [0, 1]. Let σ be a p-hom mapping from a subgraph $G_1' = (V_1', E_1', L_1')$ of G_1 to G_2.

Maximum cardinality. This metric evaluates *the number of nodes* in G_1 that σ maps to G_2. The *cardinality* of σ is defined as:

$$\mathsf{qualCard}(\sigma) = \frac{|V_1'|}{|V_1|}.$$

The *maximum cardinality problem* for p-hom (resp. 1-1 p-hom), denoted by CPH (resp. CPH^{1-1}), is to find, given G_1, G_2, $\mathsf{mat}()$ and ξ as input, a (resp. 1-1) p-hom mapping σ from a subgraph of G_1 to G_2 such that $\mathsf{qualCard}(\sigma)$ is maximum.

Observe the following. (1) If $G_1 \precsim_{(e,p)} G_2$ or $G_1 \precsim^{1-1}_{(e,p)} G_2$, then a p-hom mapping σ with maximum $\mathsf{qualCard}(\sigma)$ is a p-hom mapping from the entire G_1 to G_2. (2) The familiar *maximum common subgraph problem* (MCS) is a special case of CPH^{1-1} (recall that MCS is to find a subgraph G_1' of G_1 and a subgraph G_2' of G_2 such that (a) G_1' and G_2' are isomorphic, and (b) the cardinality of G_1' (equivalently, G_2') is maximum; see, *e.g.,* [19]).

Overall similarity. Alternatively, we consider the *overall similarity* of mapping σ. Assume a weight $w(v)$ associated with each node v, indicating relative importance of v, *e.g.,* whether v is a hub, authority, or a node with a high degree. The metric is defined to be

$$\mathsf{qualSim}(\sigma) = \frac{\Sigma_{v \in V_1'}(w(v) * \mathsf{mat}(v, \sigma(v)))}{\Sigma_{v \in V_1} w(v)}.$$

Intuitively, the higher the weight $w(v)$ is and the closer v is to its match $\sigma(v)$, the better the choice of v is. This metric favors "important" nodes in G_1 that can find highly similar nodes in G_2.

The *maximum overall similarity problem* for p-hom (resp. 1-1 p-hom), denoted by SPH (resp. SPH^{1-1}) is to compute, given G_1, G_2, $\mathsf{mat}()$ and ξ as input, a (resp. 1-1) p-hom mapping σ from a subgraph of G_1 to G_2 such that $\mathsf{qualSim}(\sigma)$ is maximum.

These optimization problems are summarized in Table 1.

Example 3.3: Consider graphs G_5 and G_6 shown in Fig. 2. There are two nodes labeled B in G_1, indicated by v_1 and v_2, respectively. A similarity matrix $\mathsf{mat_0}()$ is given as follows:

$\mathsf{mat_0}(A, A) = \mathsf{mat_0}(D, D) = \mathsf{mat_0}(E, E) = \mathsf{mat_0}(v_2, B) = 1$
$\mathsf{mat_0}(v_1, B) = 0.6$ $\quad$ $\mathsf{mat_0}(v, u) = 0$ for other cases

Let $\xi = 0.6$, and assume $w(v) = 1$ for each node v in G_5, except $w(v_2) = 6$. Then G_5 is not 1-1 p-hom to G_6: given $\mathsf{mat_0}()$ and ξ, any p-hom mapping from G_5 to G_6 has to map both v_1 and v_2 in G_5 to the B node in G_6, which is not allowed by a 1-1 mapping. Nevertheless, we can still measure the similarity of G_5 and G_6.

(1) When the maximum cardinality metric is adopted, an optimal 1-1 p-hom mapping σ_c is from a subgraph H_1 of G_5 to G_6, where H_1 contains nodes A, D, E and v_1. Here σ_c maps each node v in G_5 to a node u in G_6 that has the same label as v. The mapping σ_c has maximum cardinality with $\mathsf{qualCard}(\sigma_c) = \frac{4}{5} = 0.8$.

(2) When the maximum similarity metric is used, the optimal 1-1 p-hom mapping σ_s is from a subgraph H_2 of G_5 to G_6, where H_2 consists of nodes A and v_2 only. Here $\mathsf{qualCard}(\sigma_s) = \frac{1*1+6*1}{1+1+1+1+6}$

= 0.7. In contrast, $\mathsf{qualCard}(\sigma_c) = \frac{1*1+1*0.6+1*1+1*1}{1+1+1+1+6} = 0.36$, although σ_c maps more nodes from G_5 to G_6 than σ_s. □

4. Intractability and Approximation Hardness

We next establish complexity bounds for the decision problems and optimization problems associated with p-homomorphism and 1-1 p-homomorphism (see Appendix A for detailed proofs).

Intractability. No matter how desirable, it is intractable to determine whether a graph is p-hom or 1-1 p-hom to another. We remark that while graph homomorphism is special case of p-hom, there is no immediate reduction from the former to the latter, and vice versa; similarly for subgraph isomorphism and 1-1 p-hom.

Theorem 4.1: *Given graphs G_1 and G_2, a similarity matrix* mat() *and a threshold ξ, it is* NP*-complete to decide whether (a) $G_1 \precsim_{(e,p)} G_2$, or (b) $G_1 \precsim^{1-1}_{(e,p)} G_2$. These problems are already* NP*-hard when both G_1 and G_2 are acyclic directed graphs (*DAG*s). It is* NP*-hard for 1-1 p-hom when G_1 is a tree and G_2 is a* DAG. □

In addition, it is unrealistic to expect a polynomial time (PTIME) algorithm for finding an optimal (1-1) p-hom mapping.

Corollary 4.2: *The maximum cardinality problem and the maximum overall similarity problem are* NP*-complete for p-hom and 1-1 p-hom. These problems are already* NP*-hard for* DAG*s.* □

Approximation hardness. In light of Corollary 4.2, the best we can hope for are efficient heuristic algorithms for finding (1-1) p-hom mappings, with performance guarantees on match quality. Unfortunately, CPH, CPH^{1-1}, SPH and SPH^{1-1} are all hard to approximate. Indeed, there exist no PTIME algorithms for finding (1-1) p-hom mappings such that the quality of each mapping found is guaranteed to be within $O(1/n^{1-\epsilon})$ of its optimal counterpart.

Theorem 4.3: *Unless P = NP,* CPH, CPH^{1-1}, SPH *and* SPH^{1-1} *are not approximable within $O(1/n^{1-\epsilon})$ for any constant ϵ, where n is the number of nodes in G_1 of input graphs G_1 and G_2.* □

The hardness is verified by a certain reduction from *the maximum weighted independent set problem* (WIS). In a graph, an independent set is a set of mutually non-adjacent nodes. Given a graph with a positive weight associated with each node, WIS is to find an independent set such that the sum of the weights of the nodes in the set is maximum. It is known that WIS is NP-complete, and is hard to approximate: it is not approximable within $O(1/n^{1-\epsilon})$ for any constant ϵ, where n is the number of nodes [16].

To show the approximation bound, we need to use *approximation factor preserving reduction* (AFP-reduction) [28]. Let Π_1 and Π_2 be two maximization problems. An AFP-reduction from Π_1 to Π_2 is a pair of PTIME functions (f, g) such that

- for any instance I_1 of Π_1, $I_2 = f(I_1)$ is an instance of Π_2 such that $\mathsf{opt}_2(I_2) \geq \mathsf{opt}_1(I_1)$, where opt_1 (resp. opt_2) is the quality of an optimal solution to I_1 (resp. I_2), and
- for any solution s_2 to I_2, $s_1 = g(s_2)$ is a solution to I_1 such that $\mathsf{obj}_1(s_1) \geq \mathsf{obj}_2(s_2)$, where $\mathsf{obj}_1()$ (resp. $\mathsf{obj}_2()$) is a function measuring the quality of a solution to I_1 (resp. I_2).

AFP-reductions retain approximation bounds.

Proposition 4.4:[28] *If (f, g) is an* AFP-reduction *from problem Π_1 to problem Π_2, and if there is a* PTIME *algorithm for Π_2 with performance guarantee α, then there is a* PTIME *algorithm for Π_1 with the same performance guarantee α.* □

Here an algorithm $\mathcal{A}$ has performance guarantee α if for any instance I, $\mathsf{obj}(\mathcal{A}(I)) \geq \alpha\ \mathsf{opt}(I)$. Theorem 4.3 is verified by an AFP-reduction from WIS to each of CPH, CPH^{1-1}, SPH and SPH^{1-1}. That is, these problems are at least as hard as WIS when approximation is concerned.

5. Approximation Algorithms

Despite Theorem 4.3, we next provide approximation algorithms for each of the maximum cardinality problems (CPH, CPH^{1-1}) and the maximum overall similarity problems (SPH, SPH^{1-1}). Optimization techniques are presented in Appendix B.

One of the main results of this section is an approximation bound for CPH, CPH^{1-1}, SPH and SPH^{1-1}: although the problems are not approximable within $O(1/n^{1-\epsilon})$ (Theorem 4.3), we establish a bound $O(log^2(n_1n_2)/(n_1n_2))$. This is verified by AFP-reductions (f, g) from these problems to WIS, by constructing product graphs of G_1 and G_2 (see Appendix A for a detailed proof).

Theorem 5.1: CPH, CPH^{1-1}, SPH *and* SPH^{1-1} *are all approximable within $O(log^2(n_1n_2)/(n_1n_2))$, where n_1 and n_2 are the numbers of nodes in input graphs G_1 and G_2, respectively.* □

Theorem 5.1 suggests naive approximation algorithms for these problems. Given graphs $G_1(V_1, E_1, L_1)$, $G_2(V_2, E_2, L_2)$, a similarity matrix mat() and a similarity threshold ξ, the algorithms (1) generate a product graph by using function f in the AFP-reduction, (2) find a (weighted) independent set by utilizing the algorithms in [7, 16], and (3) invoke function g in the AFP-reduction to get a (1-1) p-hom mapping from subgraphs of G_1 to G_2.

