Iterative Repair for Seeded Graph Matching

Barak Babayov

Submitted in partial fulfillment of the requirements for the Master's Degree in the Faculty of Exact Sciences, Department of Computer Science, Bar-Ilan University

Ramat-Gan, Israel 01/2022

Iterative Repair for Seeded Graph Matching

Barak Babayov

Submitted in partial fulfillment of the requirements for the Master's Degree in the Faculty of Exact Sciences, Department of Computer Science, Bar-Ilan University

Ramat-Gan, Israel 01/2022

This work was carried out under the supervision of Prof. Yoram Louzoun and Prof. Liam Roditty Department of Computer Science, Bar-Ilan University

Contents

Abstract

The alignment of two similar graphs from different domains is a well studied problem. In many practical usages, there are no reliable labels over the vertices, leaving structural similarity as the only information available to match such graph. To simplify the matching, one often assumes a small amount of already aligned vertices - called a seed. The current state-of-the-art scalable seeded alignment algorithm is based on percolation. Namely, aligned vertices are used to align their neighbors and gradually percolate in parallel in both graphs. The 'ExpandWhenStuck' algorithm improve former percolation algorithm by generating an inaccurate artificial seed whenever the percolation is stuck, leading to better results using smaller seeds in Erdos Renyi graphs.
However, percolation based graph alignment algorithm are still limited in scale free degree distributions. We here propose 'IRMA' - Iterative Repair for graph MAtching to show that the 'ExpandWhenStuck' can be extended to high performance on real world graphs with a limited additional computational cost. IRMA starts by creating a primary alignment using 'ExpandWhenStuck', then it iteratively repairs the mistakes in the previous alignment steps.

1 Introduction

Graphs are commonly used in different real-world applications, such as social and communication networks. We denote a graph as $G = (V, E)$, where V is the set of nodes and E is the set of edges, i.e. a collections of pairs of nodes.

Our research is focused on the 'Seeded Graph Matching' problem that is a variation of the Maximum Common Subgraph (MCS) problem, specifically for large graphs, i.e. graphs that contain hundreds of thousands and even millions of nodes. Given two graphs G_1 and G_2, the $MCS(G_1, G_2)$ is the largest graph H s.t. $H \subseteq G_1, G_2$. Since the terms 'largest graph' and '$H \subseteq G$' have more than one definition, the MCS problem have several versions. The MCS is known to be NP-hard and therefore only approximate solutions have been discussed in the literature on a large scale. As it is described later in this paper, we use additional information called 'seed' as an input in order to allow large graphs yet accurate results.

In this section, we first introduce the common idea MCS problems, then we introduce the difference between Maximum Common Edges Subgraph (MCES) and Maximum Common Induced Subgraph (MCIS), and their seeded and approximate versions. Finally, we present our area of research called 'seeded graph matching' along with different variants of the problem, caring the same name.

1.1 MCS

The maximum common subgraph (MCS) problem require finding a large graph that is isomorphic to subgraphs of two input graphs simultaneously and it is a way to determine the similarity between two graphs. Because graphs are widely used to model real-world phenomena, the MCS problem has arisen in biomedical analysis, malware detection, cloud computing, source code analysis, etc. This is especially important in the task of drug design, where the successful extraction of common substructures in compounds can reduce the number of experiments needed to be conducted by humans [2]. Unfortunately, finding MCS is an NP-hard problem due to a simple reduction to the Subgraph Isomorphism problem, resulting in exponential worst-case running time algorithms.

1.2 Definitions

Graph isormorphism - Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if and only if there is a bijection $f : V_1 \to V_2$ s.t. $(u, v) \in E_1 \iff (f(u), f(v)) \in E_2$.
Mapping - We here denote any bijection function from one group of nodes to another - a mapping.
Subgraph - Graph G_1 is a subgraph of G_2 if and only if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$
Induced subgraph - Graph G_1 is an induced subgraph of G_2 if and only if $V_1 \subseteq V_2$ and $\forall v_1, v_2 \in V_1$ $(v_1, v_2) \in E_1 \iff (v_1, v_2) \in E_2$.
Common (induced) subgraph - Graph H is a common (respectively, induced) subgraph of G_1 and G_2, if H is isomorphic to (respectively, induced) subgraphs of G_1 and G_2 simultaneously.

1.3 MCIS and MCES

In this proposal, we consist with the definitions presented in [17] to the versions of MCS. A maximum common induced subgraph (MCIS) consists of a graph H with the largest number of nodes

that is a common induced subgraph of G_1 and G_2. A similar problem is the maximum common edge subgraph (MCES). A graph that is a common subgraph of G_1 and G_2, with the largest number of edges. Since in both problems each node of the subgraph is isomorphic to some $v_1 \in V_1$ and $v_2 \in V_2$, we can say the MCES (or MCIS) map v_1 to v_2. We refer to that mapping as the MCES mapping. As the mentioned mapping precisely defines the desired subgraph and its isomorphisms, we mainly discuss the mapping construction.

In many practical usages of MCES, we have some former knowledge about the similarity between the two graphs. Formally, given two large graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and the small subgroups $S_1 \subseteq V_1$, $S_2 \subseteq V_2$ with a mapping function $M : S_1 \to S_2$ s.t. $|S_1| = |S_2| << min(|V_1|, |V_2|)$, the algorithm return a mapping $F : V_1 \to V_2$ as in the MCES problem s.t. $\forall v \in S_1, F(u) = M(u)$.

State-of-the-art exact MCS detection algorithms have a worse case exponential cost [13, 7]. The usefulness of MCS detection and the inefficiency of exact MCS solvers call for the design of approximate solvers. Such solvers reduce the running time, but also the accuracy. Some of the approximate solvers based over advanced machine-learning method [2, 1], yet, approximate MCS algorithms are either limited to an input of a few hundred nodes or provide inaccurate solutions.

1.4 Seeded Graph Matching

In graph matching (GM), one is given two graphs G_1 and G_2 known to model the same data (i.e. there is an equivalence between the graphs vertices). For example, G_1 may be the friendship graph from the Facebook social network and G_2 the friendship graph from the Twitter social network for the same people. In both cases, the vertices are users and there is an edge between two vertices if the corresponding users are friends in the relevant social network. We assume that friend in one network have a higher than random probability of being friend in the second network.

The goal of GM is to create a bijection $M : V_1 \to V_2$, such that M maps vertices in V_1 to a vertex in V_2 if they represent the same real-world entities. In the example above M should connect profiles in Facebook and Twitter that belong to the same person. Given a bijection M, if $M(v_1) = v_2$, and $M(v_3) = v_4$, and the edge $(v1, v3) \in E_1$, and $(v2, v4) \in E_2$, the common edge will be defined as a shared edge. The quality measure for the quality of M is usually the *number* of shared edges.

Finding a full bijection is not always optimal, since some vertices may be absent from one of the two graphs. We thus look for a partial bijection: $M : \tilde{V_1} \subset V_1 \to \tilde{V_2} \subset V_2$, such that the *fraction* of shared edges is maximal. A single edge bijection is obviously a simple solution to that. Thus, a trade-off between the number of shared edges and the fraction of mapped vertices is often required, by adding a constraint on the number of elements in $\tilde{V_1}$.

In the presence of limited initial information on the bijection, one can use seeded GM. In seeded GM, the input contains beyond (G_1, G_2), also a small $seed \subset V_1 \times V_2$, which is a group of vertices pairs known to represent the same real-world entities in G_1 and in G_2.

1.5 Variants of Graph Matching

A similar problem emerges when the vertices have additional information named labels or metadata [12, 15]. For example, the users on Facebook and Twitter may have additional attributes, such as age, address, and user names. It is obvious that a user named "Bob Marley" in Facebook is much more likely to represent the same person as a user with the same name on Twitter than a user named "Will Smith". For that reason, labeled vertices are significantly simplifying the matching problem, and pairs of vertices with highly similar attributes can be used as a seed. In out research, we focus on seeded GM based, with no labels over the vertices.

MCS - (Maximum Common Subgraph)	Given two graphs G_1 and G_2, $MCS(G_1, G_2)$ is the largest graph H s.t. $H \subseteq G_1, G_2$. MCS might refer various of problems depending on the definitions of common subgraph and graph's size. MCS is often an umbrella term for different similar tasks.
MCIS - (Maximum Common Induced Subgraph)	Given two graphs G_1 and G_2, $MCIS(G_1, G_2)$ is a graph H with the largest amount of nodes that is a common induced subgraph of G_1 and G_2
MCES - (Maximum Common Edges Subgraph)	Given two graphs G_1 and G_2, $MCES(G_1, G_2)$ is a graph H with the largest amount of edges that is a common subgraph of G_1 and G_2
Seeded Graph Matching	Given two correlated graphs G_1 and G_2 that are known to model the same data, and a given a small group of pairs $seed \subset G_1 \times G_2$ representing the same real-world objects, we want to find a bijection $M : V_1 \to V_2$ full or partial such that M map a node in V_1 to a node in V_2 if they are representing the same real-world object.
Variants of Graph Matching	In some definitions of Graphs Matching we also given labels upon the nodes, facilitating the process of matching.

