Capturing Structural Similarity of Graphs

Wenfei Fan[1,2,3] Jianzhong Li[3] Shuai Ma[1] Hongzhi Wang[3] Yinghui Wu[1]
[1]University of Edinburgh [2]Bell Laboratories [3]Harbin Institute of Technology
{wenfei@inf.,shuai.ma@,y.wu-18@sms.}ed.ac.uk {lijzh, wangzh}@hit.edu.cn

Abstract

It is essential to identify graphs with similar structures in, *e.g.,* Web page clustering, data deduplication, schema matching and spam detection. To capture structural similarity in these areas, we propose two revisions of the conventional notions of graph simulation and subgraph isomorphism. One revision, referred to as *p-similarity*, extends graph simulation by mapping *edges* from one graph to *paths* in another graph, and by measuring *the similarity of nodes* (*e.g.,* Web page contents). The other revision, referred to as *strong p-similarity*, further restricts simulation by mapping each node in one graph to a single node in the other. When the node mapping is further required to be 1-1, subgraph isomorphism becomes a special case of 1-1 p-similarity. We give a full treatment of these revisions, from complexity bounds to (approximation) algorithms. (1) We present a cubic-time algorithm for computing p-similarity. (2) In contrast, we show that it is NP-complete to determine strong p-similarity and 1-1 p-similarity. (3) We propose optimization problems for strong (1-1) p-similarity, to maximize overall similarity or the number of nodes matched. For each of these problems, we provide an approximation algorithm with *provable guarantees* on match quality. Using real-life and synthetic data, we experimentally verify that our algorithms scale well, and that p-similarities are able to match similar graphs commonly found in practice that cannot be identified by graph simulation or subgraph isomorphism.

1. Introduction

To detect Web mirrors [7] one often needs to identify similar graphs. Indeed, to decide whether a set C of pages is a mirror of a collection C' of pages, one can represent C and C' as graphs G and G', respectively, by treating the pages as nodes and hyperlinks as edges [8, 14, 24]. Then C is a mirror of C' if G' is similar to G based on a certain graph similarity metric [8, 14, 27, 32]. The need for identifying similar graphs is also evident in Web page clustering [10], data deduplication [17], schema matching [30], pattern recognition [37], graph queries [13, 34], plagiarism detection and spam detection [5, 26], among other things.

A central question concerns how to measure the structural similarity of graphs in these contexts. A number of graph matching algorithms have been developed, typically following one of two approaches. One approach is referred to as *feature-based.* Given two graphs, it first extracts domain specific elementary structures as features from the graphs, and then measures their similarity based on the number of common features. The other, referred to as *structure-based*, directly assesses the similarity of the topology of graphs. As pointed out in [32, 37], the feature-based approach does not observe global structural connectivity, and is often less accurate than the structure-based similarity measure.

The structure-based similarity measure for node-labeled graphs often adopts one of the following two notions.

Graph simulation [4]. A graph G_1 is said to be *simulated by* graph G_2 if there exists a binary *relation* R on the nodes of G_1 and the nodes of G_2 such that for each node pair (v, u) in R, (a) v and u share the same label, and (b) for each edge (v, v') in G_1, there is an edge (u, u') in G_2 such that (v', u') is also in R, *i.e.,* each edge in G_1 is mapped to an edge in G_2.

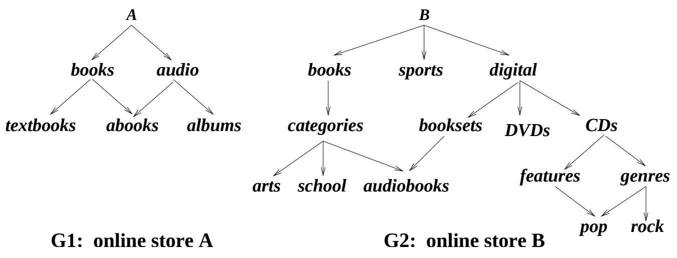


Figure 1: Graphs representing two online stores

Subgraph isomorphism [12]. Graph G_1 is *isomorphic to* a subgraph of G_2 if there exists a *bijective function* f from the nodes of G_1 to the nodes of G_2 such that (a) for any node v in G_1, v and $f(v)$ have the same label, and (b) there exists an edge from nodes v to v' in G_1 if and only if (*iff*) $(f(v), f(v'))$ is an edge in G_2.

These notions have been widely used in, *e.g.,* Web mirrors [14, 32], pattern recognition [37], social networks [34], schema matching [28] and plagiarism detection [26].

However, the conventional notions of graph simulation and subgraph isomorphism are often too restrictive to identify similar graphs in practice. Indeed, the requirement on identical label matching is an overkill for Web mirrors, where nodes are labeled with either Web page contents or URLs [7]; and moreover, edge-to-edge mappings only allow highly similar graphs to be matched.

Example 1.1: Consider an online store A and its later version B depicted as graphs G_1 and G_2 in Fig. 1, respectively. In these graphs, each node represents a Web page for sale of certain items, as indicated by its label. One wants to know whether all the items carried by store A are still provided by its later version B. If so, graph G_1 should logically be considered "similar to" G_2.

When graph simulation or subgraph isomorphism is used to measure graph similarity, G_1 is *not* found similar to G_2. Indeed, (a) nodes in G_1 may not find a node in G_2 with the same label, *e.g.,* audio and albums; and worse still, (b) there exists no sensible binary relation on the nodes V_1 of G_1 and the nodes V_2 of G_2, or mapping from V_1 to V_2, which maps edges in G_1 to edges in G_2 accordingly.

However, a page checker (*e.g.,* [36]) may find each page in store A a match in store B, *i.e.,* a page with a similar functionality:

A $\mapsto$ B, books $\mapsto$ books, audio $\mapsto$ digital, textbooks $\mapsto$ school, abooks $\mapsto$ audiobooks, albums $\mapsto$ {pop,rock} (similar to two nodes)

That is, store B has at least the same capability as its earlier version A. Hence graph G_1 should be considered similar to G_2. Observe that while edges in G_1 are not preserved by this similarity relation on nodes, each edge in G_1 is actually mapped to *paths* in G_2, *e.g.,* the edge (books, textbooks) in G_1 is mapped to the path books/categories/school in G_2. The rational behind this edge-to-path mapping is that when a node v in G_1 is mapped to a node u in G_2, then for any functionality provided by a child of v, a similar functionality can be found in G_2 by following a path from u. □

These suggest that we revise the conventional structure-based measures to match similar graphs commonly found in real world.

Contributions. We propose extensions of graph simulation and subgraph isomorphism to measure the structural similarity of node-labeled directed graphs. We also provide complexity bounds and efficient algorithms for computing these revised measures.

(1) Our first contribution is a notion of p-*similarity* that extends graph simulation in two aspects. (a) In contrast to node label equality, similarity metrics are introduced to measure *the similarity of nodes*. This allows us to, *e.g.,* leverage page checkers (*e.g.,* [36]) to match similar Web pages. (b) Edges in one graph are allowed to map to *paths* in the other, as opposed to edge-to-edge mappings.

(2) Our second contribution is a notion of *strong* p-*similarity*, which restricts p-similarity by requiring each node in one graph to be mapped to a single node in the other, rather than to a set of nodes. That is, strong p-similarity is defined in terms of a *function* for node mapping, instead of a relation. This is often needed in, *e.g.,* graph-pattern queries [13] and schema matching [30]. In schema matching, for example, one often wants to map each attribute in a source schema to a single attribute in a target schema.

We also study a further restriction of strong p-similarity by requiring node mappings to be 1-1 (injective), referred to as *1-1* p-*similarity*. Subgraph isomorphism is a special case of 1-1 p-similarity when (a) edges in one graph are only allowed to map to edges in the other, and *vice versa*, and in addition, (b) node-label equality is required instead of node similarity.

(3) Our third contribution is an $O(n^3)$-time algorithm that, given two graphs G_1 and G_2, computes p-similarity relations from G_1 to G_2 if one exists. Compared to the $O(n^2)$-time complexity of graph simulation, the algorithm tells us that despite edge-to-path mappings, p-similarity does not make our lives much harder, *i.e.,* one can still efficiently identify similar graphs using p-similarity. We also show that for any G_1 and G_2, if they are p-similar, then there exists a *unique* p-similarity relation with the maximum cardinality.

(4) Our fourth contribution consists of complexity bounds for determining strong p-similarity. In contrast to the $O(n^3)$-time complexity of p-similarity, we show that the problems for determining strong p-similarity and 1-1 p-similarity both become NP-complete.

We also study subgraph matching with strong (1-1) p-similarity. We propose optimization problems to maximize the number of nodes matched (referred to as *maximum cardinality*), or to maximize the overall similarity of mappings *w.r.t.* a certain similarity metric (*maximum similarity*). In particular, the maximum common subgraph problem [23] is a special case of maximum cardinality.

(5) Our fifth contribution is a set of approximation algorithms for finding node mappings with maximum cardinality or maximum similarity, for strong p-similarity or 1-1 p-similarity. We show that all these optimization problems are also NP-complete. Worse still, they 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. Nevertheless, we provide effective algorithms for each of these problems. These algorithms possess provable *performance guarantees* on match quality: for any graphs G_1 and G_2, the solutions found by the algorithms are provable to be within $O(\log^2(n_1n_2)/(n_1n_2))$ of the optimal solutions, where n_1 (resp. n_2) is the number of nodes in G_1 (resp. G_2).

(6) Our sixth and final contribution is an experimental study. Using real-life Web pages we evaluate the accuracy of our similarity measures. We find that (strong, 1-1) p-similarity can identify many similar graphs that conventional measures fail to detect. On a set of Web sites for online newspapers, for example, our approximation algorithms find up to 60% of matches; in contrast, graph simulation finds no match, and algorithms based on subgraph isomorphism either do not run to completion or cannot find any match. On Web sites for online stores, the accuracy of our algorithms is up to 90%, as opposed to 0% and 67% by simulation and subgraph isomorphism, respectively. Using synthetic graphs we evaluate the efficiency of our algorithms. We find that the algorithms scale well. For example, they take less than two minutes on graphs G_1 with 800 nodes and G_2 with 2000 nodes.

Applications. The notions of p-similarity and strong (1-1) p-similarity provide alternative measures for structural similarity of graphs. Their (approximation) algorithms allow us to use these measures with performance guarantees. These techniques may find practical use in the following areas.

(1) *Web page classification and duplicate page detection.* It is estimated over 30% of the Web consists of duplicate pages (cf. [7]). This highlights the need for detecting duplicate pages and for classifying Web pages, to reduce crawling and storage costs, and to improve the accuracy of page ranks. As verified by our experimental results, the notions of (strong) p-similarity allow us to effectively identify Web sites with similar structures and functionality.

(2) *Object identification.* In data integration and data cleaning one typically needs to identify objects from multiple unreliable data sources that refer to the same real-world entity [6, 17]. While object identification has been studied for decades, little is known about how to identify complex objects, *e.g.,* semistructured data. Representing such objects as graphs, (strong) p-similarity is able to identify complex objects by checking whether all the attributes of one object are also accessible via paths in another object, although the two objects may not have highly similar structures.

(3) *Schema matching.* As a first step to building schema mapping, schema matching is to map attributes from a source schema to those in a target schema such that the associated attributes are semantically related [30]. As schemas are typically represented as graphs, strong p-similarity can be used to find candidate schema matches.

(4) *Plagiarism and spam detection.* Source code can be characterized by program dependency graphs [26]. Contents and structures of documents and emails can also often be represented as graphs [5]. These promote the use of p-similarity in plagiarism detection and email classification, which typically need a similarity measure less restrictive than conventional subgraph isomorphism.