More specifically, for CPH and CPH^{1-1}, we can leverage the approximation algorithm for maximum independent sets given in [7], which is in $O(nm)$ time, where n and m are the numbers of nodes and edges in a graph, respectively. For SPH and SPH^{1-1}, we can use the algorithm of [16] for WIS, which is in $O(nm \log n)$-time. Thus the naive approximation algorithms for maximum cardinality and maximum overall similarity are in $O(|V_1|^3|V_2|^3)$-time and $O(|V_1|^3|V_2|^3 \log(|V_1||V_2|))$-time, respectively.

Although these *naive* algorithms possess performance guarantees, they incur a rather high complexity in both time and space. The cost is introduced by the product graphs, which consist of $O(|V_1||V_2|)$ nodes and $O(|V_1|^2|V_2|^2)$ edges.

We next develop more efficient algorithms that operate directly on the input graphs instead of on their product graph, retaining the same approximation bound. We first present an algorithm for CPH, and then extend the algorithm to CPH^{1-1}, SPH and SPH^{1-1}.

Approximation algorithm for CPH. The algorithm is referred to as compMaxCard and is shown in Figures 3 and 4. Given G_1, G_2, mat() and ξ as input, it computes a p-hom mapping σ from a subgraph of G_1 to G_2, aiming to maximize $\mathsf{qualCard}(\sigma)$.

The algorithm maintains the following data structures to ensure match quality. (a) A *matching list* H for nodes in G_1. For each node v in H, $H[v]$.good collects candidate nodes in G_2 that may match v via the mapping σ; and $H[v]$.minus is the set of nodes in G_2 that v cannot match via σ. (b) A set I of *pairwise contradictory matching pairs* (v, u), where v is a node in G_1 and u is a node in G_2. For any two pairs (v_1, u_1), (v_2, u_2) in I, if v_1 is mapped to u_1, then v_2 cannot be mapped to u_2, and vice versa. (c) An adjacency list H_1 for G_1. For each node v in G_1, $H_1[v]$.prev and $H_1[v]$.post store its "parents" (*i.e.,* the nodes from which there are edges to v) and "children" (*i.e.,* the nodes to which there are edges from v), respectively. (d) An adjacency matrix H_2 for the transitive closure graph G_2^+ of G_2 such that $H_2[u_1, u_2] = 1$ iff (u_1, u_2) is an edge in G_2^+, *i.e.,* there is a nonempty path from u_1 to u_2 in G_2.

Here the *transitive closure* $G^+(V, E^+, L)$ of graph $G(V, E, L)$ is the graph such that for all nodes $v, v' \in V$, $(v_1, v_2) \in E^+$ iff there is a nonempty path from v_1 to v_2 in G.

Algorithm compMaxCard

Input: Two graphs $G_1(V_1, E_1, L_1)$ and $G_2(V_2, E_2, L_2)$, a similarity matrix mat(), and a similarity threshold ξ.
Output: A p-hom mapping from subgraph of G_1 to G_2.

1. **for** each node $v \in V_1$ of graph G_1 **do**
2. $H_1[v]$.prev := $\{v' \mid v' \in V_1, (v', v) \in E_1\}$;
3. $H_1[v]$.post := $\{v' \mid v' \in V_1, (v, v') \in E_1\}$;
4. $H[v]$.good := $\{u \mid u \in V_2, \mathsf{mat}(v, u) \geq \xi\}$; $H[v]$.minus := $\emptyset$;
5. compute the transitive closure $G_2^+(V_2, E_2^+, L_2)$ of graph G_2;
6. **for** each ordered node pair (u_1, u_2) in G_2 **do**
7. **if** $(u_1, u_2) \in E_2^+$ **then** $H_2[u_1][u_2] := 1$; **else** $H_2[u_1][u_2] := 0$;
8. $\sigma_m := \emptyset$;
9. **while** sizeof(H) > sizeof(σ_m) **do**
10. $(\sigma, I) :=$ greedyMatch(H_1, H_2, H); $H := H \setminus I$;
11. **if** sizeof(σ) > sizeof(σ_m) **then** $\sigma_m := \sigma$;
12. **return** σ_m.

Figure 3: Approximation algorithm compMaxCard

The algorithm works as follows. It first constructs the adjacency list H_1 and the matching list H for G_1 (lines 1–4, Fig. 3), where for each v in G_1, $H[v]$.good collects nodes v' in G_2 such that $\mathsf{mat}(v, v') \geq \xi$, and $H[v]$.minus is initially empty. The *transitive closure* graph G_2^+ of G_2 is then computed and stored in adjacency matrix H_2 (lines 5–7). The mapping σ_m is initially $\emptyset$ (line 8), and is computed by a procedure greedyMatch as follows.

In a nutshell, greedyMatch (Fig. 4) picks a node v from H with maximal $H[v]$.good, and a candidate match u from $H[v]$.good. It then recursively computes a mapping σ_1 provided that (v, u) is a match, and a mapping σ_2 without (v, u). It returns the larger one of $\sigma_1 \cup \{(v, u)\}$ and σ_2 to decide whether (v, u) is a good choice. Meanwhile greedyMatch computes sets I_1, I_2 of pairwise contradictory matching pairs and returns the larger one of them as I. It is worth remarking that I is nonempty.

Upon receiving σ and I from greedyMatch (line 10), algorithm compMaxCard removes conflict pairs I from H (line 10) and takes the larger one of σ and σ_m. (line 11). It repeatedly invokes greedyMatch until σ_m is no smaller than H (lines 9–11), *i.e.,* when σ_m covers all the remaining nodes in H to be matched. The quality of the mapping returned (line 12) is guaranteed because (a) greedyMatch always picks the larger one of $\sigma_1 \cup \{(v, u)\}$ and σ_2, and (b) bad choices of I are removed from H at an early stage.

We next give the details of the procedures of compMaxCard.

(a) Procedure greedyMatch (Fig. 4) takes the current matching list H as input. It computes a p-hom mapping σ from a subgraph of $G_1[H]$ to G_2, and a set I of conflict pairs. It selects a candidate match (v, u) as mentioned earlier, moves other nodes in $H[v]$.good to $H[v]$.minus and sets $H[v]$.good to empty set, since v has already picked a match u (lines 2–3). Assuming that (v, u) is a match, it updates H by pruning bad matches for the parent and the children of v in G_1, via another procedure trimMatching (line 4). The updated H is partitioned into two lists, H^+ and H^-, such that for each node v' in H^+, $H[v']$.good is nonempty, *i.e.,* v' may still find a match provided that (v, u) is a match; otherwise v' is included in H^- (lines 5–9). Procedure greedyMatch then recursively computes p-hom mappings σ_1 and σ_2 for $G[H^+]$ and $G[H^-]$, respectively (lines 10–11). It compares the sizes of $\sigma_1 \cup \{(v, u)\}$ (*i.e.,* the mapping with (v, u)) and σ_2 (*i.e.,* the mapping without (v, u)), and returns the *larger* one (lines 12–13). It also computes the set I. If (v, u) is not a good choice then it is included in I_2 (line 12), the set of conflict pairs found when computing σ_2.

(b) Procedure trimMatching (Fig. 4) inputs a candidate match (v, u) and the current matching list H. It removes bad matches from H assuming that (v, u) is a match. That is, for any parent v' in both $H_1[v]$.prev and H, it moves each candidate u' from $H[v']$.good to $H[v']$.minus if there is no path from u' to u in G_2

Procedure greedyMatch

Input: Graphs H_1, H_2, and matching list H for subgraph $G_1[H]$.
Output: A p-hom mapping σ for subgraph $G_1[H]$ to G_2 and a set I of pairwise contradictory matching pairs.

1. **if** H is empty **then return** $(\emptyset, \emptyset)$;
2. pick a node v of H and a node u from $H[v]$.good;
3. $H[v]$.minus := $H[v]$.good $\setminus \{u\}$; $H[v]$.good := $\emptyset$;
4. $H :=$ trimMatching(v, u, H_1, H_2, H);
5. **for** each node v' in H **do** /* partition H into H^+ and H^- */
6. **if** $H[v']$.good is not empty
7. **then** $\{H^+[v']$.good := $H[v']$.good; $H^+[v']$.minus := $\emptyset\}$
8. **if** $H[v']$.minus is not empty
9. **then** $\{H^-[v']$.good := $H[v']$.minus; $H^-[v']$.minus := $\emptyset\}$
10. $(\sigma_1, I_1) :=$ greedyMatch(H_1, H_2, H^+);
11. $(\sigma_2, I_2) :=$ greedyMatch(H_1, H_2, H^-);
12. $\sigma := \max(\sigma_1 \cup \{(v, u)\}, \sigma_2)$; $I := \max(I_1, I_2 \cup \{(v, u)\})$;
13. **return** (σ, I);

Procedure trimMatching

Input: Node v with matching node u, H_1, H_2 and H.
Output: Updated matching list H.

1. **for** each node v' in $H_1[v]$.prev $\cap$ H **do**
/* prune the matching nodes for v's parent nodes */
2. **for** any node u' in $H[v']$.good such that $H_2[u', u] = 0$ **do**
3. $H[v']$.good := $H[v']$.good $\setminus \{u'\}$;
4. $H[v']$.minus := $H[v']$.minus $\cup \{u'\}$;
5. **for** each node v' in $H_1[v]$.post $\cap$ H **do**
/* prune the matching nodes for v's children nodes */
6. **for** any node u' in $H[v']$.good such that $H_2[u, u'] = 0$ **do**
7. $H[v']$.good := $H[v']$.good $\setminus \{u'\}$;
8. $H[v']$.minus := $H[v']$.minus $\cup \{u'\}$;
9. **return** H;

Figure 4: Procedures greedyMatch **and** trimMatching

(lines 1–4), by the definition of p-hom. Similarly, it processes v's children (lines 5–8). The updated H is then returned (line 9).