2 Related Work

2.1 MCS Solvers

Since Graph Matching is strongly related to the MCS problem, we present here the state-of-the-art exact MCIS solver along with two novel approximate solvers.

2.1.1 McSplit

McSplit [13] is state-of-the-art exact MCIS solver, based on node labeling and partitioning. The algorithm builds up a mapping M using a depth-first search, starting with $M = \emptyset$ and adding a pair (v_i, w_i) $v_i \in V_1, w_i \in V_2$ at each level of the search tree. At any stage, every node of both graphs has a label $l_i \in \{0, 1\}^{|M|}$ representing either it is a neighbor to the relevant node in each pair in M. The algorithm can only add a pair of nodes to M if they have the exact label. The labels are also used as an upper bound for the possible extension of the current M, to avoid useless routes in the search tree without fully explore them. This algorithm has no polynomial worst-case run time, yet experiments show a speedup of more than an order of magnitude over the former state-of-the-art exact MCIS solver.

2.1.2 S-GWL

Scalable Gromov-Wasserstein Learning (S-GWL) [8] is an algorithm based on the Gromov-Wasserstein discrepancy (GWD) with extremely low run-time at the expense of accuracy. Denote a measure graph as $G(V, C, \mu)$, where $V = \{v_i\}_{i=1}^{|V|}$ is the set of nodes, $C = [c_{ij}] \in \mathbb{R}^{|V|\times|V|}$is the adjacency matrix, and $\boldsymbol{\mu} = [\mu_i] \in \Sigma^{|V|}$ is a Borel probability measure defined on V. For each $p \in [1, \infty]$ and each measure graphs G_1, G_2, the Gromov-Wasserstein discrepancy between them is

$$d_{gw}(G_s, G_t) := min_{T\in\Pi(\mu_s,\mu_t)}(\sum_{i,j\in V_s}\sum_{i',j'\in V_t}|c^s_{ij} - c^t_{i'j'}|^p T_{ij}T_{i'j'})^{1/p} \tag{1}$$

where $\Pi(\mu_s, \mu_t) = \{\boldsymbol{T} \geq \boldsymbol{0}|\boldsymbol{T}\boldsymbol{1}_{|V_t|} = \boldsymbol{\mu}_s, \boldsymbol{T}^\top\boldsymbol{1}_{|V_s|} = \boldsymbol{\mu}_t\}$.
In their Algorithm, they use the $\boldsymbol{T}$ yielding $d_{gw}(G_s, G_t)$ as follows. Given two graphs G_1, G_2, they defines a graph H with k nodes and computes $d_{gw}(G_i, H)$. By taking the highest value at each row of the matching distribution, they can match each $v \in G_i$ into $u \in H$, creating an approximation to the k-partition for G_i. Then, they can solve the matching for each subgraph separately by recursion. Since the computation of d_{gw} is expensive, they approximated it is value using the regularized proximal gradient method. The total complexity received is $O((V + E)logV)$ and their accuracy is about 50% with a high variance [8, 2].

2.1.3 NeuralMCS

NeuralMCS [1] makes use of a node embedding method called Graph Matching Networks (GMN) [22] designed for graph similarity computation. Based on the GMN's result, they compute the likelihood of matching each node in G_1 to each node in G_2. These values, indicates which node pair is most likely to be in the MCIS, encodes into a matching matrix $Y \in [0, 1]^{|G_1||G_2|}$. Then, the matching starts by finding the most likely pair, and iteratively expands the extracted subgraphs by selecting the most likely pair each time and keeping the extracted subgraph connected. The procedure stops once the addition of any additional pair would lead to non-isomorphic subgraphs. Experiments on four real graphs show that the model above is 31.78 faster than the exact solver while achieving near-perfect accuracy in MCIS detection.

2.1.4 RLMCS

RLMCS [2] is a graph neural network based model for MCIS detection through reinforcement learning. The model utilizes a novel Joint Subgraph Node Embedding (JSNE) network to perform graph representation learning. This representation is fed into a Deep Q-Network (DQN) [14] to predict action distributions. Finally, They use an exploration tree based on beam search to perform subgraph extraction iteratively using the DQN results. The entire model is trained end-to-end in the reinforcement learning (RL) framework. Experiments on real graph datasets demonstrate that this model too achieves near-perfect accuracy in MCS detection while being slightly faster than supervised neural graph matching network models.

2.2 Graph Matching Solvers

GM is a necessary function in many scientific disciplines including social networking, biology, computer vision, etc [9]. In social networking Graph Matching can de-anonymize two data sets from

different domains [11, 21]. It is also used on proteins from different species in biology to trace functional equivalences [10, 18] and to discover a resembling between images in the domain of computer vision [19, 4, 20].

An important aspect of GM is scaling, since its practical use is often in large graphs. Current state-of-the-art scalable seeded GM methods are based on gradual percolation, starting from the seed and expanding through common neighbors. This class of algorithms is refereed to as Percolation Graph Matching (PGM) methods [11, 21, 6]. Despite having in some cases additional information in the form of labels, [6] showed the crucial importance of relying on edges during the process of GM. Both [3] and [9] present an improvement to [21] and are currently state-of-the-art PGM algorithms. Here we focus on [9] and improve it by presenting our Iterative Repair for graphs MAtching (IRMA) algorithm.

3 Methods

3.1 Notations

For convenience, we follow here the notations of ExpandWhenStuck (further denoted EWS) [9] with minor changes (see Table 1 for list of notations).

Pair	A pair $[u, v]$ is $[u, v] \in V_1 \times V_2$. Sometimes we refer to a pair p without specifying that it is in $V_1 \times V_2$.
Neighboring pair	Given the graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the pairs $[u, v], [u', v'] \in V_1 \times V_2$ are neighboring pairs, if there are edges $(u, u') \in E_1$ and $(v, v') \in E_2$.
Spreading out marks	In the description of the matching algorithms, we refer to a pair p spreading out marks as adding one mark to each neighboring pair of p.
$MarkedPairs$	Contains marks-counters for all marked pairs.
$Marks_t(p)$	Number of marks pair p received from other pairs until time t. $score_t(p)$ is defined accordingly.
$score_t([u, v])$	$score_t([u, v]) = marks_t([u, v]) - \epsilon * \Delta(d_{1,u}, d_{2,v})$ for an infinitesimal $\epsilon > 0$ where $d_{i,u}$ is the degree of node u in graph i.
$Marks_{i,t}(p)$	Number of marks pair p received from other pairs during the $i-th$ iteration until time t. $score_{i,t}(p)$ is defined accordingly.
$Marks_{i,t}(p)$	$Marks_{i,t}(p) = max(marks_{i,t}(p), marks_{i-1,\infty}(p))$. $\bar{score}_{i,t}(p)$ is defined accordingly.
R	The ground-truth, i.e., the set of all pairs of vertices that represent the same entity in both of the graphs.
$Seed$	$Seed \subset R$ is a small group of pairs given as input and known to contain only correct matches.
A	A set of pairs to spread out marks from. Used in EWS, initialized to $seed$.
Z	A set of pairs that already spread out marks. Used in EWS to prevent repetitions.
M	Set of all pairs matched by the algorithm. We refer to any pair in M as 'matched pair' and any other pair as a candidate.
Pairs conflict	We say that pairs $[u', v'], [u, v] \in V_1 \times V_2$ conflict if $u = u'$ and $v \neq v'$ or vice versa. If a candidate pair p conflicts a matched pair p', we say that p conflicts M.
$Weight(M)$	$\lvert\{(u, v) \lvert (u, v) \in E_1, (M(u), M(v)) \in E_2\}\rvert$
s	$s \in [0, 1]$ - the probability that an edge should be sampled in V_1 or in V_2

Table 1: Main notations

3.2 Current State-of-the-art Algorithm

The current state-of-the-art algorithm to the above problem is ExpandWhenStuck [9]. In a nutshell, EWS (see code in Figure 1) starts by adding all the seed pairs into M, and spreading

out marks to all of their neighbors. Then, at each time step t, EWS chooses a candidate pair $p' = argmax_p(score_t(p))$ that does not conflict M, adds it to M and spreads out marks to all of its neighbors (see Figure 2). When there are no pairs left with more than one mark (line 15 in Figure 1), EWS creates an artificial noisy seed (A), and uses it to further spread out marks (line 6 in figure 1). A contains all pairs that are: 1) neighbors of matched pairs 2) do not conflict M 3) never have been used to spread out marks. The novelty of EWS is the generation of an artificial seed whenever there are no more pairs with more than one mark. The artificial seed is mostly wrong. Yet, EWS manages to use it to match new correct pairs and continue the percolation. At the end of EWS, the set of mapped pairs M is returned along with $MarkedPairs$ that contains counters of marks for all marked pairs. The last is not needed in EWS, but is used in IRMA. The main contribution of EWS is a dramatic reduction in the size of the required seed set for random $G(n, p)$ networks (graph with n vertices and a probability of p for each edge).

