The VLDB Journal (2013) 22:345–368
DOI 10.1007/s00778-012-0292-8

REGULAR PAPER

High efficiency and quality: large graphs matching

Yuanyuan Zhu · Lu Qin · Jeffrey Xu Yu · Yiping Ke · Xuemin Lin

Received: 19 September 2011 / Revised: 8 August 2012 / Accepted: 19 August 2012 / Published online: 25 September 2012

Abstract Graph matching plays an essential role in many real applications. In this paper, we study how to match two large graphs by maximizing the number of matched edges, which is known as maximum common subgraph matching and is NP-hard. To find exact matching, it cannot handle a graph with more than 30 nodes. To find an approximate matching, the quality can be very poor. We propose a novel two-step approach that can efficiently match two large graphs over thousands of nodes with high matching quality. In the first step, we propose an anchor-selection/expansion approach to compute a good initial matching. In the second step, we propose a new approach to refine the initial matching. We give the optimality of our refinement and discuss how to randomly refine the matching with different combinations. We further show how to extend our solution to handle labeled graphs. We conducted extensive testing using real and synthetic datasets and report our findings in this paper.

Keywords Graph matching · Maximum common subgraph · Vertex cover

Y. Zhu · L. Qin · J. X. Yu (✉) · Y. Ke
The Chinese University of Hong Kong, Sha Tin, Hong Kong, China
e-mail: yu@se.cuhk.edu.hk

Y. Zhu
e-mail: yyzhu@se.cuhk.edu.hk

L. Qin
e-mail: lqin@se.cuhk.edu.hk

Y. Ke
e-mail: ypke@se.cuhk.edu.hk

X. Lin
University of New South Wales, Sydney, NSW, Australia
e-mail: lxue@cse.unsw.edu.au

X. Lin
NICTA, Sydney, NSW, Australia

1 Introduction

Graph proliferates in a wide variety of applications, including social networks in psycho-sociology, attributed graphs in image processing, food chains in ecology, electrical circuits in electricity, road networks in transport, protein interaction networks in biology, topological networks on the Web. Graph processing has attracted great attention from both research and industrial communities.

Graph matching is an important type of graph processing, which aims at finding correspondences between the nodes/edges of two graphs to ensure that some substructures in one graph are mapped to similar substructures in the other. Graph matching plays an essential role in a large number of concrete applications [13].

Biology: Protein–protein interaction (PPI) networks play an important role in most biological processes, in which a node corresponds to a protein and an edge indicates the interaction between two proteins. Comparative analysis of PPI networks across species provides insightful views of similarities and differences between species at systemic level, and helps to identify conserved functional components across species. Graph matching can be effectively used for such PPI networks comparisons, to maximally identify the pairs of homologous proteins from two different organisms such that PPIs are conserved between matched pairs [28,37].

Biochemistry: The genome of an organism is represented as a graph with genes as nodes and binary relations between genes as edges, and the metabolic pathway is represented as another graph with enzymes as nodes and chemical compounds as edges. These two graphs are then matched to identify FRECs (functionally related enzyme clusters) that reveal important biological features of the organisms [24].

Medicine: The electroencephalogram (EEG) signal can be transformed into a graph with the extracted energy bursts as

nodes. Graph matching is applied to the comparison of two EEG signals to analyze different brain activities in terms of latency, frequency, energy, and activated areas [7].

Video Indexing: A region adjacency graph (RAG) is constructed to represent an object, where the nodes are segmented regions in video frames. Graph matching between two RAGs can be used to retrieve similar objects in video-shot collections [12].

Schema Matching: In data integration and service interoperability, schema matching is important, which aims at identifying correspondences between metadata structures or models. Consider a comparison shopping website that aggregates product offers from multiple independent online stores. Since each website can be modeled as a graph, graph matching can also be used to solve schema matching problems [22].

In the literature, a number of algorithms have been proposed for graph matching including exact matching [19,21,31] and approximate matching [4,10,17,20,25,27,32,34]. The exact approaches are able to find the optimal matching at the cost of exponential running time, while the approximate approaches are much more efficient but can get poor matching results. More importantly, most of them can only handle small graphs with tens to hundreds of nodes. As an indication, exactly matching two undirected graphs with 30 nodes may take 100,000 s. It is important to note that real-world networks nowadays can be very large. The existing approaches cannot efficiently match graphs even with thousands of nodes with high quality.

In this paper, we study the problem of matching two large graphs, which is formulated as follows. Given two graphs G_1 and G_2, we find a one-to-one matching between the nodes in G_1 and G_2 such that the number of the matched edges is maximized. The optimal solution to the problem corresponds to the maximum common subgraph (*MCS*) between G_1 and G_2, which is an NP-hard problem, and has been studied in decades. It is known to be very difficult to find a high-quality approximate matching efficiently even for small graphs. In order to meet the needs of handling large graphs for graph matching and analysis, we propose a novel approximate solution with polynomial time complexity while still attaining high matching quality.[1]

The main contributions of this paper are summarized below. We propose a novel two-step approach, namely, matching construction and matching refinement. In the first matching construction, we propose a new anchor-selection/expansion approach to compute an initial matching. We give heuristics to select a small number of important anchors using a new similarity score, which measures how two nodes in two different graphs are similar to be matched by taking both global and local information of nodes into consideration. We compute a good initial matching by expanding from the anchors selected. The expansion is based on structural similarity among the neighbors of nodes in two graphs. In the second matching refinement, we propose a new approach to refine the initial matching. The novelty of our refinement is as follows. First, we refine a matching M to a better one, which is most likely to exist and can be identified. Second, we consider the efficiency, and focus on a subset of nodes to refine while giving every node in the graphs a chance to be refined. We show the optimality of our refinement. We also show how to randomly refine matchings with different combinations. Our refinement can improve the matching quality with small overhead for both unlabeled and labeled graphs. We conducted extensive testing using real and synthetic datasets, and confirmed the quality and efficiency of our approach. The average ratio of our approximate matching to the exact matching is above 90 %, while the computational cost is less than 1 % of the state-of-the-art exact algorithms. This is a big step compared to all the approximate algorithms to match large graphs in the literature.

The rest of the paper is organized as follows. Section 2 discusses some related work. Section 3 gives the problem statement. Section 4 gives an overview of our two-step approach. Sections 5 and 6 discuss the matching construction and matching refinement. Section 7 extends our work to handle labeled graphs. Section 8 shows the performance results. Section 9 concludes this paper.

2 Related work

We discuss exact graph matching and approximate graph matching, according to whether (sub)graph isomorphism problem or maximum common subgraph problem is involved.

For exact graph matching, in the literature, most of the algorithms use backtracking (refer to Ullmann's algorithm for subgraph and graph isomorphism [31]). Existing solutions on finding the maximum common subgraph mainly focus on the maximum common node induced subgraph, and most techniques can hardly be used for the maximum common edge induced subgraph. Among them, Mcgregor [21] proposes a backtracking search method for finding the maximum common subgraph. An improved backtracking algorithm is given in [19] with time complexity $O(m^{n+1} \cdot n)$, where n and m are the numbers of vertices of G_1 and G_2, respectively. Abu-Khzam et al. [1] propose an algorithm that combines backtracking and vertex cover enumeration to solve the maximum common node induced subgraph problem. There are also some other studies to calculate the maximum common node induced subgraph by finding the maximum clique in the association graph [18,26,29].

[1] The conference version of this work was reported in [38].

The complexity of the maximum clique approach is no better than backtracking.

For approximate graph matching, there are three categories: propagation-based method, spectral-based method, and optimization-based method.

The propagation-based method is mainly based on the intuition that two nodes are similar if their respective neighborhoods are similar. In [22], a similarity flooding approach is proposed, which starts from string-based comparison of the vertices labels to obtain an initial alignment between nodes of two graphs and refines it by an iterative fix-point computation. Blondel et al. [8] construct a similarity measure between any two nodes in any two graphs based on Kleinberg's hub and authority idea of HITS algorithm [16]. This procedure will, in general, converge to different even and odd limits which will depend upon the initial conditions. Recently, IsoRank [28] extends the propagation-based method by adding the weight of propagation into the iteration process.

Spectral-based method aims to represent and distinguish structural properties of graphs using eigenvalues and eigenvectors of graph adjacency matrices. It is based on the observation that if two graphs are isomorphic, their adjacency matrices will have the same eigenvalues and eigenvectors. Since the computation of eigenvalues can be solved in polynomial time, it is used by a lot of works in graph matching [4,10,17,20,25,32,34]. Among these works, Umeyama [32] uses the eigendecomposition of adjacency matrices of the graphs to derive a simple expression of the orthogonal matrix that optimizes the objective function. Xu and King [35] propose a solution to the weighted isomorphism problem that combines the use of eigenvalues/eigenvectors with continuous optimization techniques. These two methods are only suitable for graphs with the same number of nodes. In [6], the authors solve the problem to handle graphs with different number of nodes, using the Laplacian eigenmaps scheme to perform a generalized eigendecomposition of the Laplacian matrix. Caseli and Kosinov in [11] propose a method of projecting vertex into eigen-subspace for graph matching, which is used for inexact many-to-many graph matching other than one-to-one matching, and in [10] extend Umeyama's work to match two graphs of different sizes by choosing the largest k-eigenvalues as the projection space. Knossow et al. in [17] improve the matching result by performing eigendecomposition on the Laplacian matrix since it is positive and semidefinite. Heat-kernel [34] is used to embed the nodes of the graph into vector-space based on the graph-spectral method, and the correspondence matrix between the embedded points of two graphs is computed by a variant of the Scott and Longuet-Higgins algorithm.

The optimization-based method aims to model graph matching as an optimization problem and solve it. The representative algorithms include PATH [36] and GA [37]. In PATH [36], the graph matching problem is formulated as a convex-concave programming problem, and is approximately solved. It starts from the convex relaxation and then iteratively solves the convex-concave programming problem by gradually increasing the weight of the concave relaxation and following the path of solutions thus created. GA [37] is a gradient method based approach, which starts from an initial solution and iteratively chooses a matching in the direction of a gradient objective function.

Aside from the propagation-/spectral-based methods that compute the similarity score by iterations of random walks or spectral decomposition of adjacency matrix, Jouili and Tabbone [15] propose a vector-based node signature that can be computed straightforwardly from the adjacency matrix. Here, every node is associated with a vector containing its node degree and the incident edge weights. The similarity between two nodes is computed based on their signatures, and the graph matching problem is reduced to a bipartite graph matching problem. A survey can be found in [27].

3 Problem statement

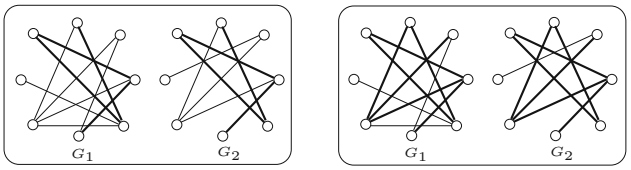
We first focus on undirected and unlabeled graphs, since the most difficult part for graph matching is the structural matching without any assistance of labels. We will discuss how to handle labeled graphs later in this paper. For a graph $G(V, E)$, we use $V(G)$ to denote the set of nodes and $E(G)$ to denote the set of edges.

Definition 1 GRAPH/SUBGRAPH ISOMORPHISM.

Graph G_1 is isomorphic to graph G_2, if and only if there exists a bijective function $f : V(G_1) \rightarrow V(G_2)$ such that for any two nodes $u_1 \in V(G_1)$ and $u_2 \in V(G_1)$, $(u_1, u_2) \in E(G_1)$ if and only if $(f(u_1), f(u_2)) \in E(G_2)$. G_1 is subgraph isomorphic to G_2, if and only if there exists a subgraph G' of G_2 such that G_1 is isomorphic to G'.

Definition 2 MAXIMUM COMMON SUBGRAPH. A graph G is the maximum common subgraph (*MCS*) of two graphs G_1 and G_2, denoted as $\mathsf{mcs}(G_1, G_2)$, if G is a common subgraph of G_1 and G_2, and there is no other common subgraph G', such that G' is larger than G.

The *MCS* of two graphs can be disconnected, and there are two kinds of *MCS*s, namely maximum common node induced subgraph (*MCSv*) and maximum common edge induced subgraph (*MCSe*). The former requires the *MCS* to be the node induced subgraph of both G_1 and G_2, and G' is larger than G iff $|V(G')| > |V(G)|$. The latter requires the *MCS* to be the edge induced subgraph of both G_1 and G_2, and G' is larger than G iff $|E(G')| > |E(G)|$. Figure 1 shows the difference between *MCSv* and *MCSe*. Given two graphs G_1 and G_2. Figure 1a shows the *MCSv* of



(a) **(b)**

Fig. 1 **a** *MCSv* and **b** *MCSe*

G_1 and G_2, whereas Fig. 1b shows the *MCSe* of G_1 and G_2. As can be seen from this example, *MCSe* can possibly get more common substructure for the given two graphs. In this paper, we adopt *MCSe* since it can possibly get more common substructure for the given two graphs, and we use *MCS* (mcs) to denote *MCSe*. Finding the *MCS* of two graphs is NP-hard.

Definition 3 GRAPH MATCHING. Given two graphs G_1 and G_2, a matching M between G_1 and G_2 is a set of vertex pairs $M = \{(u, v)|u \in V(G_1), v \in V(G_2)\}$, such that for any two pairs $(u_1, v_1) \in M$ and $(u_2, v_2) \in M$, $u_1 \neq u_2$ and $v_1 \neq v_2$. The optimal matching M of two graphs is the one with the largest number of matched edges. Finding the optimal matching M is the same as finding the *MCS*.

Problem Statement: We aim to compute the optimal matching M for two given graphs G_1 and G_2. For a given matching M, we evaluate its quality by computing $score(M)$ as follows.

$$score(M) = \frac{\sum_{(u_1,v_1)\in M} \sum_{(u_2,v_2)\in M} e_{u_1,u_2} \times e_{v_1,v_2}}{2} \quad (1)$$

where $e_{u,v} = 1$ if there is an edge between u and v, and $e_{u,v} = 0$, otherwise. Obviously, finding the optimal matching M is actually to find a matching with the maximum $score(M)$, and the maximum $score(M)$ is $|E(\mathsf{mcs}(G_1, G_2))|$.

4 An overview: construction and refinement

In the literature, in order to obtain the exact solution, backtracking search is commonly used to solve the *MCS* problem. Such backtracking search is infeasible for large graphs for two reasons.

- The search space is extremely large. For two given graphs with n and n' nodes, respectively, suppose $n < n'$, the search space of the backtracking method is $O((n'+1)^n)$, which is impractical even for $n > 30$. Although many heuristics can be used to reduce the search space, the complexity of the search space can hardly be reduced.
- For each partial solution generated in the backtracking, we need to decide whether we need to expand the current solution further or cut the current branch down. Suppose the current solution can match $|E_c|$ edges, and the current best solution can match $|E_b|$ edges. We need to estimate an upper bound $|E_u|$ of the maximum matching size between the unmatched parts of the two graphs. If $|E_c| + |E_u| \leq |E_b|$, we can cut the current branch down to reduce the search space. A tight upper bound can cut many useless branches and thus largely reduce the search space. However, a good upper bound can hardly be derived. In the literature, as far as we know, the best upper bound is given in [26], which is derived by only considering the degree of each node in the two graphs. Obviously, such an upper bound is very loose because it does not consider any structural information of the two graphs.

Based on the above two reasons, we know that using backtracking, the search space can hardly be reduced to an acceptable range especially when the sizes of the graphs are large.

It is known that the *MCS* problem is NP-hard, and it is also known that it is very difficult to obtain a tight, or even useful, approximation bound, because finding a maximum common subgraph of two graphs is equivalent to finding a maximum clique in their association graph, which cannot be approximated with ratio n^ϵ for any constant $\epsilon > 0$ unless P = NP [3]. For the quality of the *MCS* result, Almohamad and Duffaa in [2] give a bound of $O(n^2)$ based on the number of mismatched edges, where n is the size of the larger graph. This means that it may mismatch all the edges. Raymond et al. in [26] provide an upper bound for the size of the *MCS*, which is computed by sorting the degree sequences of two graphs separately followed by summarizing the corresponding smaller degrees. The bound is almost the smaller graph, without considering any structural information of the two graphs, which does not provide much information. For the time complexity, in [2], it is $O(n^6 L)$, where n is the size of the graph and L is the size of an LP model formulated for graph matching (at least n). It cannot handle graphs with more than 100 nodes.

In this paper, we propose a novel approach to solve the graph matching problem in two steps: matching construction and matching refinement. In the matching construction step, we construct the initial matching M by identifying anchors of two graphs G_1 and G_2 followed by expanding from the anchors. We do so based on a new similarity between nodes in the two different graphs, which combines both global and local information of nodes. In the matching refinement step, we refine the initial matching. We prove that our refinement is efficient and can generate a matching with good quality. The framework of the algorithm is shown in Algorithm 1 and is self-explained. We discuss the first step in Sect. 5, and the second step in Sect. 6. We show how to extend our solution to handle labeled graphs in Sect. 7.

Algorithm 1 match(G_1, G_2)

Require: two graphs, G_1 and G_2;
Ensure: a graph matching between G_1 and G_2;
1: $\mathcal{A} \leftarrow$anchor-selection (G_1,G_2); {refer to Algorithm 2}
2: $M \leftarrow$ anchor-expansion (G_1, G_2, $\mathcal{A}$); {refer to Algorithm 3}
3: $M \leftarrow$ refine(G_1, G_2, M); {refer to Algorithm 7}
4: **return** M;

5 Matching construction

In this section, we discuss how to select anchors and how to expand from the selected anchors to obtain the initial matching M for two graphs G_1 and G_2, using a new node similarity matrix S. The node similarity between $u \in G_1$ and $v \in G_2$ is very important because it indicates how likely the two nodes will be matched when computing the matching M.