Example 5.1: We illustrate how compMaxCard computes a p-hom mapping from a subgraph of G_p to G of Fig. 1. For the lack of space we consider subgraphs G_1' and G_2' of G_p and G, respectively, where G_1' is induced by {books, textbooks, abooks}, and G_2' by {books, categories, booksets, school, audiobooks}. We use the similarity matrix $\mathsf{mat}_e()$ of Example 3.1, and fix $\xi = 0.5$. In the following, the nodes labeled with '$*$' are the nodes chosen at line 2 in the procedure greedyMatch.

After step 7, the algorithm constructs an initial matching list H for G_1' (see below), an adjacency matrix H_2 for the transitive closure graph of G_2', and an adjacent list H_1 (G_2' and H_1 are omitted).

Nodes in H	good	bad
books*	{books*, booksets}	$\emptyset$
textbooks	{school}	$\emptyset$
abooks	{audiobooks}	$\emptyset$

The algorithm then calls greedyMatch to produce a subgraph p-hom mapping from G_1' to G_2'. At step 2 of greedyMatch, it maps books to books. After step 9, it splits H into H^+ and H^-, and H^+ is further partitioned into H_a^+ and H_a^- by mapping abooks to audiobooks (shown below with empty lists omitted).

	Nodes	good	minus
H^+	textbooks	{school}	$\emptyset$
	abooks*	{audiobooks*}	$\emptyset$
H^-	books*	{booksets*}	$\emptyset$
H_a^+	textbooks*	{school*}	$\emptyset$

For these lists, σ and I are as follows (empty sets omitted).

	σ	**I**
H_a^+	{(textbooks, school)}	{(textbooks, school)}
H^-	{(books, booksets)}	{(books, booksets)}
H^+	{(textbooks, school), (abooks, audiobooks)}	{(textbooks, school)}
H	{(books, books), (textbooks, school), (abooks, audiobooks)}	{(books, books), (books, booksets)}

After removing I from H, the size of H becomes smaller than that of σ_m, and compMaxCard returns {(abooks, audiobooks), (textbooks, school), (books, books)} as the p-hom mapping. □

Analysis. Algorithm compMaxCard possesses the performance guarantee given in Theorem 5.1 (see Appendix A for a proof).

Proposition 5.2: *For any $G_1(V_1, E_1, L_1)$, $G_2(V_2, E_2, L_2)$, mat() and ξ, algorithm* compMaxCard *finds a p-hom mapping σ from a subgraph of G_1 to G_2 such that* qualCard(σ) *is within $O(log^2(|V_1||V_2|)/(|V_1||V_2|))$ of the optimal quality.* □

One can verify that algorithm compMaxCard is in $O(|V_1|^3|V_2|^2 + |V_1||E_1||V_2|^3)$ time, and is in $O((|V_1| + |V_2|)^2)$ space.

Algorithm compMaxCard can be readily converted to approximation algorithms for CPH$^{1-1}$, SPH and SPH$^{1-1}$, as follows.

Approximation algorithm for CPH$^{1-1}$. A 1-1 p-hom mapping requires that no two nodes in G_1 are mapped to the same node in G_2. Minor changes to compMaxCard suffice to do this: we add an extra step to procedure greedyMatch such that after node v in H is mapped to u in G_2, we remove u from $H[v']$.good and add u to $H[v']$.minus for each node v' in H other than v. The extra step changes neither the worst-case complexity nor the performance guarantee of compMaxCard. This yields an approximation algorithm for CPH$^{1-1}$, referred to as compMaxCard$^{1-1}$.

Approximation algorithms for SPH **and** SPH$^{1-1}$. We develop an approximation algorithm, referred to as compMaxSim, for the maximum overall similarity problem SPH. The algorithm borrows a trick from [16]. The strategy of [16] for computing WIS is as follows. It first removes nodes with weights less than W/n, where W is the maximum node weight and n is the number of nodes in a graph. It then partitions the remaining nodes into $\log n$ groups based on theirs weights, such that the weight of each node in group i ($1 \leq i \leq \log n$) is in the range $[W/2^i, W/2^{i-1}]$. Then for each i, it applies an algorithm for computing maximum independent sets (*e.g.,* the algorithm of [7]) to the subgraph induced by the group i of nodes, and returns the maximum of the solutions to these groups.

Along the same lines, compMaxSim first partitions the initial matching list H into $\log(|V_1||V_2|)$ groups, and then it applies compMaxCard to each group. It returns σ with the maximum qualSim(σ) among p-hom mappings for all these groups. Similarly, an approximation algorithm is developed for SPH$^{1-1}$, referred to as compMaxSim$^{1-1}$. It is easy to verify that these algorithms are in $O(\log(|V_1||V_2|)(|V_1|^3|V_2|^2 + |V_1||E_1||V_2|^3))$ time, and possess the same performance guarantee as compMaxCard.

6. Experimental Study

We next present an experimental study of our matching methods in Web mirror detection. Using real-life and synthetic data, we conducted two sets of experiments to evaluate the ability and scalability of our methods for matching similar Web sites vs. (a) conventional graph simulation [17] and subgraph isomorphism [9], and (b) vertex similarity based on similarity flooding [21].

Experimental setting. We used real-life data and synthetic data.

(1) Real-life data. The real-life data was taken from the Stanford WebBase Project [2], in three categories: Web sites for online stores, international organizations and online newspapers, denoted by *sites 1, 2* and 3, respectively. For each Web site, we found an archive that maintained different versions of the same site.

Using the Web data we generated our graphs as follows. We randomly chose a Web site A in each category. We then produced a set T_A of Web graphs, using data from the archive for A. In each graph, each node was labeled with the content of the page. The similarity between two nodes was measured by the textual similarity of their contents based on *shingles* [8].

Skeletons. These Web graphs are typically large. We thus considered their skeletons that retain only those nodes with a degree above a certain threshold. For each graph G in T_A, we produced its *skeleton* G_s, which is a subgraph of G such that for each node v in G_s, its degree deg$(v) \geq$ avgDeg$(G) + \alpha \times$ maxDeg(G), where avgDeg(G) and maxDeg(G) are the average and maximum node degree in G, respectively, and α is a constant in [0, 1].

Selection of Web graphs. For each Web site A, we generated T_A consisting of 11 graphs representing different versions of A. Based on T_A, we fixed α = 0.2 and produced a set of Web skeletons. Unfortunately, these graphs were beyond the capability of the algorithms we could find for computing maximum common subgraphs [1]. To favor [1], we also chose top 20 nodes with the highest degree, and constructed another set of skeletons. The information about the Web graphs and skeletons is reported in Table 2.

Since each set of the graphs represents different versions (snapshots) of the same Web site, they *should match* each other. Based on this, we evaluated the accuracy of our algorithms. More specifically, after T_A was generated, we sorted the 11 graphs based on their *timestamp* to get a Web graph sequence [23]. We treated the oldest one as *pattern* G_1, and tested whether various approaches could match the 10 later versions to G_1. We used the percentage of matches found as the accuracy measure for all the algorithms.

(2) Synthetic data. We also designed a generator to produce graphs, controlled by two parameters: the number m of nodes and the noise rate noise%. Given m, we first randomly generated a graph *pattern* G_1 with m nodes and $4 \times m$ edges. We then produced a set of 15 graphs G_2 by introducing noise into G_1, with added complexity to make it hard to match G_1. More specifically, G_2 was constructed from G_1 as follows: (a) for each edge in G_1, with probability noise%, the edge was replaced with a path of from 1 to 5 nodes, and (b) each node in G_1 was attached with a subgraph of at most 10 nodes, with probability noise%. The nodes were tagged with labels randomly drawn from a set L of $5 \times m$ distinct labels. The set L was divided into $\sqrt{5 \times m}$ disjoint groups. Labels in different groups were considered totally different, while labels in the same group were assigned similarities randomly drawn from $[0, 1]$.

(3) Algorithms. We have implemented the following, all in Java: (a) all of our algorithms: compMaxCard, compMaxCard$^{1-1}$, compMaxSim, and compMaxSim$^{1-1}$, (b) the graph simulation algorithm of [17], (c) the algorithm of CDK [1] for finding a maximum common subgraph, denoted by cdkMCS, and (d) vertex similarity based on the similarity flooding (SF) algorithm of [21] (we also tested the algorithm of [6], which had results similar to those of SF; for the lack of the space we only report the results of SF).

The experiments were run on a machine with an AMD Athlon 64×2 Dual Core CPU and 2GB of memory. Each experiment was repeated over 5 times and the average is reported here.

Experimental results. We next present our experimental results. In both sets of experiments, we fixed the threshold for matching to be 0.75; *i.e.,* a graph G_1 is said to match G_2 if there is a mapping σ from G_1 to G_2 such that qualCard$(\sigma) \geq 0.75$ (resp. qualSim(σ); see Section 3). We also assumed a uniform weight $w(v) = 1$ for all nodes v when measuring the overall similarity. We used a unified accuracy measure defined above. This is because it is impractical to determine whether two graphs exactly match or not, and the two input graphs were guaranteed to match in all the experiments when generated. Recall that the problems are NP-hard (see Section 4).

Web Sites	Web graphs $G(V, E, L)$				Skeletons 1 ($\alpha = 0.2$)		Skeletons 2 (top-20)	
	# of nodes	# of edges	avgDeg(G)	maxDeg(G)	# of nodes	# of edges	# of nodes	# of edges
Site 1	20,000	42,000	4.20	510	250	10,841	20	207
Site 2	5,400	33,114	12.31	644	44	214	20	20
Site 3	7,000	16,800	4.80	500	142	4,260	20	37

Table 2: Web graphs and skeletons of real life data

Algorithms	Accuracy (%)						Scalability (seconds)					
	Skeletons 1 ($\alpha = 0.2$)			Skeletons 2 (top-20)			Skeletons 1 ($\alpha = 0.2$)			Skeletons 2 (top-20)		
	site 1	site 2	site 3	site 1	site 2	site 3	site 1	site 2	site 3	site 1	site 2	site 3
compMaxCard	80	100	60	80	100	60	3.128	0.108	1.062	0.078	0.066	0.080
compMaxCard$^{1-1}$	40	100	30	80	100	40	2.847	0.097	0.840	0.054	0.051	0.064
compMaxSim	80	100	50	90	100	60	3.197	0.093	0.877	0.051	0.051	0.062
compMaxSim$^{1-1}$	20	80	10	90	100	40	2.865	0.093	0.850	0.053	0.049	0.039
SF	40	30	20	80	80	70	60.275	3.873	7.812	0.067	0.158	0.121
cdkMCS	N/A	N/A	N/A	67	100	0	N/A	N/A	N/A	156.931	189.16	0.82

Table 3: Accuracy and scalability on real life data

Exp-1: Accuracy and efficiency on real-life data. In the first set of experiments, we evaluated the accuracy and efficiency of (1-1) p-hom against the conventional notions of graph matching as well as vertex similarity (SF), using the sets of Web *skeletons*.