ExpandWhenStuck

1: $A \leftarrow seed$ is the initial set of seed pairs, $M \leftarrow seed$;
2: $Z \leftarrow \emptyset$ is the set of used pairs
3: $MarkedPairs \leftarrow \emptyset$ is the set of all marked pairs along with their number of marks
4: **while** ($|A| > 0$) **do**
5: **for** all pairs $[i, j] \in A$ **do**
6: Add the pair [i,j] to Z and add one mark to all of its neighboring pairs;
7: **end for**
8: **while** there exists an unmatched pair with at least 2 marks in $MarkedPairs$ that does not conflict M **do**
9: among those pairs select the one maximizing $score(p)$;
10: Add p=[i,j] to the set M;
11: **if** [i,j] $\notin Z$ **then**
12: Add one mark to all of its neighboring pairs and add the pair [i,j] to Z;
13: **end if**
14: **end while**
15: $A \leftarrow$ all neighboring pairs [i,j] of matched pairs M s.t. [i,j]$\notin Z$, $i \notin V_1(M)$ and $j \notin V_2(M)$;
16: **end while**
17: **return** M, $MarkedPairs$;

Figure 1: Current state-of-the-art algorithm for Seeded Graph Matching. Base over percolation method, which is highly scale-able, to gradually map neighboring pairs of matched pairs.

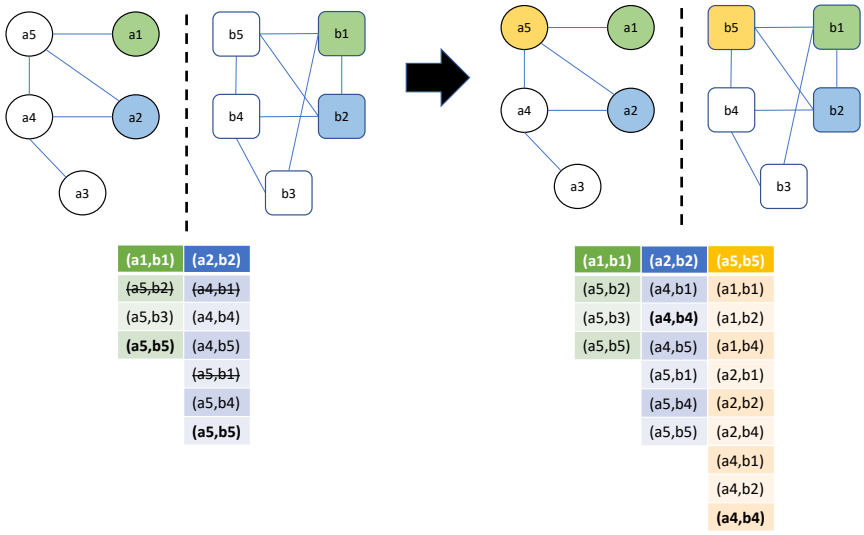


Figure 2: **ExpandWhendStuck example.** Given the seed {(a1,b1),(a2,b2)} on the left hand side, (a1,b1) spread out marks to the Cartesian product {a5}*{b2,b3,b5} (green column) and (a2,b2) spread out marks to {a4,a5}*{b1,b4,b5} (blue column). Marked pairs conflicting the seed have been crossed out while (a5,b5) has been marked in bold for having the most marks. On the right hand side (a5,b5) been painted in yellow for getting into M and marks have been spread out to {a1,a2,a4}*{b1,b2,b4} (yellow column). We marked (a4,b4) in bold for being the next pair to be inserted into M.

3.3 Evaluation Methods and Stopping Criteria

We use precision and recall to evaluate the performance of algorithms: (i) Precision refers to the fraction of errors in the set of matched vertices (i.e., pairs in M that are in R), and (ii) Recall is the size of the intersect of M and R out of the size of R as

$$Precision = \frac{\Lambda(M)}{|M|}, Recall = \frac{\Lambda(M)}{|R|}, \tag{2}$$

where $\Lambda(M)$ is the number of correct pairs in M and R is the set of all pairs of vertices that represent the same entity in both of the graphs.

To compare the performance of GM algorithms, we also report the F1–score, defined as:

$$F1 - score = 2\frac{precision \times recall}{precision + recall}. \tag{3}$$

The scores above (Recall, Precision, and F1) can only be computed based on known ground truth, yet it is useful to approximate the quality of a solution during the run time, assuming no known

ground truth. Namely, given an input to the seeded-GM problem and two possible maps $M, M' : V_1 \to V_2$ we want to determine which of the two is better. We use $weight(M)$ as a score for the quality of a mapping M:

$$weight(M) = |\{(u,v)|(u,v) \in E_1, (M(u), M(v)) \in E_2\}|. \quad (4)$$

3.4 Generating correlated graphs

To examine the performance of seeded GM algorithms, one needs a pair of graphs, where at least a part of the vertices corresponds to the same entities. EWS used a simple probabilistic sampling method over a single given graph to create two correlated graphs with different levels of similarity.

We follow the same method. Specifically, given $G = (V, E)$ and s, we generate $G_1 = (V_1, E_1)$, and $G_2 = (V_2, E_2)$ by twice randomly and independently removing edges $e \in E$ with a probability of $1 - s$. We then remove vertices with no edges. The edge overlap between G_1 andG_2 increases with s. EWS used only $s \in [0.7, 0.9]$. Using IRMA, we show that a high F1 score can be obtained for the range of $s \in [0.4, 0.9]$. Note that given (G, s), a vertex $v \in G$ with k neighbors contained in both G_1, G_2 has an expected degree of s^2k in $G1 \cap G_2$. We name s the 'graphs-overlap'.

3.5 Data Sets

For fully simulated graphs, we use the above sampling method over $G(n, p)$ Erdos-Renyi graphs [5], defined as a graph with n vertices, where every edge of the possible $\binom{n}{2}$ exists with a probability of p. It is common to mark two graphs created by sampling from an Erdos-Renyi graph as $G_1, G_2 = G(n, p, s)$ [16].

To test our algorithm on graphs that better represent real-world data, yet to control their level of similarity, we used sampling over the following real-world graphs (further denoted as graph 1, graph 2, and so on, according to their order here (Table 2)).

Number	Name	Nodes	Edges	Average degree
1	Fb-pages-media	27,900	206,000	14
2	Soc-brightkite	56,700	212,900	7.8
3	Soc-epinions	26,600	100,100	7.9
4	Soc-gemsec-HU	47,500	222,900	9.4
5	Soc-sign-Slashdot081106	77,300	516,600	12.1
6	Deezer_europe_edges	28,300	92,800	6.6

Table 2: Real world data set graphs.

1. Fb-pages-media - Data collected about Facebook pages (November 2017). These datasets represent verified Facebook page graphs of different categories. Vertices represent the pages and edges are mutual likes among them (http://networkrepository.com/fb-pages-media.php).

2. Soc-brightkite - Brightkite is a location-based social networking service provider where users shared their locations by checking-in. The dataset contains all links among users (http://networkrepository.com/soc-brightkite.php).

3. Soc-epinions - Controversial Users Demand Local Trust Metrics: An Experimental Study on epinions.com Community (http://networkrepository.com/soc-epinions.php).

4. Soc-gemsec-HU - The data was collected from the music streaming service Deezer (November 2017). These datasets represent friendship graphs of users from 3 European countries. Vertices represent the users and edges are the mutual friendships. We re-indexed the vertices in order to achieve a certain level of anonymity. The edge files that contain the edges - vertices are indexed from 0. The json files contain the genre preferences of users - each key is a user id, the genres loved are given as lists. Genre notations are consistent across users. In each dataset users could like 84 distinct genres. Liked genre lists were compiled based on the liked song lists. The countries included are Romania, Croatia and Hungary (http://networkrepository.com/soc-gemsec-HU.php).

5. Soc-sign-Slashdot081106 - Slashdot Zoo signed social network from November 6 2008. It is noteworthy that this graph was also used in [9] (http://networkrepository.com/soc-sign-Slashdot081106.php).