In addition, p-similarities and their computation techniques also shed light on pattern recognition [15, 37], graph-pattern query evaluation [13] and social networks [34], among other things.

Organization. We introduce p-similarity in Section 2, and give its computation algorithm in Section 3. In Section 4 we define strong p-similarity and 1-1 p-similarity, and show their NP-completeness. The optimization problems are also introduced there. Approximation algorithms are provided in Section 5, followed by experimental results in Section 6. Related work is discussed in Section 7, followed by topics for future work in Section 8. All proofs are in [2].

2. From Graph Simulation to P-Similarity

We first review basic notations, and then define p-similarity.

2.1 Graphs, Subgraphs, Paths and Node Similarity

A node-labeled, directed *graph* is defined as $G = (V, E, L)$, where (1) V is a finite 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 node v in V, $L(v)$ is the *label* of v. The label $L(v)$ may

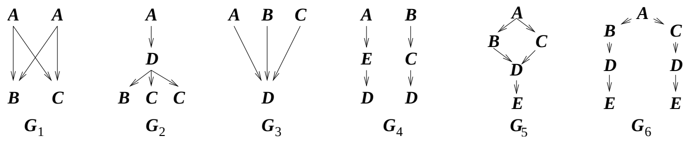


Figure 2: Graph simulation vs. p-similarity

simulation	p-similarity	strong p-similarity	1-1 p-similarity
$\preceq_{(e,e)}$	$\preceq_{(e,p)}$	$\precsim^{s}_{(e,p)}$	$\precsim^{1-1}_{(e,p)}$

Table 1: Notations: Simulation and p-similarity

indicate, *e.g.,* the content or URL of a page if G represents a set of Web pages [7, 8], or an attribute if G depicts a schema [30].

A *node-induced subgraph* (or simply a *subgraph*) of a graph G is defined to be a subset of V together with all those edges whose endpoints are both in this subset. More specifically, a *graph* $H = (V_s, E_s, L_s)$ is a *subgraph* of graph $G = (V, E, L)$, denoted as $G[V_s]$, if for each node $v \in V_s$, v is in V, $L_s(v) = L(v)$, and moreover, $(v, v') \in E_s$ iff $v, v' \in V_s$ and $(v, v') \in E$.

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 $i \in [1, n-1]$. A path is *nonempty* if it consists of at least one edge. Abusing notations 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 $i \in [2, n]$.

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

We assume a *similarity matrix* $\mathsf{mat}()$ on V_1 and V_2. For each pair (v, u) of nodes in $V_1 \times V_2$, $\mathsf{mat}(v, u)$ is a number in the range $[0, 1]$, indicating how close the labels of v and u are. The matrix $\mathsf{mat}()$ may be produced by a page checker for comparing the contents of Web pages [36], or by an algorithm for identifying attribute correspondences across schemas [25]. We also use a *similarity threshold* ξ to indicate the suitability of mapping v to u, such that v can be mapped to u only if $\mathsf{mat}(v, u) \geq \xi$.

2.2 P-Similarity and Graph Simulation

P-similarity. Graph G_1 is said to be *p-similar* to G_2 *w.r.t.* similarity matrix $\mathsf{mat}()$ and threshold ξ, denoted by $G_1 \preceq_{(e,p)} G_2$ when $\mathsf{mat}()$ and ξ are clear in the context, if there exists a binary relation $R_{(e,p)} \subseteq V_1 \times V_2$ such that for each $(v, u) \in R_{(e,p)}$,

(1) $\mathsf{mat}(v, u) \geq \xi$, *i.e.,* the *similarity* of the labels of v and u is above the threshold;

(2) for each edge (v, v') in E_1, there exists a nonempty *path* $u/\ldots/u'$ in G_2 such that $(v', u') \in R_{(e,p)}$, *i.e.,* all the children of v match some *descendants* of u via $R_{(e,p)}$; and

(3) for every node v in V_1 there exist (possibly multiple) nodes u in V_2 such that (v, u) is in $R_{(e,p)}$, *i.e.,* each node in G_1 is mapped to a nonempty set of nodes in G_2.

We refer to $R_{(e,p)}$ as a *p-similarity relation* from G_1 to G_2.

Graph simulation. Recall (from *e.g.,* [4]) that G_1 is said to be *simulated* by G_2, denoted by $G_1 \preceq_{(e,e)} G_2$, if there is a binary relation $R_{(e,e)} \subseteq V_1 \times V_2$ such that for each $(v, u) \in R_{(e,e)}$, (1) $L_1(v) = L_2(u)$, *i.e.,* v and u have the *same* label; (2) for each edge (v, v') in E_1, there is an edge (u, u') in E_2 such that $(v', u') \in R_{(e,e)}$, *i.e.,* all the children of v match *children* of u; and (3) for each node v in V_1, $(v, u) \in R_{(e,e)}$ for some u in V_2.

Note that graph simulation is a special case of p-similarity when (1) node label equality is required instead of similarity, and (2) edges in G_1 are only allowed to map to edges in G_2.

Example 2.1: Consider graphs G_1 and G_2 depicted in Fig. 1. As observed in Example 1.1, $G_1 \not\preceq_{e,e} G_2$. In contrast, suppose that a page checker yields a matrix $\mathsf{mat_e}()$:

$\mathsf{mat_e}(A, B)$ = $\mathsf{mat_e}$(audio, digital) = 0.7
$\mathsf{mat_e}$(books, books) = 1.0
$\mathsf{mat_e}$(abooks, audiobooks) = 0.9
$\mathsf{mat_e}$(books, booksets) = $\mathsf{mat_e}$(textbooks, school) = 0.6
$\mathsf{mat_e}$(albums, rock) = $\mathsf{mat_e}$(albums, pop) = 0.65
$\mathsf{mat_e}$(v, u)= 0, for all other node pairs

Then $G_1 \preceq_{(e,p)} G_2$ *w.r.t.* $\mathsf{mat_e}()$ and any threshold $\xi \leq 0.6$. Indeed, the node mapping given in Example 1.1 is a p-similarity relation that maps edges in G_1 to paths in G_2:

$(A$, books)	$\mapsto$	B/books
$(A$, audio)	$\mapsto$	B/digital
(books, textbooks)	$\mapsto$	books/categories/school
(books, abooks)	$\mapsto$	books/categories/audiobooks
(audio, albums)	$\mapsto$	{digital/CDs/genres/pop, digital/CDs/genres/rock}

Since node albums in G_1 is mapped to both rock and pop in G_2, the edge (audio, albums) in G_1 is mapped to two paths in G_2.

As another example, consider graphs depicted in 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 ξ = 0.5. One can see the following.

(1) $G_1 \not\preceq_{e,e} G_2$ since there is no binary relation that preserves the edges of G_1 in G_2, *e.g.,* (A, B) in G_1 cannot find a match in G_2. In contrast, $G_1 \preceq_{(e,p)} G_2$; an example p-similarity relation consists of (v, u) for each v in G_1 and u in G_2 that have the same label. Here edge (A, B) in G_1 is mapped to a path $A/D/B$ in G_2.

(2) $G_3 \not\preceq_{e,e} G_4$ but $G_3 \preceq_{(e,p)} G_4$, for the same reason as (1).

(3) $G_5 \preceq_{(e,e)} G_6$ and $G_5 \preceq_{(e,p)} G_6$. Indeed, a p-similarity relation $\preceq_{(e,p)}$ is the same as the one in (1), so is a simulation relation. Observe that the node labeled D in G_5 is *necessarily* mapped to both nodes labeled D in G_6; similarly for the E node in G_5. $\square$

This example shows that there exist graphs G_1, G_2 such that G_1 is p-similar to G_2 whereas G_1 is *not* simulated by G_2. In contrast:

Proposition 2.1: *For any graphs G_1, G_2, matrix* $\mathsf{mat}()$ *and threshold ξ, if $G_1 \preceq_{(e,e)} G_2$ then $G_1 \preceq_{(e,p)} G_2$.* $\square$

Proof sketch: A simulation relation is also a p-similarity relation since each edge is a path and every similarity relation subsumes equality. $\square$

This shows that simulation is a stronger notion than p-similarity. As remarked earlier, simulation is often too restrictive: a graph can only be mapped to *highly similar* graphs by simulation.

Remark. For the ease of reference we summarize various notions of similarity in Table 1, including those given in this section and notations to be introduced in Section 4.

One might be tempted to further relax simulation, by allowing path-to-path mappings instead of edge-to-path mappings. This can be formalized by the notion of $\preceq_{(p,p)}$.

We say that $G_1 \preceq_{(p,p)} G_2$ *w.r.t.* $\mathsf{mat}()$ and ξ if there is a binary relation $R_{(p,p)} \subseteq V_1 \times V_2$ such that for each (v, u) in $R_{(p,p)}$, conditions (1) and (3) for p-similarity hold and moreover, (2) for each nonempty *path* $v/\ldots/v'$ in G_1, there exists a nonempty *path* $u/\ldots/u'$ in G_2 such that $(v', u') \in R_{(p,p)}$, *i.e.,* all the descendants of v map to some descendants of u via $R_{(p,p)}$.

The result below tells us, however, that $\preceq_{(p,p)}$ and $\preceq_{(e,p)}$ are *equivalent*, *i.e.,* $\preceq_{(p,p)}$ does not gain any extra power.

Proposition 2.2: *For any graphs G_1 and G_2, $G_1 \preceq_{(p,p)} G_2$ iff $G_1 \preceq_{(e,p)} G_2$.* $\square$

Proof sketch: If $G_1 \preceq_{(p,p)} G_2$, then each edge in G_1 is mapped to a path in G_2 via $R_{(p,p)}$, since an edge is a nonempty path itself. Thus from $R_{(p,p)}$ a p-similarity relation can be readily derived. Conversely, suppose that $G_1 \preceq_{(e,p)} G_2$. Then for each nonempty path $\rho = v_1/\ldots/v_k$ in G_1, one can find a nonempty matching path in G_2 by concatenating nonempty paths that match (v_i, v_{i+1}) via $R_{(e,p)}$. This yields an $R_{(p,p)}$ relation. □

In light of this result we shall focus on p-similarity.

3. An Algorithm for Computing P-Similarity

There are possibly multiple p-similarity relations from G_1 to G_2. We want to find the *maximum* p-similarity relation $R_{(e,p)}$, *i.e.,* $R_{(e,p)}$ has the maximum cardinality (the number of elements) among all p-similarity relations. Intuitively, $R_{(e,p)}$ covers all similar matches from nodes of G_1 to nodes of G_2.

In this section we focus on the following problem.

Given G_1, G_2, mat() and ξ, *the problem for computing the maximum p-similarity* is to find the maximum p-similarity relation from G_1 to G_2 *w.r.t.* mat() and ξ, if there exists one.

The main result of this section is an $O(n^3)$-time algorithm for computing maximum p-similarity. Compared with the $O(n^2)$-time complexity of graph simulation, this tells us that while the power of p-similarity is not for free, it does not increase the complexity too much. Moreover, we show that there is a *unique* maximum p-similarity relation if $G_1 \preceq_{(e,p)} G_2$.

Overview. An algorithm was proposed in [21] that, given a single graph $G = (V, E, L)$, finds a (maximum) simulation relation from G to G in $O(|V|\,|E|)$ time, if there is one. We extend this algorithm to compute the maximum p-similarity relation from a graph G_1 to another graph G_2. To simplify the discussion, we first present a simple algorithm to illustrate conceptual-level computation. We then show how the algorithm can be improved to be in cubic-time.