5.1 Global and local node similarity

Let G_1 and G_2 be two graphs. The new node similarity matrix S we propose takes both global and local node similarities into consideration when matching nodes in two graphs.

$$S[u, v] = S_g[u, v] \times S_l[u, v] \tag{2}$$

Here, S is a $|V(G_1)| \times |V(G_2)|$ matrix, in which the element $S[u, v] \in [0, 1]$ represents the similarity of two nodes, u in G_1 and v in G_2. S is based on S_g and S_l, where S_g measures global similarity between u and v in the entire graphs G_1 and G_2, and S_l measures local similarity between u and v in their neighborhoods. We will introduce an existing global similarity below followed by the discussion on our new local similarity in this section.

Global node similarity: In the literature, the global similarity for nodes in two graphs can be the spectral-based similarity. The representative study is Umeyama's work [32] which is improved by [17]. Suppose G_1 and G_2 are two undirected graphs with the same number of nodes n. The Laplacian matrix $L_{n\times n}$ of graph G with n nodes is defined as $L = D - A$, where A is the adjacency matrix and D is the diagonal degree matrix. $A[u_1, u_2] = 1$ if $(u_1, u_2) \in E(G)$, and 0 otherwise. $D[u_1, u_1] = \sum_{(u_1,u_2)\in E(G)} A[u_1, u_2]$. We denote the Laplacian matrices of G_1 and G_2 as L_1 and L_2, respectively. Suppose the eigenvalues of L_1 and L_2 are $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n$ and $\beta_1 \geq \beta_2 \geq \cdots \geq \beta_n$, respectively. Since L_1 and L_2 are symmetric and positive-semidefinite, we have $L_1 = U_1\Lambda_1 U_1^T$ and $L_2 = U_2\Lambda_2 U_2^T$, where U_1 and U_2 are orthogonal matrices, and $\Lambda_1 = diag(\alpha_i)$ and $\Lambda_2 = diag(\beta_i)$. If G_1 and G_2 are isomorphic, there exists a permutation matrix P such that $PU_1\Lambda_1 U_1^T P^T = U_2\Lambda_2 U_2^T$. Let $P = U_2 D' U_1^T$ where $D' = diag(d_1, \ldots, d_n)$ and $d_i \in \{+1; -1\}$ accounts for the sign ambiguity in the eigendecomposition. When G_1 and G_2 are isomorphic, the optimum permutation matrix is P, which maximizes $tr(P^T \bar{U}_2 \bar{U}_1^T)$, where $\bar{U}_1$ and $\bar{U}_2$ are matrices that have the absolute value of each element of U_1 and U_2, respectively. When the numbers of nodes in G_1 and G_2 are not the same, we only choose the largest c eigenvalues [17]. Let $c = \min\{|V(G_1)|, |V(G_2)|\}$, and $\bar{U}_1'$ and $\bar{U}_2'$ be the first c columns of $\bar{U}_1$ and $\bar{U}_2$, respectively, the global similarity matrix can be computed with Eq. (3).

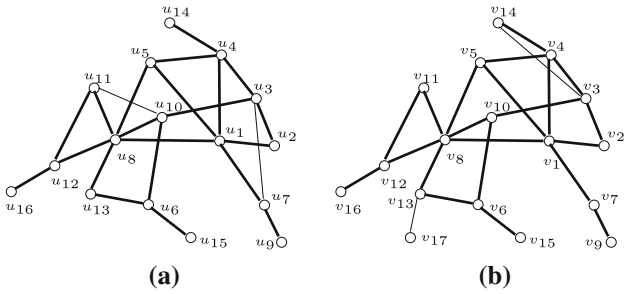


Fig. 2 Two graphs. **a** Graph G_1, **b** Graph G_2

$$S_g = \bar{U}_1' \bar{U}'^T_2 \tag{3}$$

Here, $S_g[u, v] \in [0, 1]$ is the global node similarity between the node u in $V(G_1)$ and the node v in $V(G_2)$.

Example 1 shows an example of matching two graphs using the global node similarity.

Example 1 Consider the two graphs in Fig. 2. We first compute their global node similarity matrix S_g shown in Fig. 3a. We construct a bipartite graph G_b with $|V(G_1)| + |V(G_2)|$ nodes, and for any $u \in V(G_1)$ and $v \in V(G_2)$, we add an edge $(u, v) \in E(G_b)$ with weight $S_g[u, v]$. We compute the maximum weighted bipartite matching of G_b and get the matching as $M = \{(u_1, v_1), (u_2, v_2), (u_3, v_7), (u_4, v_4), (u_5, v_5), (u_6, v_{12}), (u_7, v_{13}), (u_8, v_8), (u_9, v_{17}), (u_{10}, v_{10}), (u_{11}, v_3), (u_{12}, v_6), (u_{13}, v_{14}), (u_{14}, v_{15}), (u_{15}, v_{16}), (u_{16}, v_9)\}$. In this way, the number of matched edges is 10, which is far away from the optimal solution mcs(G_1, G_2), 21 (bold edges in Fig. 2). Comparing to the optimal solution, u_3 is mismatched to v_7 because they have a high global similarity, but obviously, the local structure near u_3 and the local structure near v_7 differ much.

Alternative global node similarity measures: Besides the global node similarity based on eigendecomposition, there are other global similarity measures based on the node importance in the graph in the literature, such as Katz score [9,14] and random walk with restart (RWR) [30]. A Katz score is a weighted count of the number of walks originating (or terminating) at a given node. The walks are weighted inversely by their length so that long and highly indirect walks count less, while short and direct walks count larger. The Katz score is given by the formula $\mathbf{r} = (I - bA)^{-1} bA\mathbf{u}$, where $\mathbf{r}$ is the

$$\begin{pmatrix}
.94 & .52 & .79 & .77 & .72 & .57 & .47 & .83 & .35 & .63 & .53 & .56 & .52 & .43 & .37 & .42 & .35 \\
.57 & .84 & .43 & .49 & .75 & .53 & .78 & .42 & .71 & .57 & .73 & .43 & .45 & .74 & .52 & .57 & .44 \\
.70 & .56 & .73 & .70 & .73 & .68 & .69 & .54 & .54 & .52 & .61 & .69 & .48 & .53 & .40 & .49 & .32 \\
.57 & .52 & .75 & .82 & .67 & .82 & .44 & .64 & .37 & .63 & .45 & .67 & .61 & .54 & .59 & .42 & .51 \\
.59 & .60 & .71 & .68 & .90 & .65 & .52 & .52 & .46 & .73 & .59 & .44 & .59 & .53 & .56 & .41 & .53 \\
.43 & .53 & .63 & .66 & .63 & .70 & .58 & .42 & .59 & .70 & .61 & .85 & .65 & .63 & .54 & .58 & .51 \\
.42 & .41 & .49 & .39 & .49 & .69 & .66 & .39 & .53 & .65 & .39 & .54 & .81 & .42 & .58 & .42 & .63 \\
.83 & .47 & .53 & .66 & .59 & .45 & .47 & 1.00 & .39 & .58 & .40 & .37 & .45 & .44 & .31 & .31 & .29 \\
.24 & .40 & .28 & .26 & .36 & .53 & .48 & .27 & .54 & .39 & .56 & .46 & .59 & .50 & .87 & .76 & .92 \\
.77 & .61 & .85 & .88 & .88 & .64 & .70 & .64 & .54 & .86 & .48 & .44 & .61 & .64 & .43 & .36 & .41 \\
.56 & .66 & .79 & .79 & .86 & .86 & .53 & .50 & .55 & .77 & .54 & .58 & .57 & .71 & .61 & .48 & .46 \\
.37 & .54 & .62 & .50 & .59 & .95 & .59 & .46 & .62 & .70 & .41 & .63 & .86 & .60 & .62 & .45 & .58 \\
.50 & .70 & .42 & .47 & .59 & .59 & .57 & .36 & .63 & .74 & .59 & .54 & .54 & .66 & .57 & .52 & .48 \\
.29 & .45 & .34 & .35 & .44 & .55 & .34 & .38 & .42 & .37 & .70 & .49 & .48 & .53 & .86 & .78 & .87 \\
.37 & .70 & .48 & .49 & .51 & .51 & .55 & .28 & .76 & .50 & .72 & .62 & .49 & .81 & .63 & .87 & .63 \\
.36 & .66 & .54 & .49 & .55 & .61 & .58 & .28 & .80 & .54 & .59 & .50 & .57 & .78 & .69 & .76 & .71
\end{pmatrix}$$

(a)

$$\begin{pmatrix}
.94 & .57 & .75 & .74 & .77 & .50 & .45 & .86 & .23 & .74 & .51 & .54 & .58 & .45 & .27 & .31 & .26 \\
.59 & .88 & .57 & .64 & .72 & .61 & .77 & .48 & .42 & .60 & .76 & .58 & .58 & .77 & .41 & .47 & .38 \\
.77 & .62 & .76 & .73 & .77 & .58 & .57 & .66 & .30 & .65 & .54 & .60 & .57 & .54 & .30 & .36 & .27 \\
.65 & .69 & .81 & .88 & .73 & .72 & .53 & .66 & .28 & .67 & .54 & .69 & .70 & .60 & .41 & .37 & .38 \\
.72 & .66 & .72 & .72 & .93 & .57 & .51 & .64 & .28 & .80 & .57 & .51 & .66 & .54 & .35 & .33 & .34 \\
.46 & .61 & .65 & .67 & .59 & .78 & .71 & .44 & .43 & .60 & .66 & .85 & .66 & .74 & .47 & .53 & .46 \\
.54 & .61 & .63 & .60 & .61 & .67 & .66 & .50 & .34 & .69 & .52 & .63 & .82 & .55 & .41 & .38 & .43 \\
.85 & .53 & .59 & .66 & .68 & .43 & .44 & .96 & .23 & .70 & .42 & .42 & .52 & .44 & .24 & .26 & .23 \\
.22 & .35 & .28 & .27 & .29 & .49 & .46 & .23 & .63 & .29 & .47 & .42 & .42 & .46 & .93 & .80 & .96 \\
.85 & .63 & .77 & .79 & .85 & .53 & .56 & .74 & .28 & .89 & .48 & .48 & .63 & .55 & .29 & .29 & .28 \\
.63 & .76 & .77 & .82 & .83 & .74 & .58 & .56 & .34 & .75 & .62 & .66 & .72 & .70 & .42 & .40 & .36 \\
.46 & .64 & .66 & .61 & .62 & .87 & .68 & .48 & .41 & .64 & .60 & .77 & .86 & .69 & .47 & .43 & .45 \\
.56 & .79 & .57 & .62 & .65 & .63 & .64 & .44 & .39 & .69 & .69 & .66 & .69 & .69 & .43 & .44 & .39 \\
.29 & .44 & .37 & .37 & .38 & .55 & .46 & .32 & .47 & .33 & .63 & .51 & .45 & .56 & .78 & .80 & .78 \\
.27 & .46 & .37 & .37 & .34 & .48 & .50 & .23 & .75 & .32 & .54 & .48 & .38 & .58 & .79 & .86 & .79 \\
.29 & .49 & .42 & .40 & .38 & .56 & .55 & .25 & .72 & .36 & .53 & .47 & .45 & .62 & .77 & .87 & .78
\end{pmatrix}$$

(b)

Fig. 3 Node similarity matrices: **a** S_g and **b** S

$N \times 1$ column vector containing Katz score for each node, I is the $N \times N$ identity matrix, $\mathbf{u}$ is a $N \times 1$ column vector with all entries equal to 1, and $b \in (0, 1)$ is the attenuation factor, which is $1/(d+1)$ by default in [14], where d is the maximum degree of the graph. The extent to which the weights attenuate with length is controlled by b. The RWR score is given by the formula $\mathbf{r} = (1-c)(I - cW)^{-1}\mathbf{u}$, where W is a transmit matrix where $W(i, j) = A(i, j)/\sum_i A(i, j)$, and $c \in [0, 1]$ is the positive probability, which means a surfer at a node will jump to a random node with probability $1 - c$. Under this random walk, the importance of a node v is the expected sum of the importance of all the nodes u that link to v. For two nodes, u in graph G_1 and v in graph G_2, they are considered highly similar if both have a high Katz/RWR score. The similarity matrix of two graphs becomes $S'_g = \mathbf{r_1 r_2}^T$. In Sect. 8, we report the effectiveness of these global node similarity measures.

The global node similarity gives a node similarity measure from the global point of view. However, when G_1 and G_2 are not sufficiently similar to each other, using global node similarity only is not sufficient to get a good matching because the global node similarity does not consider the local information for nodes in two graphs. We need a local node similarity.

Local node similarity: For any node v in graph G and $k \geq 0$, we define the k-neighborhood of v, $N_k(v)$, as the set of nodes in $V(G)$ such that $v \notin N_k(v)$ and for any $u \in N_k(v)$, the shortest distance from v to u is no more than k. The shortest distance is defined as the number of edges in the shortest path from v to u. The k-neighborhood subgraph of v in G, denoted as G_v^k, is defined as the induced subgraph over $N_k(v) \cup \{v\}$ in G. For two nodes $u \in V(G_1)$ and $v \in V(G_2)$, we measure their local node similarity by comparing the k-neighborhood subgraphs of them. Suppose $d(u)$ and $d(v)$ are the degrees of node u and v in G_1 and G_2, respectively, and suppose $d_{1,1}, d_{1,2}, \ldots$ is the degree sequence of node set $N_k(u)$ in G_u^k sorted in non-increasing order, and $d_{2,1}, d_{2,2}, \ldots$ is the degree sequence of node set $N_k(v)$ in G_v^k sorted in non-increasing order. Let $n_{min} = \min\{|N_k(u)|, |N_k(v)|\}$. We define a $|V(G_1)| \times |V(G_2)|$ local node similarity matrix S_l as follows.

$$S_l[u, v] = \frac{(n_{min} + 1 + D(u, v))^2}{(|V(G_u^k)| + |E(G_u^k)|)(|V(G_v^k)| + |E(G_v^k)|)} \quad (4)$$

$$D(u, v) = \frac{\min\{d(u), d(v)\} + \sum_{i=1}^{n_{min}} \min\{d_{1,i}, d_{2,i}\}}{2} \quad (5)$$

Here, $D(u, v)$ consists of two parts. The first part $\min\{d(u), d(v)\}$ is the ideal contribution of edges when matching u with v, and the second part $\sum_{i=1}^{n_{min}} \min\{d_{1,i}, d_{2,i}\}$ is the ideal contribution of edges when matching nodes in $N_k(u)$ with nodes in $N_k(v)$. We show that S_l has the following properties.

(1) $0 < S_l[u, v] \leq 1$.
(2) $S_l[u, v] \geq \frac{(|V(\mathsf{mcs}(G_u^k, G_v^k))| + |E(\mathsf{mcs}(G_u^k, G_v^k))|)^2}{(|V(G_u^k)| + |E(G_u^k)|)(|V(G_v^k)| + |E(G_v^k)|)}$.
(3) If G_u^k and G_v^k are isomorphic, and u matches v in the optimal matching of G_u^k and G_v^k, then $S_l[u, v] = 1$.
(4) If G_u^k is subgraph isomorphic to G_v^k, and u matches v in the optimal matching of G_u^k and G_v^k, we have $S_l[u, v] = \frac{|V(G_u^k)| + |E(G_u^k)|}{|V(G_v^k)| + |E(G_v^k)|}$.

For (1), it is obvious that $S_l[u, v] > 0$ holds, because both $(n_{min}+1+D(u, v))^2 > 0$ and $(|V(G_u^k)|+|E(G_u^k)|)(|V(G_v^k)| + |E(G_v^k)|) > 0$. $S_l[u, v] \leq 1$ can be showed as follows. Since $\min\{d(u), d(v)\} \leq d(u)$ and $\min\{d_{1,i}, d_{2,i}\} \leq d_{1,i}$, $D(u, v) \leq \frac{d(u) + \sum_{i=1}^{n_{min}} d_{1,i}}{2} = |E(G_u^k)|$. Similarly, $D(u, v) \leq |E(G_v^k)|$. By combining such two inequations with the fact that $n_{min} + 1 \leq |V(G_u^k)|$ and $n_{min} + 1 \leq |V(G_v^k)|$, we have $S_l[u, v] \leq 1$. For (2), since the node number of either G_u^k or G_v^k appearing in mcs can never exceed the minimum node number of G_u^k and G_v^k, $|V(\mathsf{mcs}(G_u^k, G_v^k))| \leq n_{min} + 1$. Also, $D(u, v)$ is known to be an upper bound of $|E(\mathsf{mcs}(G_u^k, G_v^k))|$, which is proved in [26]. Thus, this inequation holds. Here, $S_l[u, v]$ is an upper bound of such similarity, if we treat the right side of the equation in the property (2) as an accurate similarity of two nodes based on their *MCS*. For (3), this can be obtained based on the illustration of the first property, since when they are isomorphism, we have $n_{min} + 1 = |V(G_u^k)| = |V(G_v^k)|$ and $D(u, v) =$

$|E(G_u^k)| = |E(G_v^k)|$. For (4), it is because when G_u^k is subgraph isomorphic to G_v^k, we have $(n_{min} + 1 = |V(G_u^k)|$ and $D(u, v) = \frac{d(u) + \sum_{i=1}^{n_{min}} d_{1,i}}{2} = |E(G_u^k)|$, which leads to $S_l[u, v] = \frac{|V(G_u^k)| + |E(G_u^k)|}{|V(G_v^k)| + |E(G_v^k)|}$.

Note that our local similarity [Eq. (4)] is different from the vector-based node signature [15] which deals with edge weights. For an undirected and unweighted graph, the edge weights for all its incident edges are 1. This means that the node signature in [15] is merely its node degree, and measuring the similarity of two nodes by their degrees is not sufficient, because there might be many pairs of nodes, which share the same degree but are with different structures. In our local similarity measure, we do not only consider the degrees of two nodes but also consider their k-neighborhoods. [15] is one specific case of our local similarity when $k = 0$ for undirected and unweighted graphs.