In this set of experiments, graph simulation did *not* find matches *in almost all the cases*. This shows that the graph simulation algorithm, which aim at finding matches for an entire graph, is too restrictive when matching Web sites. As a result, we opt to report the results of our approximation algorithms, cdkMCS and SF only.

The accuracy and efficiency results are shown in Table 3. (1) In most cases, our algorithms found more than 50% of matches. (2) The p-hom algorithms found more matches than the 1-1 p-hom ones since the latter pose stronger requirements than the former. (3) All algorithms found more matches on *sites 1* and *2* than *site 3* since a typical feature of *site 3* (online news papers) is its timeliness, reflected by the rapid changing of its contents and structures.

On all graphs in skeletons 1, cdkMCS *did not run* to completion. While compMaxCard and compMaxSim found more than 50% of matches, SF found no more than 40%. On skeletons 2, all of our algorithms found more matches than cdkMCS. In particular, on *site 3* cdkMCS found no matches at all. In contrast, our algorithms found up to 60% of matches on the same data. Compared with SF, all of our algorithms performed better on sites 1 and 2, whereas SF did better on site 3. However, when the size of Web sites increased, the performance of SF deteriorated rapidly.

Our algorithms took less than 4 seconds in all these cases, while cdkMCS took 180 seconds even for graphs with only 20 nodes. Note that although *sites 2* and *3* are about the same size, the running times of cdkMCS on them are not comparable. While the running time of SF was comparable to our algorithms on small Web sites (skeleton 2), it took much longer on large sites (skeleton 1).

From the results we can see the following: our algorithms (1) perform well on both the accuracy and efficiency on different types of Web sites, (2) find more matches than cdkMCS and SF, and (3) are much more efficient and robust than the other two methods.

Exp-2: Accuracy and efficiency on synthetic data. In the second set of experiments, using graphs randomly generated, we evaluated the performance of our algorithms and the graph simulation algorithm of [17], denoted by graphSimulation. However, we could not evaluate cdkMCS and SF, since cdkMCS did not run to completion on large graphs, and SF found constantly *0%* of matches.

We investigated (a) the accuracy of our four algorithms, and (b) the efficiency of these algorithms and graphSimulation. We do not show the accuracy of graphSimulation as it found *0%* of matches in all the cases. We evaluated the effects of the following parameters on the performance: the number of nodes m in G_1, the noise ratio noise% and the node similarity threshold ξ. In each setting, the accuracy was measured by the percentage of matches found between G_1 and a set of 15 graphs (G_2) as mentioned above.

(1) Varying the size of G_1. To evaluate the impact of graph sizes on the accuracy and the scalability, we fixed noise% $= 10\%$ and $\xi = 0.75$, while varying m from 100 to 800, where the number of nodes in G_2 was in the range $[260, 2225]$.

The accuracy results are reported in Fig. 5(a), which show that our approximation algorithms have accuracy above 65%, and are insensitive to the size of G_1. The scalability results are reported in Fig. 6(a), which show that all the algorithms scale well with the size m. The larger G_1 is, the longer the algorithms take, as expected.

(2) Varying the noise. We evaluated the accuracy and performance of the algorithms *w.r.t.* noise%: fixing $m = 500$ and $\xi = 0.75$, we varied noise% from 2% to 20%, where the number of nodes in G_2 was in the range $[650, 2100]$ accordingly.

Figure 5(b) shows that the accuracy of our algorithms is sensitive to the noise rate. But the accuracy is still above 50% even when noise% = 20% and G_2 had 2000 nodes. Figure 6(b) shows that while the scalability of graphSimulation is sensitive to noise%, our algorithms are not. All these algorithms scale well with noise%.

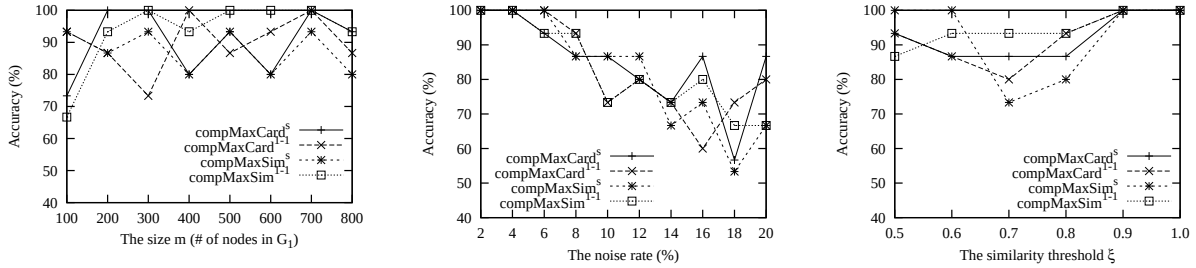
(3) Varying the similarity threshold. Finally, we evaluated the impact of ξ: fixing $m = 500$ and noise% $= 10\%$, we varied ξ from 0.5 to 1.0, where the number of nodes in G_2 was about $1,300$.

Figure 5(c) shows that the accuracy of our approximation algorithms is not very sensitive to ξ, with accuracy above 70% in all the cases. When ξ is between 0.6 and 0.8, the accuracy is relatively lower. This is because (a) when ξ is low ([0.5, 0.6]), it is relatively easy for a node in G_1 to find its matching nodes in G_2; (b) when ξ is high (above 0.8), the chances for each node in G_1 to find its copy in G_2 are higher, by the construction of G_2. Figure 6(c) tells us that the scalability of all these algorithms is indifferent to ξ.

Summary. From the experimental results we find the following. (a) The notions of (1-1) p-hom are able to identify a large number of similar Web sites that are not matched by graph simulation, subgraph isomorphism and vertex similarity. On a set of organization sites, the accuracy of all of our algorithms is above 80%, as opposed to 0%, 0% and 30% by graphSimulation, cdkMCS and SF, respectively. (b) Our algorithms scale well with the sizes of the graphs, noise rates, and similarity threshold. They seldom demonstrated their worst-case complexity. Even for G_1 of 800 nodes and G_2 of 2000 nodes, all of our algorithms took less than two minutes.

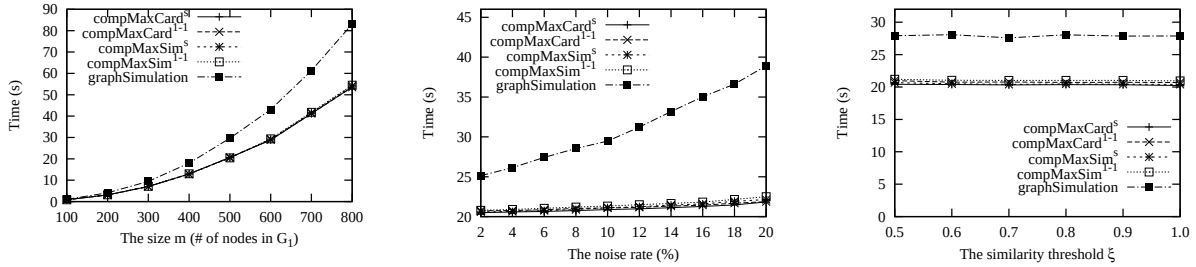
7. Conclusion

We have proposed several notions for capturing graph similarity,



(a) Accuracy *w.r.t.* the size m (b) Accuracy *w.r.t.* the noise rate (c) Accuracy *w.r.t.* the threshold ξ

Figure 5: Accuracy on synthetic data



(a) Scalability *w.r.t.* the size m (b) Scalability *w.r.t.* the noise rate (c) Scalability *w.r.t.* the threshold ξ

Figure 6: Scalability on synthetic data

namely, p-hom, 1-1 p-hom, and quantitative metrics by maximizing either the number of nodes matched or the overall similarity. These notions support edge-to-path mappings and node similarity. We have established the intractability and the hardness to approximate for these problems. Despite the hardness, we have developed approximation algorithms for these problems, with provable guarantees on match quality. We have verified the effectiveness of our techniques using Web site matching as a testbed. Our experimental results have shown our methods are able to identify a number of similar Web sites that cannot be matched either by the conventional notions of graph matching or by vertex similarity alone.

This work is a first step to revising the conventional notions of graph matching. We are exploring areas in which our techniques are effective, beyond Web mirror detection. We also plan to improve our algorithms by leveraging indexing and filtering of [27, 30]. Another topic is to compare the accuracy and efficiency of our methods with the counterparts of the feature-based approaches.

Acknowledgments. Fan, Ma and Wu are supported in part by EPSRC E029213/1.

8. References

[1] Chemistry development kit (cdk). http://sourceforge.net/projects/cdk/.

[2] Stanford webbase. http://diglib.stanford.edu:8091/testbed/doc2/WebBase.

[3] M. Aery and S. Chakravarthy. eMailSift: Email classification based on structure and content. In *ICDM*, 2005.

[4] K. Bharat and A. Broder. Mirror, mirror on the Web: a study of host pairs with replicated content. *Comput. Netw.*, 31(11-16), 1999.

[5] K. Bharat et al. A comparison of techniques to find mirrored hosts on the WWW. *J. Am. Soc. Inf. Sci.*, 51(12), 2000.