Algorithm. Our algorithm, referred to as compSimilarity, is shown in Fig. 3. Given $G_1 = (V_1, E_1, L_1)$, $G_2 = (V_2, E_2, L_2)$, mat() and ξ, it returns a maximum p-similarity relation from G_1 to G_2 if $G_1 \preceq_{(e,p)} G_2$, and returns empty set $\emptyset$ otherwise.

Before we illustrate the algorithm, we first present notations it uses. The *transitive closure* $G^+(V, E^+, L)$ of graph $G(V, E, L)$ is a graph such that for all nodes $v, v' \in V$, $(v, v') \in E^+$ iff there is a nonempty path from v to v' in G, *i.e.,* v' is a descendant of v. One can compute G^+ in $O(|V||E|)$ time (see [29] for details).

The algorithm also uses the following variables. (a) A set sim(v) for each node v in graph G_1, which maintains candidate nodes in G_2 to which v is possibly p-similar. (b) A set post(v) consisting of all the children of node v in G_1, *i.e.,* if $(v, v') \in E_1$ then $v' \in$ post(v). (c) A set desc(u) for each node u in graph G_2, which contains all the descendants of u in G_2. Note that desc(u) in G_2 is the same as post(u) in G_2^+, *i.e.,* for any node u', $u' \in$ desc(u) iff (u, u') is an edge in the transitive closure G_2^+ of G_2. Thus for all $u \in V_2$, desc(u) can be computed in $O(|V_2|\,|E_2|)$ time.

Algorithm compSimilarity first constructs sim(v) for each node $v \in V_1$, by collecting all such nodes u in G_2 that mat$(v, u) \geq \xi$. Then it removes those nodes u from sim(v) to which v cannot be p-similar. That is, u is not a candidate match if there exists a child v' of v ($v' \in$ post(v)) such that v' cannot find p-similar matches reachable from u in G_2, *i.e.,* desc$(u) \cap$ sim$(v') = \emptyset$. The process repeats until no further changes can be made. Then the nodes remaining in sim(v) are those to which v is p-similar.

More specifically, compSimilarity first computes sim(v) and post(v) for each node v in G_1 (lines 1–3). For each node u in graph G_2, it finds its descendant set desc(u) (lines 4–6), by computing the transitive closure of G_2. After the prepossessing, for each node v in G_1, it repeatedly removes invalid p-similar nodes from sim(v) based on the definition of p-similarity (lines 7–9). Finally, it checks whether the surviving matches form a p-similarity relation, *i.e.,* whether sim(v) is nonempty for each $v \in V_1$. If so, it returns a p-similarity relation; it returns $\emptyset$ otherwise (lines 10–12).

```
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 $\xi$.
Output: A maximum $p$-similarity relation from $G_1$ to $G_2$.

1.  for each node $v \in V_1$ of graph $G_1$ do
2.    sim$(v)$ := $\{u \mid u \in V_2$ and mat$(v, u) \geq \xi\}$;
3.    post$(v)$ := $\{v' \mid (v, v') \in E_1\}$;
4.  compute the transitive closure $G_2^+(V_2, E_2^+, L_2)$ of graph $G_2$;
5.  for each node $u \in V_2$ of graph $G_2$ do
6.    desc$(u)$ := $\{u' \mid (u, u') \in E_2^+\}$;
7.  repeat until no further changes
8.    if there exist nodes $v \in V_1$, $v' \in$ post$(v)$ and $u \in$ sim$(v)$
          such that desc$(u) \cap$ sim$(v')$ is empty
9.    then remove $u$ from sim$(v')$;
10. if for each $v \in V_1$, sim$(v)$ is nonempty
11. then return $\{(v, u) \mid v \in V_1, u \in$ sim$(v)\}$
12. else return $\emptyset$.
```

Figure 3: Algorithm compSimilarity

Example 3.1: We illustrate how algorithm compSimilarity finds p-similarity relation for graphs in Fig. 1. For the lack of space we consider subgraphs G_1' and G_2' of G_1 and G_2, respectively, where G_1' is induced by {books, textbooks, abooks}, and G_2' by {books, categories, booksets, school, audiobooks}. We use the similarity matrix mat$_e$() of Example 2.1, and fix $\xi = 0.5$.

After step 6, compSimilarity produces the following.

Nodes in G_1'	sim()	post()
books	{books, booksets}	{textbooks, abooks}
textbooks	{school}	$\emptyset$
abooks	{audiobooks}	$\emptyset$

Nodes in G_2'	desc()
books	{categories, school, audiobooks}
booksets	{audiobooks}
categories	{school, audiobooks}
school	$\emptyset$
audiobooks	$\emptyset$

Then for each node v in G_1', the algorithm removes nodes from sim(v) to which v is not p-similar. For example, booksets is removed from sim(books) because the child textbooks of books in G_1' cannot find p-similar nodes reachable from booksets in G_2'. After step 9, we get a nonempty set of p-similar nodes in G_2' for each node in G_1', and the algorithm returns a p-similarity relation {(books, books), (textbooks, school), (abooks, audiobooks)}. □

The result below shows that maximum p-similarity relations are unique. Furthermore, the algorithm correctly finds it.

Proposition 3.1: *For any G_1, G_2, mat() and ξ, $G_1 \preceq_{(e,p)} G_2$ iff (1) there is a unique maximum p-similarity relation from G_1 to G_2 w.r.t. mat() and ξ, and (2) algorithm compSimilarity correctly finds the maximum p-similarity relation.* □

Proof sketch: It is easy to see that after the loop (lines 7–9), for each u remaining in sim(v), v is p-similar to u. Thus if the algorithm returns a nonempty set R, then R is a p-similarity relation. Moreover, R is maximum because (a) compSimilarity starts with all possible p-similar nodes for each node v in G_1; and (b) the loop only drops those nodes to which v cannot possibly be p-similar. Furthermore, it can be verified that the order of removing nodes is irrelevant. Thus maximum p-similarity relations are unique, and the algorithm finds it as long as it exists. □

Improvement and analysis. For the complexity of the algorithm, observe the following. (a) It takes $O(|V_1||V_2| + |V_1||E_1|)$ time to pre-process the nodes in G_1 (lines 1–3). (b) Computing the transitive closure of G_2 (line 4) and processing the nodes in G_2 (lines 5–6) can be combined and done at the same time, which takes $O(|V_2||E_2|)$ time in total. However, the loop (lines 7–9) may be expensive. Below we improve the loop to be within cubic-time.

The idea is to trade space for time. To do this we introduce and maintain more structures, including (a) a set $\mathsf{remv}(v)$ for each node v in graph G_1, consisting of nodes in G_2 that should be removed from $\mathsf{sim}(v)$, which is computed by inspecting $\mathsf{sim}(v')$ and $\mathsf{remv}(v')$ for each child v' of v; and (b) an array num on any nodes v in G_1 and u in G_2 such that $\mathsf{num}(v, u) = |\mathsf{sim}(v) \cap \mathsf{desc}(u)|$, *i.e.,* the number of common nodes between $\mathsf{sim}(v)$ and $\mathsf{desc}(u)$.

Using these variables we reduce the cost of the loop. Recall that the loop does the following: for any nodes v in G_1 and any node $u \in \mathsf{sim}(v)$, if there is an edge (v, v') in G_1 such that $\mathsf{sim}(v') \cap \mathsf{desc}(u)$ is empty, then u must be removed from $\mathsf{sim}(v)$. Observe the following. (1) $\mathsf{sim}(v') \cap \mathsf{desc}(u) = \emptyset$ iff $\mathsf{num}(v', u)$ = 0; thus the former can be checked in constant time by inspecting $\mathsf{num}(v', u)$. (2) Both $\mathsf{remv}(v)$ and $\mathsf{num}(v, u)$ can be *incrementally maintained* in response to node removal from $\mathsf{remv}(v')$ and $\mathsf{num}(v', u)$ of *children* v' of v; thus they do not have to be computed starting from scratch. (3) Once u in $\mathsf{remv}(v)$ is inspected, it will not appear in $\mathsf{remv}(v)$ again. In the worst case, it takes at most $O(|V_2|)$ time to make $\mathsf{sim}(v)$ empty. (4) The loop terminates if for all node v in G_1, $\mathsf{remv}(v)$ is empty. Hence there are at most $|E_1||V_2|$ iterations. In addition, if $\mathsf{sim}(v)$ becomes empty for some node v in G_1, the algorithm terminates and returns an empty relation. (5) The total maintenance cost in all these iterations due to the node removal is in $O(|E_1||V_2|^2)$ time. Taken together, the loop takes at most $O(|E_1||V_2|^2)$ time. Hence the algorithm is in $O((|E_1| + |V_2|)(|V_1| + |V_2|^2))$ time including the pre-processing.

From this analysis and Proposition 3.1 it follows:

Proposition 3.2: *Given G_1=(V_1, E_1, L_1), G_2=(V_2, E_2, L_2),* $\mathsf{mat}()$ *and ξ, whether or not $G_1 \preceq_{(e,p)} G_2$ can be decided in $O((|E_1| + |V_2|)(|V_1| + |V_2|^2))$ time.* □

4. Strong P-Similarity and Intractability

Graph simulation and p-similarity aim to find a binary relation on the nodes of two graphs G_1 and G_2. In such a relation a node in G_1 may match multiple nodes in G_2. As remarked earlier, in some applications we need a stronger notion, namely, a function such that each node of G_1 is only mapped to a single node in G_2.

In this section we first define strong p-similarity, a restriction of p-similarity by using functions instead of relations. We show that, no matter how desirable, it becomes NP-complete to compute strong p-similarity. We then present optimization problems for computing strong p-similarity. We show that these problems are also intractable, and worse still, are hard to approximate.

4.1 Strong P-Similarity and 1-1 P-Similarity

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

Strong p-similarity. Graph G_1 is said to be *strongly p-similar* to G_2 *w.r.t.* a similarity matrix $\mathsf{mat}()$ and a similarity threshold ξ, denoted by $G_1 \precsim^s_{(e,p)} G_2$, if there exists a mapping σ from 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 child of v is mapped to a *single descendant* of u.
Here σ is called *a p-similarity mapping* from G_1 to G_2.

Example 4.1: Consider graphs G_1 and G_2 of Fig. 1. We have seen from Example 2.1 that $G_1 \preceq_{(e,p)} G_2$. The p-similarity relation given there can be changed into a p-similarity mapping by mapping node albums to *either* pop *or* rock, but *not* both.

To further illustrate the difference between p-similarity and strong p-similarity, let us consider the graphs of Fig. 2. As we have seen in Example 2.1, $G_1 \preceq_{(e,p)} G_2$, $G_3 \preceq_{(e,p)} G_4$ and $G_5 \preceq_{(e,p)} G_6$. For strong p-similarity, observe the following.
(1) $G_1 \precsim^s_{(e,p)} G_2$. A p-similarity mapping is defined by mapping *both A* nodes in G_1 to the A node in G_2, node B in G_1 to the B node in G_2, and node C in G_1 to *any* of the two C nodes in G_2.
(2) $G_3 \not\precsim^s_{(e,p)} G_4$. In the p-similarity relation given in Example 2.1, the D node in G_3 is mapped to *both* of the two D nodes in G_4. However, this is no longer allowed by strong p-similarity, and mapping the D node in G_3 to *only one* of the D nodes in G_4 does not make a p-similarity mapping, because either the edge (A, D) or (B, D) in G_3 can no longer find a matching path in G_4.
(3) $G_5 \not\precsim^s_{(e,p)} G_6$, for the same reason as (2). □

1-1 p-similarity. As remarked earlier, one often wants a similarity mapping to map each node in G_1 to a *distinct* node in G_2. We say that a graph G_1 is *1-1 p-similar to* G_2, denoted by $G_1 \precsim^{1-1}_{(e,p)} G_2$, if there exists a 1-1 (injective) p-similarity 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-similarity mapping* from G_1 to G_2.