Example 2 Reconsider the two graphs in Fig. 2. Let $k = 2$. The similarity matrix S of G_1 and G_2 is shown in Fig. 3b. We construct a bipartite graph G_b with $|V(G_1)|+|V(G_2)|$ nodes, and for any $u \in V(G_1)$ and $v \in V(G_2)$, we add an edge $(u, v) \in E(G_b)$ with weight $S[u, v]$ (instead of $S_g[u, v]$). We compute the maximum weighted bipartite matching of G_b and get the matching $M = \{(u_1, v_1), (u_2, v_2), (u_3, v_3), (u_4, v_4), (u_5, v_5), (u_6, v_{12}), (u_7, v_{13}), (u_8, v_8), (u_9, v_{17}), (u_{10}, v_{10}), (u_{11}, v_{14}), (u_{12}, v_6), (u_{13}, v_{11}), (u_{14}, v_{15}), (u_{15}, v_9), (u_{16}, v_{16})\}$. The number of matched edges is 13, which is better than 10 when only using the global similarity. But it is still much less than the optimal solution, 21.

A problematic approach to compute M using S: Umeyama [32] computes a matching M by applying the Hungarian algorithm to the node similarity matrix, which can be with S we newly proposed or S_g given in [32]. Using all the similar node pairs computed, a matching M can be found. In order to compute a matching, Umeyama constructs a bipartite graph G_b that includes $|V(G_1)| + |V(G_2)|$ nodes. For any node $u \in V(G_1)$ and node $v \in V(G_2)$, an edge (u, v) is added to G_b with weight $S[u, v]$ (or $S_g[u, v]$). The maximum weighted bipartite matching of G_b leads to a matching M of graphs G_1 and G_2. Such an approach has two drawbacks.

- Similarity optimality does not mean matched edge optimality, while our aim is to maximize the number of matched edges in two graphs. It is possible that two nodes are very similar in terms of S (or S_g) but the two nodes do not have many incident edges that help to increase the number of matched edges. As an example, suppose node $u_1 \in V(G_1)$ and node $v_1 \in V(G_2)$ all have degree 1, and $S[u_1, v_1] = 1.0$, and node $u_2 \in V(G_1)$ and node $v_2 \in V(G_2)$ all have degree 10, and $S[u_2, v_2] = 0.9$. Suppose (u_1, v_1) is in conflict with (u_2, v_2) when computing the maximum weighted bipartite matching. In constructing the initial matching, the algorithm may give up (u_2, v_2) because it has a lower similarity. But obviously, giving up (u_1, v_1) is a better solution because (u_2, v_2) can contribute a larger number of matched edges, although u_2 and v_2 have lower node similarity.
- This approach only considers the matching of individual nodes in two graphs, and does not consider whether the nodes around them can be well matched when it matches two nodes. In other words, matching $u \in V(G_1)$ with $v \in V(G_2)$ does not consider whether the nodes around u and v can be matched using the maximum weighted bipartite matching. When the nodes around u and v are mismatched, even if u and v are similar, it can significantly affect the quality of the final matching M.

Algorithm 2 anchor-selection (G_1, G_2)

Require: two graphs G_1 and G_2;
Ensure: a list of matched anchor pairs $\mathcal{A}$;
1: compute the similarity matrix S;
2: $\mathcal{A} \leftarrow \emptyset$; $\mathcal{S}_1 \leftarrow \emptyset$; $\mathcal{S}_2 \leftarrow \emptyset$;
3: **for all** $u \in V(G_1)$ and $v \in V(G_2)$ in decreasing order of their similarity $S[u, v]$ **do**
4: **if** $S[u, v] \geq \tau$ **and** $\min\{d(u), d(v)\} \geq \delta$ **and** $u \notin \mathcal{S}_1$ **and** $v \notin \mathcal{S}_2$ **then**
5: $\mathcal{A} \leftarrow \mathcal{A} \cup \{(u, v)\}$; $\mathcal{S}_1 \leftarrow \mathcal{S}_1 \cup \{u\}$; $\mathcal{S}_2 \leftarrow \mathcal{S}_2 \cup \{v\}$;
6: **return** $\mathcal{A}$;

Algorithm 3 anchor-expansion $(G_1, G_2, \mathcal{A})$

Require: two graphs, G_1 and G_2, and the anchor pairs $\mathcal{A}$;
Ensure: a graph matching M;
1: $M \leftarrow \mathcal{A}$; $\mathcal{Q} \leftarrow \emptyset$; $\mathcal{S}_1 \leftarrow \emptyset$; $\mathcal{S}_2 \leftarrow \emptyset$;
2: **for all** $(u, v) \in \mathcal{A}$ **do**
3: $\mathcal{S}_1 \leftarrow \mathcal{S}_1 \cup \{u\}$; $\mathcal{S}_2 \leftarrow \mathcal{S}_2 \cup \{v\}$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup (N(u) \times N(v))$;
4: **while** $\mathcal{Q} \neq \emptyset$ **do**
5: remove (u, v) from Q with the largest similarity $S_l[u, v]$;
6: **if** $u \notin \mathcal{S}_1$ **and** $v \notin \mathcal{S}_2$ **then**
7: $M \leftarrow M \cup \{(u, v)\}$; $\mathcal{S}_1 \leftarrow \mathcal{S}_1 \cup \{u\}$; $\mathcal{S}_2 \leftarrow \mathcal{S}_2 \cup \{v\}$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup (N(u) \times N(v))$;
8: **return** M;

5.2 Anchor selection and expansion

In our approach, we solve the two drawbacks as follows. Instead of matching all the nodes, we first match some important nodes as anchors. Every two anchors matched have high similarity and large degrees, and can contribute a large number of matched edges. Then, we expand from the anchors to match the other nodes using the local similarity S_l as the measure. Thus, our solution consists of two steps, namely anchor selection and anchor expansion.

The anchors selected play two important roles in matching construction. (1) The matching anchors contribute a large number of edges to the matching M. (2) The anchors are the

references to start with when matching the other nodes. For two nodes $u \in V(G_1)$ and $v \in V(G_2)$, we select (u, v) as matched anchors, if they satisfy the following two conditions.

(1) $\min\{d(u), d(v)\} \geq \delta$, where δ is the larger average degree of the two graphs, that is,

$$\delta = \max\left\{\frac{2 \times |E(G_1)|}{|V(G_1)|}, \frac{2 \times |E(G_2)|}{|V(G_2)|}\right\}.$$

(2) $S[u, v] \geq \tau$, where τ is a threshold and generally $\tau > 0.5$, and is one sensitive threshold that has impacts on graph matching. We will discuss it in Sect. 5.3.

The algorithm for anchor selection is shown in Algorithm 2. Given two graphs G_1 and G_2, it outputs a list of anchor pairs, denoted as $\mathcal{A}$. In the algorithm, $\mathcal{S}_1$ and $\mathcal{S}_2$ denote the sets of matched nodes in $V(G_1)$ and $V(G_2)$, respectively. Line 1 computes the similarity matrix S [Eq. (2)]. Line 3 tries to match the pairs (u, v) for all $u \in V(G_1)$ and $v \in V(G_2)$ in the decreasing order of their similarity. In this way, the most similar pairs will have a large chance to be matched as the anchors. Line 4 selects the nodes that satisfy the two conditions for anchor selection that are not matched before. If the conditions in line 4 are all satisfied, we add the pair (u, v) into the list $\mathcal{A}$ and add the matched nodes u and v into $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively in line 5. After checking all pairs, line 6 returns $\mathcal{A}$ as the anchor pairs.

Example 3 Consider the two graphs in Fig. 2. Suppose $\tau = 0.94$, using Algorithm 2, we can get the set of anchor pairs to be $\mathcal{A} = \{(u_1, v_1), (u_8, v_8)\}$. Obviously, the correct matching of the two pairs is very important in the final matching of G_1 and G_2. For the pair (u_9, v_{17}), although it satisfies the similarity constraint, it destroys the degree constraint. Obviously, expanding from the pair (u_9, v_{17}) to match other pairs is a bad choice.

Theorem 1 *The time complexity of Algorithm 2 is* $O(|V(G_1)|^2 \cdot (|V(G_1)| + |E(G_1)|) + |V(G_2)|^2 \cdot (|V(G_2)| + |E(G_2)|))$.

Proof 1 Algorithm 2 is to select anchors. Computing the global node similarity matrix needs $O(|V(G_1)|^3 + |V(G_2)|^3)$ time, and computing the local node similarity matrix needs $O(|V(G_1)|^2 \cdot |E(G_1)| + |V(G_2)|^2 \cdot |E(G_2)|)$ time. In lines 3-5, sorting all pairs needs $O(|V(G_1)| \cdot |V(G_2)| \cdot (\log(|V(G_1)|) + \log(|V(G_2)|)))$ time. Hence, the overall time complexity of Algorithm 2 is $O(|V(G_1)|^2 \cdot (|V(G_1)| + |E(G_1)|) + |V(G_2)|^2 \cdot (|V(G_2)| + |E(G_2)|))$. □

We illustrate the anchor expansion algorithm (Algorithm 3) to obtain a matching M. Let $\mathcal{A}$ be the anchor pairs (u, v) selected already. Initially, $M = \mathcal{A}$. Let $N(u)$ and $N(v)$ denote the immediate neighbors of u and v in graphs G_1 and G_2, respectively. For every matched pair (u, v) in the initial M, we put all $(N(u) \times N(v))$ pairs in a queue $\mathcal{Q}$, where $\mathcal{Q}$ is the set of candidate matching pairs sorted in decreasing order of their local similarity. In an iterative manner, we remove the pair (u, v) with the largest local similarity $S_l[u, v]$ [Eq. (4)] from $\mathcal{Q}$. If both u and v have not been matched before, we add (u, v) to M and put their all $(N(u) \times N(v))$ immediate neighbor pairs into $\mathcal{Q}$ for further consideration. We repeat it until $\mathcal{Q} = \emptyset$.

The example of anchor expansion is given below.

Example 4 Given the two graphs in Fig. 2. After we get the set of anchor pairs $\mathcal{A} = \{(u_1, v_1), (u_8, v_8)\}$. Using Algorithm 3, we can construct our matching $M = \{(u_1, v_1), (u_2, v_7), (u_3, v_3), (u_4, v_4), (u_5, v_5), (u_6, v_6), (u_7, v_2), (u_8, v_8), (u_9, v_9), (u_{10}, v_{10}), (u_{11}, v_{13}), (u_{12}, v_{12}), (u_{13}, v_{11}), (u_{14}, v_{14}), (u_{15}, v_{15}), (u_{16}, v_{16})\}$. The number of matched edges is 18.

Theorem 2 *The time complexity of Algorithm 3 is* $O(|V(G_1)| \cdot |V(G_2)| \cdot \min\{|V(G_1)|, |V(G_2)|\})$.

Proof 2 Algorithm 3 is to expand from the anchors selected. The dominant part of the time complexity is line 5 and line 7. For line 5, there are at most $|V(G_1)| \cdot |V(G_2)|$ pairs, and for each pair, it needs $O(|V(G_1)| \cdot |V(G_2)|)$ to obtain the one with the largest similarity. For line 7, there are at most $\min\{|V(G_1)|, |V(G_2)|\}$ matched pairs, and for each pair, it needs $O(|V(G_1)| \cdot |V(G_2)|)$ to compute the cartesian product. Therefore, the overall time complexity is $O(|V(G_1)| \cdot |V(G_2)| \cdot \min\{|V(G_1)|, |V(G_2)|\})$. □

5.3 Discussion on τ for anchor selection

In the matching construction step, the threshold τ used in anchor-selection (Algorithm 2) is an important factor for the matching quality. It should be neither too large nor too small. When τ is too large, very few nodes will be selected as anchors, which lead to more nodes to be mismatched in anchor-expansion. The reason is that anchor-expansion is designed as a greedy algorithm and can only achieve local optimum. For a node in a graph, the more steps it needs to be expanded from an anchor, the higher the probability to be mismatched. When τ is too small, a large number of anchor pairs may be selected, and many mismatched anchors are thus involved. Expanding from these mismatched anchors will hardly lead to a good matching result. We explain it using an example.

Example 5 Reconsider Example 3 in Sect. 5. Suppose we set τ to a very small value, that is, $\tau = 0.78$, for graphs in Fig. 2. A large set of anchor pairs is obtained: $\mathcal{A} = \{(u_1, v_1), (u_4, v_4), (u_5, v_5), (u_6, v_{12}), (u_7, v_{13}), (u_8, v_8),$

$(u_{10}, v_{10}), (u_{12}, v_6)\}$. If we then run anchor-expansion based on this $\mathcal{A}$, we obtain the matching result $M = \{(u_1, v_1), (u_2, v_2), (u_3, v_3), (u_4, v_4), (u_5, v_5), (u_7, v_{13}), (u_8, v_8), (u_9, v_{17}), (u_{10}, v_{10}), (u_{12}, v_6), (u_{13}, v_{11}), (u_{14}, v_{14}), (u_{15}, v_{16}), (u_{16}, v_{15})\}$. The number of matched edges is 16, which is not as good as the matching result in Example 4 (The number of matched edges is 18). This is because a small τ makes several mismatched anchor pairs $\{(u_6, v_{12}), (u_7, v_{13}), (u_{12}, v_6)\}$ included in $\mathcal{A}$. Expanding from such mismatched anchor pairs will make the matching result ineffective. On the other hand, suppose we set τ to a very high value, that is, $\tau = 0.98$. No anchor pair will be selected. Under such circumstances, one node will be randomly selected from each graph to form an anchor pair, and expanding from such a random anchor pair is unlikely to lead to a good matching.

It is difficult to determine an appropriate τ for graph matching, because τ is not related to graph sizes or degree distributions. One attempt we made is to collect some statistics from the similarity matrix S to see whether there is certain relationship between S and τ. First, we set τ to be a small value 0.5 and select an initial list of anchor pairs $\mathcal{A}_\mathcal{I}$ using Algorithm 2. Then, we divide the interval [0.5, 1] to disjoint small intervals (bins) with equal length l, and count the number of anchor pairs (u, v) in a bin if $S[u, v]$ is in its interval. We take τ to be used as the middle value of the bin, which has the smallest number of anchor pairs. This means that we choose a significant gap as threshold τ to be used. However, in our extensive experimental studies, the matching results obtained by using such a τ are not necessarily guaranteed to be good. In order to ensure a good initial graph matching, we select the best τ by exploring all τ with every step of l in [0.5, 1], for example, $0.5, 0.5 + l, 0.5 + 2l, \ldots$, and choosing the τ value that results in the best graph matching M (the output of anchor-expansion). Such process is shown in Algorithm 4. With Algorithm 4, Algorithm 1 can be modified by replacing line (1–2) with construct-opt(G_1, G_2). In order to further reduce the cost, as a heuristics, we adopt a hill climbing method. We set the default value of $\tau = 0.9$, and set the step length l to 0.02, and choose the direction by comparing the matching result for $\tau = 0.9 - l$ and $\tau = 0.9 + l$ to start climbing. The climbing will stop when the current value is less than the best value we have obtained or the value of τ is out of range [0.5, 1]. Such a hill climbing method stops within several steps and achieves good results, as can be seen in our testing.

Theorem 3 *The time complexity of Algorithm 4 is* $O(|V(G_1)|^2 \cdot (|V(G_1)| + |E(G_1)|) + |V(G_2)|^2 \cdot (|V(G_2)| + |E(G_2)|))$.

Proof 3 The dominant part of Algorithm 4 is line 2 and lines 3–7. For line 2, the time complexity of anchor-selection is shown to be $O(|V(G_1)|^2 \cdot (|V(G_1)| + |E(G_1)|) + |V(G_2)|^2 \cdot (|V(G_2)| + |E(G_2)|))$ in Theorem 1. For lines 3–7, since the time complexity of anchor-expansion is shown to be $O(|V(G_1)| \cdot |V(G_2)| \cdot \min\{|V(G_1)|, |V(G_2)|\})$ in Theorem 2, the total complexity of lines 3–7 is $O(c \cdot |V(G_1)| \cdot |V(G_2)| \cdot \min\{|V(G_1)|, |V(G_2)|\})$ where c is the steps that we need in the loop. Usually, $c = 0.5/l$ is a small constant for a given l. So the total complexity of Algorithm 4 is $O(|V(G_1)|^2 \cdot (|V(G_1)| + |E(G_1)|) + |V(G_2)|^2 \cdot (|V(G_2)| + |E(G_2)|))$. □

Algorithm 4 construct-opt (G_1, G_2)

Require: two graphs, G_1 and G_2;
Ensure: a graph matching between G_1 and G_2;
1: $\tau \leftarrow 0.5$; $M_{opt} \leftarrow \emptyset$;
2: $\mathcal{A}_\mathcal{I} \leftarrow$anchor-selection (G_1, G_2); {Algorithm 2}
3: **for all** $\tau_i = 0.5 + i \times l$ such that $\tau_i \in [0.5, 1]$ **do**
4: $\mathcal{A} \leftarrow \{(u, v) | (u, v) \in \mathcal{A}_\mathcal{I}, S[u, v] \geq \tau_i\}$;
5: $M \leftarrow$ anchor-expansion $(G_1, G_2, \mathcal{A})$; {Algorithm 3}
6: **if** $score(M) > score(M_{opt})$ **then**
7: $M_{opt} \leftarrow M$;
8: **return** M_{opt};

The time complexity of Algorithm 4 remains unchanged, compared to Algorithm 2 and Algorithm 3, because it only repeats anchor-expansion constant times. It is worth noting that anchor-selection is the dominant factor and anchor-expansion can be done very quickly in practice compared to anchor-selection. Some results are shown in Table 3 in Sect. 8, where the time of anchor-expansion means the total expansion time including the τ selection.