6. Deezer_europe_edges - A social network of Deezer users which was collected from the public API in March 2020. Vertices are Deezer users from European countries and edges are mutual follower relationships between them. The vertex features are extracted based on the artists liked by the users. The task related to the graph is binary node classification - one has to predict the gender of users. This target feature was derived from the name field for each user (http://snap.stanford.edu/data/feather-deezer-social.html).

3.6 Second Run

The EWS algorithm has a greedy property. At step t, it adds to M the pair maximizing $score_t(p)$. Assume that at time t the algorithm chooses some wrong pair $[u', v]$, in particular, $score_t([u', v]) > score_t([u, v])$ for the correct pair $[u, v]$. Let us further assume that $[u, v]$ will receive many marks from neighbors later in the algorithm, such that $score_\infty([u', v]) < score_\infty([u, v])$. Such a scenario is highly likely, since we argue that correct pairs tend to reach a high score. Yet, $[u, v]$ will never get into M as it conflicts with another pair in it ($[u', v]$).

We use the accumulation of scores ($score_\infty([u', v]) < score_\infty([u, v])$) to improve the matching. We suggest to perform a second iteration in which wrong matched pairs could be fixed. Since matched pairs are spreading out marks at the second iteration as well, let us define $mark_{i,t}(p)$ as the number of marks gained by pair p during the i-th iteration until time t; $score_{i,t}(p)$ is defined accordingly as $marks_{i,t}([u, v]) - \epsilon * \Delta(d_{1,u}, d_{2,v})$. Base on those scores we can establish $\bar{mark}_t(p) = max(mark_{1,\infty}(p), mark_{2,t}(p))$, ($\bar{score}_t(p)$ defined accordingly) the second iteration uses $\bar{score}_t(p)$ as presented in Figure 3. First it initializes M to be the seed set, then, at time t, while there exists a candidate pair with $\bar{mark}_t(p) > 1$, we add into M the pair p maximizing $\bar{score}_t(p)$ among those who do not conflict M and then spread marks out of it.

Second Run

1: $M_1, MarkedPairs_1 \leftarrow ExpandWhenStuck(A_0)$ is the map and set of all marks from the first run
2: $MarkedPairs_2 \leftarrow \emptyset, M_2 \leftarrow A_0$
3: spread out marks from all pairs in M_2
4: **while** there exists an unmatched pair with at least 2 marks in $MarkedPairs_1$ or $MarkedPairs_2$ that does not conflict M_2 **do**
5: Among those pairs select the pair $p = argmax_p\{\bar{score}_t(p)\}$;
6: Spread out marks from p (updating $MarkedPairs_2$) and add it to the set M_2;
7: **end while**
8: **return** M_2

Figure 3: Second-Run starts by running ExpandWhenStuck (iteration 1), then it builds M_2 based on the information gained during both first and second iterations using the function $\bar{score}_t(p)$.

3.7 Iterative approach

The next natural stage will be to add a third run of the algorithm. In fact, we can use $\bar{mark}_{i,t}(p) = max(mark_{i,t}(p), mark_{i-1,\infty}(p))$ to run over and over again until no better map is achieved. We present a pseudo code to the Repairing-Iteration and IRMA in Figures 4 and 5, respectively. IRMA starts with the initialization of M to be the seed set, then it performs a standard EWS iteration. In the following iterations, at time t, while there exists a candidate pair with $\bar{mark}_t(p) > 1$, IRMA adds to M the pair p maximizing $\bar{score}_t(p)$ among those that do not conflict M, and spread marks out of it.

IRMA stops the iterations when the mapping quality stops increasing. Formally, the i-th iteration starts by initializing $M = seed$, then, at time t, it adds to M the candidate pair obeying:

$$p = argmax_p\{\bar{score}_{i,t}(p)\}, \tag{5}$$

and spreads marks out of it - updating $mark_{i,t}$. The iteration ends when no candidate pair p, that does not conflicts M, satisfies $\bar{mark}_{i,t}(p) > 1$.

Since we do not know the real mapping, we use $weight(M)$ to estimate the quality of the score at the current iteration (section 3.3). We compute $weight(M_i)$ $\forall i$ where M_i is the matching at the end of the i-th iteration. Whenever $weight(M_i) \leq weight(M_{i-1})$, the algorithm stops and returns M_{i-1}. In practice, to avoid many redundant iterations with a limited increase in $weight(M)$, the algorithm stops when $weight(M) \leq (1 + \delta) * weight(M_{i-1})$, where δ was empirically set to 0.01. Note that this does not ensure convergence of the mapping, only of its score.

Repairing Iteration

1: $MarkedPairs_i$ is an input of all marks from the previous iteration
2: $MarkedPairs_{i+1} \leftarrow \emptyset$, $M \leftarrow A_0$;
3: Spread out marks from all pairs in M
4: **while** there exists an unmatched pair with at least 2 marks in $MarkedPairs_i$ or $MarkedPairs_{i+1}$ that does not conflict M **do**
5: Among those pairs select the pair $p = argmax_p\{\bar{score}_t(p)\}$;
6: Add p to the set M;
7: Add one mark in $MarkedPairs_{i+1}$ to all of its neighboring pairs and add the p to Z
8: **end while**
9: **return** $M, MarkedPairs_{i+1}$

Figure 4: Repairing Iteration is a generalization 'Second Run'. It receive the marks from previous iteration along with the seed as input and builds a map considering the marks gained at the previous and current iteration.

IRMA (Iterative Repair for graph MAtching)

1: $M, MarkedPairs \leftarrow ExpandWhenStuck(A_0)$
2: $M_0 = \emptyset$
3: **while** $weight(M) > (1+\delta) * weight(M_0)$ **do**
4: $M_0 = M$
5: $M, MarkedPairs \leftarrow RepairingIteration(MarkedPairs)$
6: **end while**
7: **return** M

Figure 5: IRMA builds a primary map using ExpandWhenStuck and repeatedly improves it by running 'Repairing Iteration'. It uses $wight(M)$ as an indication for convergence by the stop condition appears in line 3.

3.8 Expansion Boost

Each iteration is built of the main while-loop with a break condition of having no candidate pair, with at least two marks, that does not conflict M. The threshold of marks has been determined to two as a trade-off between the scores of precision and recall. If one sets the threshold to a high value, M will only contain pairs with high confidence, yet the percolation will stop early - leading to high precision but low recall. Similarly, setting the threshold to one will increase recall at the expense of precision.

We suggest performing one "noisy iteration" with a threshold of one after IRMA has been converged, expecting the precision to drop at the expense of an increase in recall. Then, we run regular repairing iterations again to gradually restore the precision. The idea is that while wrong pairs could not comply with the next iterations' threshold of two marks, correct pairs might lead to unexplored areas of the graphs. Such pairs gain marks by their newly revealed neighbors, *a-posteriori* justifying their insertion to M.

3.8.1 Break Condition

Since the iterations performed after the 'noisy iteration' are meant to filter out pairs with lower certainty, we can no longer expect $|weight(M)|$ to increase between iterations. In practice, the second stage of IRMA (after the expansion boost), restores the precision within few iterations and afterwords has no significant improvement from M_i to M_{i+1}. We thus empirically set the algorithm to always stop after four repairing iterations.

3.9 parallel version

In order to develop a GPU version of IRMA, we propose a parallel version. The bottleneck of EWS is in spreading out marks from $[u, v]$, costs $deg_1(u) * deg_2(v)$ updates to the queue of marks. Ideally, we would like to perform multiple mark spreading steps in parallel. However, this is not immediately possible since the pair chosen at time t depends on the marks that have been spread out earlier, including those of time $t-1$. Section 6.3 in [9] presents a paralleled version in which the main loop has been split into epochs. This version of EWS starts by spreading out marks from the seed, then at each epoch the algorithm greedily takes all possible pairs from the queue one by one - without spreading any mark. When the queue is eventually empty, it simultaneously spreads marks from all pairs selected at the current epoch, creating the queue to the next epoch. This method has the advantage of being extremely fast, allowing input graphs with millions of vertices, and has been argued not to fundamentally affect the performance of the algorithm.

We used a similar logic to parallelize iterations as follows (see Figure 6 for pseudo code). The i-th iteration gets $queue_{i-1}$ as input and starts by greedily adding all possible pairs from the queue into M_i one by one - without spreading any mark. Then, when the queue is eventually empty, we simultaneously spread out marks from all pairs in M_i creating $queue_i$. Iteration i returns M_i and $queue_i$.