Example 4.2: Consider again G_1 and G_2 of Fig. 1. One can verify that the p-similarity mapping given in Example 4.1 is a 1-1 p-similarity mapping, *i.e.,* $G_1 \precsim^{1-1}_{(e,p)} G_2$.

As another example, consider G_1 and G_2 of Fig. 2. While $G_1 \precsim^s_{(e,p)} G_2$, $G_1 \not\precsim^{1-1}_{(e,p)} G_2$. In particular, the p-similarity mapping given in Example 4.1 is not injective, since it maps both A nodes in G_1 to the same A node in G_2. □

Note that subgraph isomorphism is a special case of 1-1 p-similarity: G_1 is isomorphic to a subgraph of G_2 iff there exists a 1-1 p-similarity 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, rather than to a path, (b) adopts node label equality rather than node similarity, 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-similarity only requires edges from G_1 to find a match in G_2, but not the other way around.

Intractability. In contrast to the $O(n^3)$-time complexity of p-similarity, it is intractable to determine strong p-similarity and 1-1 p-similarity. Recall that the subgraph isomorphism problem is also intractable. However, the proofs below are quite different from the proof for the subgraph isomorphism problem.

Theorem 4.1: *Given graphs G_1 and G_2, a similarity matrix* $\mathsf{mat}()$ *and a threshold ξ, it is* NP*-complete to decide*
- *whether $G_1 \precsim^s_{(e,p)} G_2$, and*
- *whether $G_1 \precsim^{1-1}_{(e,p)} G_2$.*

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-similarity even when G_1 is a tree and G_2 is a* DAG. □

Proof sketch: For the upper bound, we develop NP algorithms for these problems: guess a mapping from nodes in G_1 to those in G_2, and then check whether the mapping is a p-similarity mapping (resp. 1-1 p-similarity mapping). The latter can be done in PTIME.

The lower bound for strong p-similarity is verified by reduction from 3SAT, an NP-complete problem (cf. [19]). Given a 3SAT instance ϕ, we construct graphs G_1 and G_2 such that $G_1 \precsim^s_{(e,p)} G_2$ iff ϕ is satisfiable. Furthermore, G_1 and G_2 are DAGs.

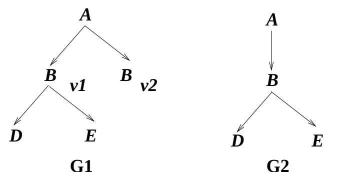


Figure 4: Cardinality vs. overall similarity

The lower bound for 1-1 p-similarity is shown by reduction from X3C, which is NP-complete (cf. [19]). Given an X3C instance, the reduction constructs a tree G_1 and a DAG G_2 such that there exists an exact cover for the instance iff $G_1 \precsim^{1-1}_{(e,p)} G_2$. □

4.2 Optimization Problems of Strong P-Similarity

It is common in practice that there may not exist strong (1-1) p-similarity mappings from the entire G_1 to G_2. Thus one often wants to find mappings from subgraphs of G_1 to G_2 instead. There are possibly many such mappings, and it is natural to want the one with the "best quality" *w.r.t.* a certain quality metric.

Below we present two quality metrics, and state optimization problems for strong (resp. 1-1) p-similarity accordingly.

Consider a strong (resp. 1-1) p-similarity mapping σ from a subgraph $G'_1 = (V'_1, E'_1, L'_1)$ of G_1 to G_2.

Maximum cardinality. This metric is based on *the number of nodes* in G_1 that σ maps to G_2. More specifically, the cardinality of σ is defined as:

$$\mathsf{qualCard}(\sigma) = \frac{|V'_1|}{|V_1|}.$$

The *maximum cardinality problem* for strong (resp. 1-1) p-similarity, referred to as $\mathsf{CSP}^{\mathsf{s}}$ (resp. CSP^{1-1}), is to find, given graphs G_1, G_2, a matrix $\mathsf{mat}()$ and a threshold ξ as input, a p-similarity mapping σ from a subgraph of G_1 to G_2 such that $\mathsf{qualCard}(\sigma)$ is maximum.

Observe the following. (1) If $G_1 \precsim^{s}_{(e,p)} G_2$ or $G_1 \precsim^{1-1}_{(e,p)} G_2$, then a p-similarity mapping σ with maximum $\mathsf{qualCard}(\sigma)$ is a p-similarity mapping from the entire G_1 to G_2. (2) Recall the familiar *maximum common subgraph problem* (MCS; see, *e.g.,* [23]): it 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. Note that MCS is a special case of CSP^{1-1} when only edge-to-edge mappings and node label equality are allowed, and when backward edge preservation is required.

Optimal overall similarity. Alternatively, we consider the *overall similarity* of mapping σ. Here we assume a weight $w(v)$ associated with each node v, indicating relative importance of the nodes. 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 strong (resp. 1-1) p-similarity, referred to as $\mathsf{SSP}^{\mathsf{s}}$ (resp. SSP^{1-1}), is to compute, given G_1, G_2, $\mathsf{mat}()$ and ξ as input, a p-similarity 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 2.

Example 4.3: Consider graphs G_1 and G_2 shown in Fig. 4. 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

$\mathsf{CSP}^{\mathsf{s}}$	maximum cardinality for strong p-similarity
CSP^{1-1}	maximum cardinality for 1-1 p-similarity
$\mathsf{SSP}^{\mathsf{s}}$	maximum overall similarity for strong p-similarity
SSP^{1-1}	maximum overall similarity for 1-1 p-similarity

Table 2: Notations: Optimization problems

Let $\xi = 0.6$, and assume a weight function such that $w(v) = 1$ for each node v in G_1, except $w(v_2) = 6$. Then G_1 is not 1-1 p-similar to G_2. Indeed, given $\mathsf{mat}_0()$ and ξ, any p-similar mapping from G_1 to G_2 has to map both v_1 and v_2 in G_1 to the B node in G_2, which is not allowed by a 1-1 mapping. In light of this we want to find a 1-1 p-similarity mapping from a subgraph of G_1 to G_2.

(1) When the maximum cardinality metric is adopted, an optimal 1-1 p-similarity mapping σ_c is from a subgraph H_1 of G_1 to G_2, where H_1 contains nodes A, D, E and v_1. Here σ_c maps each node v in G_1 to a node u in G_2 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-similarity mapping σ_s is from a subgraph H_2 of G_1 to G_2, 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_1 to G_2 than σ_s. □

Intractability. Unless P = NP, it is unrealistic to have a polynomial time (PTIME) algorithm for finding an optimal (1-1) p-similarity mapping, no matter whether the maximum overall similarity metric or the maximum cardinality metric is adopted. The proof of the result below is a variation of the proof of Theorem 4.1.

Corollary 4.2: *The maximum cardinality problem and the maximum overall similarity problem are* NP*-complete for both strong p-similarity and 1-1 p-similarity. The problems are already* NP*-hard even when only* DAG*s are considered.* □

4.3 Approximation Hardness

In light of Corollary 4.2 we aim to develop efficient heuristic algorithms for finding (1-1) subgraph p-similarity mappings, with performance guarantees, *i.e.,* the quality of the mappings found by the algorithms is within a bound of that of the optimal mappings.

Unfortunately, both the maximum cardinality problems ($\mathsf{CSP}^{\mathsf{s}}$, CSP^{1-1}) and the maximum overall similarity problems ($\mathsf{SSP}^{\mathsf{s}}$, SSP^{1-1}) are hard to approximate. Indeed, the result below shows that there exist no PTIME algorithms for finding (1-1) p-similarity 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: $\mathsf{CSP}^{\mathsf{s}}$, CSP^{1-1}, $\mathsf{SSP}^{\mathsf{s}}$ *and* SSP^{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), a well-studied optimization problem. 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 with maximum weight, *i.e.,* 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 graph nodes [20].

To show the approximation bound, we need to use *approximation factor preserving reduction* (AFP-reduction) [35]. 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).

The result below shows that AFP-reductions retain approximation bounds. Here an algorithm $\mathcal{A}$ has performance guarantee α if for any instance I, $\mathsf{obj}(\mathcal{A}(I)) \geq \alpha\, \mathsf{opt}(I)$.

Proposition 4.4:[35] *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.* $\square$

Theorem 4.3 is verified by an AFP-reduction from WIS to each of $\mathsf{CSP}^{\mathsf{s}}$, CSP^{1-1}, $\mathsf{SSP}^{\mathsf{s}}$ and SSP^{1-1}. That is, these problems are at least as hard as WIS when approximation is concerned.

5. Computing Strong Similarities

Despite Theorem 4.3, in this section we provide approximation algorithms for each of the maximum cardinality problems ($\mathsf{CSP}^{\mathsf{s}}$, CSP^{1-1}) and the maximum overall similarity problems ($\mathsf{SSP}^{\mathsf{s}}$, SSP^{1-1}). In addition, we introduce optimization techniques for improving the performance of these algorithms.

5.1 Approximation Algorithms: An Overview

One of the main results of this section is an approximation bound for $\mathsf{CSP}^{\mathsf{s}}$, CSP^{1-1}, $\mathsf{SSP}^{\mathsf{s}}$ and SSP^{1-1}.

Theorem 5.1: $\mathsf{CSP}^{\mathsf{s}}$, CSP^{1-1}, $\mathsf{SSP}^{\mathsf{s}}$ *and* SSP^{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.* $\square$

Proof sketch: This is verified by AFP-reductions (f, g) from these problems to WIS, by constructing product graphs of G_1 and G_2. It is known that WIS is approximable within $O(log^2 n/n)$ [20]. $\square$

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 $\mathsf{mat}()$ 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 [9, 20], and (3) invoke function g in the AFP-reduction to get a (1-1) p-similarity mapping from subgraph of G_1 to G_2.

More specifically, for $\mathsf{CSP}^{\mathsf{s}}$ and CSP^{1-1}, we can leverage the approximation algorithm for maximum independent sets given in [9], which is in $O(nm)$ time, where n and m are the numbers of nodes and edges in a graph, respectively. For $\mathsf{SSP}^{\mathsf{s}}$ and SSP^{1-1}, we can use the algorithm of [20] 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.

In Section 5.2 we develop more efficient algorithms that operate directly on the input graphs instead of on their product graph, while retaining the same performance guarantees.

5.2 Algorithms with Performance Guarantees

We first develop an approximation algorithm for $\mathsf{CSP}^{\mathsf{s}}$, and then extend the algorithm to deal with CSP^{1-1}, $\mathsf{SSP}^{\mathsf{s}}$ and SSP^{1-1}.

Approximation algorithm for $\mathsf{CSP}^{\mathsf{s}}$. We propose an algorithm, referred to as $\mathsf{compMaxCard}^{\mathsf{s}}$ and shown in Figures 5 and 6. Given graphs G_1, G_2, matrix $\mathsf{mat}()$ and threshold ξ as input, it computes a p-similarity mapping σ from a subgraph of G_1 to G_2, aiming to maximize $\mathsf{qualCard}(\sigma)$. It is a nontrivial extension of the algorithm of [9] for finding maximum independent sets.

```
Algorithm compMaxCard^s
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 $\xi$.
Output: A $p$-similarity mapping from subgraph of $G_1$ to $G_2$.

1.  for each node $v \in V_1$ of graph $G_1$ do
        /* create adjacency-list representation $H_1$ for $G_1$ */
2.      $H_1[v]$.prev := $\{v' \mid v' \in V_1, (v', v) \in E_1\}$;
3.      $H_1[v]$.post := $\{v' \mid v' \in V_1, (v, v') \in E_1\}$;
        /* create the matching-list representation $H$ for $G_1$ */
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
        /* create adjacency-matrix representation $H_2$ for $G_2^+$ */
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($\sigma_m$) do
10.     $(\sigma, I)$ := greedyMatch($H_1, H_2, H$);   $H := H \setminus I$;
11.     if sizeof($\sigma$) > sizeof($\sigma_m$) then $\sigma_m := \sigma$;
12. return $\sigma_m$.
```

Figure 5: Approximation algorithm $\mathsf{compMaxCard}^{\mathsf{s}}$

The algorithm maintains several data structures to ensure the quality of mappings. (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 and children, respectively. (d) An adjacency matrix H_2 for the *transitive closure* graph G_2^+ of G_2. For any edge (u_1, u_2) in G_2, $H_2[u_1, u_2]$ = 1 iff there is an edge from u_1 to u_2 in G_2^+, *i.e.,* a nonempty path from u_1 to u_2 in G_2.