6 Matching refinement

The initial matching M is computed using the heuristics that match the anchors first followed by matching the nodes around the anchors in a top-down fashion. The heuristics used cannot guarantee that all the anchors are correctly matched. In this section, we propose a new approach to refine the initial matching M. It is important to note that our strategy is to refine the initial matching and is not to find a completely new matching. By refinement, we mean the following two things. First, we are not to explore all possibilities without a goal when we refine a matching. In other words, we refine a matching M to a better one which is most likely to exist and can be identified. Second, we consider the efficiency when refining a matching. In our approach, each time we focus on a subset of nodes to refine by excluding a subset of nodes and including a subset of nodes. The set of nodes to be excluded from refinement at one time is neither large nor small. Also, we give every node in the graphs a chance to be refined.

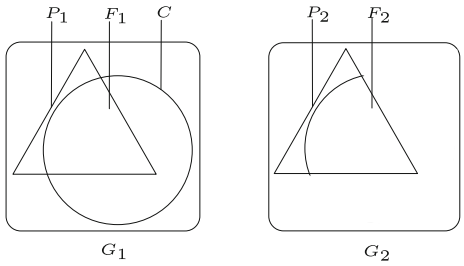


Fig. 4 Vertex cover refinement

6.1 Vertex cover based refinement

We use a vertex cover C to refine a matching M. A vertex cover C of a graph G is a subset of nodes in $V(G)$, that is, $C \subseteq V(G)$, such that for every edge $(u, v) \in E(G)$, we have $u \in C$ or $v \in C$. A minimum vertex cover of graph G is a vertex cover with the minimum number of nodes. A vertex cover C of G is a minimal vertex cover, if there does not exist a vertex cover C' of G such that $C' \subset C$. A set of nodes C is a vertex cover of graph G if and only if its complement $I = V(G) - C$ is an independent set of G. Here, an independent set I of G is a subset of nodes in $V(G)$, that is, $I \subseteq V(G)$, such that for any $u \in I$ and $v \in I$, $(u, v) \notin E(G)$.

Below, we introduce some notations we use to refine a matching M based on vertex cover. Suppose we match two graphs G_1 and G_2, and M is a matching found. Let P_1 and P_2 be the matched nodes in G_1 and G_2, respectively, using the matching M. For any $(u, v) \in M$, we have $u \in P_1$ and $v \in P_2$. Given a cover C of G_1, we use F_1 to denote $C \cap P_1$. For any subset of nodes $S \subseteq P_1$, we use $M[S]$ to denote the corresponding matched part of S in P_2 using matching M. For any subset of nodes $S \subseteq P_2$, we use $M^{-1}[S]$ to denote the matched part of S in P_1 using matching M. Let $F_2 = M[F_1]$. The relationships among G_1, G_2, P_1, P_2, F_1, F_2 and C are illustrated in Fig. 4.

The vertex cover structure plays an important role when we match two graphs G_1 and G_2. It allows us to focus on one graph G_1, with the assistance of its vertex cover. The intuition is as follows. By definition, a vertex cover of G_1 is the set of nodes that covers all possible edges in G_1. This implies that a node in the vertex cover can possibly have many edges to cover (or possibly have many matched edges with another graph G_2).

A vertex cover C of G_1 divides $V(G_1)$ into three parts, $F_1 = C \cap P_1$, $C - F_1$ and $V(G_1) - C$. The implications are given below. The nodes in F_1 are most likely to lead to good matches, based on the definition of vertex cover. We exclude nodes in F_1 to refine. We include nodes in $V(G_1) - C$ to refine, because the complement of the vertex cover $V(G_1) - C$ is an independent set. Such a property makes it possible to apply some efficient polynomial algorithms for optimizing the matching. For $C - F_1$, we will first discuss how to refine by excluding nodes in $C - F_1$, and then discuss how to include nodes $C - F_1$ to refine.

6.2 Refinement and its optimality

Given two graphs G_1 and G_2, a matching M, and a vertex cover C of G_1, we give a refinement $M^+(C)$ of M, and show its optimality below.

First, we show how to obtain a refinement $M^+(C)$ of M. We build a complete weighted bipartite graph G_b. On one side, G_b includes the nodes in $V(G_1) - C$, and on the other side, G_b includes the nodes in $V(G_2) - F_2$. For any node $u \in V(G_1) - C$ and node $v \in V(G_2) - F_2$, we add an edge (u, v) in $E(G_b)$, and the weight of the edge (u, v) is defined as follows.

$$w(u, v) = |M[N(u) \cap F_1] \cap (N(v) \cap F_2)| \tag{6}$$

where $N(u)$ and $N(v)$ are the sets of immediate neighbors of u and v in graphs G_1 and G_2, respectively. Intuitively, $w(u, v)$ is the contribution of the matched edges if we match u in graph G_1 with v in graph G_2. Next, we find the maximum weighted bipartite matching M_b of G_b using the Hungarian algorithm, such that the total weight of edges in M_b is maximized. We obtain our new matching $M^+(C)$ as follows.

$$M^+(C) = (M \cap (F_1 \times F_2)) \cup M_b \tag{7}$$

where $F_1 \times F_2$ is the cartesian product of F_1 and F_2. It includes all pairs (u, v) such that $u \in F_1$ and $v \in F_2$.

Example 6 Continue with Example 4. To make it simpler, let's only consider part of the matching in the initial matching. Suppose the first graph in Fig. 5a is the partial graph induced by nodes $\{u_6, u_8, u_{11}, u_{12}, u_{13}\}$ in G_1, and the second graph in Fig. 5b is the partial graph induced by nodes $\{v_6, v_8, v_{11}, v_{12}, v_{13}\}$ in G_2. In the initial matching M generated in Example 4, only three edges are matched, which is showed as the bold edges in Fig. 5a, b. We have $P_1 = \{u_6, u_8, u_{11}, u_{12}, u_{13}\}$ and $P_2 = \{v_6, v_8, v_{11}, v_{12}, v_{13}\}$. Suppose $C = \{u_6, u_8, u_{12}\}$, we have $F_1 = C \cap P_1 = \{u_6, u_8, u_{12}\}$ and $F_2 = M[F_1] = \{v_6, v_8, v_{12}\}$. In the bipartite graph G_b, the left part consists of the nodes in $V(G_1) - C$, which is $\{u_{11}, u_{13}\}$, and the right part consists of the nodes in $V(G_2) - F_2$, which is $\{v_{11}, v_{13}\}$. The graph G_b is shown in Fig. 5c. For the edge (u_{11}, v_{11}), its weight is 2 because if we match node u_{11} with node v_{11}, 2 edges will be matched in the original graphs, that is, edge (u_{11}, u_8) is matched to edge (v_{11}, v_8), and edge (u_{11}, u_{12}) is matched to edge (v_{11}, v_{12}). The maximum weighted bipartite matching of G_b is $M_b = \{(u_{11}, v_{11}), (u_{13}, v_{13})\}$. Modifying the result in Example 4 using the new matching, we can improve the number of matched edges from 18 to 20. Similarly, we can refine matching pairs $\{(u_2, v_7), (u_7, v_2)\}$ to be $\{(u_2, v_2), (u_7, v_7)\}$

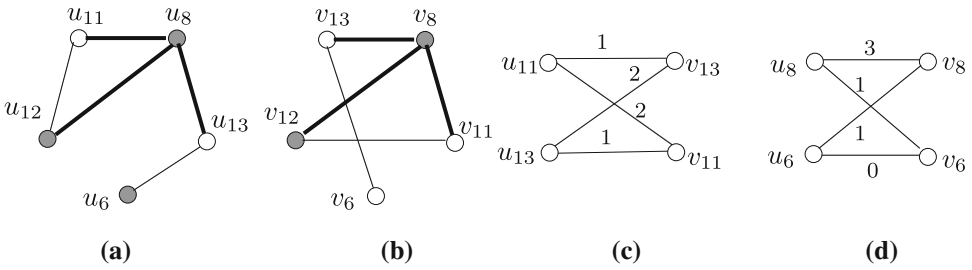


Fig. 5 Vertex cover refinement example. **a** G_1, **b** G_2, **c** $C = \{u_6, u_8, u_{12}\}$, **d** $C = \{u_{11}, u_{12}, u_{13}\}$

such that it will improve the number of matched edges to 21, which is the optimal value in this example.

Second, we give the optimality of $M^+(C)$ over a matching space $\mathcal{M}$. The space $\mathcal{M}$ is a set of matchings between nodes in G_1 and G_2, such that for any matching M', $M' \in \mathcal{M}$ if and only if $M' \cap (F_1 \times F_2) = M \cap (F_1 \times F_2)$ and $M' \cap ((C - F_1) \times V(G_2)) = \emptyset$. For the matching M, a matching $M' \in \mathcal{M}$, if and only if the matching for nodes in F_1 is not changed and the matching for nodes in $C - F_1$ is $\emptyset$. The second condition can also be expressed as $M'[C - F_1] = \emptyset$.

Theorem 4 *Suppose* $min = \min\{|V(G_1)| - |C|, |V(G_2)| - |F_2|\}$ *and* $max = \max\{|V(G_1)| - |C|, |V(G_2)| - |F_2|\}$, *then we have:*

(1) $|\mathcal{M}| = \sum_{i=0}^{min} \frac{min! \times max!}{i! \times (min - i)! \times (max - i)!}$ *and* $\frac{max!}{(max - min)!} \leq |\mathcal{M}| \leq (max + 1)^{min}$,
(2) $M \in \mathcal{M}$,
(3) $M^+(C) \in \mathcal{M}$ *and*
(4) $M^+(C)$ *is optimal in* $\mathcal{M}$.

Proof 4 We prove it step by step.

(1) To make things simple and without loss of generality, we assume $|V(G_1)| - |C| \leq |V(G_2)| - |F_2|$, then $min = |V(G_1)| - |C|$ and $max = |V(G_2)| - |F_2|$. Since $V(G_1) - C$ and $V(G_2) - F_2$ are the included parts of G_1 and G_2, respectively, we only consider the number of different matchings between $V(G_1) - C$ and $V(G_2) - F_2$. Suppose in $V(G_1) - C$, there are i nodes that participate in the matching in $\mathcal{M}$, there are C^i_{min} different selections of the i nodes, and for each selection, there are P^i_{max} different matchings between the i nodes and nodes in $V(G_2) - F_2$. There are totally $C^i_{min} \times P^i_{max}$ different matchings for a certain i. Since $i \in [0, min]$, the total number of different matchings is

$$|\mathcal{M}| = \sum_{i=0}^{min} C^i_{min} \times P^i_{max} = \sum_{i=0}^{min} \frac{min! \times max!}{i! \times (min - i)! \times (max - i)!}$$

When $i = min$, we have:

$$|\mathcal{M}| \geq C^i_{min} \times P^i_{max} = \frac{max!}{(max - min)!}$$

If we remove the constraint that different nodes in $V(G_1) - C$ must match different nodes in $V(G_2) - F_2$, each node in $V(G_1) - C$ will have $max + 1$ choices include max nodes in $V(G_2) - F_2$ and an empty match. The number of different relaxed matchings is then changed to $(max+1)^{min}$ which is an upper bound of $|\mathcal{M}|$.

(2) We only need to prove that M satisfies the two conditions of $\mathcal{M}$. For the first condition, obviously, $M \cap (F_1 \times F_2) = M \cap (F_1 \times F_2)$. For the second condition, the part $C - F_1$ is the nodes in C that are not matched in M, so $M[C - F_1] = \emptyset$. As a result, $M \cap ((C - F_1) \times V(G_2)) = \emptyset$.

(3) We need show that $M^+(C)$ satisfies the two conditions of $\mathcal{M}$.

– For the first condition, we have:

$$\begin{aligned} &M^+(C) \cap (F_1 \times F_2) \\ &= ((M \cap (F_1 \times F_2)) \cup M_b) \cap (F_1 \times F_2) \\ &= (M \cap (F_1 \times F_2)) \cup (M_b \cap (F_1 \times F_2)) \end{aligned}$$

Since M_b only includes nodes in $V(G_1) - C$ and $V(G_2) - F_2$, we have $M_b \cap (F_1 \times F_2) = \emptyset$. As a result, $M^+(C) \cap (F_1 \times F_2) = M \cap (F_1 \times F_2)$.

– For the second condition, we have:

$$\begin{aligned} &M^+(C) \cap ((C - F_1) \times V(G_2)) \\ &= ((M \cap (F_1 \times F_2)) \cup M_b) \cap ((C - F_1) \times V(G_2)) \\ &= ((M \cap (F_1 \times F_2)) \cap ((C - F_1) \times V(G_2))) \\ &\cup (M_b \cap ((C - F_1) \times V(G_2))) \end{aligned}$$

Moreover, we have $(M \cap (F_1 \times F_2)) \cap ((C - F_1) \times V(G_2)) = \emptyset$, because $M \cap ((C - F_1) \times V(G_2)) = \emptyset$ is already proved in (2) and $M_b \cap ((C - F_1) \times V(G_2)) = \emptyset$ due to the fact that M_b does not contain any nodes in $C - F_1$. Thus, we have $M^+(C) \cap ((C - F_1) \times V(G_2)) = \emptyset$.

(4) For any matching $M' \in \mathcal{M}$, we define a matching M'_b as $M'_b = M' \cap ((V(G_1) - C) \times (V(G_2) - F_2))$. We use $score_b(M_b)$ to denote the total weight for

the bipartite matching M_b of the bipartite graph G_b. We claim: (a) M'_b is a bipartite matching of G_b; (b) $score(M') = score_b(M'_b) + score(M \cap (F_1 \times F_2))$; (c) $score(M^+(C)) = score_b(M_b) + score(M \cap (F_1 \times F_2))$.

– For (a), it is obvious because of two reasons. (1) M'_b only contains the nodes in $V(G_1) - C$ and $V(G_2) - F_2$, which is exactly the set of nodes in G_b. (2) Any edge in M'_b is also an edge of G_b since G_b is a complete bipartite graph.
– For (b), we have:

$$\begin{aligned} &score(M') \\ &= score(M' \cap ((C \cup (V(G_1) - C)) \\ &\quad \times (F_2 \cup (V(G_2) - F_2)))) \\ &= score((M' \cap (C \times F_2)) \\ &\quad \cup (M' \cap (C \times (V(G_2) - F_2))) \\ &\quad \cup (M' \cap ((V(G_1) - C) \times F_2)) \\ &\quad \cup (M' \cap ((V(G_1) - C) \times (V(G_2) - F_2)))) \end{aligned}$$

Since $C \times F_2$, $C \times (V(G_2) - F_2)$, $(V(G_1) - C) \times F_2$ and $(V(G_1) - C) \times (V(G_2) - F_2)$ are mutually exclusive with each other, we have:

$$\begin{aligned} &score(M') \\ &= score(M' \cap (C \times F_2)) \\ &\quad + score(M' \cap (C \times (V(G_2) - F_2))) \\ &\quad + score(M' \cap ((V(G_1) - C) \times F_2)) \\ &\quad + score(M' \cap ((V(G_1) - C) \times (V(G_2) - F_2))) \end{aligned}$$

Since $M'[F_1] = F_2$ and $M'[C - F_1] = \emptyset$, we have $M'[C] = M'[F_1] \cup M'[C - F_1] = F_2$, and thus $M' \cap (C \times (V(G_2) - F_2)) = \emptyset$. Since $M'^{-1}[F_2] = F_1$ and $F_1 \subseteq C$, we have $M' \cap ((V(G_1) - C) \times F_2) = \emptyset$. We also have:

$$\begin{aligned} &M' \cap (C \times F_2) \\ &= M' \cap (((C - F_1) \cup F_1) \times F_2) \\ &= (M' \cap ((C - F_1) \times F_2)) \cup (M' \cap (F_1 \times F_2)) \\ &= M' \cap (F_1 \times F_2) \end{aligned}$$

The last equation is due to $M' \cap ((C - F_1) \times F_2) = \emptyset$ because $M'[C - F_1] = \emptyset$. Since $M' \cap (F_1 \times F_2) = M \cap (F_1 \times F_2)$, we can derive:

$$\begin{aligned} &score(M') \\ &= score(M' \cap (F_1 \times F_2)) \\ &\quad + score(M' \cap ((V(G_1) - C) \times (V(G_2) - F_2))) \\ &= score(M \cap (F_1 \times F_2)) + score(M'_b) \end{aligned}$$

We only need to prove $score(M'_b) = score_b(M'_b)$. Since $V(G_1) - C$ is a independent set which only have edges with C and C is the excluded part of the matching M', we can derive that $score(M' \cap ((V(G_1) - C) \times (V(G_2) - F_2)))$ only consists of the contributions of the edges $(u, v) \in E(G_1)$ such that $u \in C$ and $v \in V(G_1) - C$. From the construction of G_b, the contribution for each $v \in V(G_1) - C$ in M' is just the weight $(v, M'[v])$ in the bipartite graph G_b. This implies $score(M'_b) = score_b(M'_b)$.
– For (c), it can be easily derived from (b) because $M^+(C) \in \mathcal{M}$ and $M_b = M^+(C) \cap ((V(G_1) - C) \times (V(G_2) - F_2))$.

Since M_b is the maximum weight bipartite matching of G_b, we have $cost_b(M_b) \geq cost_b(M'_b)$. Hence, we have:

$$\begin{aligned} score(M^+(C)) &= score_b(M_b) + score(M \cap (F_1 \times F_2)) \\ &\geq score_b(M'_b) + score(M \cap (F_1 \times F_2)) \\ &= score(M') \end{aligned}$$

As a result, $M^+(C)$ is the optimal solution in $\mathcal{M}$. The theorem holds. □

Theorem 4 shows that the size of $\mathcal{M}$ is exponentially large. Both M and $M^+(C)$ are elements in $\mathcal{M}$, and $M^+(C)$ is the optimal matching for all matchings in $\mathcal{M}$. It implies that $M^+(C)$ is the best among a large number of matchings in $\mathcal{M}$ and $score(M^+(C)) \geq score(M)$. For two graphs with 2,000 nodes each, the number of nodes in a vertex cover can be assumed as 1,000 (50 %) reasonably. $M^+(C)$ is the best among a factorial of 1,000 (1,000!) possible matchings.