Parallel Repairing Iteration

1: $MarkedPairs_{i-1}$ is an input of all marks from the previous iteration.
2: $MarkedPairs_i \leftarrow \emptyset$, $M_i \leftarrow A_0$;
3: **while** there exists an unmatched pair with at least 2 marks in $MarkedPairs_{i-1}$ or $MarkedPairs_i$ that does not conflict M_i **do**
4: Among those pairs select the pair $p = argmax_p\{score_{i-1,\infty}(p)\}$;
5: Add p to the set M_i;
6: **end while**
7: $MarkedPairs_i \leftarrow$ marks that has been spread out from all M_i in parallel
8: **return** $M_i, MarkedPairs_i$

Figure 6: In order to allow spreading out marks in parallel, we no longer base over the marks of current iterations. It enables us to perform this stage simultaneously at the end of the iteration (line 7).

4 Results

4.1 ExpandWhenStuck on Erdos-Renyi graphs

IRMA is an improvement of EWS [9]. We first tested the accuracy of EWS on a set of fully simulated graphs as done in the original paper (see section 3.5 for detailed information). As a remainder we use a base graph (simulated or given) to create two partially overlapping graphs by twice removing edges from it. The fraction of edges from the original graph (G) maintained in each of the partially overlapping graphs G_1, G_2 is denoted s. We use a value of s in the range of $[0.4, 0.8]$. Using $s \in \{0.7, 0.8, 0.9\}$ and $p = 20/n$ on Erdos-Renyi graphs, EWS reported matching correctly almost every node by a few dozen seeds. To further research the performance of EWS, we used graphs with $p \in \{\frac{20}{n}, \frac{30}{n}, \frac{40}{n}\}$ and expanded the graphs overlap (s) to the range $[0.4, 0.8]$.

Figures 7 - 9 present the algorithm results (precision, recall, and F1, respectively) with different graphs-overlap (s) values. According to eq. 2, since EWS percolates smoothly on $G(n, p, s)$ graphs (i.e, $|M| = |R|$), precision, recall, and F1-score should be identical. In addition, one can notice that all accuracy measures are highly sensitive to s. The reason is that a correct pair $[u, v] \in G_1 \times G_2$, corresponding to vertex $w \in G$ with degree d, has an expected number of $s^2 d$ common neighbors, which also, approximately the number of marks it will get. In random $G(n, p)$ graphs, most vertices have similar degree (a normal distribution of degrees), and if $s^2 * E(d) > 2$, EWS typically works. More precisely, [9] present a simplified version to the EWS that is much easier to analyze and computes the threshold seed size in order for the algorithm to correctly match $G(n, p, s)$ graphs. Under several assumptions they found an upper bound of $O(\frac{1}{np^4 s^4})$.

Figures 10 - 12 represent the algorithm results (precision, recall, and F1, respectively) along different seed sizes. Those figures consistent both with the claim that precision, recall, and F1 should be identical when running over $G(n, p, s)$ graphs and with the expected advantage of high degree graphs and high graphs-overlap (s).
In order to better understand the algorithm, in Figure 13 we produced several indices during a single run of the algorithm on $G(10^4, 10/10^4, 0.7)$ graphs with $|seed| = 50$.

- Sub-figure 13a represents a binomial degree distribution with an expectation of $np = 10$.
- Sub-figure 13b represents the local precision using a sliding window of 30. Mistakes accumulate only at the beginning and at the end of the algorithm, most likely due to a low number of marks (as can be seen in sub-figure d).
- Sub-figure 13c represents the degree of pairs inserted to M, ordered by insertion time (we define pair's degree to be the sum of its nodes' degrees). In both sub-figures c and d, sliding windows have been used to smooth the graphs. Correctly matched pairs has been differentiated from wrong ones by color. The straight orange line in the range $[2000, 9000]$ represents the absence of mistakes in this time window. It is clear from the plot that high degree pairs tend to gain marks much faster than low ones, thus were matched earlier. We suggest that at the beginning of the algorithm the degree of wrong pairs is very much like correct ones since not enough marks have been spread to distinguish between them. However, wrong pairs at the end of the algorithm have match higher degree since it increases their probability to gain marks, allowing them to be matched despite being incorrect.

- Sub-figure 13d represents the number of marks each pair had at the moment of insertion into M, ordered by insertion time. The number of marks is gradually decreasing over time up to the point where only low degree pairs remain as candidates pairs (see sub-figure c). From this point on, the number of marks is gradually decreasing as candidate pairs have lower and lower degrees, and they are unable to gain many marks. The wrong pairs have fewer marks than correct ones since mistakes typically happen when there are only candidates with few marks. The noticeable difference between them is caused by the sliding window used to smooth the plots.

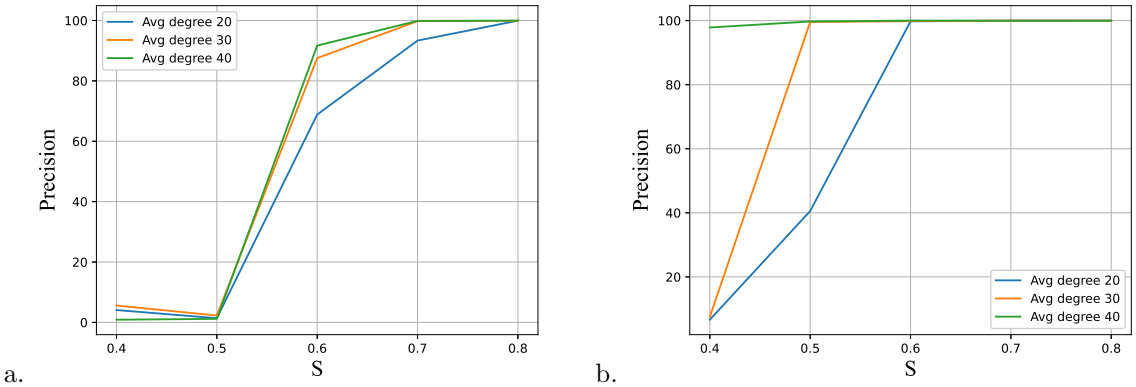
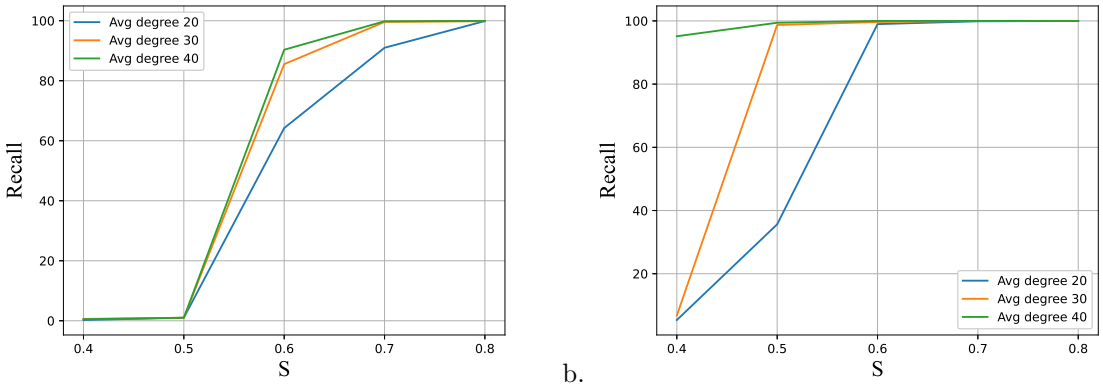
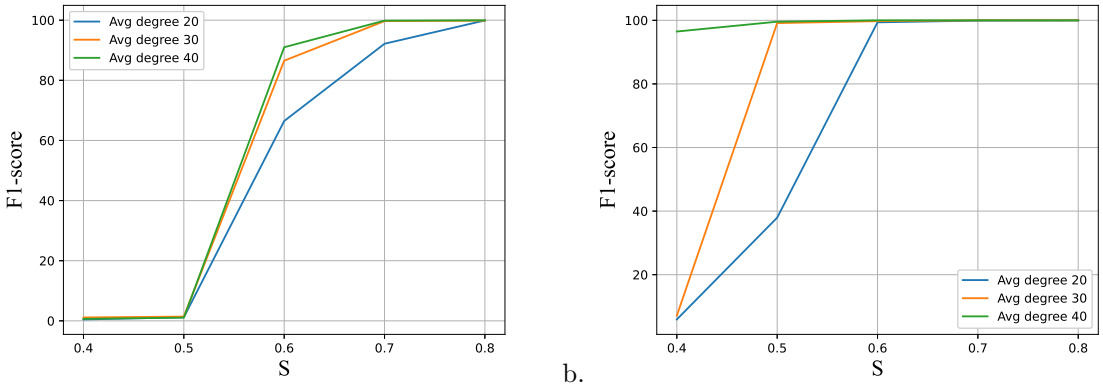


Figure 7: ExpandWhenStuck algorithm: Precision as a function of graphs-overlap (s). Sub-figure a and sub-figure b use 30 and 200 random seeds, respectively, and run over $G(10^4, 0.002, s)$ (blue), $G(10^4, 0.003, s)$ (orange) and $G(10^4, 0.004, s)$ (green). Precision has the exact same values as recall, and F1-score (Figures 8,9). Another observation is a high sensitivity to the graphs-overlap and the graphs average degree.