The algorithm works as follows. It first constructs the adjacency list H_1 and the matching list H for G_1 (lines 1–4), 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. It then computes the *transitive closure* graph G_2^+ of G_2 and creates the adjacency matrix H_2 of G_2^+ (line 5–7). It is to find a p-similarity mapping σ_m from a subgraph of G_1 to G_2. The mapping σ_m is initially $\emptyset$ (line 8), and is computed by a procedure, referred to as greedyMatch, as follows.

In a nutshell, greedyMatch (Fig. 6) 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. In this way it decides whether (v, u) is a good choice or not. At the same time it computes sets I_1, I_2 of pairwise contradictory matching pairs, for mappings with (v, u) as a match and without, respectively; it returns as I the larger one of I_1 and I_2. It is worth remarking that I is nonempty.

Upon receiving σ and I from greedyMatch (line 10), algorithm $\mathsf{compMaxCard}^{\mathsf{s}}$ removes conflict pairs I from H (line 10) and takes σ as σ_m if σ is larger than 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 $\mathsf{compMaxCard}^{\mathsf{s}}$.

(a) Procedure greedyMatch (Fig. 6) takes the current matching list

Procedure greedyMatch

Input: Graphs H_1, H_2, and matching list H for subgraph $G_1[H]$.
Output: A p-similarity 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
        /* separate $H$ 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 $(\sigma, I)$;
```

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 6: Procedures greedyMatch **and** trimMatching

H as input. It computes a p-similarity 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-similarity 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. 6) takes as input a candidate match (v, u) and the current matching list H. It removes bad matches from H assuming that (v, u) is a match. More specifically, for any parent v' in both $H_1[v]$.prev and H, it moves every candidate match u' from $H[v']$.good to $H[v']$.minus if there exists no path from u' to u in G_2 (*i.e.,* $H_2[u', u] = 0$; lines 1–4), by the definition of strong p-similarity. Similarly, it processes v's children (lines 5–8). The updated H is then returned (line 9).

Example 5.1: We illustrate how compMaxCard$^{\mathsf{s}}$ computes a p-similarity mapping from a subgraph of G_1 to G_2 of Fig. 1. For the lack of space we use subgraphs G'_1 and G'_2, mat() and ξ given in Example 3.1. After step 7, the algorithm constructs an adjacent list H_1 and an initial matching list H for G'_1 (see below), and an adjacency matrix H_2 for the transitive closure graph of G'_2 (omitted).

Nodes in H_1	prev	post
books	$\emptyset$	{textbooks,abooks}
textbooks	{books}	$\emptyset$
abooks	{books}	$\emptyset$

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

The algorithm then calls greedyMatch to produce a subgraph p-similarity 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 (nodes in G_1) becomes smaller than the one of σ_m. At this point compMaxCard$^{\mathsf{s}}$ returns {(abooks, audiobooks), (textbooks, school), (books, books)} as the p-similarity mapping found. $\square$

Correctness and complexity. Algorithm compMaxCard$^{\mathsf{s}}$ indeed possesses the performance guarantee given in Theorem 5.1.

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$^{\mathsf{s}}$ *finds a p-similarity 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.* $\square$

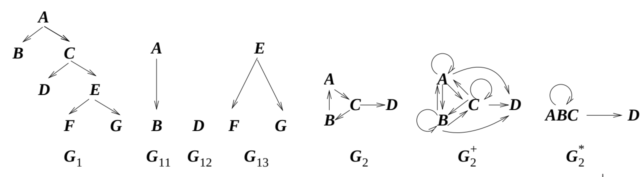
Proof sketch: Algorithm greedyMatch generates the same mappings as the naive approximation algorithm described in Section 5.1. Thus the approximation bound of that algorithm carries over to greedyMatch. See [2] for a detailed proof. $\square$

For the complexity, observe the following. (a) Algorithm compMaxCard$^{\mathsf{s}}$ calls greedyMatch at most $|V_1||V_2|$ times (line 10). (b) greedyMatch recursively calls itself at most $|V_1||V_2|$ times (lines 10–11). For each call, it takes $O((d^-(v) + d^+(v))|V_2| + |V_1|)$ time, where $d^-(v)$ and $d^+(v)$ are the numbers of v's parents and children, respectively. Putting these together, algorithm compMaxCard$^{\mathsf{s}}$ is in $O(|V_1|^3|V_2|^2 + |V_1||E_1||V_2|^3)$ time. Note that (1) $\sum_{v \in V_1} d^+(v) = \sum_{v \in V_1} d^-(v) = |E_1|$, and (2) $|V_2|$ can actually be replaced by the number of nodes in G_2 that σ maps nodes in G_1 to, which is often much smaller than $|V_2|$.

For the space complexity, compMaxCard$^{\mathsf{s}}$ maintains single copies for H_1 and H_2 in the entire process. Each time when the matching list H is split into H^+ and H^-, H^+ and H^- can reuse H's space. Hence the total space used is in $O((|V_1| + |V_2|)^2)$.

We next show that algorithm compMaxCard$^{\mathsf{s}}$ can be readily converted to approximation algorithms for CSP$^{1-1}$, SSP$^{\mathsf{s}}$ and SSP$^{1-1}$.

Approximation algorithm for CSP$^{1-1}$. A 1-1 p-similarity mapping requires that no two nodes in G_1 are mapped to the same node



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

Figure 7: Reducing the graph size

in G_2. Minor changes to compMaxCard^s 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^s. This yields an approximation algorithm for CSP$^{1-1}$, referred to as compMaxCard$^{1-1}$.

Approximation algorithms for SSP^s **and** SSP$^{1-1}$. We develop an approximation algorithm, referred to as compMaxSim^s, for the maximum overall similarity problem SSP^s. The algorithm borrows a trick from [20]. The strategy of [20] 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 [9]) to the subgraph induced by group i, and returns the maximum of the solutions to these induced groups.

Along the same lines, compMaxSim^s first partitions the initial matching list H into $\log(|V_1||V_2|)$ groups, and then it applies compMaxCard^s to each group. It returns σ with the maximum qualSim(σ) among p-similarity mappings for all these groups. Similarly, an approximation algorithm is developed for SSP$^{1-1}$, 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^s.

5.3 Optimization Techniques

We next propose techniques to improve the efficiency of our algorithms, while retaining or improving their match quality.

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-similar match in G_2. Obviously the nodes in S_1 do not contribute to any p-similarity 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-similarity 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. 7(a) is connected, in which node C has no p-similar 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 5.3: *Let graph G_1 consist of k pairwise disconnected components G_{11}, ..., and G_{1k}. If σ_i is a maximum p-similarity mapping from a subgraph of G_{1i} to G_2, then $\bigcup_{i=1}^{i=k}(\sigma_i)$ is a maximum p-similarity 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. 7(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 [16].

The partitioning strategy may improves 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-similarity 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 7(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-similarity mapping they find from a subgraph of G_1 to G_2^+ is also a strong (1-1) p-similarity 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^+ [29].

6. Experimental Study

We next present an experimental study of our algorithms for identifying (strong, 1-1) p-similar graphs. Using real-life and synthetic data, we conducted two sets of experiments to evaluate the ability and scalability of our methods for matching similar graphs vs. conventional graph simulation and subgraph isomorphism.

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 [3]. Three types of Web data were extracted, in different categories: Web sites for online stores, international organizations and online newspapers. 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 a certain 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 its corresponding URL. The similarity between two nodes was measured by the distance between their contents (whenever available) or URLs. We denoted these Web sites by *sites 1, 2* and 3, respectively.

These Web graphs are typically large, having thousands of nodes. 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 node degree and the maximum node degree in G, respectively, and α is a constant in [0, 1]. Skeleton G_s is the subgraph of G induced by these nodes (see Section 2), consisting of the nodes and their edges.

More specifically, 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. Unfor-

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 3: 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^{s}	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^{s}	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
cdkMCS	N/A	N/A	N/A	67	100	0	N/A	N/A	N/A	156.931	189.16	0.82

Table 4: Accuracy and scalability on real life data

tunately, these graphs were beyond the capability of the algorithms we could find for computing maximum common subgraphs [1]. To cope with this, we also chose top 20 nodes with the highest degree, and constructed another set of Web skeletons. The information about those Web graphs and skeletons is reported in Table 3.

Observe that each set of the graphs represent different versions of the same Web site, and hence they *should match* with each other. By using skeletons we reduce the impact of noisy nodes. Based on this, we evaluated the accuracy of our algorithms.

(2) Synthetic data. We also designed a generator to produce graphs for experiments, controlled by two parameters: the number m of nodes and the noise rate noise%. Given m, we first randomly generated graph G_1 with m nodes and $4 \times m$ edges. We then produced a set of 15 graphs G_2 by introducing artificial noise into G_1, with added complexity to make it hard for G_1 to match. 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 between 1 and 5 nodes, and (b) for each node in G_1, with probability noise%, the node was attached with a subgraph of between 1 to 10 nodes. 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 algorithms, all in Java: (a) all the algorithms developed in this paper, namely, compSimilarity, compMaxCard^{s}, compMaxCard$^{1-1}$, compMaxSim^{s}, and compMaxSim$^{1-1}$, (b) the graph simulation algorithm of [21], and (c) the algorithm given in CDK [1] for finding a maximum common subgraph, denoted by cdkMCS.

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(σ) ≥ 0.75 (resp. qualSim(σ); see Section 4). We also assumed a uniform weight $w(v) = 1$ for all nodes v when measuring the overall similarity.

Exp-1: Accuracy and efficiency on real-life data. In the first set of experiments, we evaluated the accuracy and efficiency of (strong, 1-1) p-similarity against the conventional notions of graph simulation and subgraph isomorphism, using the sets of Web *skeletons*.