6.3 Randomly refinement excluding $C - F_1$

If M itself is an optimal matching in $\mathcal{M}$, or the selected vertex cover C includes most nodes in G_1 that are not well matched, it is possible that $M^+(C)$ cannot improve M. As an example, suppose $C = \{u_{11}, u_{12}, u_{13}\}$, in Example 6, then the new bipartite graph G_b is the one shown in Fig. 5d. In other words, using the maximum weighted bipartite matching of G_b, the matching $M^+(C)$ might be the same with M. The reason is that the mismatched nodes are excluded by the vertex cover C to refine. We give an approach based on two strategies to solve such a problem. (1) Making C smaller, such that more mismatched nodes can be included and thus can be used to refine. (2) Iteratively refining the current matching using different vertex covers, such that every mismatched node will have a chance to be included to refine. The first strategy is based on the following Lemma.

Lemma 1 *For any two vertex covers C_1 and C_2 of G_1, if $C_1 \subseteq C_2$, then $score(M^+(C_1)) \geq score(M^+(C_2))$.*

Proof 5 Suppose $\mathcal{M}(C_1)$ and $\mathcal{M}(C_2)$ are the matching spaces generated by C_1 and C_2, respectively. We use $F_1(C_1)$, $F_1(C_2)$, $F_2(C_1)$, and $F_2(C_2)$ to denote F_1 generated by C_1,

Algorithm 5 select-random-cover (G)

Require: a graph G;
Ensure: a randomly selected minimal cover of G;
1: $\mathcal{L} \leftarrow$ shuffled nodes in $V(G)$; $C \leftarrow \emptyset$;
2: **for all** $u \in \mathcal{L}$ **do**
3: **if** $\exists(u, v) \in E(G)$, s.t. $v \notin C$ **then** $C \leftarrow C \cup \{u\}$;
4: **for all** $u \in C$ **do**
5: **if** $C - \{u\}$ is a vertex cover of G **then** $C \leftarrow C - \{u\}$;
6: **return** C;

Algorithm 6 refine (G_1, G_2, M)

Require: two graphs G_1 and G_2, and the matching M;
Ensure: a refined matching M;
1: **while** M is updated or it is the first iteration **do**
2: **for** $i = 1$ **to** X **do**
3: $G \leftarrow$ random selection between G_1 and G_2;
4: $C \leftarrow$ select-random-cover (G);
5: compute $M^+(C)$;
6: **if** $score(M^+(C)) > score(M)$ **then** $M \leftarrow M^+(C)$;
7: **return** M;

F_1 generated by C_2, F_2 generated by C_1, and F_2 generated by C_2, respectively. Since we have $F_1(C_1) \subseteq F_1(C_2)$ and $F_2(C_1) \subseteq F_2(C_2)$, we need show $\mathcal{M}(C_2) \subseteq \mathcal{M}(C_1)$. For any $M' \in \mathcal{M}(C_2)$, we have:

$$M' \cap (F_1(C_2) \times F_2(C_2)) = M \cap (F_1(C_2) \times F_2(C_2))$$
$$M' \cap ((C_2 - F_1(C_2)) \times V(G_2)) = \emptyset$$

Since $F_1(C_1) \times F_2(C_1) \subseteq F_1(C_2) \times F_2(C_2)$, we derive:

$$M' \cap (F_1(C_1) \times F_2(C_1)) = M \cap (F_1(C_1) \times F_2(C_1))$$

We also have $C_1 - F_1(C_1) \subseteq C_2 - F_1(C_2)$, which yields:

$$M' \cap ((C_1 - F_1(C)) \times V(G_2)) = \emptyset$$

Thus, we have $M' \in \mathcal{M}(C_1)$, that is, $\mathcal{M}(C_2) \subseteq \mathcal{M}(C_1)$. Since $M^+(C_1)$ and $M^+(C_2)$ are the optimal solutions in $\mathcal{M}(C_1)$ and $\mathcal{M}(C_2)$ respectively, we have $score(M^+(C_1)) \geq score(M^+(C_2))$.

In order to make C small, a straightforward way is to find a minimum vertex cover of G_1. This method is not practical for two reasons. (1) Finding a minimum vertex cover of a graph is NP-hard. (2) In a minimum vertex cover, the mismatched nodes do not have a chance to be included to refine. To avoid these, we use a minimal vertex cover instead, because (1) a minimal vertex cover is easy to be found and (2) the number of different minimal vertex covers for a graph is much larger than the number of different minimum vertex covers. Thus, a minimal vertex cover gives the mismatched nodes in a minimum vertex cover more chances to be refined.

The approach to randomly select a minimal vertex cover of graph G is shown in Algorithm 5. First, in line 1, we shuffle all nodes in the graph and put them into a list $\mathcal{L}$, such that any permutation of $V(G)$ has the same probability in $\mathcal{L}$. In lines 2–3, we find a vertex cover of G by adding node in $\mathcal{L}$ one by one. For any node to be added, we add it into the vertex cover if and only if it contributes at least one edge to the currently covered edges (line 3). This operation can be implemented as follows. For every node in the graph, we maintain its number of uncovered edges, which is initially set to be the degree of the corresponding node. Every time before we add a new node into the cover, we first check its number of uncovered edges. If it is 0, we skip the node, and continue to add the next one in $\mathcal{L}$. Otherwise, we add the node into the cover, and traverse its adjacent nodes in the graph. For each adjacent node, we decrease its number of uncovered edges by 1. In such a way, the total complexity for line 2–3 is $O(|E(G)|)$, since every edge in G is visited at most once. Lines 4–5 make the current vertex cover minimal by removing those useless nodes, such that the removal of such nodes does not influence any edge currently covered. The following lemma shows that, for any minimal cover C of a graph G, there are considerable number of ways for Algorithm 5 to generate C.

Lemma 2 *For any minimal vertex cover C of graph G, there are at least $|C|! \times |V(G) - C|!$ permutations of $V(G)$, such that Algorithm 5 generates C.*

Proof 6 We construct the $|C|! \times |V(G) - C|!$ permutations as follows. For each permutation, we put C in the front in any order followed by $V(G) - C$ in any order. The number of such permutations is $|C|! \times |V(G) - C|!$. Now we prove for any such permutation, Algorithm 5 can generate C. Since C is minimal, in the first $|C|$ loops of lines 2–3 of Algorithm 5, the conditions in line 3 are all satisfied, and in the last $|V(G) - C|$ loops of lines 2–3, the conditions in line 3 are all unsatisfied because C is already a vertex cover of G. So after the loop in lines 2-3, C is generated. Since C is already minimal, the loop in lines 4–5 will eliminate no node. Thus, Algorithm 5 can generate C. □

The main refine approach is an iterative algorithm shown in Algorithm 6. We iteratively update the current matching until the matching is not improved in a certain iteration. In each iteration (lines 2–6), we try X times to find a new random minimal vertex cover C (line 4), generate the matching $M^+(C)$ using the method introduced above (line 5), and update the current matching if $M^+(C)$ is a better matching (line 6). Here, X is a constant (≥ 1) in order to avoid selecting a bad cover to terminate the whole process. In our experiments, when $X = 5$ and $X = 10$ over 92 and 99% of the nodes have a chance to be included to refine. We use $X = 5$. Note that in line 3, we choose C to be a vertex cover of either G_1 or G_2 with the same probability to increase the randomness.

Theorem 5 *The time complexity of Algorithm 6 is $O(m \cdot n^3)$, for $m = \min\{|E(G_1)|, |E(G_2)|\}$ and $n = \max\{|V(G_1)|, |V(G_2)|\}$.*

Algorithm 7 refine (G_1, G_2, M)

Require: two graphs G_1 and G_2, and the matching M;
Ensure: a refined matching M;
1: **while** M is updated or it is the first iteration **do**
2: **for** $i = 1$ **to** X **do**
3: $G_F \leftarrow$ random selection between $G_1[P_1]$ and $G_2[P_2]$;
4: $F \leftarrow$ select-random-cover (G_F);
5: compute $M^*(F)$;
6: **if** $score(M^*(F)) > score(M)$ **then** $M \leftarrow M^*(F)$;
7: **return** M;

Proof 7 Algorithm 6 is the main refinement. The while loop in line 1 will repeat for at most m times because the optimal solution can match at most m edges and in each loop, the number of edges for the latest solution will be increased for at least 1. In each loop, the dominant part is finding the maximum weight bipartite matching using the Hungarian algorithm which can be done in $O(n^3)$. Since X is a constant, the total time complexity for Algorithm 6 is $O(m \cdot n^3)$. □

Theorem 5 shows an upper bound of the time complexity for Algorithm 6. In practice, the processing time for the algorithm is much smaller than the upper bound because the initial matching M has already matched a lot of edges. In the case when m and n are large, Algorithm 6 can be very slow. We discuss two approaches to make Algorithm 6 faster, with possible loss of matched edges. The goal is the same as before to match as many edges as possible. The first approach is to stop the iteration when the algorithm converges slowly, that is, no larger than δ new matched edges are found in a certain iteration. In such a way, the m part in the time complexity can be largely reduced. The second approach is to enlarge the size of the vertex cover, for example, adding some nodes with the minimum number of unmatched edges into the current vertex cover. The bipartite matching is only conducted on the nodes that are not in the vertex cover. If the size of the vertex cover increases, the number of nodes used in the bipartite matching decreases, thus the time used for matching nodes decreases. In such a way, the n^3 part in the time complexity can be reduced.

6.4 Randomly refinement including $C - F_1$

In this section, we show that $M^+(C)$ can be further improved. Recall that in our previous approach to compute $M^+(C)$, the nodes in $C - F_1$ of G_1 are excluded to refine. In order to refine the nodes in $C - F_1$, we build a new weighted bipartite graph G_b^* as follows. On one side, G_b^* includes all nodes in $V(G_1) - F_1$, and on the other side, G_b^* includes all nodes in $V(G_2) - F_2$. For any node $v \in V(G_1) - F_1$ and node $u \in V(G_2) - F_2$, there is an edge $(u, v) \in E(G_b^*)$ with weight defined in Eq. (6). Suppose the maximum weighted bipartite matching of G_b^* is M_b^*, the new matching $M^*(F_1)$ is defined as follows.

$$M^*(F_1) = (M \cap (F_1 \times F_2)) \cup M_b^* \tag{8}$$

We now define a matching space $\mathcal{M}^*$. For any matching M' between graphs G_1 and G_2, $M' \in \mathcal{M}^*$ if and only if it satisfies the following two conditions.

(1) $M' \cap (F_1 \times F_2) = M \cap (F_1 \times F_2)$
(2) $F_1 \cup (V(G_1) - P_1 - M'^{-1}[V(G_2) - P_2])$ is a vertex cover of G_1.

Theorem 6 *$\mathcal{M} \subseteq \mathcal{M}^*$ and suppose $M_{\mathcal{M}}^*$ is the optimal solution among all matchings in $\mathcal{M}^*$, we have $score(M^*(F_1)) \geq score(M_{\mathcal{M}}^*) \geq score(M^+(C))$.*

Proof 8 For $\forall\, M' \in \mathcal{M}$, we have $M' \cap (F_1 \times F_2) = M \cap (F_1 \times F_2)$ and $M'[C - F_1] = \emptyset$. The first condition is the same as the first condition of $\mathcal{M}^*$. Since $M'[C - F_1] = \emptyset$, we have $(C - F_1) \cap M'^{-1}[V(G_2) - P_2] = \emptyset$. We also have $(C - F_1) \cup M'^{-1}[V(G_2) - P_2] \subseteq V(G_1) - P_1$, accordingly, $C - F_1 \subseteq V(G_1) - P_1 - M'^{-1}[V(G_2) - P_2]$, and thus $C \subseteq F_1 \cup (V(G_1) - P_1 - M'^{-1}[V(G_2) - P_2])$. Since C is a vertex cover of G_1, $F_1 \cup (V(G_1) - P_1 - M'^{-1}[V(G_2) - P_2])$ is a vertex cover of G_1, hence we have $M' \in \mathcal{M}^*$. Thus $\mathcal{M} \subseteq \mathcal{M}^*$ holds.

We now prove that $score(M^*(F_1)) \geq score(M_{\mathcal{M}}^*)$. Suppose $C^* = F_1 \cup (V(G_1) - P_1 - M_{\mathcal{M}}^{*-1}[V(G_2) - P_2])$, we know C^* is a vertex cover of G_1. Since $M_{\mathcal{M}}^*[V(G_1) - P_1] \subseteq V(G_2) - P_2$, we have:

$$\begin{aligned} &M_{\mathcal{M}}^*[C^* - F_1] \\ &\quad = M_{\mathcal{M}}^*[V(G_1) - P_1 - M_{\mathcal{M}}^{*-1}[V(G_2) - P_2]] = \emptyset \end{aligned}$$

Thus, we have $M_{\mathcal{M}}^* \in \mathcal{M}$ using vertex cover C^*, which implies (see the proof of Theorem 4):

$$score(M_{\mathcal{M}}^*) = score(M \cap (F_1 \times F_2)) + score_b(M_b)$$

where M_b is the maximum weight bipartite matching of G_b generated by C^*. We also have $score(M^*(F_1)) \geq score(M \cap (F_1 \times F_2)) + score_b(M_b^*)$. Since $G_b \subseteq G_b^*$, we have:

$$\begin{aligned} score(M^*(F_1)) &\geq score(M \cap (F_1 \times F_2)) + score_b(M_b^*) \\ &\geq score(M \cap (F_1 \times F_2)) + score_b(M_b) \\ &= score(M_{\mathcal{M}}^*). \end{aligned}$$

We last prove $score(M_{\mathcal{M}}^*) \geq score(M^+(C))$. This can be derived directly from $\mathcal{M} \subseteq \mathcal{M}^*$ since $M_{\mathcal{M}}^*$ is optimal in $\mathcal{M}^*$ and $M^+(C)$ is optimal in $\mathcal{M}$. □

Theorem 6 implies that the new space $\mathcal{M}^*$ is larger than the space $\mathcal{M}$ in refinement excluding $C - F_1$, and the new matching $M^*(F_1)$ is no worse than the optimal matching in $\mathcal{M}^*$. This implies that $score(M^*(F_1)) \geq score(M^+(C))$, where $M^+(C)$ is the optimal matching in $\mathcal{M}$. It is worth noticing that the cover C of G_1 does not participate in the

construction of $M^*(F_1)$ directly. The matching $M^*(F_1)$ can be computed as long as F_1 is generated, and F_1 can be computed easily by the following lemma.

Lemma 3 *Suppose $G_1[P_1]$ is the subgraph of G_1 induced by P_1. If C is a vertex cover of G_1, then $C \cap P_1$ is a vertex cover of $G_1[P_1]$, and if C_{P_1} is a vertex cover of $G_1[P_1]$, then there exists a vertex cover C of G_1 such that $C_{P_1} \subseteq C$.*

Proof 9 We first prove that if C is a vertex cover of G_1, then $C \cap P_1$ is a vertex cover of $G_1[P_1]$. Suppose $C \cap P_1$ is not a vertex cover of $G_1[P_1]$, then there exists an edge $(u, v) \in E(G_1[P_1])$ such that $u \notin C \cap P_1$ and $v \notin C \cap P_1$. Note that C is a vertex cover of G_1, we have $u \in C$ or $v \in C$. Without loss of generality, we suppose $u \in C$. Since $u \notin C \cap P_1$, we have $u \in C - (C \cap P_1)$, which contradicts with $u \in V(G_1[P_1])$. Thus, $C \cap P_1$ is a vertex cover of $G_1[P_1]$.

We then prove that if C_{P_1} is a vertex cover of $G_1[P_1]$, then there exists a vertex cover C of G_1 such that $C_{P_1} \subseteq C$. We only need to prove that $C = C_{P_1} \cup (V(G_1) - P_1)$ is a vertex cover of G_1. For any $(u, v) \in E(G_1)$, if $u \in P_1$ and $v \in P_1$, (u, v) is covered by C because C_{P_1} is a vertex cover of $G_1[P_1]$. Otherwise, without loss of generality, we suppose $u \notin P_1$, then $u \in V(G_1) - P_1 \subseteq C$, so (u, v) is also covered by C. As a result, all edged in $E(G_1)$ can be covered by C, thus C is a vertex cover of G_1. □

Based on Lemma 3, we can derive that the vertex cover of $G(P_1)$, F_1, is enough to generate $M^*(F_1)$. Our new refinement algorithm is shown in Algorithm 7 which is the refine used in Algorithm 1. We use $X = 5$. Comparing to Algorithm 6, there are two major modifications. The first is about the cover computing in lines 3–4, instead of computing the cover of G_1 (or G_2 if we select G_2 as the first graph in line 3), we only compute the vertex cover of $G_1[P_1]$ (or $G_2[P_2]$). For the second modification, instead of computing $M^+(C)$, we compute our new matching $M^*(F)$.

7 Labeled graph handling

In previous sections, we concentrate on unlabeled graphs. In this section, we discuss how to handle node-labeled graphs. Given a set of node-labels Σ_V, a node-labeled graph is denoted by $G = (V, E, l)$. Here, l is a labeling function: $V \rightarrow \Sigma_V$. We use $l(u)$ to denote the label of u for every node $u \in V(G)$. The definitions of graph isomorphism and graph matching are given as follows.