[6] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. V. Dooren. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM Rev.*, 46(4), 2004.

[7] R. B. Boppana and M. M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32(2), 1992.

[8] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. *Comput. Netw. ISDN Syst.*, 29(8-13), 1997.

[9] H. Bunke. Graph matching: Theoretical foundations, algorithms, and applications. *Vision Interface*, 3, 2000.

[10] L. Chen, A. Gupta, and M. E. Kurul. Stack-based algorithms for pattern matching on dags. In *VLDB*, 2005.

[11] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang. Fast graph pattern matching. In *ICDE*, 2008.

[12] J. Cho, N. Shivakumar, and H. Garcia-Molina. Finding replicated Web collections. *SIGMOD Rec.*, 29(2), 2000.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[14] W. Fan and P. Bohannon. Information preserving XML schema embedding. *TODS*, 33(1), 2008.

[15] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[16] M. M. Halldórsson. Approximations of weighted independent set and hereditary subset problems. *J. Graph Algorithms Appl.*, 4(1), 2000.

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

[18] S. Joshi et al. A bag of paths model for measuring structural similarity in Web documents. In *KDD*, 2003.

[19] V. Kann. On the approximability of the maximum common subgraph problem. In *STACS*, 1992.

[20] C. Liu, C. Chen, J. Han, and P. S. Yu. Gplag: Detection of software plagiarism by program dependence graph analysis. In *SIGKDD*, 2006.

[21] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *ICDE*, 2002.

[22] E. Nuutila. An efficient transitive closure algorithm for cyclic digraphs. *Inf. Process. Lett.*, 52(4), 1994.

[23] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web graph similarity for anomaly detection. Technical report, 2008.

[24] I. Sanz, M. Mesiti, G. Guerrini, and R. B. Llavori. Fragment-based approximate retrieval in highly heterogeneous XML collections. *Data Knowl. Eng.*, 64(1):266–293, 2008.

[25] A. Schenker, M. Last, H. Bunke, and A. Kandel. Classification of Web documents using graph matching. *IJPRAI*, 18(3), 2004.

[26] D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *PODS*, 2002.

[27] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. In *ICDE*, 2008.

[28] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[29] Webconfs. Similar page checker. www.webconfs.com/similar-page-checker.php.

[30] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *SIGMOD*, 2005.

[31] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou. Edit distance evaluation on graph structures. In *VLDB*, 2009.

[32] L. Zou, L. Chen, and M. T. Özsu. Distance-join: Pattern match query in a large graph database. In *VLDB*, 2009.

Appendix A: Proofs

Proof of Theorem 4.1 (a)

The p-hom problem is to determine, given two graphs $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$, whether $G_1 \preceq_{(e,p)} G_2$. We show that the p-hom problem is NP-complete even when both G_1 and G_2 are DAGs.

We first show that this problem is in NP. An NP algorithm is given as follows: first guess a binary relation $R \subseteq V_1 \times V_2$, and then check whether it is a p-hom mapping. It is in polynomial time (PTIME) to check whether R is a function and whether it is a p-hom mapping from G_1 to G_2.

We next show that this problem is NP-hard by reduction from the 3SAT problem, which is NP-complete (cf. [15]).

An instance ϕ of 3SAT is of the form $C_1 \wedge \cdots \wedge C_n$ where all the variables in ϕ are $x_1, \ldots, x_m$, each clause C_j ($j \in [1, n]$) is of the form $y_{j_1} \vee y_{j_2} \vee y_{j_3}$, and moreover, for $i \in [1, 3]$, y_{j_i} is either $x_{p_{ji}}$ or $\overline{x_{p_{ji}}}$ for $p_{ji} \in [1, m]$. Here we use $x_{p_{ji}}$ to denote the occurrence of a variable in the literal i of clause C_j. The 3SAT problem is to determine whether ϕ is satisfiable.

Given an instance ϕ of the 3SAT problem, we construct two DAGs G_1, G_2 and a similarity matrix mat() such that $G_1 \preceq_{(e,p)} G_2$ if and only if ϕ is satisfiable. The similarity threshold ξ is set to 1.

(1) The DAG $G_1 = (V_1, E_1, L_1)$ is defined as follows:
- $V_1 = \{R_1, C_1, \ldots, C_n, X_1, \ldots, X_m\}$;
- $E_1 = \{(R_1, X_i), (X_{p_{j1}}, C_j), (X_{p_{j2}}, C_j), (X_{p_{j3}}, C_j)\}$ for each $i \in [1, m]$ and each $j \in [1, n]$; and
- we simply let $L_1(v) = v$ for each node $v \in V_1$.

Intuitively, graph G_1 encodes the instance ϕ of 3SAT. Node X_i ($i \in [1, m]$) denotes variable x_i, and node C_j ($j \in [1, n]$) represents clause C_j. Node R_1 is the root of graph G_1, which connects to all X_i nodes ($i \in [1, m]$). An edge (X_i, C_j) in E_1 encodes that variable x_i appears in clause C_j, *i.e.,* x_i is one of the three variables $x_{p_{j1}}, x_{p_{j2}}$ and $x_{p_{j3}}$.

For example, consider an instance for the 3SAT problem: $\phi = C_1 \wedge C_2$, where $C_1 = x_1 \vee x_2 \vee x_3$ and $C_2 = x_2 \vee x_3 \vee x_4$. The corresponding graph G_1 is depicted in Fig. 7 (G_1).

(2) The DAG $G_2 = (V_2, E_2, L_2)$ is defined as follows:
- $V_2 = \{R_2, T, F, X_{T1}, X_{F1}, \ldots, X_{Tm}, X_{Fm}, 0_1, \ldots, 7_1, \ldots, 0_n, \ldots, 7_n\}$.
- $E_2 = \{(R_2, T), (R_2, F)\} \cup \{(T, X_{Ti}), (F, X_{Fi})\} \cup E_2'$, where $i \in [1, m]$.
- E_2' contains 7×3 edges for each clause $C_j = y_{j_1} \vee y_{j_2} \vee y_{j_3}$ of ϕ ($j \in [1, n]$), and there are in total $21n$ edges in E_2'.
  (a) Treating true as 1 and false as 0, we represent the truth assignments of clause C_j in terms of 8 nodes $C_j(\rho)$, where ρ ranges over all truth assignments of variables $x_{p_{j1}}$, $x_{p_{j2}}$ and $x_{p_{j3}}$. Each node $C_j(\rho)$ is a three-bit constant $y_{j_1} y_{j_2} y_{j_3}$ with a subscript j, determined by $\rho(x_{p_{j1}}), \rho(x_{p_{j2}})$ and $\rho(x_{p_{j3}})$, *e.g.,* 2_1.
  (b) For each truth assignment ρ of $x_{p_{j1}}$, $x_{p_{j2}}$ and $x_{p_{j3}}$ *that makes* C_j *true*, E_2' consists of the following edges: $(X_{Tp_{jk}}, C_j(\rho))$ if $\rho(X_{p_{jk}})$ = true, or $(X_{Fp_{jk}}, C_j(\rho))$ if $\rho(X_{p_{jk}})$ = false, where $k \in [1, 3]$.
- $L_2(u) = u$ for each $u \in V_2$.

Intuitively, graph G_2 encodes the truth assignments of the variables that satisfy the clauses in the instance ϕ of 3SAT. Node X_{Ti} ($i \in [1, m]$, resp. X_{Fi}) means assigning variable x_i a true (resp. false) value. Nodes $\{0_j, \ldots, 7_j\}$ represent $C_j(\rho)$, which are denoted as a three-bit constant *w.r.t.* the truth assignments of the three variables in clause C_j. Node R_2 is the root of graph G_2. Nodes T and F are simply included for the ease of exposition. Edges from X_{Ti} or X_{Fi} to nodes $\{0_j, \ldots, 7_j\}$ encode the relationships between the truth assignments of the variables ($x_{p_{j1}}$, $x_{p_{j2}}$ and $x_{p_{j3}}$) and the corresponding $C_j(\rho)$.

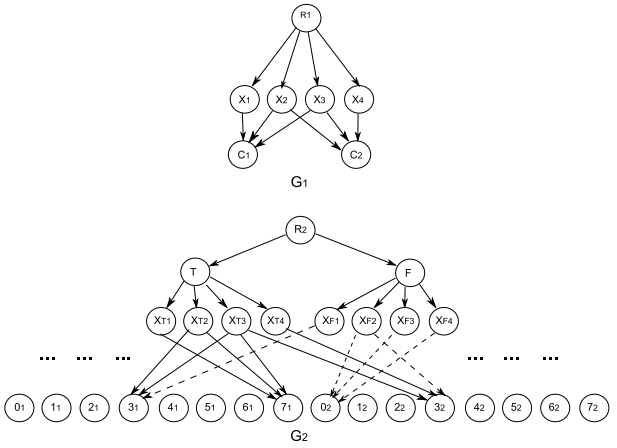


Figure 7: An example reduction for p-hom

For example, graph G_2 corresponding to the 3SAT instance ϕ given above is shown in Fig. 7. Observe that both G_1 and G_2 are DAGs.

(3) The similarity matrix mat() is defined as follows:
- mat$[R_1, R_2] = 1$;
- mat$[X_i, X_{Ti}] = 1$ and mat$[X_i, X_{Fi}] = 1$ for $i \in [1, m]$;
- mat$[C_j, 0_j] = 1, \ldots,$ mat$[C_j, 7_j] = 1$ for $j \in [1, n]$;
- mat$[v, u] = 0$ for any other nodes $v \in V_1$ and $u \in V_2$.

The matrix mat() guarantees that (a) the root R_1 of G_1 must be mapped to the root R_2 of G_2, (b) node X_i ($i \in [1, m]$) in G_1 is mapped to either node X_{Ti} (true) or X_{Fi} (false) of G_2, and (c) node C_j in G_1 ($j \in [1, n]$) is mapped to one of the nodes $\{0_j, \ldots, 7_j\}$ of G_2.