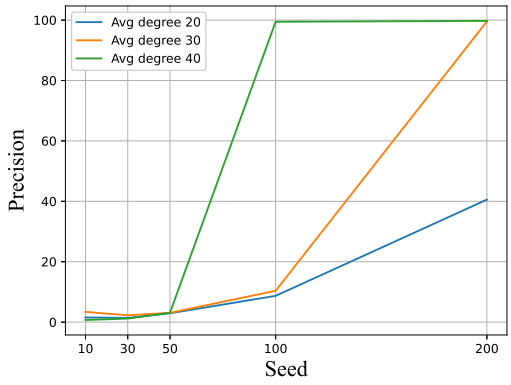
a. b.

Figure 8: ExpandWhenStuck algorithm: Recall as a function of graphs-overlap (s). Sub-figure a and sub-figure b use 30 and 200 random seeds, respectively, and run over $G(10^4, 0.002, s)$ (blue), $G(10^4, 0.003, s)$ (orange) and $G(10^4, 0.004, s)$ (green).

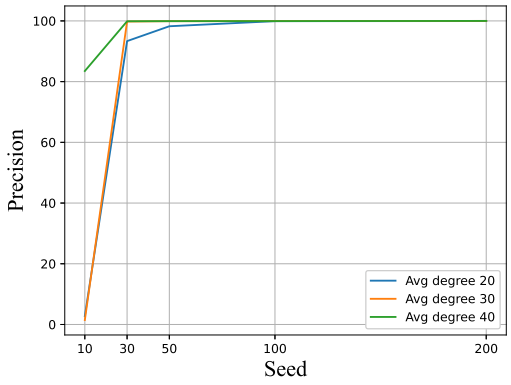


a. b.

Figure 9: ExpandWhenStuck algorithm: F1-score as a function of graphs-overlap (s). Sub-figure a and sub-figure b use 30 and 200 random seeds, respectively, and run over $G(10^4, 0.002, s)$ (blue), $G(10^4, 0.003, s)$ (orange) and $G(10^4, 0.004, s)$ (green).

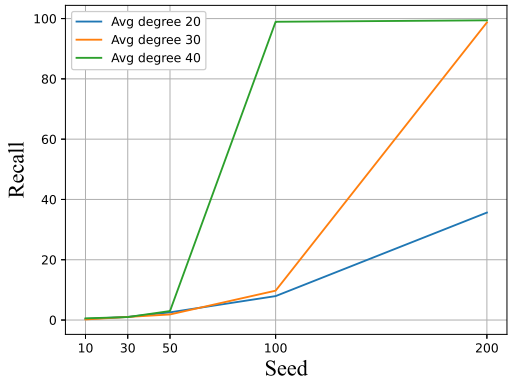


a.

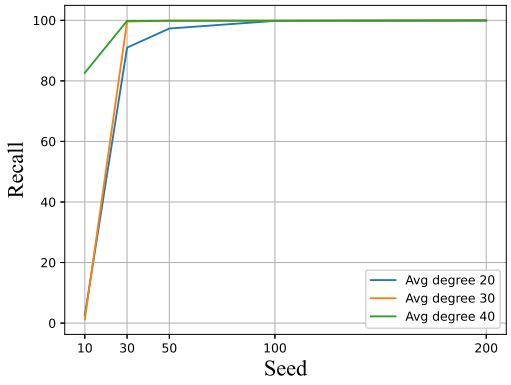


b.

Figure 10: ExpandWhenStuck algorithm: Precision as a function of seed size. Sub-figure a and sub-figure b use graphs-overlap (s) of 0.5 and 0.7, respectively, and run over $G(10^4, 0.002, s)$ (blue), $G(10^4, 0.003, s)$ (orange) and $G(10^4, 0.004, s)$ (green).

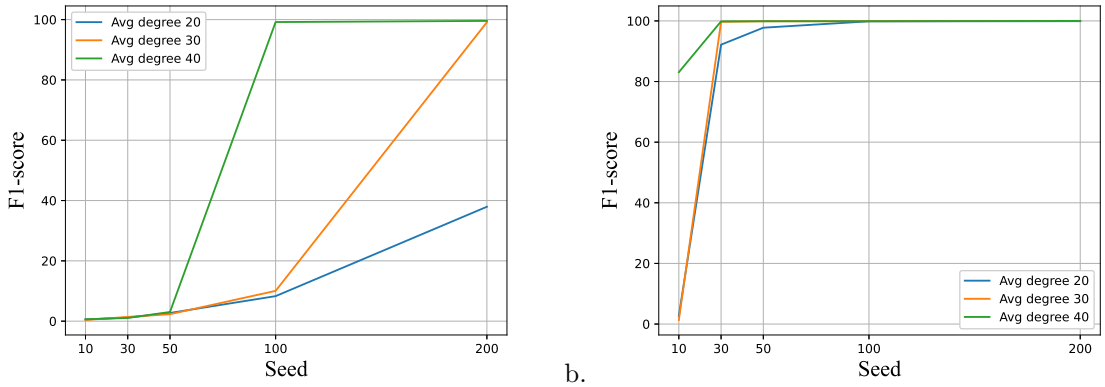


a.



b.

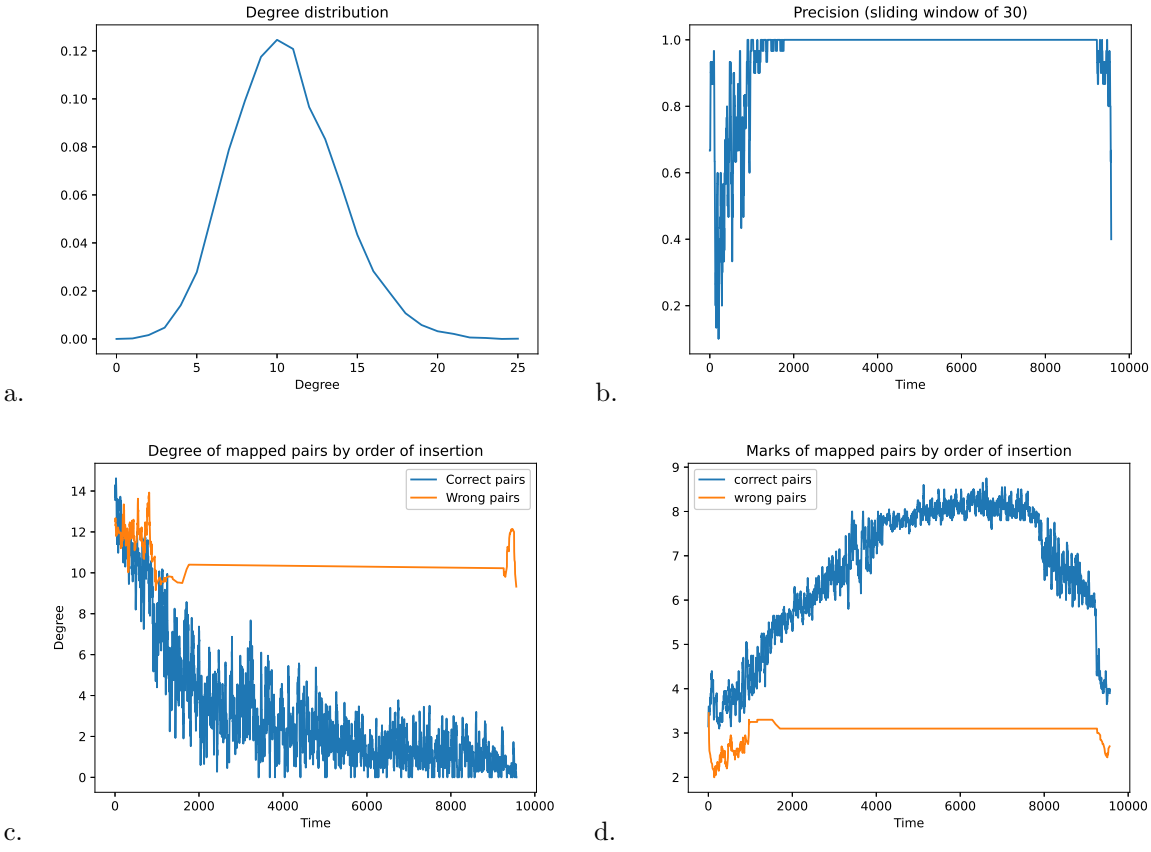
Figure 11: ExpandWhenStuck algorithm: Recall as a function of seed size. Sub-figure a and sub-figure b use graphs-overlap (s) of 0.5 and 0.7, respectively, and run over $G(10^4, 0.002, s)$ (blue), $G(10^4, 0.003, s)$ (orange) and $G(10^4, 0.004, s)$ (green).



a. b.