Definition 4 GRAPH/SUBGRAPH ISOMORPHISM (LABELED GRAPH). Graph $G_1(V, E, l_1)$ is isomorphic to graph $G_2(V, E, l_2)$, if and only if there exists a bijective function $f : V(G_1) \rightarrow V(G_2)$ such that for any two nodes $u_1 \in V(G_1)$ and $u_2 \in V(G_1)$, $(u_1, u_2) \in E(G_1)$ if and only if $l_1(u_1) = l_2(f(u_1))$, $l_1(u_2) = l_2(f(u_2))$, and $(f(u_1), f(u_2)) \in E(G_2)$. G_1 is subgraph isomorphic to G_2, if and only if there exists a subgraph G' of G_2 such that G_1 is isomorphic to G'.

Definition 5 GRAPH MATCHING (LABELED GRAPH). Given two graphs $G_1(V, E, l_1)$ and $G_2(V, E, l_2)$, a matching M between G_1 and G_2 is a set of pairs $M = \{(u, v) | u \in V(G_1), v \in V(G_2), l_1(u) = l_2(v)\}$, such that for any two pairs $(u_1, v_1) \in M$ and $(u_2, v_2) \in M$, $u_1 \neq u_2$ and $v_1 \neq v_2$. The optimal matching M of two graphs is the one with the largest number of matched edges. Finding the optimal matching M is the same as *MCS*.

For handling node-labeled graphs, the matching construction and matching refinement need to be modified.

In the matching construction step, recall that we need to compute node similarity S [Eq. (2)], which equals to the multiplication of global node similarity S_g [Eq. (3)] and local node similarity S_l [Eq. (4)]. Since it is difficult to incorporate node-label information into the spectral-based similarity when computing the global node S_g, we incorporate node-label information into the local node similarity S_l. In doing so, let G_u^k be a k-neighborhood subgraph for a node u in G, we handle node-labels for a node $u' \in G_u^k$ using $L(u')$, which is a multi-set consisting of all labels in the adjacent nodes of u' in G_u^k. This is because the same label may appear multiple times in the adjacent nodes of u'. Given two graphs, G_1 and G_2, suppose u in G_1 and v in G_2, and let the corresponding k-neighborhood subgraphs be G_u^k and G_v^k. We construct a bipartite graph, B_{uv}, with $|N_k(u)|$ nodes in G_u^k on one side and $|N_k(v)|$ nodes in G_v^k on the other. There is an edge in B_{uv}, between two nodes $u' \in N_k(u)$ and $v' \in N_k(v)$, iff the node-labels of the two nodes are the same ($l_1(u') = l_2(v')$). Such an edge (u', v') is associated with an edge weight $w(u', v') = |L(u') \cap L(v')|$. Based on the bipartite graph B_{uv} constructed, we compute the maximum weighted bipartite matching M_{uv} over B_{uv} using the Hungarian algorithm. $D(u, v)$ [Eq. (5)] used in S_l [Eq. (4)] for unlabeled graphs is modified as follows.

$$D(u, v) = \frac{|L(u) \cap L(v)|}{2} + \frac{\sum_{(u', v') \in M_{uv}} w(u', v')}{2} \quad (9)$$

It is important to note that Eq. (9) is computed on the condition that the node-labels of u and v are the same ($l_1(u) = l_2(v)$). In other words, $S_l[u, v]$ [Eq. (4)] can be only computed using Eq. (9) for labeled graphs, when the node-labels of u and v are the same. Otherwise, we define $S_l[u, v] = 0$ for labeled graphs.

All the four properties that hold for unlabeled graphs hold for labeled graphs.

(1) $0 \leq S_l[u, v] \leq 1$.

(2) $S_l[u, v] \geq \frac{(|V(\mathsf{mcs}(G_u^k, G_v^k))| + |E(\mathsf{mcs}(G_u^k, G_v^k))|)^2}{(|V(G_u^k)| + |E(G_u^k)|)(|V(G_v^k)| + |E(G_v^k)|)}$ if the node-labels of u and v are the same.

(3) If G_u^k and G_v^k are isomorphic, and u is matched to v in the optimal matching of G_u^k and G_v^k, then $S_l[u, v] = 1$.

(4) If G_u^k is subgraph isomorphic to G_v^k, and u matches v in the optimal matching of G_u^k and G_v^k, we have $S_l[u, v] = \frac{|V(G_u^k)| + |E(G_u^k)|}{|V(G_v^k)| + |E(G_v^k)|}$.

In addition, Eq. (5) is a special case of Eq. (9) when all nodes have the same label (equally unlabeled). This can be explained as follows. When all the nodes in the two graphs have the same label, the intersection of the adjacency label sets will be the minimum degree, $|L(u) \cap L(v)| = \min\{d(u), d(v)\}$, and therefore $w(u', v')$ in Eq. (9) is the same as $\min\{d(u'), d(v')\}$. As a result, in such a case, the value of $D(u, v)$ computed using Eq. (9) is the same of $D(u, v)$ computed using Eq. (5).

In the matching refinement step, the only change we need to make for labeled graphs is the computing of the weight $w(u, v)$ [Eq. (6)] for each edge (u, v) in the constructed bipartite graph G_b. In handling labeled graphs, for each $(u, v) \in E(G_b)$, when the node-labels of u and v are the same, since $N(u) \cap F_1$ in G_1 and $N(v) \cap F_2$ in G_2 have been fixed, $M[N(u) \cap F_1] \cap (N(v) \cap F_2)$ is also fixed and should not be changed. When the node-labels of u and v are different, Eq. (6) cannot be used to compute $w(u, v)$, because u does not match v, and we simply set $w(u, v) = 0$. Equation (6) is modified as follows for handling labeled graphs.

$$w(u, v) = \begin{cases} |M[N(u) \cap F_1] \cap (N(v) \cap F_2)|, & \text{if } l_1(u) = l_2(v) \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

All other equations, theorems, and algorithms for unlabeled graphs can be directly applied to labeled graphs.

8 Performance studies

We compare our algorithms, cons (matching construction only) and consR (matching construction plus matching refinement), with five state of the art graph matching algorithms: ume, heat, iso, path, and GA. Here, ume is the improved Umeyama algorithm in [17,32], and heat, iso, path, and GA are the algorithms proposed in [28,34,36], and [37], respectively. We implement ume, heat and our algorithm using Visual C++ 2005 and Matlab R2009a. The C++ part calls Matlab to compute the eigenvalues and eigenvectors of the matrix, and executes the rest of the algorithm in C++. For iso, path, and GA, we download the source code of the graph matching package GraphM.[2] All tests were conducted on a 2.66 GHz CPU and 3.43 GB memory PC running Windows XP.

Table 1 The PN dataset

Graph	g1	g2	g3	g4	g5
$\|V\|$	118	443	1,454	1,723	5,300
$\|E\|$	179	590	1,923	2,394	6,094

We evaluated the algorithms using both real and synthetic datasets. The real datasets include the Power Network and the NCI dataset. The synthetic datasets are generated using two models, namely Scale-Free/Power Law Model and Erdos Renyi Model. We use software Pajek[3] to generate graphs under these two models. We mainly focus on testing unlabeled graphs, and discuss the results for labeling graphs in Sect. 8.5.

Power Network (PN) is the electrical power grid of the western US selected from the University of Florida Sparse Matrix Collection.[4] The nodes are generators, transformers, and substations, and the edges are the high-voltage transmission lines between them. The graphs are proved to be power law networks [5,33]. The dataset contains graphs with number of nodes varying from 39 to 5,300. The information of graphs used in our testing is shown in Table 1.

NCI dataset (NCI) contains the compound structures from the National Cancer Institute Open Database.[5] The NCI dataset contains 233,281 connected graphs. The average node number is 21.17 and the average node degree is 2.2. According to the number of nodes in a graph, we selected 5 groups, containing graphs with node numbers 10 ± 4, 15 ± 4, 20 ± 4, 25 ± 4, and 30 ± 4, respectively.

Scale-Free/Power Law Model (SF) is a network model whose node degrees follow the power law distribution or at least asymptotically. We generate graphs with node numbers 100, 500, 1,000, 2,500, and 5,000, respectively, with default value 1,000. The average node degree is 4.

Erdos Renyi Model (ER) is a classical random graph model. It defines a random graph as N nodes connected by M edges that are chosen randomly from the $N(N-1)/2$ possible edges. We generate graphs with node numbers 100, 500, 1,000, 2,500, and 5,000, respectively, with default value 1,000. The average node degree is 4.

For NCI, since each graph is small, we can compute the optimal matching for each pair of graphs using backtracking method. For graphs in other datasets, it is impossible to compute the optimal matching. Therefore, for any graph G_1, we generate G_2 by randomly inserting/deleting a certain percent of nodes/edges followed by shuffling all nodes.

[2] http://cbio.ensmp.fr/graphm/.

[3] http://vlado.fmf.uni-lj.si/pub/networks/pajek/.

[4] http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/bcspwr/bcspwr.html.

[5] http://cactus.nci.nih.gov/download/nci/.

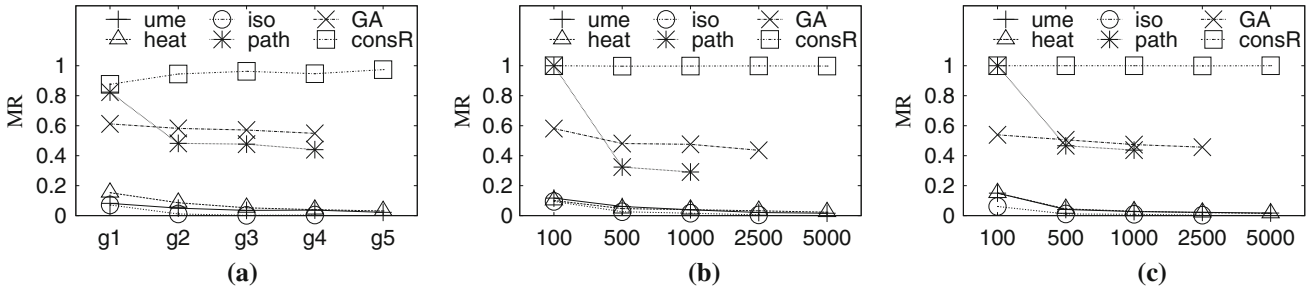


Fig. 6 Vary graph size. **a** PN, **b** SF, **c** ER

Table 2 Parameters

Parameter	Range	Default
#nodes	100, 500, 1,000, 2,500, 5,000	1,000
Average degree	2, 3, 4, 5, 6	4
p	0.1, 0.15, 0.2, 0.25, 0.3	0.2
τ	0.82, 0.86, 0.90, 0.94, 0.98	–
k	0, 1, 2, 3, 4	2
#labels	0, 2, 4, 8, 16, 32	–

We set the parameters and the default values used in our algorithms in Table 2, where τ is the threshold used in anchor-selection (Algorithm 2). Since τ affects the quality of matching, as default, we use the approach discussed in Sect. 5.3 to determine such a τ where $l = 0.02$ (used in Algorithm 4). We report the total time including τ selection in our experimental studies (Algorithm 1 with replacement of lines (1-2) with construct-opt(G_1, G_2)). We will also report the sensitivity of τ, and in such a case, we use Algorithm 1 as it is. For k in the k-neighborhood definition, we vary it from 0 to 4 and find $k = 2$ achieves the best result in most cases. Thus, we set k to 2 as default in the experiment.

We use matching ratio as our measure for the matching quality, which is computed as follows:

$$MR = \frac{actual\ matched\ edges}{optimal\ matched\ edges}$$

8.1 Comparison with the approximate algorithms

Vary Graph Size: We vary the number of nodes in the graphs from 100 to 5,000 and test the matching ratio of each algorithm on PN, SF, and ER. Their results are shown in Fig. 6a–c, respectively. For all cases, the matching ratios for ume, heat and iso algorithms are no larger than 0.2. It is because all the three algorithms get the matching by maximizing the total weight of the similarity matrix and consider little about their neighborhood information of two nodes. When the numbers of nodes are no large than 120, the performance of path is similar to consR, and reaches a matching ratio above 0.9. When the sizes of the graphs increase, the matching ratio of path decrease. GA performs better than path for large graphs, but still much worse than consR in all cases. When the sizes of the graphs are above 2,500, iso, path, and GA cannot generate a result for some test cases under our current computing environment. path even cannot generate a result for graphs with more than 2,000 nodes.

Vary p: We vary p from 0.10 to 0.3, and the results for PN, SF, and ER datasets are shown in Fig. 7a–c, respectively. For all cases, the matching ratios for ume, heat, and iso algorithms are still no larger than 0.2 for the same reason as analyzed in Fig. 6. The matching ratios for path and GA are constant when p changes. consR performs best in all cases.

Vary Initial Matching: We compare ume, heat, iso, path, and GA with our matching construction algorithm cons. Figure 8a shows the results for PN. In most cases, cons performs best among all algorithms. The only exception is the case when the numbers of nodes in graphs are smaller than 120, where path performs better than cons. Figure 8d shows the results by applying our matching refinement algorithm to these five algorithms. Accordingly, they are denoted as umeR, heatR, isoR, pathR, and GAR. For ume, heat, and iso, the matching ratios increase 0.5 after refinement in all cases. Our consR algorithm performs best in all cases after refinement even for the case when the numbers of nodes in the graphs are smaller than 120. Figure 8b, e show the testing results for SF. Our algorithms perform best both before and after refinement in all cases. The results for ER are shown in Fig. 8c, f. The performances for all test cases in ER are similar to those in SF.

Vary Initial Matching on Small Graphs: Since path outperforms cons when graph size is around 100 but is beaten by cons when graph size is 500 in Fig. 8, we evaluate two sets of graphs, by varying node numbers in two separate ranges, (0, 100) and (100, 500). The results are shown in Fig. 9. Figure 9a, b show the result for SF and ER with the graph size in (0, 100). path has a better performance than cons in most cases. Figure 9c, d show the matching ratio for SF and ER with the graph size in (100, 500). path only has a better or similar performance than cons when the graph size is less than 200, and it is outperformed by cons when the graph size is larger than 200. Figure 9e–h show the match-

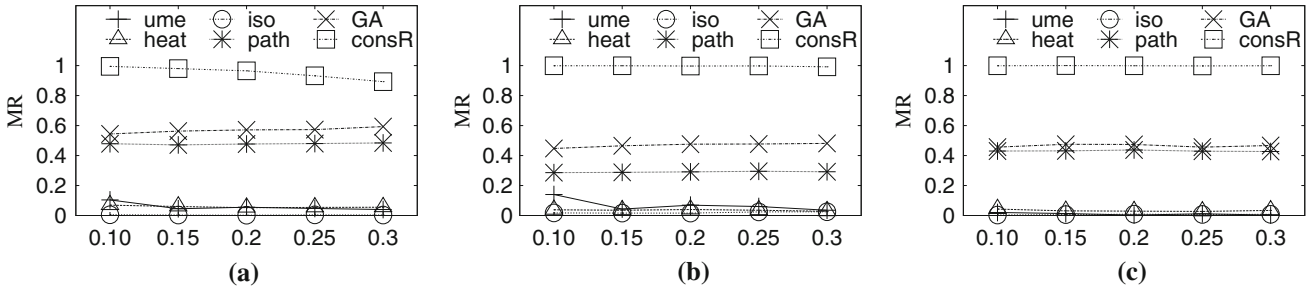


Fig. 7 Vary p. **a** PN, **b** SF, **c** ER

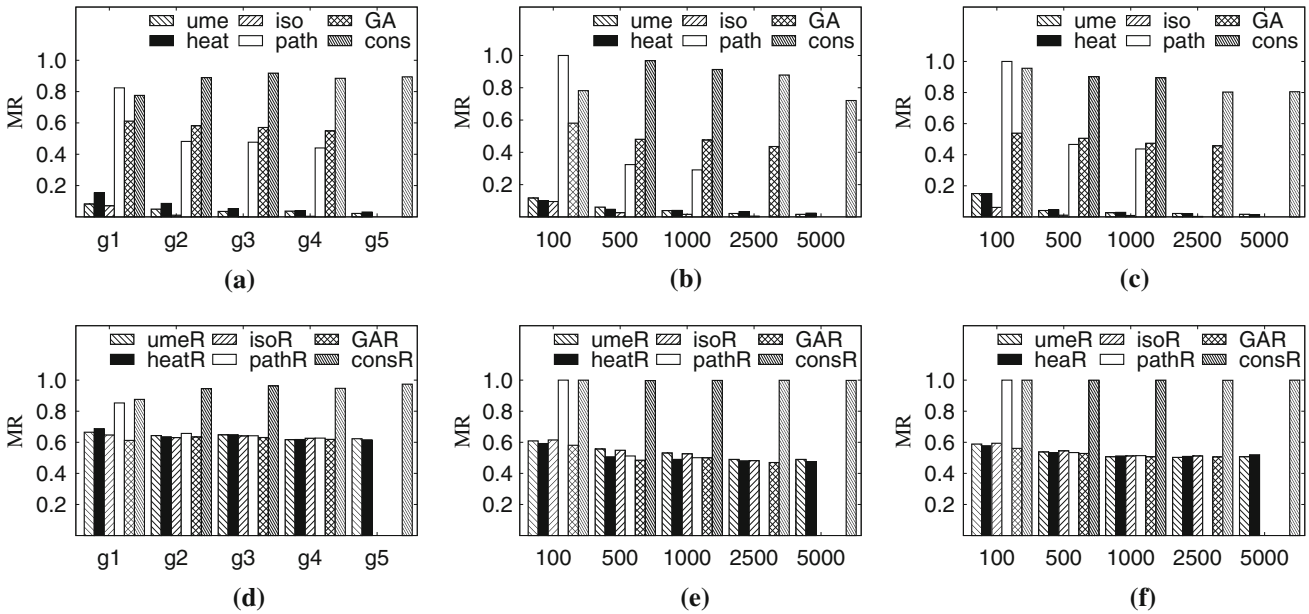


Fig. 8 Vary initial matching. **a** PN (construction), **b** SF (construction), **c** ER (construction), **d** PN (refinement), **e** SF (refinement), **f** ER (refinement)