It is easy to verify that the above construction is in PTIME. We next verify that this is indeed a reduction from the 3SAT instance, *i.e.,* there is a p-hom mapping from G_1 to G_2 if and only if the 3SAT instance ϕ is satisfiable.

Assume that there is a p-hom mapping λ from G_1 to G_2. We show that there is a truth assignment ρ that makes ϕ true. The truth assignment ρ is defined as follows. For each variable x_i ($i \in [1, m]$), $\rho(x_i)$ = true if $\lambda(X_i) = X_{Ti}$, and $\rho(x_i)$ = false if $\lambda(X_i) = X_{Fi}$. Note that node X_i in G_1 cannot be mapped to both nodes X_{Ti} and X_{Fi} in G_2 since λ is a function. For each node C_j ($j \in [1, n]$), $\lambda(C_j)$ guarantees that ρ must make clause C_j true, by the construction of graph G_2. Hence the truth assignment ρ indeed makes ϕ true.

Conversely, if there is a truth assignment ρ that makes ϕ true, we show that there is a p-hom mapping λ from G_1 to G_2. The p-hom mapping λ is defined as follows: (1) $\lambda(R_1) = R_2$; (2) for each $i \in [1, m]$, $\lambda(X_i) = X_{Ti}$ if $\rho(x_i)$ = true, and $\lambda(X_i) = X_{Fi}$ if $\rho(x_i)$ = false; and (3) for each $j \in [1, n]$, $\lambda(C_j) = C_j(\rho)$ defined as above. It is easy to verify that λ is indeed a p-hom mapping. □

Proof of Theorem 4.1 (b)

We show that the 1-1 p-hom problem ($G_1 \preceq^{1-1}_{(e,p)} G_2$) is NP-complete even when G_1 is a *tree* and G_2 is a DAG.

We first show that this problem is in NP. An NP algorithm is given as follows: first guess a binary relation $R \subseteq V_1 \times V_2$, and then check whether it is a 1-1 p-hom mapping. It is in polynomial time (PTIME) to check whether R is an injective function and whether it is a p-hom mapping from G_1 to G_2.

We next show that this problem is NP-hard by reduction from

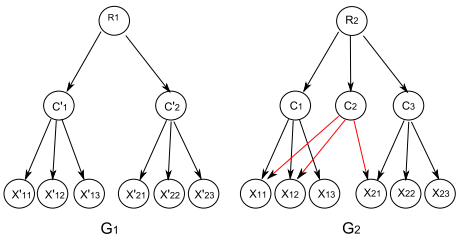


Figure 8: An example reduction for 1-1 p-hom

the exact cover by 3-sets problem (X3C), which is NP-complete (cf. [15]). Given a finite set $X = \{x_1, \ldots, x_{3q}\}$ with $|X| = 3q$ and a collection $S = \{C_1, \ldots, C_n\}$ of 3-element subsets of X, where $C_i = \{x_{i1}, x_{i2}, x_{i3}\}$ for $i \in [1, n]$, the X3C problem is to decide whether there exists an exact cover for X, that is, a sub-collection $S' \subseteq S$ such that S' is a partition of X, *i.e.,* every element of X occurs in exactly one member of S'.

Given an instance I of X3C, we construct two graphs G_1 and G_2 and a similarity matrix $\mathsf{mat}()$ such that there is a 1-1 p-hom mapping from G_1 to G_2 if and only if there exists an exact cover for I. The similarity threshold ξ is set to 1.

(1) The tree $G_1 = (V_1, E_1, L_1)$ is defined as follows:
- $V_1 = \{R_1, C'_1, \ldots, C'_q, X'_{11}, X'_{12}, X'_{13}, \ldots, X'_{q1}, X'_{q2}, X'_{q3}\}$;
- $E_1 = \{(R_1, C'_i), (C'_i, X'_{i1}), (C'_i, X'_{i2}), (C'_i, X'_{i3})\}$ for each $i \in [1, q]$; and
- $L_1(v) = v$ for each node $v \in V_1$.

Intuitively, the tree G_1 encodes the structure of an exact cover S' for the X3C instance I. If there exists such an S', then S' consists of exactly q subsets, and each contains three distinct elements. Node R_1 is the root node of tree G_1. Nodes C'_i ($i \in [1, q]$) denote the subsets in the solution S'. Moreover, we encode the three elements for each subset C'_i ($i \in [1, q]$) with three distinct nodes X'_{i1}, X'_{i2} and X'_{i3}. Edges from node C'_i to nodes X'_{i1}, X'_{i2} and X'_{i3} indicate their relationships.

For example, consider an instance of X3C, where $X = \{X_{11}, X_{12}, X_{13}, X_{21}, X_{22}, X_{23}\}$ and $S = \{C_1, C_2, C_3\}$ such that $C_1 = \{X_{11}, X_{12}, X_{13}\}$, $C_2 = \{X_{11}, X_{12}, X_{21}\}$ and $C_3 = \{X_{21}, X_{22}, X_{23}\}$. The tree G_1 is depicted in Fig. 8 (G_1).

(2) The DAG $G_2 = (V_2, E_2)$ is defined as follows:
- $V_2 = \{R_2, C_1, \ldots, C_n, X_{11}, X_{12}, X_{13}, \ldots, X_{q1}, X_{q2}, X_{q3}\}$;
- $E_2 = \{(R_2, C_i)\} \cup \{(C_i, X_{jk})\}$, where $i \in [1, n]$, $1 \leq j \leq p$, and $X_{jk} \in C_i$ for all $k \in [1, 3]$; and
- $L_2(u) = u$ for each node $u \in V_2$.

Intuitively, DAG G_2 encodes the instance of the X3C problem. Node R_2 is the root of G_2. For each $i \in [1, n]$, node C_i represents the 3-element subset C_i in S, and nodes X_{i1}, X_{i2}, X_{i3} denotes the three elements of C_i. Again, edges from node C_i to nodes X_{i1}, X_{i2} and X_{i3} indicate their relationships.

Referring the X3C instance given above, the DAG G_2 is shown in Fig. 8 (G_2).

(3) The similarity matrix $\mathsf{mat}()$ is defined as follows:
- $\mathsf{mat}[R_1, R_2] = 1$;
- $\mathsf{mat}[C'_i, C_j] = 1$ for $i \in [1, q]$ and $j \in [1, n]$;
- $\mathsf{mat}[X'_{ik}, X_{jg}] = 1$ for $i, j \in [1, q]$ and $k, g \in [1, 3]$;
- $\mathsf{mat}[v, u] = 0$ for any other nodes $v \in V_1$ and $u \in V_2$.

The similarity matrix $\mathsf{mat}()$ guarantees that (a) the root R_1 of G_1 must be mapped to the root R_2 of G_2, (b) node C'_i ($i \in [1, q]$) of G_1 is mapped to node C_j ($j \in [1, n]$) of G_2, and (c) node X'_{ik} in G_1 ($i \in [1, q]$ and $k \in [1, 3]$) is mapped to node X_{jg} in G_2 ($j \in [1, q]$ and $g \in [1, 3]$).

It is easy to verify that the above construction is in PTIME. We next verify that this is indeed a reduction from the X3C instance, *i.e.,* there is a 1-1 p-hom mapping from G_1 to G_2 if and only if there is an exact cover for the X3C instance.

First, suppose that there exists a 1-1 p-hom mapping λ from G_1 to G_2. From the mapping λ, we construct $S' = \{\lambda(C'_i)\}$ for each $C'_i \in V_1$ of G_1 ($i \in [1, q]$). We next show that S' is an exact cover for the X3C instance.

Since the mapping λ is injective, it is easy to verify that (1) $|S'| = q$, and (2) for any two distinct nodes C'_i and C'_j ($i, j \in [1, q]$ and $i \neq j$) in G_1, $\lambda(C'_i) \neq \lambda(C'_j)$, *i.e.,* they are mapped to distinct nodes in G_2. From this it follows that if S' is not an exact cover of S, there must exist $\lambda(C'_i), \lambda(C'_j) \in S'$ ($1 \leq i, j \leq q$ and $i \neq j$) such that $\lambda(C'_i) \cap \lambda(C'_j) \neq \emptyset$. However, this implies that there exist two distinct nodes X'_{ik} (a child of node C'_i) and X'_{jg} (a child of node C'_j) in G_1 such that $\lambda(X'_{ik}) = \lambda(X'_{jg})$, which is *impossible* since λ is injective. Hence, S' is indeed an exact cover.

To illustrate this, let us consider an example. Let λ be a 1-1 p-hom mapping from G_1 to G_2 shown in Fig. 8 such that (1) $\lambda(R_1) = R_2$, (2) $\lambda(C'_1) = C_1$, $\lambda(C'_2) = C_3$, and (3) $\lambda(X'_{ik}) = X_{ik}$ for each $i \in [1, 2]$ and each $k \in [1, 3]$. Consider $S' = \{\lambda(C'_1), \lambda(C'_2)\} = \{C_1, C_3\}$. It is easy to verify that S' is an exact cover for the X3C instance given above.

Conversely, suppose there is an exact cover S' for the X3C instance. We show that there is 1-1 p-hom mapping λ from G_1 to G_2. Assume *w.l.o.g.* that $S' = \{C_{j_1}, \ldots, C_{j_q}\}$ such that $j_i \in [1, n]$ and $C_{j_i} \in S$ for $i \in [1, q]$.

We define a mapping λ as follows: (1) $\lambda(R_1) = R_2$, (2) $\lambda(C'_i) = C_{j_i}$ for $i \in [1, q]$, and (3) $\lambda(X'_{ik}) = X_{j_i k}$ for $i \in [1, q]$ and $k \in [1, 3]$, where $C_{j_i} = \{X_{j_i 1}, X_{j_i 2}, X_{j_i 3}\}$ and $X'_{i1}, X'_{i2}, X'_{i3}$ are the children of C'_i in G_1. Then it is easy to verify that λ is a 1-1 p-hom mapping, using an argument similar to the one given above.