Figure 12: ExpandWhenStuck algorithm: F1-score as a function of seed size. Sub-figure a and sub-figure b use graphs-overlap (s) of 0.5 and 0.7, respectively, and run over $G(10^4, 0.002, s)$ (blue), $G(10^4, 0.003, s)$ (orange) and $G(10^4, 0.004, s)$ (green).

ExpandWhenStuck on $G(10^4, \frac{10}{10^4}, 0.7)$ with $|seed| = 50$



a. b. c. d.

Figure 13: ExpandWhenStuck over $G(10^4, 0.001, 0.7)$ with $|seed| = 50$: Sub-figure b indicates high performances, with mistakes only A) at the beginning of the algorithm - when not enough marks have been spread (see sub-figure c), and B) at the end - when only low degree pairs were left as candidates (see sub-figure d).

4.2 ExpandWhenStuck on Real World Graphs

We extend our tests on ExpandWhenStuck using real-world graphs, see section 3.5 for detailed information about the selected graphs. We again use each of those graphs to create two overlapping graphs by the above sampling method. Given (G, s) we generate G_1, G_2 by inserting each of G's edges to G_1 and G_2 independently with probability s.

Figures 14, 15 represent precision, recall and F1-score of ExpandWhenStuck over graphs 2 and graph 4 (other graphs are similar) with graphs overlap (s) value of $0.4, 0.5, 0.6, 0.7, 0.8$ (a-e, respectively) as a function of seed size. Figures 16, 17 represents precision, recall and F1-score of ExpandWhenStuck over graphs 1 and graph 3 with different seed size as a function of the graphs

overlap (s). One can still see the high sensitivity to the graphs-overlap value and the threshold nature of the seed size. Yet, even with higher seed size, performances are significantly lower, and there is a consistent large gap between recall and precision. The proportions between precision and recall are equal to the one between $|M|$ and $|R|$ (eq. 6), i.e., the algorithm struggle to percolate through the entire graphs.

$$\frac{recall}{precision} = \frac{\frac{\Lambda(M)}{|G_1 \cap G_2|}}{\frac{\Lambda(M)}{|M|}} = \frac{|M|}{|R|} \tag{6}$$

Since in scale free degree distributions most vertices obey $d < E(d)$, many correct pairs cannot collect enough marks. As a result, M often contains only part of the possible pairs to match.

In order to better understand the algorithm on real-world graphs, in Figure 18 we produced several indices during a single run of the algorithm on graph 6 with graphs-overlap (s) of 0.7 and $|seed| = 480$.

- Sub-figure 18a represents a power-law degree distribution characterized by a long left tail.
- Sub-figure 18b represents the local precision using a sliding window of 30. The reason for lower precision in power-law degree distribution graphs is its cluster nature. The graph is built such that there are a lot of nodes sharing many common neighbors, creating wrong candidates pairs with many marks. The drop in the precision around times 5000 and 9000 was caused by a generation of an artificial seed, adding a lot of wrong marks in exchange for keeping on the percolation process.
- Sub-figure 18c represents the degree of pairs inserted to M by the order of insertion, i.e., at $time = t$ we can see the degree of pair p inserted to M at time t (we define pair's degree to be the sum of its nodes' degrees). In both sub-figures c and d, sliding windows have been used to smooth the graphs correct pairs that have been differentiated from wrong ones by color. It is clear from the plot that high degree pairs tend to gain marks much faster than low ones, thus been matched earlier. At time 5000 percolation has stopped, therefore artificial seed has been created and spread out marks. At this point many new pairs gained marks, so again, high degree pairs were matched first and gradually all other pairs. We suggest that at the beginning of the algorithm the degree of wrong pairs is lower than the degree of correct ones since low degree nodes are harder to match. However, wrong pairs at the end of the algorithm have much higher degree since it increases their probability to gain marks, especially after the artificial seed spread out many wrong marks, allowing them to be matched despite being incorrect.
- Sub-figure 18d represents the number of pairs' marks, also by order of insertion to M. The two picks around time 5000 and 9000 corresponding to the generation of an artificial seed, adding a lot of marks to the candidate pairs.

$$\frac{recall}{precision} = \frac{\frac{\Lambda(M)}{|G_1 \cap G_2|}}{\frac{\Lambda(M)}{|M|}} = \frac{|M|}{|G_1 \cap G_2|} \tag{7}$$

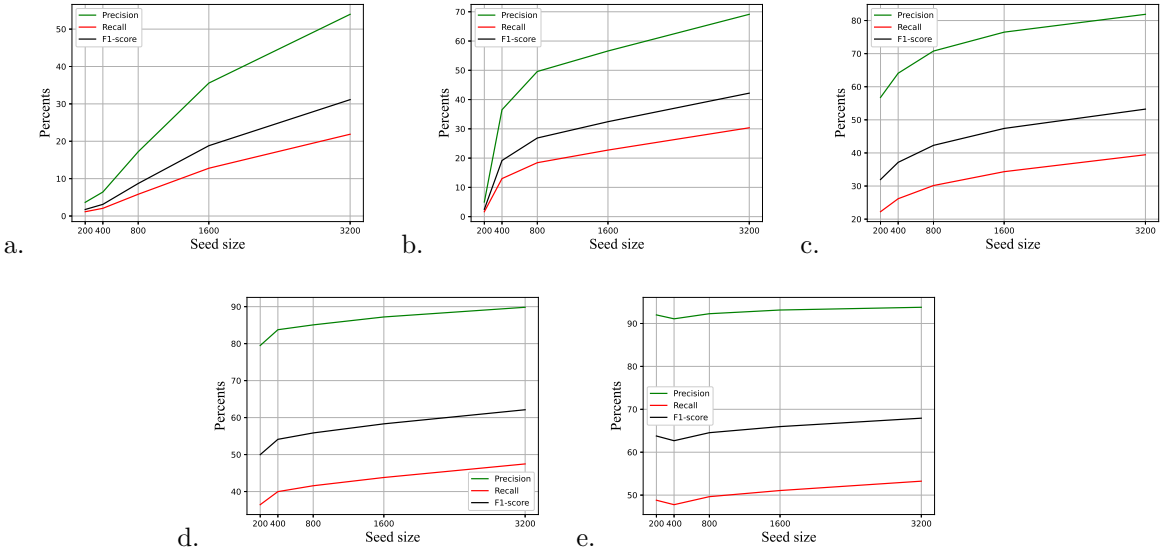


Figure 14: ExpandWhenStuck algorithm: Precision, recall and F1-score over graph 2 as a function of seed size. Sub-figures a-e use increasing s values of $0.4, 0.5, 0.6, 0.7, 0.8$, respectively. Even with higher seed size, performances are significantly lower compared to Erdos-Renyi graphs. In addition, there is a consistent large gap between recall and precision.

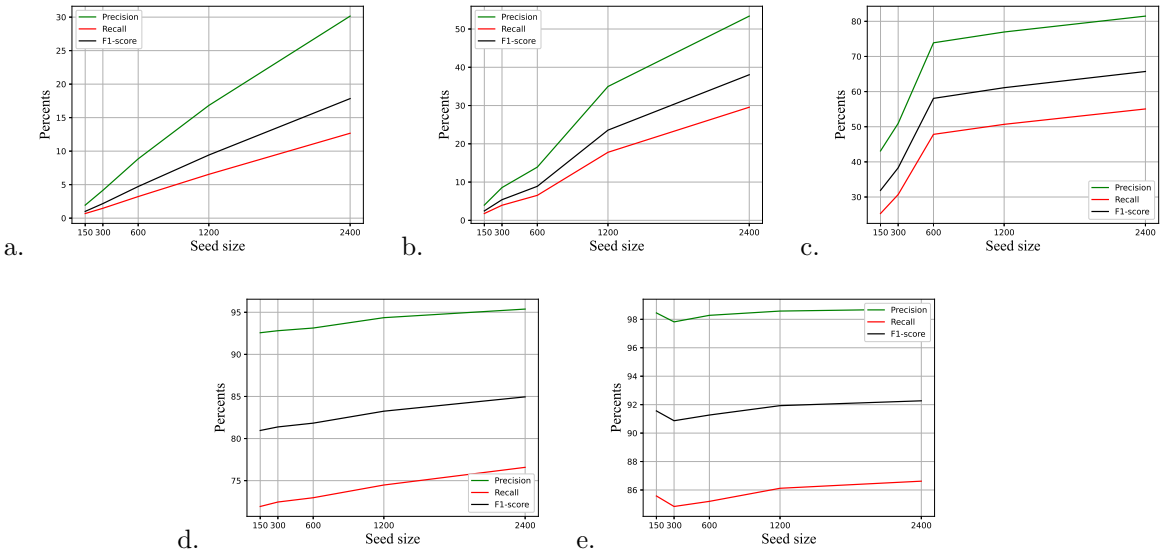


Figure 15: ExpandWhenStuck algorithm: Precision, recall and F1-score over graph 4 as a function of seed size. Sub-figures a-e use increasing s values of $0.4, 0.5, 0.6, 0.7, 0.8$, respectively.