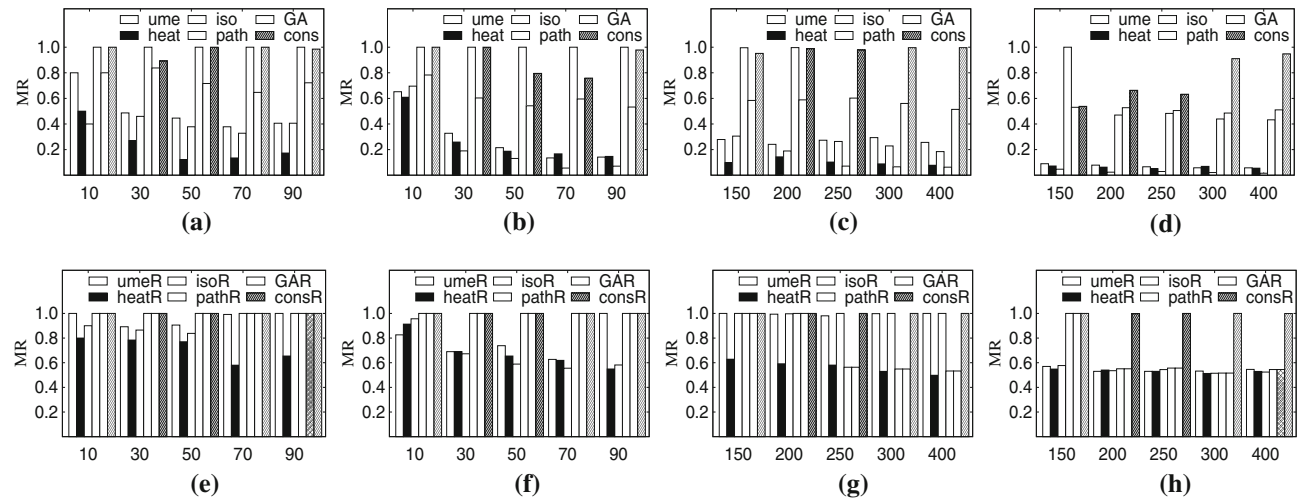


Fig. 9 Vary initial matching on small graphs. **a** SF ($n < 100$, construction), **b** ER ($n < 100$, construction), **c** SF ($n < 500$, construction), **d** ER ($n < 500$, construction), **e** SF ($n < 100$, refinement), **f** ER ($n < 100$, refinement), **g** SF ($n < 500$, refinement), **h** ER ($n < 500$, refinement)

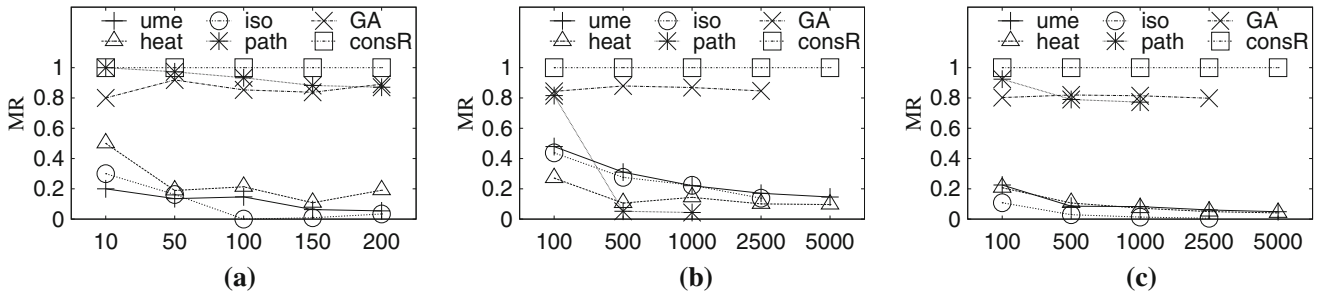


Fig. 10 Comparison on random pairs. **a** Vary graph size (NCI), **b** Vary graph size (SF), **c** Vary graph size (ER)

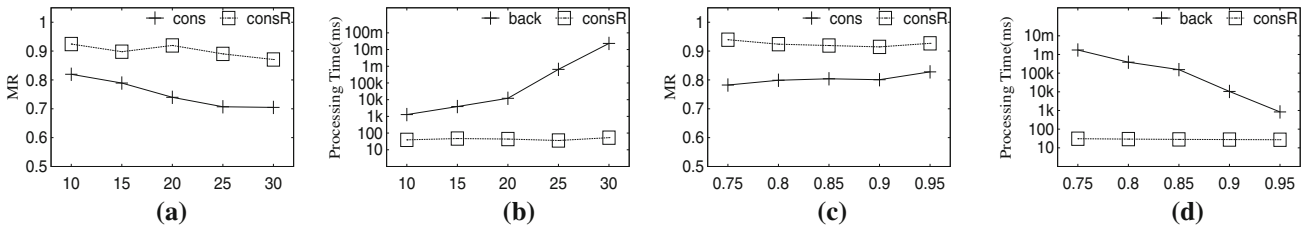


Fig. 11 Test the NCI dataset. **a** Vary # nodes (NCI), **b** Vary # nodes (NCI), **c** Vary optional solution (NCI), **d** Vary optional solution (NCI)

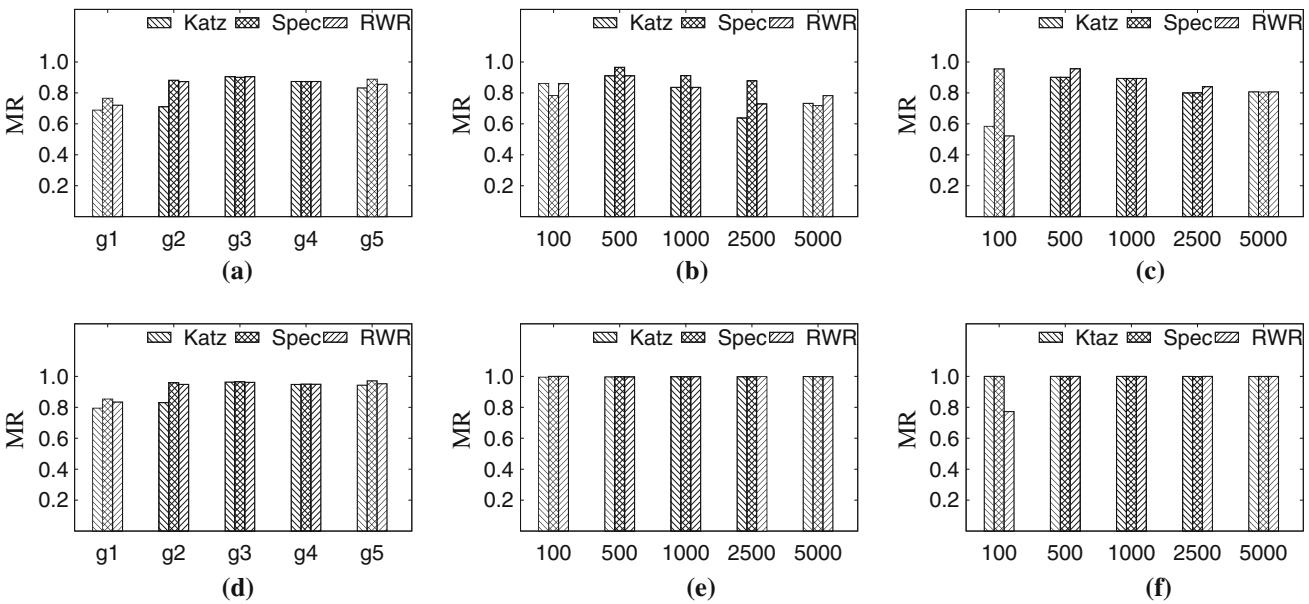


Fig. 12 Vary global similarity measures. **a** PN (construction), **b** SF (construction), **c** ER (construction), **d** PN (refinement), **e** SF (refinement), **f** ER (refinement)

ing ratio after refinement. consR performs the best for all cases. Even though, for the graphs with the size less than 200, path slightly outperforms cons, our refinement algorithm consR outperforms pathR. In addition, path is quite time-consuming as shown in Table 3. We suggest using consR rather than pathR even for small graphs with size less than 200 considering both effectiveness and efficiency.

Comparison on Random Pairs: We compare ume, heat, iso, path, and GA with our algorithm consR on the datasets where every two graphs are randomly selected or generated. For two randomly selected graphs, since we do not know the optimal matched edges, the matching ratio MR for each algorithm is the relative ratio of its number of matched edges to the best matched edges among all these six algorithms. For

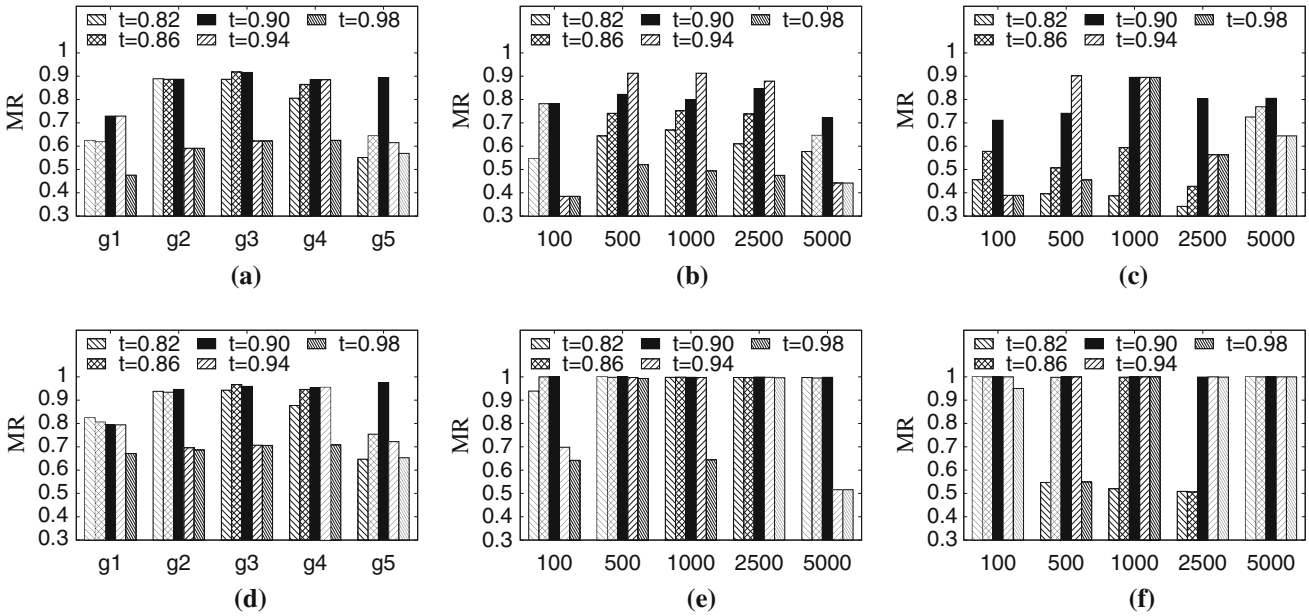


Fig. 13 Vary τ. **a** PN (construction), **b** SF (construction), **c** ER (construction), **d** PN (refinement), **e** SF (refinement), **f** ER (refinement)

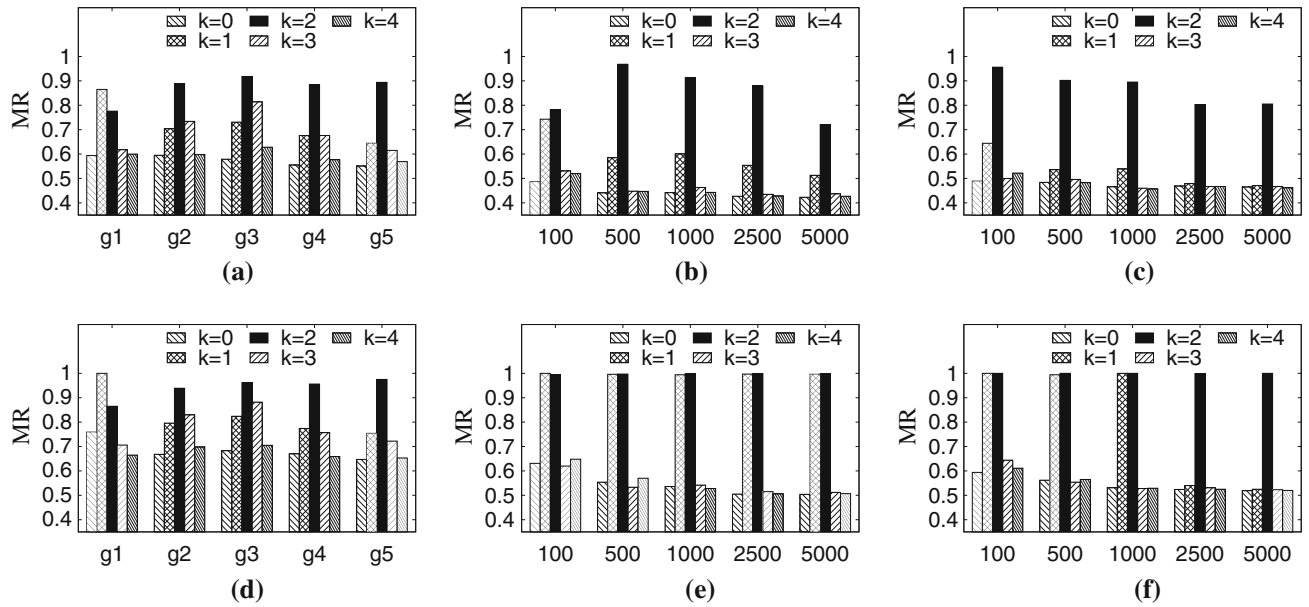


Fig. 14 Vary k. **a** PN (construction), **b** SF (construction), **c** ER (construction), **d** PN (refinement), **e** SF (refinement), **f** ER (refinement)

real dataset NCI, we randomly select 5 groups, containing graphs with node numbers 10 ± 4, 50 ± 4, 100 ± 4, 150 ± 4, and 200 ± 4, respectively. Figure 10a shows the matching ratio for NCI dataset. path performs better than GA which is similar as shown in Fig. 9. consR performs best among all algorithms for all cases. We also generate 5 groups of two random graphs according to SF (ER) model, with the number of nodes ranging from 100 to 5,000. Figure 10b shows the matching ratio for SF dataset. consR performs best among all the algorithms for all cases. path and GA have similar performance when the number of nodes is 100. However path performs badly when the graph size becomes larger. The underlying reason is that path can be easily trapped in a small local maximum in solving the linear combination of the convex relaxation and concave relaxation, since the structures of two randomly generated graphs according to SF

Table 3 Processing time comparison (PN)

Graph	g1	g2	g3	g4	g5
selection (s)	0.05	0.45	20.41	33.11	1,221.58
expansion (s)	0.02	0.08	1.11	0.97	32.92
refinement (s)	0.05	0.72	6.81	10.45	205.05
consR (s)	0.11	1.25	28.33	44.53	1,459.54
ume (s)	0.05	0.48	20.63	33.41	1,561.58
heat (s)	0.11	2.06	72.79	116.47	4,117.50
iso (s)	0.12	2	185	57	–
path (s)	14	411	11,700	22,200	–
GA (s)	1	8	150	253	–

model might be quite different and unbalanced. Figure 10c shows the matching ratio for ER model. path and GA have relatively stable performance and consR performs the best among all the algorithms.

Efficiency Testing: We test the efficiency of the algorithms with PN. The processing time is shown in Table 3. We divide the whole processing time of our algorithm into three phases, namely selection, expansion, and refinement, denoting the processing time for anchor selection, anchor expansion, and matching refinement, respectively. The processing time for consR is the sum of the time for all the three phases. For the three phases, in all cases, the most costly part is anchor selection, because it involves calculating the eigenvalues for matrices. In all test cases, our consR algorithm is faster than iso, heat, path, and GA, and is similar to ume. For the largest graph g5, the total processing time for consR is smaller than 25 min, while heat needs more than 1 h and iso, path, and GA even cannot generate a result under our current computing environment.

8.2 Comparison with the exact algorithm

With NCI, we randomly select 10 pairs of graphs from each group of graphs with different node numbers. For each pair, we compute the optimal matching by backtracking using the best upper bound introduced in [26]. For each group, we record the average processing time for backtracking and our algorithm as well as the average matching ratio in the two steps of our algorithm. The results are shown in Fig. 11a, b. When node number increases from 10 to 30, the processing time of backtracking increases from 1 to 100,000 s and consR algorithm consumes $\leq$0.1 s in all cases. The average accuracy (matching ratio) for cons is 0.8 and the average accuracy for consR is 0.95.

We randomly select 1,000 pairs from the group of graphs with 20 ± 4 nodes. For each pair, we compute the ratio of the optimal matched edges to the size of the smaller graph, and vary the ratio from 0.75 to 0.95. For each ratio, we compute the average processing time and average accuracy of our algorithm. The results are shown in Fig. 11c, d. When the ratio increases from 0.75 to 0.95, the processing time of the backtracking algorithm decreases from 10,000 to 1 s. This is because when the ratio is large, the upper bound used to cut branches in the backtracking is tight [26], thus the algorithm stops early. Our consR algorithm consumes no more than 0.1 s in all cases. The average accuracy (matching ratio) for our cons algorithm is 0.8 and the average accuracy for our consR algorithm is 0.95.

8.3 Parameter and scalability testing

Vary Global Similarity Measures: We compare the matching results of our algorithm cons and consR using three different global similarity measures: spectral similarity, Katz score, and RWR score, denoted as Spec, Katz and RWR, respectively. We set the attenuation factor b used in Katz $1/(d+1)$ by default [14], where d is the maximum degree of the graph. For the parameter c in RWR, we use the same setting as [30]. Figure 12 shows the performances of these three global similarity measures. The matching ratio of construction is shown in Fig. 12a–c. Katz is outperformed by Spec and RWR in most cases. Spec performs best in most cases. RWR performs better in a few cases. However, the differences among the three are marginal. The underlying reason is that both Katz score and RWR score originate from the idea of random walks where the stationary distribution converges to the largest eigenvector. Our refinement algorithm can refine, using any of them, to a better result as shown in Fig. 12d–f.