For each set of Web skeletons, we randomly chose one as G_1, and attempted to match it with the rest (G_2) in the set. As remarked earlier, G_1 and G_2 should logically match. We use the percentage of matches found as the accuracy measure for all algorithms. We fixed the node similarity threshold ξ to be 0.75.

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

The accuracy and efficiency results are shown in Table 4. (1) In most cases, our algorithms found more than 50% of matches. (2) The strong p-similarity algorithms found more matches than the 1-1 p-similarity 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. (4) For skeletons 1, cdkMCS did not run to completion. For skeletons 2, 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.

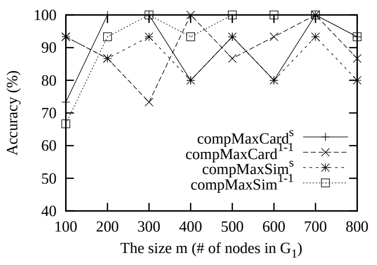
Our algorithms took less than 4 seconds in all these cases, while cdkMCS took 180 seconds even for graphs with only 20 nodes. Note that although *sites 2* and *3* are about the same size, the running time of cdkMCS on them is not comparable.

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

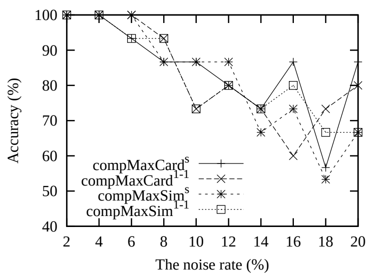
Exp-2: Efficiency on synthetic data. In the second set of experiments, we evaluated the accuracy and scalability of our algorithms using graphs randomly generated as described above. We also evaluated the performance of the graph simulation algorithm of [21], denoted by graphSimulation. However, we could not evaluate algorithm cdkMCS [1], since it only works on small graphs.

We investigated (a) the accuracy of our four approximation algorithms, and (b) the efficiency of these algorithms and that of graphSimulation and compSimilarity. In these experiments the accuracy of compSimilarity is constantly 100% whereas it is 0% for graphSimulation in all the cases. Thus we do not show the accuracy of these methods. 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.

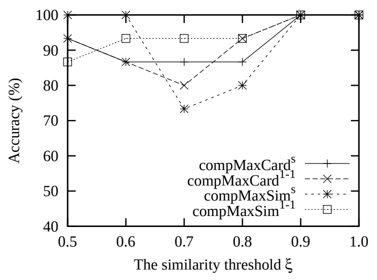
(1) Varying the size of G_1. To evaluate the impact of graph sizes on the accuracy and the scalability, we fixed noise% $= 10\%$ and $\xi = 0.75$, while varying m from 100 to 800, where the number of nodes in G_2 was in the range $[260, 2225]$.



(a) Accuracy *w.r.t.* the size m

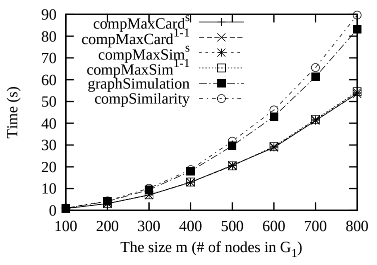


(b) Accuracy *w.r.t.* the noise rate

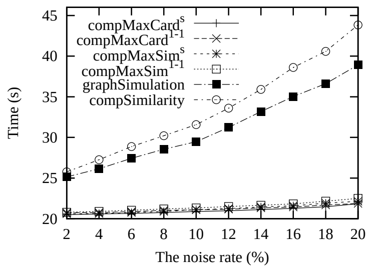


(c) Accuracy *w.r.t.* the threshold ξ

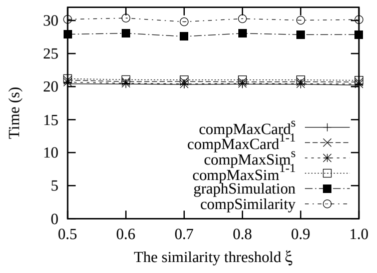
Figure 8: Accuracy on synthetic data



(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 9: Scalability on synthetic data

The accuracy results are reported in Fig. 8(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. 9(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. The approximation algorithms have comparable running time, while compSimilarity and graphSimulation are comparable. Here the approximation algorithms actually outperformed both compSimilarity and graphSimulation, although their worst-case complexity bounds are not as good as those of the latter ones. This is because the worst case hardly occurred in the experiments.

(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 8(b) shows that the accuracy of our approximation algorithms is sensitive to the noise rate. Nevertheless, the accuracy is still above 50% even when noise% = 20% and G_2 had 2000 nodes.

Figure 9(b) shows that while the scalability of compSimilarity and graphSimulation is sensitive to noise%, the approximation 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 8(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 9(c) tells us that the scalability of all these algorithms is indifferent to ξ.

Summary. Our experimental results demonstrate the effectiveness and efficiency of our matching methods. From the results we find the following. (a) The notions of (strong, 1-1) p-similarity are capable of identifying a large number of similar graphs that cannot be matched by simulation or subgraph isomorphism. (b) Our algorithms scale well with the sizes of the graphs, the noise rates, and with the node similarity threshold. The algorithms 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. Related Work

A variety of models have been proposed for measuring structural similarity of graphs: graph simulation [4, 14], subgraph isomorphism [34, 37], graph embedding [13, 18] and graph edit distance [11] (see [12, 15, 33] for surveys). While graph simulation is defined in terms of binary relations on nodes, the others are defined in terms of functions from nodes in one graph to nodes in the other. As remarked earlier, p-similarity and strong (1-1) p-similarity extend graph simulation and subgraph isomorphism by supporting (a) edge-to-path mappings rather than edge-to-edge mappings, and (b) node similarity instead of node label equality, in order to capture graph structural similarity in emerging applications.

Closer to the notion of *strong* p-similarity are the models studied for graph-pattern queries (*e.g.,* [13]) and XML schema embedding [18]. Graph-pattern queries support a descendant-or-self axis '//' and a child axis '/', which are to be mapped to a path and an edge in a data graph, respectively. On one hand, strong p-similarity only allows '//'; on the other hand, strong p-similarity takes node similarity into account, which is not considered in [13]. Algorithms are given in [13] for DAG queries. While the algorithms are claimed to be in PTIME, it is unlikely that they work on general DAGs since the problem is NP-complete for DAGs, as shown by Theorem 4.1. In this line of research no approximation algorithms have been developed for matching similar graphs.

A notion of schema embedding is introduced in [18], which is a special case of strong p-similarity with two additional constraints: prefix-free and type matching. Node similarity is also considered in [18]. While heuristic algorithms are provided in [18], they do not have performance guarantees on match quality, as opposed to this work. Further, extensions of simulation are not studied in [18].

The edit distance of two graphs G_1 and G_2 is defined to be the number of deletions and insertions of nodes and edges required to transform G_2 to G_1. While 1-1 p-similarity can be mimicked by graph editing when only deletions are allowed, this simulation involves excessive deletions of nodes or edges from G_2 to which no nodes or edges of G_1 are mapped.

Optimization problems have also been studied for similar graph matching [12, 15, 33]. These problems are often defined based on similarity metrics that measure the ratio of the size of a maximum common subgraph to the size of original graphs (*e.g.,* [31, 32, 38]). These metrics allow edge-to-edge mappings only, and typically do not take weights of nodes into account. Another metric is based on paths [22], which assesses overlapping root-leaf paths in the graph representations of documents, but ignores the topology of the graphs. We are not aware of any previous work that has studied maximum cardinality and maximum overall similarity, which are proposed in this work for strong p-similarity and 1-1 p-similarity,

Based on various similarity metrics, a number of graph matching algorithms have been developed (see [12, 15] for surveys). Closest to ours are [21, 9, 20], which provide a quadratic-time algorithm for computing simulation relations and approximation algorithms for computing maximum (weighted) independent sets, respectively. Our algorithms in Sections 3 and 5 are extensions of these algorithms. There have also been algorithms for identifying subgraph isomorphism and maximum common subgraphs (*e.g.,* [13, 31, 34, 38]). Stack-based algorithms are studied for matching DAGs [13], by leveraging indexing to avoid computing transitive closures, and by adopting filtering for early pruning. An algorithm for computing maximum common edge subgraphs is proposed in [31], which makes use of a maximum clique formulation as well as pruning heuristics. Algorithms for approximate graph matching can be found in [34, 37], based on indexing and feature filtering. Unfortunately, these algorithms do not possess provable guarantees on match quality, as opposed to approximation algorithms of this paper. On the other hand, the indexing and filtering techniques of [31, 34, 37] can be used to improve the performance of our algorithms.

8. Conclusions

We have proposed several notions for capturing structural similarity of graphs, by allowing edges in one graph to map to paths in the other, and by incorporating node similarity. For graph matching based on these notions, we have provided complexity bounds and algorithms. (a) We have presented an $O(n^3)$-time algorithm for p-similarity, comparable to its simulation counterpart. (b) We have established the intractability for strong (1-1) p-similarity and their optimization problems, and shown that these problems are hard to approximate. (c) Despite the hardness, we have developed approximation algorithms for these problems, with provable guarantees on match quality. As verified by our experimental results, these techniques are able to identify many similar graphs that conventional graph simulation and subgraph isomorphism fail to find, and yield a promising tool for graph matching in emerging applications.

Admittedly our experimental study is still preliminary. We are experimenting with more real-life data in various domains, and hope to identify areas in which (strong, 1-1) p-similarity is most effective. We also plan to improve the efficiency of our algorithms, by leveraging indexing and filtering techniques developed in [31, 34, 37]. Another topic for future work is to compare our techniques with approaches based on feature graphs and graph edit distance.

9. References

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

[2] Full version. *http://homepages.inf.ed.ac.uk/sma1/gm-full.pdf.*

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

[4] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web. From Relations to Semistructured Data and XML.* Morgan Kaufman, 2000.

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

[6] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer, 2006.

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

[8] K. Bharat, A. Broder, J. Dean, and M. R. Henzinger. A comparison of techniques to find mirrored hosts on the WWW. *J. Am. Soc. Inf. Sci.*, 51(12), 2000.

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

[10] 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.

[11] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8), 1997.

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

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

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

[15] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3), 2004.

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

[17] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1), 2007.

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