For instance, $S' = \{C_1, C_3\}$ is an exact cover for the X3C instance in Fig. 8. Then the corresponding 1-1 p-hom mapping λ is constructed as follows: (1) $\lambda(R_1) = R_2$, (2) $\lambda(C'_1) = C_1$ and $\lambda(C'_2) = C_3$, (3) $\lambda(X'_{ik}) = X_{ik}$ for $i \in [1, 2]$ and $k \in [1, 3]$. □

Proof of Corollary 4.2

We show that the maximum cardinality problem (MCP) and the maximum overall similarity problem (MSP) are NP-complete for both p-hom and 1-1 p-hom. These problems are already NP-hard when only DAGs are considered.

Given graphs G_1, G_2, similarity matrix mat, threshold ξ, and a rational number K, MCP (resp. MSP) for p-hom (resp. 1-1 p-hom) is to determine whether there exists a p-hom (resp. 1-1 p-hom) mapping σ from G_1 to G_2 such that $\mathsf{qualCard}(\sigma) \geq K$ (resp. $\mathsf{qualSim}(\sigma) \geq K$).

It is easy to verify that these problems are in NP. We next show that there exists a reduction from the p-hom problem to MCP (MSP) for p-hom, and the reduction from the 1-1 p-hom problem to MCP (MSP) for 1-1 p-hom is identical.

Given an instance $I_1 = (G_1, G_2, \mathsf{mat}, \xi)$ of the p-hom problem, we construct an instance $I_2 = (G_1, G_2, \mathsf{mat}', \xi, K)$ of MCP (MSP) such that (1) $K = 1$, (2) $\mathsf{mat}'(v, u) = 1$ for each node v in G_1 and each node u in G_2 such that $\mathsf{mat}(v, u) \geq \xi$, and $\mathsf{mat}'(v, u) = \mathsf{mat}(v, u)$ otherwise. The reduction is trivially in PTIME.

If there is a p-hom mapping σ such that $\mathsf{qualCard}(\sigma) \geq 1$ for MCP or $\mathsf{qualSim}(\sigma) \geq 1$ for MSP in instance I_2, then it is easy to verify that the mapping σ contains all nodes of G_1. From this, it follows that there exists a solution for instance I_1 if and only if there exists a solution for instance I_2. □

Proof of Theorem 4.3

We show that CPH, CPH^{1-1}, SPH and SPH^{1-1} are not approximable within $O(1/n^{1-\epsilon})$ for any constant ϵ, where n is the num-

ber of nodes in G_1, and G_1 and G_2 are input graphs.

We show that there exists an AFP-reduction (f, g) (see Section 4 for a detailed description) from the WIS problem to the SPH problem, from which the conclusion follows since the WIS problem is not approximable within $O(|V_1|^{1-\epsilon})$ for any constant ϵ [16].

We first construct algorithm f. Given an instance I_1 of the WIS problem as its input, algorithm f outputs an instance I_2 of the SPH problem. The instance I_1 is an undirected graph $G(V, E)$ with a positive weight $w(v)$ on each node v. The instance I_2 consists of the following: (1) two directed graphs $G_1(V_1, E_1, L_1)$ and $G_2(V_2, E_2, L_2)$ such that $V_1 = V_2 = V$, E_1 contains the set of (arbitrarily directed) edges in E, $E_2 = \emptyset$, and $L_1(v) = L_2(v) = v$ for each node $v \in V$; (2) a similarity matrix mat() such that $\mathsf{mat}(v, u) = 1$ iff $L_1(v) = L_2(u)$ for any nodes v in G_1 and u in G_2, and $\mathsf{mat}(v, u) = 0$ otherwise; (3) for each node $v \in V_1$, its weight is equal to $w(v)$ on G; and (4) a similarity threshold $\xi = 1$. It is easy to verify that algorithm f is in PTIME.

We then construct algorithm g. Given a feasible solution $s_2 = \{(v_1, v_1), \ldots, (v_n, v_n)\}$ of the SPH instance I_2, algorithm g outputs $s_1 = \{v_1, \ldots, v_n\}$. Algorithm g is trivially in PTIME.

We now show that $(f,\ g)$ is an AFP-reduction from the WIS problem to the SPH problem. Let us consider the following.

Claim 1. Let $s_1 = \{v_1, \ldots, v_n\}$ be a set of nodes of G in the WIS instance I_1, and $s_2 = \{(v_1, v_1), \ldots, (v_n, v_n)\}$ be a mapping of the SPH instance I_2. Then s_2 is p-hom mapping from subgraph $G_1[s_1]$ to graph G_2 in I_2 iff s_1 is an independent set of G in I_1.

This suffices. For if it holds, then we can easily verify that algorithm g produces a solution of the WIS instance I_1, $\mathsf{obj}_1(s_1) = \mathsf{obj}_2(s_2)$, and $\mathsf{opt}_2(I_2) = \mathsf{opt}_1(I_1)$. Recall that (1) $\mathsf{obj}_1()$ (resp. $\mathsf{obj}_2()$) is a function measuring the quality of a solution to I_1 (resp. I_2); and (2) opt_1 (resp. opt_2) is the quality of an optimal solution to I_1 (resp. I_2). From this it follows that $(f,\ g)$ is indeed an AFP-reduction from the WIS problem to the SPH problem.

We next prove **Claim 1**. First, suppose that s_1 is an independent set in I_1. By the definition of p-hom, it is easy to verify that s_2 is a p-hom mapping from subgraph $G_1[s_1]$ to G_2 in I_2.

Conversely, suppose that s_2 is a p-hom mapping from subgraph $G_1[s_1]$ to graph G_2 in I_2. We then show that s_1 is an independent set of graph G in I_1. By the definition of p-hom, (1) each node v_i ($i \in [1, n]$) of s_1 in G_1 is mapped to node $s_2(v_i) = v_i$ in G_2; and (2) for any nodes v_i, v_j ($i \neq j$) in s_1, (v_i, v_j) is not in E_1 since G_2 has no edges at all ($E_2 = \emptyset$). Hence, s_1 is an independent set of graph G_1 in I_2. By the construction of graph G_1 in algorithm f, s_1 is indeed an independent set of graph G in I_1.

For the SPH^{1-1} problem, the above AFP-reduction suffices since the p-hom mapping constructed is indeed injective.

For the CPH (resp. CPH^{1-1}) problem, by setting the weights of all nodes in G_1 to 1, the revised AFP-reduction (f, g) for SPH given above suffices again. □

Proof of Theorem 5.1

We show that CPH, CPH^{1-1}, SPH and SPH^{1-1} are all approximable within $O(log^2(n_1n_2)/(n_1n_2))$, where n_1 and n_2 are the numbers of nodes in input graphs G_1 and G_2, respectively.

It suffices to show that there exists an AFP-reduction (f, g) from the SPH problem to the WIS problem, from which the conclusion follows since the WIS problem is approximable within $O(log^2 n)/n$, where $n = n_1n_2$ is the number of graph nodes [16].

We first design algorithm f. Given an SPH instance I_1 as its input, algorithm f produces a WIS instance I_2. The instance I_1 consists of (1) two directed graphs $G_1(V_1, E_1, L_1)$ and $G_2(V_2, E_2, L_2)$, (2) a similarity matrix mat() on the nodes of G_1 and G_2, and (3) a similarity threshold ξ. Algorithm f first computes the *transitive closure* $G_2^+(V_2, E_2^+, L_2)$ of graph G_2, and then produces an undirected graph $G(V, E)$ with a positive weight on each node based on graphs G_1 and G_2^+. The graph G, a product graph of G_1 and G_2, is built as follows:

(1) $V = \{[v, u] \mid v \in V_1, u \in V_2, \mathsf{mat}(v, u) \geq \xi\}$.

(2) For any nodes $[v_1, u_1]$, $[v_2, u_2]$ in V, there exists an edge from $[v_1, u_1]$ to $[v_2, u_2]$ in E iff they satisfy the following conditions: (a) $v_1 \neq v_2$; (b) if there is a loop (v_1, v_1) (resp. (v_2, v_2)) in G_1, then there must exist a loop (u_1, u_1) (resp. (u_2, u_2)) in G_2^+; and (c) if $(v_1, v_2) \in E_1$, then $(u_1, u_2) \in E_2^+$.

(3) For each node $[v, u]$ in G, its weight is equal to $\mathsf{mat}(v, u)$.

Finally, algorithm f produces a graph $G^c(V, E^c)$, which is the WIS instance I_2. Graph $G^c(V, E^c)$ is the *complement* graph of $G(V, E)$ such that an edge $e \in E^c$ iff $e \notin E$. Here graph G^c allow no self-loops. It is easy to verify that algorithm f runs in PTIME.

We then design algorithm g as follows. Given a feasible solution $s_2 = \{[v_1, u_1], \ldots, [v_n, u_n]\}$ of the WIS instance I_2, g outputs $s_1 = \{(v_1, u_1), \ldots, (v_n, u_n)\}$. Algorithm g is obviously in PTIME.

We now show that $(f,\ g)$ is an AFP-reduction from the SPH problem to the WIS problem. Let us consider the following.

Claim 2. Let $s_2 = \{[v_1, u_1], \ldots, [v_n, u_n]\}$ be a set of nodes in G^c, and $s_1 = \{(v_1, u_1), \ldots, (v_n, u_n)\}$. Then s_2 is an independent set in graph G^c iff s_1 is a p-hom mapping from subgraph $G_1[V_1']$ to G_2 such that $V_1' = \{v_1, \ldots, v_n\}$.

If **Claim 2** holds, then we can easily verify that (1) algorithm g produces a solution (a p-hom mapping) for the SPH instance I_1, (2) $\mathsf{obj}_1(s_1) = \mathsf{obj}_2(s_2)$, and (2) $\mathsf{opt}_2(I_2) = \mathsf{opt}_1(I_1)$. From this it follows that $(f,\ g)$ is indeed an AFP-reduction.