Vary τ: We first test the sensitivity of τ, and show the representative τ values from 0.82 to 0.98 which is the similarity threshold used in anchor selection (Algorithm 2). We list MR for the two steps: construction and refinement for the 5 graphs. As shown in Fig. 13a, with the PN dataset, for all the 5 graphs, when increasing τ, the matching ratio increases to a peak value followed by decreasing. This is because when τ is small, the number of anchors is large, some mismatched anchors are thus involved. When τ is large, the number of anchors is too small, thus few nodes are referenced when expanding. The peak value of MR varies from 0.7 to 0.9. For different graphs, the τ values that generate the peak value are different. In Fig. 13d, when increasing τ, the MR value also increases followed by decreasing. Comparing to Fig. 13a, after refinement, the matching ratio increases by 10 % on average, and the average MR is above 0.95 when fixing τ to be the default value 0.9. The results for the SF dataset when varying τ from 0.82 to 0.98 in the 5 cases are shown in Fig. 13b, e. For the matching construction step, the matching ratio increases followed by decreasing, and the peak value is 0.8 on average. After refinement, most matching ratios

increase to 1, except for several cases when τ is too large. This is because when τ is large, few anchors are selected, thus in the expansion step, a lot of nodes are mismatched. The large number of mismatched nodes can hardly be refined perfectly. Figure 13c, f show the situations in the ER dataset, the performances in the construction step are similar to the SF dataset. In the refinement step, when τ is small, the matching can hardly be refined perfectly. This is because when τ is small, a lot of anchors are selected, and in the ER dataset, all degrees have a uniform distribution. Thus, the number of mismatched anchors is large, which induces the bad performances of the refinement step.

Vary k: We vary k which is the k-neighborhood used for local similarity computation. Figure 14a shows the results for the construction step, with the PN dataset, when varying k from 0 to 4. The performance is best when k is 1 or 2. This is because when k is small, very little local information is involved in the local similarity, and when k is large, too much noise is added to the local similarity. Figure 14d shows that, after refinement, the curves for all graphs are similar as those in the construction step, but the MR values increase by 10 % on average. When k is the default value 2, the average matching ratio can reach 0.95 in most cases. Figure 14b, e show the results of the two steps on the SF dataset when varying k. In all cases, $k = 2$ always leads to the best matching for the construction step. In the refinement step, the matching ratios for $k = 1$ in all cases largely increase, although their performances in the first step are not so good. This is because when $k = 1$, the anchor selection can select good pairs of anchors, and the errors induced by the anchor expansion are easier to be repaired in the refinement step. The results on the ER dataset when varying k are shown in Fig. 14c, f. For the construction step, the performances are similar to that on the SF dataset. For the refinement step, when the number of nodes is large, the errors induced in the construction step when $k = 1$ can hardly be repaired. This is because in the ER dataset, the degrees for all nodes are uniformly distributed. When $k = 1$, even the anchor selection cannot select good anchor pairs. As a result, it is hard for the refinement step to generate a good matching.

Vary p and degree: We compare our algorithms cons and consR by varying p from 0.1 to 0.3. The results on PN are shown in Fig. 15a. When p increases, the matching ratio for both cons and consR decreases. This is because when p increases, the similarity of the graphs to be matched decreases. consR is 10 % better than cons on average. The average matching ratio for consR in all cases is above 0.95. The results for SF and ER when varying p are similar to PN. Figure 15d shows the testing results on SF when varying the average degree.[6] It shows that when the average degree increases, the matching ratio for cons decreases. This is because when the average degree is large, the anchor expansion algorithm keeps a lot of pairs in the queue in early stages, and thus increases the probability for nodes to be mismatched. consR is consistent and increases the matching ratio to 1 in most cases. The performance on ER is similar to the performance on SF when varying average degree. The results when varying p in the SF and ER datasets are shown in Fig. 15b, c, respectively. When p increases, the matching ratios of cons decrease in both datasets. The matching ratios with SF decrease slower, because with SF the degrees of nodes show a power law distribution. Although the similarity of the two graphs decreases, the anchors also have a large chance to be matched correctly. Our consR algorithm can correct the errors caused in the initial matching perfectly, and increase the average matching ratio to above 0.98 in both datasets.

[6] We cannot vary degrees for real datasets like PN.

8.4 Sensitivity of randomness (PN)

In the refinement step, each vertex cover is randomly selected. In order to test the sensitivity of such randomness, for each test case, we run our program for 100 times, and calculate their average match ratio (MR) as well as the standard deviation of the 100 match ratios. The average match ratio (MR) and the standard deviation for the PN dataset are shown in Table 4. For each test case, the standard deviation is very small. This means that no matter how the vertex covers are randomly selected, the program will generate almost the same result. There are two reasons. First, the nodes that largely influence the stability of results are those with large degrees, and with high probability such nodes will be selected in a vertex cover. Second, our refinement algorithm is an iterative algorithm and it stops when converged. In each iteration, every vertex cover has chances to be refined. The results for SF and ER datasets are similar with those in the PN dataset.

8.5 Effectiveness of label distribution

In this experiment, we compare our algorithms, cons and consR, with five other algorithms: ume, heat, iso, path, and GA on labeled graphs. Since ume and heat are algorithms originally designed for unlabeled graphs, we modify them to handle labeled graphs by setting the similarity score of two nodes with different labels to 0 before applying Hungarian algorithm on the similarity matrix to obtain the maximum weight bipartite matching. For iso, path, and GA, we use the setting for the case of constrained graph matching [37] in which only nodes with the same label can be matched with each other. In other words, we set the node similarity to 1 for two nodes $u \in G_1$ and $v \in G_2$ with the same label and 0 otherwise, and set the objective as only maximizing the number of matched edges with the label constraint.

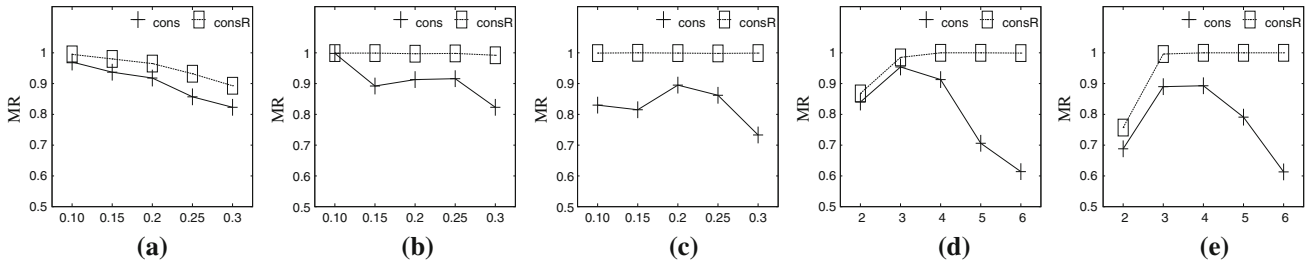


Fig. 15 Vary p and $degree$. **a** Vary p (PN), **b** vary p (SF), **c** vary p (ER), **d** vary $degree$ (SF), **e** vary $degree$ (ER)

Table 4 Sensitivity of randomness

Graph	g1	g2	g3	g4	g5
Average MR	0.860	0.950	0.961	0.949	0.975
SD	0.008	0.008	0.002	0.003	0.002

We evaluate these algorithms on both real dataset PN (Power Law model) and synthetic dataset ER (Erdos Renyi model). In PN dataset, we use graph g3 as default, and in ER dataset, we use the graph with 1,000 nodes as default. We evaluate two different node-label distributions: uniform distribution and power law distribution. For the former, we assign the labels to nodes in the graph uniformly, and for the latter, we assign the labels to nodes in the graph according to a power law probability distribution $p(x) = C \cdot x^{-\alpha}$. Since for most of real networks, $2 < \alpha < 3$ [23], we set $\alpha = 2.5$ in our experiment. All other parameters are set to the default values listed in Table 2.

We vary the number of labels from 0 to 32, and show the matching results in Fig. 16. Figure 16a shows the matching result on PN dataset with uniform label distribution. For all cases, the matching ratios for ume, heat, and iso algorithms are no larger than 0.2. As analyzed for the unlabeled graphs, the reason is that all the three algorithms get the matching by maximizing the total weight of the similarity matrix and consider little about their neighborhood information of two nodes. When the number of labels increases, the matching ratios of path and GA decrease slightly, but then they reach a good result when the number of the labels is 32. This is because these two algorithms get the matching result by starting from an initial solution and linearly interpolate between the convex and concave relaxations gradually to find a local optimum of the problem. Therefore, when the number of labels increases, 0 values in the node similarity matrix will increase, which leads to the reduction in the feasible solution space as well as the local optimum solution space. However, when the number of labels is very large, the feasible solution space is limited to a very small space, which make finding a good initial solution and good local optimum solution or even global optimum solution easier. The matching ratios of cons and consR increase as the number of labels increases, because more the number of labels is, higher distinctive ability the labels will have. This means that, with the help of node-labels, we will not mismatch the nodes that have similar neighborhood structures but different node-labels, and we can reduce the number of mismatched edges. Overall, our algorithm performs best for all testing cases. Figure 16b shows the matching result on ER dataset with uniform label distribution, which share the similar curves with Fig. 16a. Figure 16c, d show the matching ratios of PN and ER with power law label distribution, respectively. Both of them share similar pattern with Fig. 16a, b. The main difference is that matching ratios of path and GA at 32 are not as large as that in Fig. 16a, b. This is because the feasible solution space is not reduced noticeably, since, for power law distribution, no matter how large the number of labels is, there are only a few labels appearing frequently, which make the labels less distinctive.

9 Conclusions

In this paper, we study how to find a matching of two large graphs or how to score how similar two large graphs can

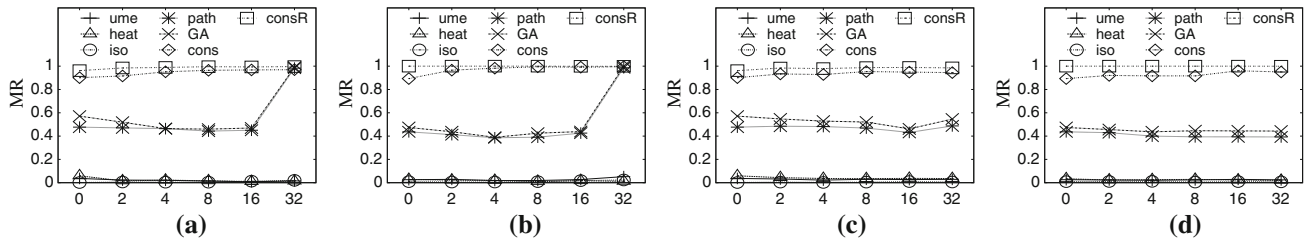


Fig. 16 Vary number of distinct labels. **a** Uniform labels (PN), **b** Uniform labels (ER), **c** Power law labels (PN), **d** Power law labels (ER)

be in terms of the possibly maximum number of matched edges. This is known to be NP-hard. We give a new two-step approach which ensures high efficiency and high quality. Our solution can be applied to both unlabeled and labeled graphs. We conducted extensive testing using real and synthetic datasets, and confirmed the quality and efficiency of our approach.

Acknowledgments The work was supported by the Research Grants Council of the Hong Kong SAR, China (419109), ARC Discovery Grants (ARCDP0987557, ARCDP110102937, ARCDP120104168), and NSFC61021004.

References

1. Abu-Khzam, F.N., Samatova, N.F., Rizk, M.A., Langston, M.A.: The maximum common subgraph problem: faster solutions via vertex cover. In: AICCSA, pp. 367–373 (2007)
2. Almohamad, H.A., Duffuaa, S.O.: A linear programming approach for the weighted graph matching problem. IEEE Trans. Pattern Anal. Mach. Intell. **15**(5), 522–525 (1993)
3. Arora, S., Safra, S.: Approximating clique is np-complete. In: Proceedings of the 33rd IEEE Symposium on Foundations on Computer Science, pp. 2–13 (1992)
4. Bai, X., Yu, H., Hancock, E.: Graph matching using spectral embedding and alignment. In: Proceedings of International Conference on Pattern Recognition, pp. 398–401 (2004)
5. Barabási, A., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509 (1999)
6. Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput. **15**(6), 1373–1396 (2003)
7. Bernard, M., Richard, N., Paquereau, J.: Functional brain imaging by eeg graph-matching. In: 27th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC'05), pp. 5309–5312 (2005)
8. Blondel, V., Gajardo, A., Heymans, M., Senellart, P., Van Dooren, P.: A measure of similarity between graph vertices: Applications to synonym extraction and web searching. Siam Rev. **46**(4), 647–666 (2004)
9. Bonchi, F., Esfandiar, P., Gleich, D.F., Greif, C., Lakshmanan, L.V.S.: Fast matrix computations for pair-wise and column-wise commute times and katz scores. CoRR abs/1104.3791 (2011)
10. Caelli, T., Kosinov, S.: An eigenspace projection clustering method for inexact graph matching. IEEE Trans. Pattern Anal. Mach. Intell. **26**(4), 515–519 (2004)
11. Caelli, T., Kosinov, S.: Inexact graph matching using eigen-subspace projection clustering. Int. J. Pattern Recognit. Artif. Intell. **18**(3), 329–354 (2004)
12. Chevalier, F., Domenger, J.P., Benois-Pineau, J., Delest, M.: Retrieval of objects in video by similarity based on graph matching. Pattern Recogn. Lett. **28**(8), 939–949 (2007)
13. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. IJPRAI **18**(3), 265–298 (2004)
14. Foster, K.C., Muth, S.Q., Potterat, J.J., Rothenberg, R.B.: A faster katz status score algorithm. Comput. Math. Organ. Theory **7**(4), 275–285 (2001)
15. Jouili, S., Tabbone, S.: Graph matching based on node signatures. In: GbRPR, pp. 154–163 (2009)
16. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM **46**(5), 604–632 (1999)
17. Knossow, D., Sharma, A., Mateus, D., Horaud, R.: Inexact matching of large and sparse graphs using laplacian eigenvectors. In: Proceedings of the 7th IAPR-TC-15 International Workshop on Graph-Based Representations in Pattern Recognition, p. 153. Springer (2009)
18. Koch, I.: Enumerating all connected maximal common subgraphs in two graphs. Theor. Comput. Sci. **250**(1–2), 1–30 (2001)
19. Krissinel, E., Henrick, K.: Common subgraph isomorphism detection by backtracking search. Softw. Practice Experience **34**(6), 591–607 (2004)
20. Lee, W., Duin, R.: An inexact graph comparison approach in joint eigenspace. In: Proceedings of the 2008 Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition, p. 44. Springer (2008)
21. McGregor, J.: Backtrack search algorithms and the maximal common subgraph problem. Softw. Practice Experience **12**(1), 23–34 (1982)
22. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In: ICDE, pp. 117–128 (2002)
23. Newman, M.E.J.: Power laws, pareto distributions and zipf's law. Contemp. Phys. **46**, 323–351 (2005)
24. Ogata, H., Fujibuchi, W., Goto, S., Kanehisa, M.: A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. Nucleic Acids Res. **28**(20), 4021–4028 (2000)
25. Qiu, H., Hancock, E.: Graph matching and clustering using spectral partitions. Pattern Recognit. **39**(1), 22–34 (2006)
26. Raymond, J., Gardiner, E., Willett, P.: Rascal: Calculation of graph similarity using maximum common edge subgraphs. Comput. J. **45**(6), 631 (2002)
27. Riesen, K., Jiang, X., Bunke, H.: Exact and inexact graph matching: methodology and applications. In: Managing and Mining Graph Data (Chapter 7) (2010)
28. Singh, R., Xu, J., Berger, B.: Pairwise global alignment of protein interaction networks by matching neighborhood topology. In: Research in Computational Molecular Biology, pp. 16–31. Springer (2007)
29. Suters, W., Abu-Khzam F., Zhang, Y., Symons, C., Samatova, N., Langston, M.: A new approach and faster exact methods for the maximum common subgraph problem. Comput. Comb. 717–727 (2005)
30. Tong, H., Faloutsos, C., Pan, J.-Y.: Random walk with restart: fast solutions and applications. Knowl. Inf. Syst. **14**(3), 327–346 (2008)
31. Ullmann, J.: An algorithm for subgraph isomorphism. J. ACM (JACM) **23**(1), 42 (1976)
32. Umeyama, S.: An eigendecomposition approach to weighted graph matching problems. IEEE Trans. Pattern Anal. Mach. Intell. **10**(5), 695–703 (1988)
33. Watts, D., Strogatz, S.: Collective dynamics of 'small-world' networks. Nature **393**(6684), 440–442 (1998)
34. Xiao, B., Hancock, E., Wilson, R.: A generative model for graph matching and embedding. Comput. Vis. Image Underst. **113**(7), 777–789 (2009)
35. Xu, L., King, I.: A PCA approach for fast retrieval of structural patterns in attributed graphs. IEEE Trans. Syst. Man Cybern. B Cybern. **31**(5), 812–817 (2001)
36. Zaslavskiy, M., Bach, F., Vert, J.: A path following algorithm for the graph matching problem. IEEE Trans. Pattern Anal. Mach. Intell. **31**(12), 2227–2242 (2009)
37. Zaslavskiy, M., Bach, F., Vert, J.: Global alignment of protein-protein interaction networks by graph matching methods. Bioinformatics **25**(12), i259 (2009)
38. Zhu, Y., Qin, L., Yu, J.X., Ke, Y., Lin, X.: High efficiency and quality: large graphs matching. In: CIKM (2011)