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

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

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

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

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

[24] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The Web as a graph: Measurements, models, and methods. In *COCOON*, 1999.

[25] W.-S. Li and C. Clifton. Semantic integration in heterogeneous databases using neural networks. In *VLDB*, 1994.

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

[27] G. S. Manku, A. Jain, and A. D. Sarma. Detecting near-duplicates for Web crawling. In *WWW*, 2007.

[28] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *VLDB*, 1998.

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

[30] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 2001.

[31] J. W. Raymond, E. J. Gardiner, and P. Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *Comput. J.*, 45(6), 2002.

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

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

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

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

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

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

[38] X. Yan, F. Zhu, P. S. Yu, and J. Han. Feature-based similarity search in graph structures. *TODS*, 31(4), 2006.

APPENDIX: Proofs

Corollary 1: *The p-similarity problem is in* PTIME. □

Proof: Following from Proposition 3.2.

Proof of Theorem. 4.1. (a) The strong p-similarity problem ($G_1 \precsim^{s}_{(e,p)} G_2$) is NP-complete even when both G_1 and G_2 are acyclic directed graphs (DAGs).

Proof: We first show that this problem is in NP. An NP algorithm is as follows: guess a relation, and then check whether it is a strong p-similarity mapping. It is easy to check wether a relation is a mapping (function), and can be done in polynomial time, so does checking whether it is p-similar according to Corollary 1.

We next show that this problem is NP-hard by reduction from the 3SAT problem, which is NP-complete (cf. [19]). Given an instance $\phi = C_1 \wedge \cdots \wedge C_n$ of 3SAT, where all the variables in ϕ are $x_1, \ldots, x_m$, 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 indicate the occurrence of a variable in literal i of clause C_j. The 3SAT problem is to determine whether ϕ is satisfiable or not.

More specifically, given an instance ϕ of the 3SAT problem, we will construct two DAGs G_1, G_2 and a similarity matrix att such that $G_1 \precsim^{s}_{(e,p)} G_2$ if and only if ϕ is satisfiable.

(1) The DAG $G_1 = (V_1, E_1)$ 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 $j \in [1, n]$.

Intuitively, graph G_1 encodes the instance ϕ of 3SAT. Node X_i ($i \in [1, m]$) in V_1 corresponds to variable x_i, and node C_j ($j \in [1, n]$) in V_1 corresponds to 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 $x_{p_{j3}}$.

(2) The DAG $G_2 = (V_2, E_2)$ 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 $i \in [1, m]$;

∘ For each clause $C_j = y_{j_1} \vee y_{j_2} \vee y_{j_3}$ of ϕ ($j \in [1, n]$), E_2' contains 7×3 edges.

(a) If we take true as 1, false as 0, and C_j as a three-bit constant $y_{j1} y_{j2} y_{j3}$, then the domain of clause $C_j(\rho)$ is $\{0, 1, \ldots, 7\}$, where ρ is a truth assignment of variables $x_{p_{j1}}$, $x_{p_{j2}}$ and $x_{p_{j3}}$.

(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: $(X_{Tp_{jk}}, C_j(\rho)_j)$ if $\rho(X_{p_{jk}})$ = true, or $(X_{Fp_{jk}}, C_j(\rho)_j)$ if $\rho(X_{p_{jk}})$ = false, where $k \in [1, 3]$.

Intuitively, graph G_2 encodes the truth assignments of variables and clauses in the instance ϕ of 3SAT that make them true. Node X_{Ti} ($i \in [1, m]$, resp. X_{Fi}) means assigning variable x_i a true (resp. false) value. Nodes $\{0_j, \ldots, 7_j\}$ corresponds to $C_j(\rho)$, measured as a three-bit constant *w.r.t.* the truth assignment of the three variables contained in clause C_j. Node R_2 is the root of graph G_2. Nodes T and F are redundant, but make the graph structure clearer. Edges from X_{Ti} or X_{Fi} to nodes $\{0_j, \ldots, 7_j\}$ encodes the relationships between the truth assignments of variables ($x_{p_{j1}}$, $x_{p_{j2}}$ and $x_{p_{j3}}$) and the corresponding values of $C_j(\rho)$.

(3) The similarity matrix att is defined as follows:

∘ att$[R_1, R_2]$ = 1;

∘ att$[X_i, X_{Ti}]$ = 1 and att$[X_i, X_{Fi}]$ = 1 for $i \in [1, m]$;

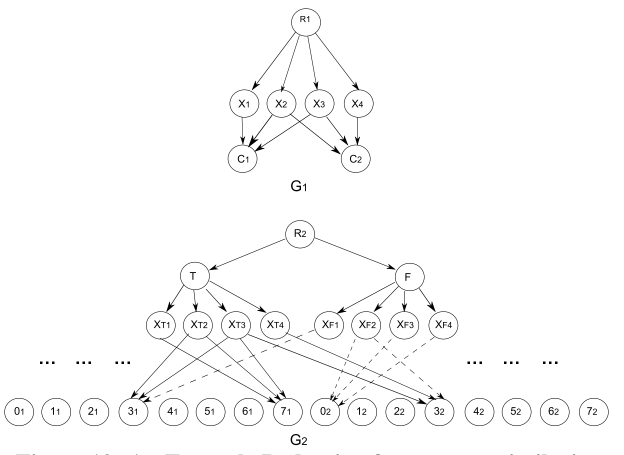


Figure 10: An Example Reduction for strong p-similarity

∘ att$[C_j, 0_j]$ = 1, …, att$[C_j, 7_j]$ = 1 for $j \in [1, n]$;

∘ att$[v, u]$ = 0 for any other nodes $v \in V_1$ and $u \in V_2$

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

Here is a simple example 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 graphs G_1 and G_2 are shown in Fig.10. Observe that both G_1 and G_2 are DAGs.

Now we verify that G_1, G_2 and att are indeed a reduction from the 3SAT problem, *i.e.,* there is a strong p-similarity mapping from G_1 to G_2 if and only if the 3SAT instance ϕ is satisfiable.

Assume that there is a strong p-similarity 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 can not be mapped to both nodes X_{Ti} and X_{Fi} in G_2 since λ is a mapping (function). For each node C_j ($j \in [1, n]$), $\lambda(C_j)$ guarantees that ρ must make clause C_j true. Otherwise, there are no edges from its corresponding variables to $\lambda(C_j)$ that makes clause C_j false. This is guaranteed by the construction of graph G_2. Following from these, we show that 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 strong p-similarity mapping λ from G_1 to G_2. The p-similarity mapping λ is defined as follows:

∘ $\lambda(R_1) = R_2$;

∘ For $i \in [1, m]$, $\lambda(X_i) = X_{Ti}$ if ρ = true, or $\lambda(X_i) = X_{Fi}$ if ρ = false;

∘ For $i \in [1, n]$, $\lambda(C_i) = C_i(\rho)_j$, defined as before.

It is easy to verify that λ is indeed a strong p-similarity mapping.

Proof of Theorem. 4.1. (b) The 1-1 p-similarity problem ($G_1 \precsim^{1-1}_{(e,p)} G_2$) is NP-complete even when G_1 is a tree and G_2 is a DAG.

Proof: We first show that this problem is in NP. An NP algorithm is as follows: guess a relation, and then check whether it is a 1-1 p-similarity mapping. It is easy to check whether a relation is a 1-1 mapping (function), and can be done in polynomial time, so does checking whether it is p-similar according to Corollary 1.

We next show that this problem is NP-hard by reduction from the **exact cover by 3-sets** problem (X3C), which is NP-complete (cf. [19]). 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

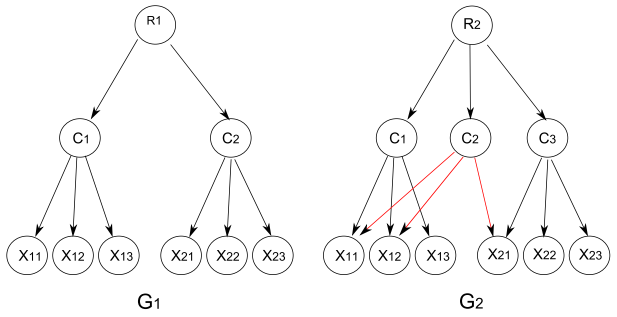


Figure 11: An Example Reduction for 1-1 p-similarity

$C_i = \{x_{i1}, x_{i2}, x_{i3}\}$ 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'.

More specifically, given an instance I of the X3C problem, we will construct two graphs G_1 and G_2 and a similarity matrix att such that there is a 1-1 p-similarity mapping from G_1 to G_2 if and only if I has an exact cover.

(1) The tree $G_1 = (V_1, E_1)$ 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 $i \in [1, q]$.

Intuitively, tree G_1 is an exact cover (a solution) for the X3C instance I. Node R_1 is the root node of tree G_1. Node C_i ($i \in [1, q]$) corresponds to a 3-element subset in S. Nodes X_{i1}, X_{i2} and X_{i3} ($i \in [1, q]$) correspond to the three elements x_{i1}, x_{i2} and x_{i3} of subset C_i. Their relationships are reflected by edges from nodes X_{i1}, X_{i2} and X_{i3} to node C_i in G_1.

(2) The DAG $G_2 = (V_2, E_2)$ is defined as follows:

- $V_2 = \{R_2, C_1, \ldots, C_n, X_{11}, X_{12}, X_{13}, \ldots, X_{q1}, X_{q2}, X_{q3}\}$;
- $E_2 = \{(R_2, C_i)\} \cup \{(C_i, X_{ik})\}$, where $i \in [1, n]$ and $X_{ik} \in C_i$ for $k \in [1, 3]$.

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

(3) The similarity matrix att is defined as follows:

- att$[R_1, R_2] = 1$;
- att$[C_i, C_j] = 1$ for $i \in [1, q]$ and $j \in [1, n]$;
- att$[X_{ik}, X_{jg}] = 1$ for $i, j \in [1, q]$ and $k, g \in [1, 3]$;
- att$[v, u] = 0$ for any other nodes $v \in V_1$ and $u \in V_2$.

The similarity matrix att 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 must be mapped to nodes C_j ($j \in [1, n]$) of G_2, (c) node X_{ik} in G_1 ($i \in [1, q]$ and $k \in [1, 3]$) must be mapped to nodes X_{jg} in G_2 ($j \in [1, n]$ and $g \in [1, 3]$).

Here is a simple example instance for the X3C problem, 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 corresponding tree G_1 and DAG G_2 are shown in Fig.11.

Now we verify that G_1, G_2 and att are indeed a reduction from the X3C problem, *i.e.,* there is a 1-1 p-similarity mapping from G_1 to G_2 if and only if there is an exact cover for the X3C problem.

Assume that there is 1-1 p-similarity mapping λ from G_1 to G_2, we show that there is an exact cover S' for the X3C problem. First, following from (R_1, R_2) is included in the mapping λ, all nodes in G_1 are forced to be included in the mapping. Second, due to the injective constraint, the mapping between the nodes X_{ik} ($i \in [1, q]$ and $k \in [1, 3]$) in G_1 and the nodes X_{jg} ($j \in [1, q]$ and $g \in [1, 3]$) in G_2 must bijective. Third, the injective constraint also requires that (1) each node C_i ($i \in [1, q]$) in G_1 must be mapped to one and at most one node C_j ($1 \leq j \leq n$) in G_2, (2) for any distinct nodes C_i and C_j ($i, j \in [1, q]$ and $i \neq j$) in G_1, they must be mapped to distinct nodes in G_2, *i.e.,* $\lambda(C_i) \neq \lambda(C_j)$.

Let $S' = \{\lambda(C_i)\}$ for each $C_i \in V_1$ of G_1 ($i \in [1, q]$), and we show that S' is an exact cover for the X3C problem. First, from the above description, it is easy to know that $|S'| = q$. If S' is not an exact cover of S, there must exist $\lambda(C_i), \lambda(C_j) \in S'$ ($i \neq j$ and $1 \leq i, j \leq q$) such that $\lambda(C_i) \cap \lambda(C_j) \neq \emptyset$. Following from this, there must exist X_{ik} and X_{jg} in G_1 such that $\lambda(X_{ik}) = \lambda(X_{jg})$, which contradicts the injectivity constraint, thus λ is not a valid p-similarity mapping.

For instance, suppose there is a mapping λ for the example in Fig.11, where $\lambda(C_1) = C_1$ and $\lambda(C_2) = C_2$. Without loss of generality, we also assume that $\lambda(X_{11}) = X_{11}$, $\lambda(X_{12}) = X_{12}$ and $\lambda(X_{13}) = X_{13}$. In order to get a valid p-similarity mapping, we have to map X_{21}, X_{22}, X_{23} of G_1 to X_{11}, X_{12}, X_{21} of G_2. Here the injectivity constraint will force a node in X_{21}, X_{22}, X_{23} of G_1 to have the same mapping as some node in X_{11}, X_{12}, X_{13} of G_1. This shows λ violates the injectivity constraint.

Following from these, we know S' is indeed an exact cover for the X3C problem.