We next prove **Claim 2**. Assume that s_2 is an independent set of G^c. We show that s_1 is a p-hom mapping from $G_1[V_1']$ to G_2. Since s_2 is an independent set of G^c, s_2 is a clique of G. Hence there exists an edge in graph G between any nodes $[v_i, u_i]$ and $[v_j, u_j]$ ($i \neq j$) of s_2. The construction of G guarantees the following: (a) if there is an edge from nodes $[v_i, u_i]$ to $[v_j, u_j]$, then $v_i \neq v_j$, and (b) if there is an edge from v_i to v_j in G_1, then there must exist a path from v_i to v_j in G_2; (c) nodes with self-loops in G_1 must be mapped to nodes with self-loops in G_2^+. Condition (a) guarantees that s_1 is a function; and conditions (a), (b) and (c) together guarantee that s_1 is indeed a p-hom mapping.

Conversely, if s_1 is a p-hom mapping from $G_1[V_1']$ to G_2, we show that s_2 is an independent set of G^c. This is trivial since for any nodes $[v_i, u_i]$ and $[v_j, u_j]$ ($i \neq j$) in s_2, there is an edge $([v_i, u_i], [v_j, u_j])$ in G, and thus no edge in G^c.

To prove the statement for the SPH^{1-1} problem, for each node pair $[v_1, u]$ and $[v_2, u]$ ($v_1 \neq v_2$), we further add an edge $([v_1, u], [v_2, u])$ to G^c given above. This suffices to guarantee that the independent set s_2 corresponds to a 1-1 p-hom mapping.

For the CPH (resp. CPH^{1-1}) problem, by setting the weights of all nodes in G^c to 1, the AFP-reduction (f, g) for SPH (resp. SPH^{1-1}) given above suffices. □

Proof of Proposition. 5.2

We show that given any graphs $G_1(V_1, E_1, L_1)$, $G_2(V_2, E_2, L_2)$, mat() and ξ, algorithm compMaxCard finds a p-hom mapping σ from a subgraph of G_1 to G_2 such that $\mathsf{qualCard}(\sigma)$ is within $O(log^2(|V_1||V_2|)/(|V_1||V_2|))$ of the optimal quality.

As pointed out in Section 5, the AFP-reductions in Theorem 5.1, together with the algorithm for the WIS problem [16] serve as naive approximation algorithms for these problems. These algorithms have the performance guarantee given above. Thus, all we need to

Algorithm ISRemoval
Input: An undirected graph $G(V, E)$.
Output: A clique C of G.

1.i :=1; (I_1, C_1) := Ramsey(G);
2.**while** G is not empty **do**
3. $G := G \setminus I_i$; /*remove independent set I_i from G*/
4. $i := i + 1$; (C_i, I_i) := Ramsey(G);
5.**return** $\max(C_1, C_2, \ldots, C_i)$.

Procedure Ramsey
Input: An undirected graph $G(V, E)$.
Output: An independent set I and a clique C of G.

1.**if** $G = \emptyset$ **then return** $(\emptyset, \emptyset)$;
2.choose some node v of G **do**
3. (C_1, I_1) := Ramsey$(\mathcal{N}(v))$;
/*subgraph $\mathcal{N}(v)$ of G consists of the neighbors of v*/
4. (C_2, I_2) := Ramsey$(\overline{\mathcal{N}}(v))$;
/*subgraph $\overline{\mathcal{N}}(v)$ of G consists of the non-neighbors of v*/
5.$I := \max(I_1, I_2 \cup \{v\})$; $C := \max(C_1 \cup \{v\}, C_2)$;
6.**return** (I, C).

Figure 9: Algorithm ISRemoval

do is to show that given the same input, algorithm compMaxCard produces the same output as those naive algorithms.

To show this, it suffices to show that algorithm compMaxCard simulates algorithm ISRemoval, in a non-trivial way, for finding a maximum clique on the product graph (shown in Fig. 9). Algorithm ISRemoval is the dual of algorithm CliqueRemoval for finding a maximum independent set [7]. Recall that the maximum independent set problem on graph G is equivalent to the maximum clique problem on the complement graph G^c of G, and vice versa.

One can easily see how algorithm compMaxCard in Fig. 3 mimics algorithm ISRemoval. We next show, in detail, how procedure greedyMatch in Fig. 4 simulates procedure Ramsey (see a detailed explanation in [7]). This is based on the following connections:

(1) The matching-list H for graph G_1 corresponds to the product graph $G = G_1 \times G_2$, and each node v in G_1 and another node u in $H[v]$.good or $H[v]$.minus together correspond to the node $[v, u]$ in the product graph G. From these it follows that lines 1 and 2 of greedyMatch simulate lines 1 and 2 of Ramsey, respectively.

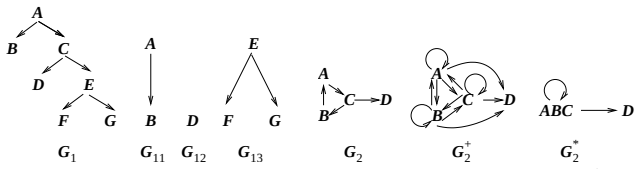
(2) The matching-lists H^+ and H^- correspond to $\mathcal{N}([v, u])$ and $\overline{\mathcal{N}}([v, u])$, respectively, where nodes v, u come from line 2 of greedyMatch. Since computing the neighbors or non-neighbors of a node on graphs is trivial, it is not explicitly addressed in Ramsey. In greedyMatch, however, we need to distinguish neighbors from non-neighbors in the matching-list H, instead of the product graph directly. Procedure trimMatching in Fig. 4 is thus introduced to solve this problem. Indeed, it is trimMatching that makes it possible to operate on the product graph directly.

(3) Procedure greedyMatch(H_1, H_2, H) returns (σ, I), where σ and I correspond to a clique and an independent set in the product graph G respectively, as defined in the proof of Theorem 5.1. From this it follows that lines 10, 11, 12 and 13 of greedyMatch simulate lines 3, 4, 5 and 6 of Ramsey, respectively.

Putting all these together, we have shown that compMaxCard indeed simulates ISRemoval, *i.e.,* given the same input, they always produce the the same output. □

Appendix B: Optimization Techniques

We next propose techniques to improve the efficiency of our algorithms given in Section 5, while retaining or even improving their match quality. These techniques had been implemented when conducting the experiments reported in Section 6.



(a) Partitioning graph G_1 (b) Compressing graph G_2^+

Figure 10: Reducing the graph size

Partitioning graph G_1. Consider the set S_1 of nodes in G_1 such that for any node $v \in S_1$, mat$(v, u) < \xi$ for each node u in G_2. That is, no node in S_1 can find a p-hom match in G_2. Obviously the nodes in S_1 do not contribute to any p-hom mapping from any subgraph of G_1 to graph G_2. Therefore, we only need to consider the subgraph $G_1[V_1 \setminus S_1]$ of G_1 instead of entire G_1, when computing p-hom mappings from G_1 to G_2.

Observe that $G_1[V_1 \setminus S_1]$ may become disconnected even if G_1 is connected. For example, G_1 depicted in Fig. 10(a) is connected, in which node C has no p-hom nodes in G_2. After removing node C from G_1, the remaining subgraph has three pairwise disconnected components G_{11}, G_{12} and G_{13}. It is easy to show:

Proposition 1: *Let graph G_1 consist of k pairwise disconnected components $G_{11}, \ldots$, and G_{1k}. If σ_i is a maximum p-hom mapping from a subgraph of G_{1i} to G_2, then $\bigcup_{i=1}^{i=k}(\sigma_i)$ is a maximum p-hom mapping from a subgraph of G_1 to G_2.* □

This allows us to treat each component separately, and take as the final mapping the union of those mappings for the components. Better yet, if some group G_{1i} contains a single node v, *e.g.,* G_{12} in Fig. 10(a), a match is simply $\{(v, u)\}$, where mat$(v, u) \geq$ mat(v, u') for any other node u' in G_2. Note that finding pairwise disconnected components is linear-time equivalent to finding strongly connected components, which is in linear time [13].

The partitioning strategy may improve match quality. To see this let us examine the approximation bound $y = log^2 n/n$. Obviously, (1) if $n = e^2 \approx 7.39$, y is maximal, where e is the base of the natural logarithms; (2) when $n \geq e^2$, y is monotonically decreasing; and (3) if $n \leq e^2$, it is affordable to use an exact algorithm to find the exact maximum p-hom mapping. Thus when $n \geq e^2$, the larger n is, the worse the performance guarantee is. This tells us that reducing G_1 to $G_1[V_1 \setminus S_1]$ and partitioning $G_1[V_1 \setminus S_1]$ to disconnected components indeed improve match quality.

Compressing graph G_2^+. Each strongly connected component (SCC) in G_2 forms a *clique* in its transitive closure graph G_2^+. By a *clique* in G we mean a set C of nodes such that subgraph $G[C]$ is a complete graph (*i.e.,* any pair of nodes is connected by an edge).

We can replace each clique in G_2^+ with a single node with a self-loop, whose label is the bag of all node labels in the clique. We denote the compressed graph by $G_2^*(V_2^*, E_2^*)$, where each node in V_2^* represents a (maximum) clique in G_2^+, and there exists an edge from nodes u_1^* to u_2^* in G_2^* iff there is an edge from a node in clique u_1^* to a node in clique u_2^* in G_2^+. For example, Figure 10(b) shows a graph G_2, its transitive closure graph G_2^+ and its compressed graph G_2^*. Note that G_2^+ is often much smaller than G_2.

By capitalizing on bags of labels, our algorithms can be modified such that any strong (1-1) p-hom mapping they find from a subgraph of G_1 to G_2^+ is also a strong (1-1) p-hom mapping from a subgraph of G_1 to G_2, with the same quality. By compressing G_2 to G_2^+, the performance of the algorithms is significantly improved. The compressing process incurs little extra cost since SCCs of G_2 can be identified during the computation of G_2^+ [22].