Conversely, if there is an exact cover S' for the X3C problem, we show that there is 1-1 p-similarity mapping λ from G_1 to G_2. We assume that $S' = \{C'_1, \ldots, C'_q\}$, where C'_i ($i \in [1, q]$) is some 3-element subset C_j ($1 \leq j \leq n$) in S.

We define a mapping λ as follows: (1) $\lambda(R_1) = R_2$, (2) $\lambda(C_i) = C'_i$ for $i \in [1, q]$, and (3) $\lambda(X_{ik}) = X'_{ik}$ for $i \in [1, q]$ and $k \in [1, 3]$, where $C_i = \{X_{i1}, X_{i2}, X_{i3}\}$ and $C'_i = \{X'_{i1}, X'_{i2}, X'_{i3}\}$. Then it is trivial to verify that λ is a 1-1 p-similarity mapping.

For instance, let $S' = \{C_1, C_3\}$ be an exact cover for the X3C problem in Fig.11. Let λ be a mapping defined 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 $k \in [1, 3]$. Then λ is a 1-1 p-similarity mapping.

Following the above description, we show that the 1-1 p-similarity problem is NP-complete. □

Proof of Corollary. 4.2. The maximum cardinality problem and the maximum overall similarity problem are NP-complete for both strong p-similarity and 1-1 p-similarity. The problems are already NP-hard even when only DAGs are considered.

Proof sketch: Let $w(v) = 1$ for each node v in G_1. Then if there exists a strong p-similarity or 1-1 p-similarity in the reductions of Theorem. 4.1, it is maximum.

Proof of Theorem. 4.3. CSP$^{\mathsf{s}}$, CSP$^{1-1}$, SSP$^{\mathsf{s}}$ and SSP$^{1-1}$ 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.

Proof sketch: We show that there exists an AFP-reduction (f, g) from the WIS problem to the SSP$^{\mathsf{s}}$ problem, following from which we can prove this result.

We first show how algorithm f is defined. Given an instance I_1 of the WIS problem as input, which is a graph $G(V, E)$ with a positive weight $w(v)$ on each node v, f outputs an instance I_2 of the SSP$^{\mathsf{s}}$ problem, which consists of (1) two graphs $G_1(V_1, E_1, L_1)$ and $G_2(V_2, E_2, L_2)$ such that $V_1 = V_2 = V$, $E_1 = E$, $E_2 = \emptyset$ and $L_1(v) = L_2(u)$ iff $v = u$, (2) a similarity matrix mat() such that mat$(v, u) = w(v)$ iff $L_1(v) = L_2(u)$ for any nodes v in G_1 and u in G_2 and mat$(v, u) = 0$ for other cases, and (3) a similarity threshold $\xi = 1$. It is easy to verify that algorithm f runs in PTIME.

Let $s_1 = \{v_1, \ldots, v_n\}$ be a set of nodes in G. Next we show

$s_2 = \{(v_1, v_1), \ldots, (v_n, v_n)\}$ is a strong p-similarity mapping from subgraph $G'_1[s_1]$ of G_1 to G_2 iff s_1 is an independent set of G. If s_1 is an independent set, then it is easy to know s_2 is a strong p-similarity mapping from $G'_1[s_1]$ to G_2, following from the definition of strong p-similarity. Conversely, if s_2 is a strong p-similarity mapping from $G'_1[s_1]$ to G_2, then we can easily verify for any nodes v_i, v_j $(i \neq j)$ in s_1, (v_i, v_j) is not in E_1. Otherwise, since there are no edges in graph G_2, there is no way to find a descendant v_j of v_i in graph G_2, and this contradicts the definition of strong p-similarity. Following from this, it is easy to define algorithm g. That is, given a feasible solution $s_2 = \{(v_1, v_1), \ldots, (v_n, v_n)\}$ of the $\mathsf{SSP^s}$ problem, g outputs a feasible solution $s_1 = \{v_1, \ldots, v_n\}$ of the WIS problem.

Now it is easy to verify that the weight of s_1 is equal to the one of s_2 (*i.e.,* $\mathsf{obj}_1(s_1) = \mathsf{obj}_2(s_2)$), and s_1 is the maximum independent set in I_1 iff s_2 is the *maximum overall similar* strong p-similarity mapping in I_2 (*i.e.,* $\mathsf{opt}_2(I_2) = \mathsf{opt}_1(I_1)$). Following from these, we show $(f,\ g)$ is indeed an AFP-reduction from the WIS problem to the $\mathsf{SSP^s}$ problem. Therefore, if the $\mathsf{SSP^s}$ problem is approximable within $O(|V_1|^{1-\epsilon})$ for some constant ϵ, then the WIS problem must be approximable within $O(|V_1|^{1-\epsilon})$, which contradicts the hardness result for the WIS problem [20].

The strong p-similarity mapping above is indeed a 1-1 p-similarity mapping. Following from this, the $\mathsf{SSP^s}$ problem is verified. Let $w(v) = 1$ for each node v in G_1, the AFP-reduction (f, g) above works for the $\mathsf{CSP^s}$ problem and the CSP^{1-1} problem. □

Proof of Theorem. 5.1. $\mathsf{CSP^s}$, CSP^{1-1}, $\mathsf{SSP^s}$ and SSP^{1-1} are all approximable within $O(log^2(n_1 n_2)/(n_1 n_2))$, where n_1 and n_2 are the numbers of nodes in input graphs G_1 and G_2, respectively.

Proof sketch: We show that there exists an AFP-reduction (f, g) from the $\mathsf{SSP^s}$ problem to the WIS problem, following from which we can show this result.

We first show how algorithm f is defined. Its input is an instance I_1 of the $\mathsf{SSP^s}$ problem, which consists of (1) two graphs $G_1(V_1, E_1, L_1)$ and $G_2(V_2, E_2, L_2)$, (2) similarity matrix $\mathsf{mat}()$ on nodes between 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. Note that there is a path from nodes u_1 to u_2 in graph G_2 iff there is an edge from nodes u_1 to u_2 in graph G_2^+. Then f produces graph $G(V, E)$ with a positive weight on each node based on graphs G_1 and G_2^+. Graph G is defined as follows:

(1) $V = \{[v, u] | v \in V_1, u \in V_2, \mathsf{mat}(v, u) \geq \xi\}$. That is, G is a product graph of G_1 and G_2.

(2) For any nodes $[v_1, u_1]$, $[v_2, u_2]$ in G, there exists an edge from $[v_1, u_1]$ to $[v_2, u_2]$ iff they satisfy one of the following conditions:

- ◦ $(v_1, v_2) \in E_1$, $v_1 \neq v_2$ and $(u_1, u_2) \in E_2^+$. Moreover, 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^+;
- ◦ $(v_1, v_2) \notin E_1$, $v_1 \neq v_2$, and u_1, u_2 are any nodes in G_2 such that $\mathsf{mat}(v_1, u_1) \geq \xi$ and $\mathsf{mat}(v_2, u_2) \geq \xi$. Moreover, 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^+.

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

Finally, algorithm f produces a graph $G^c(V, E^c)$, which is an instance I_2 of the WIS problem. Graph $G^c(V, E^c)$ is the *complementary* graph of $G(V, E)$ such that an edge $e \in E^c$ iff $e \notin E$. Note that graph G contains no self-loops. Also, we do not allow self-loops in G^c. It is easy to verify that algorithm f runs in PTIME.

Let $s_2 = \{[v_1, u_1], \ldots, [v_n, u_n]\}$ be a set of nodes in G^c. Next we show s_2 is an independent set in graph G^c iff $s_1 = \{(v_1, u_1), \ldots, (v_n, u_n)\}$ is a strong p-similarity mapping from subgraph $G_1[V'_1]$ of G_1 to G_2 such that $V'_1 = \{v_1, \ldots, v_n\}$.

If s_2 is an independent set of G^c, then 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 the edges in G guarantees the following: (1) If there is an edge from nodes $[v_i, u_i]$ to $[v_j, u_j]$, then $v_i \neq v_j$. Moreover, 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, *i.e.,* there exists an edge from v_i to v_j in G_2^+; And (2) nodes with self-loops in G_1 must be mapped to nodes with self-loops in G_2^+. Now it is easy to verify that s_1 is a strong p-similarity mapping from $G_1[V'_1]$ to G_2, following from the definition of strong p-similarity.

Conversely, if s_1 is a strong p-similarity mapping from $G_1[V'_1]$ to G_2, then we can easily show that for any nodes $[v_i, u_i]$ and $[v_j, u_j]$ $(i \neq j)$ of s_2, there exists an edge $([v_i, u_i], [v_j, u_j])$ in G, following from which s_2 is an independent set of G^c.

Putting these together, it is easy to define algorithm g. That is, given a feasible solution $s_2 = \{[v_1, u_1], \ldots, [v_n, u_n]\}$ of the WIS problem, g outputs a feasible solution $s_1 = \{(v_1, u_1), \ldots, (v_n, u_n)\}$ of the $\mathsf{SSP^s}$ problem.

Now it is easy to verify that the weight of s_1 is equal to the one of s_2 (*i.e.,* $\mathsf{obj}_1(s_1) = \mathsf{obj}_2(s_2)$), and s_1 is the *maximum overall similar* strong p-similarity mapping in I_1 iff s_2 is the maximum independent set in I_2 (*i.e.,* $\mathsf{opt}_2(I_2) = \mathsf{opt}_1(I_1)$). Following from these, we show $(f,\ g)$ is indeed an AFP-reduction from the $\mathsf{SSP^s}$ problem to the WIS problem. Since the WIS problem is approximable within $O(n/log^2 n)$, where n is the number of graph nodes [20], then the $\mathsf{SSP^s}$ problem must be approximable within $O((|V_1||V_2|)/log^2(|V_1||V_2|))$.

For the SSP^{1-1} problem, we need to add some extra edges to graph G_c above. For each node pair $[v, u_1]$ and $[v, u_2]$ $(u_1 \neq u_2)$, add an edge $([v, u_1], [v, u_2])$ to G_c. In this way, we guarantee the independent s_2 corresponds to a 1-1 p-similar mapping.

For the $\mathsf{CSP^s}$ (resp. CSP^{1-1}) problem, the AFP-reduction (f, g) for $\mathsf{SSP^s}$ (resp. SSP^{1-1}) can be used. Moreover, all nodes are assigned equal weights in this case. □

Proof of Proposition. 5.2. For any two graphs $G_1(V_1, E_1, L_1)$, $G_2(V_2, E_2, L_2)$, $\mathsf{mat}()$ and ξ, algorithm $\mathsf{compMaxCard^s}$ finds a p-similarity 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.

Proof sketch: Finding an independent set in a graph G is equivalent to finding a clique in its complementary graph, and vice versa. Accordingly, MAXIMUM INDEPENDENT SET is the same problem as MAXIMUM CLIQUE on the complementary graph, and vice versa.

Algorithm $\mathsf{compMaxCard^s}$ is a simulation of algorithm ISRemoval shown in Fig. 12, which aims to find a maximum clique of the graph. Algorithm ISRemoval is the dual of algorithm CliqueRemoval in [9], which aims to find a maximum independent set of the graph. One can easily find the resemblance between $\mathsf{compMaxCard^s}$ and ISRemoval.

The tricky part is the procedure greedyMatch, which corresponds to algorithm Ramsey (see a detailed explanation in [9]). (σ, I) is the return of procedure $\mathsf{greedyMatch}(H_1, H_2, H)$, where σ and I correspond to a clique and an independent set in the product graph respectively.

Basically, greedyMatch simulates Ramsey as follows:

- ◦ Line 1 of greedyMatch → line 1 of Ramsey;
- ◦ Line 2 of greedyMatch → line 2 of Ramsey;
- ◦ In greedyMatch H^+ and H^- correspond to $\mathcal{N}(v)$ and $\overline{\mathcal{N}}(v)$ respectively. Lines 3–9 of greedyMatch compute H^+ and H^-, where Ramsey does not explicitly mention this part;

```
Algorithm ISRemoval
Input: An undirected graph $G(V, E)$.
Output: An independent set $I$ of $G$.

1. $i$ :=1;  $(I_1, C_1)$ := Ramsey($G$);
2. while $V$ is not empty do
3.   $G := G \setminus C_i$;
4.   $i := i + 1$;  $(C_i, I_i)$ := Ramsey($G$);
5. return $\max(I_1, I_2, \ldots, I_i)$.

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

1. if $V = \emptyset$ 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 12: Algorithm ISRemoval

- ○ Lines 10 and 11 of greedyMatch $\rightarrow$ lines 3 and 4 of Ramsey respectively;
- ○ Line 12 of greedyMatch $\rightarrow$ line 5 of Ramsey;
- ○ Line 13 of greedyMatch $\rightarrow$ line 6 of Ramsey.

Putting these together, we conclude our algorithm compMaxCard$^{\mathsf{s}}$ guarantees to find a p-similarity 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.