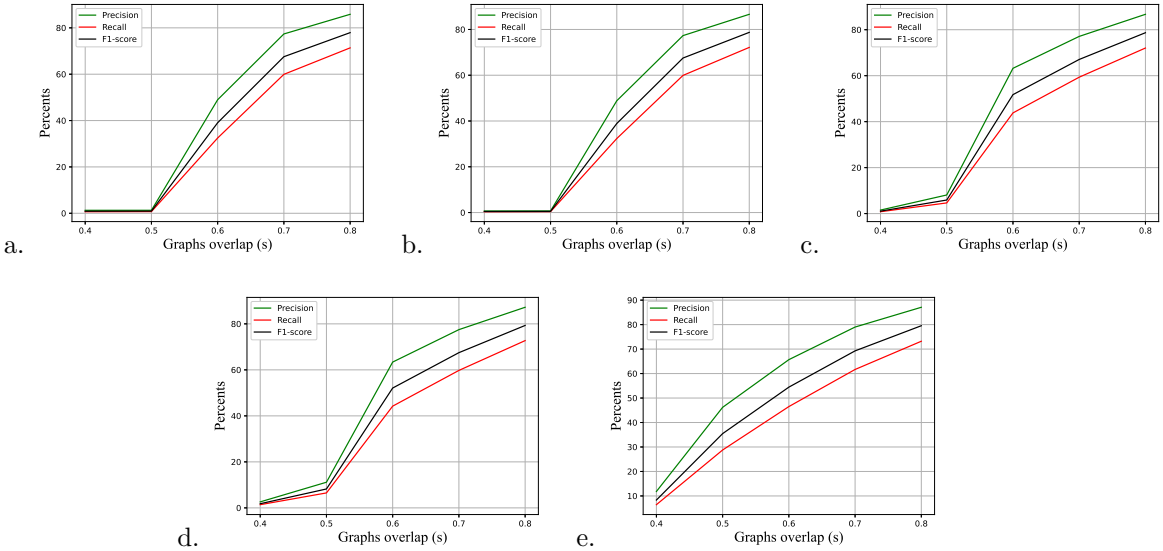


Figure 16: ExpandWhenStuck algorithm: Precision, recall and F1-score over graph 1 as a function of graphs overlap (s). Sub-figures a-e use increasing seed size of $25, 50, 100, 200, 400$, respectively.

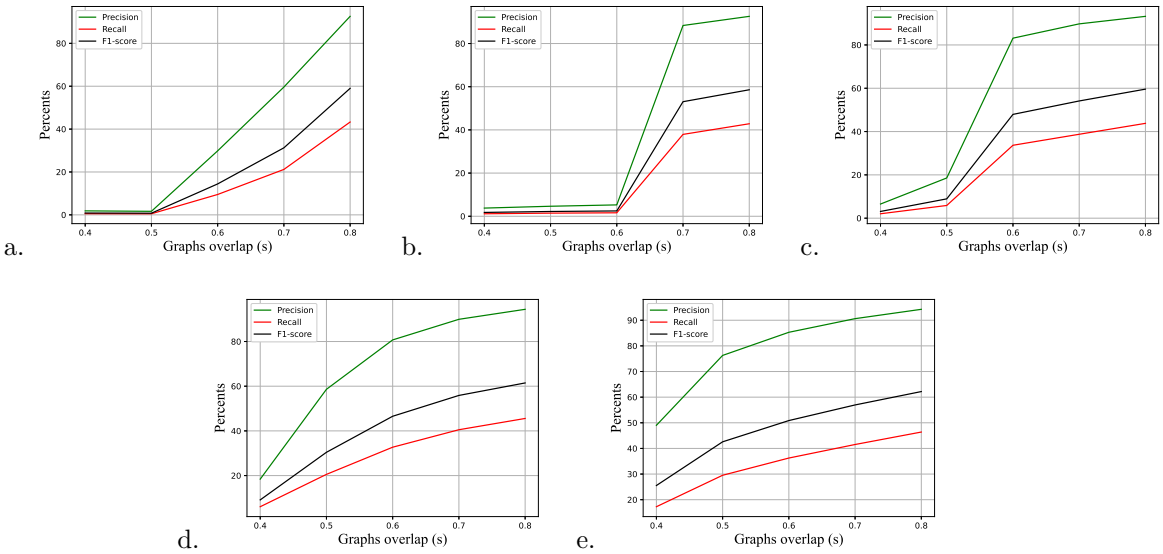


Figure 17: ExpandWhenStuck algorithm: Precision, recall and F1-score over graph 3 as a function of graphs overlap (s). Sub-figures a-e use increasing seed size of $50, 100, 200, 400, 800$, respectively.

ExpandWhenStuck over Graph6, s=0.7, $|seed| = 480$

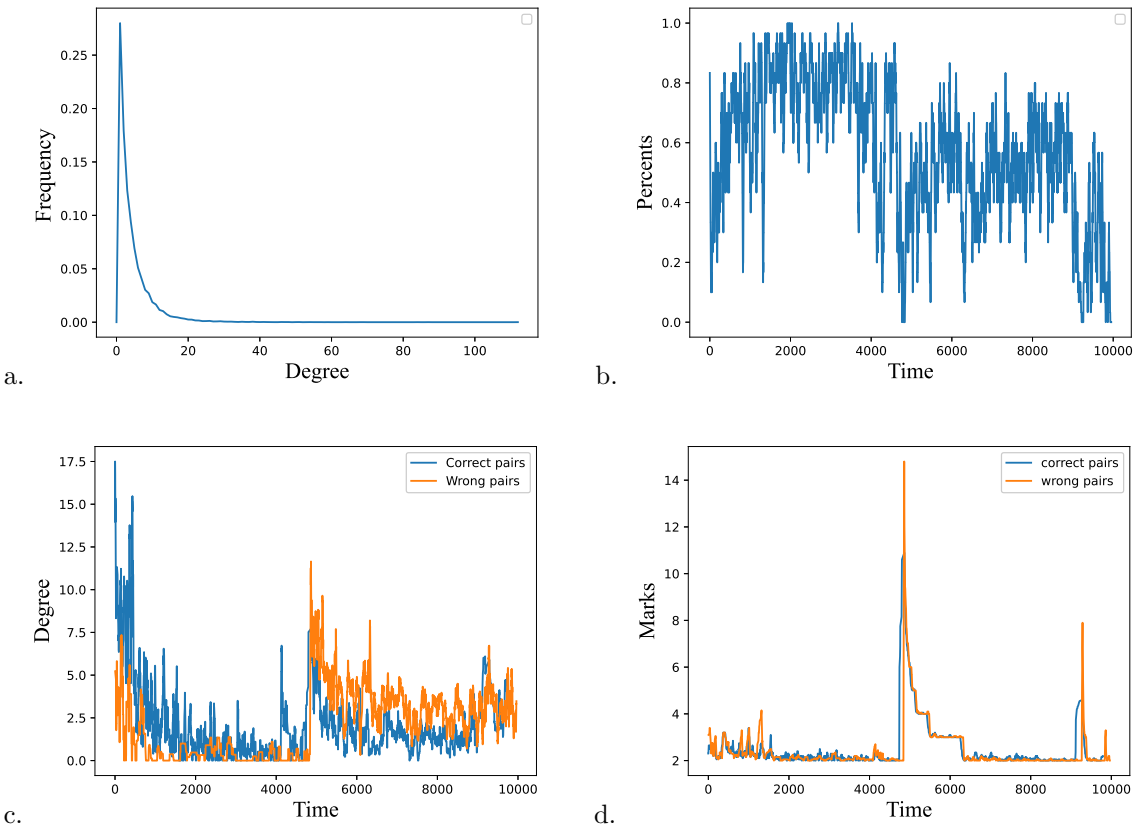


Figure 18: ExpandWhenStuck over Graph6 with graphs overlap (s) of 0.7 with $|seed| = 480$. Sub-figures b indicates much lower performances, caused both by many low-degree nodes (see sub-figure a) and the clustered nature of graphs with power-law degree distribution.

4.3 Parallel Version of ExpandWhenStuck

Although [9] reports that their parallel version can be run on graphs with millions of nodes without fundamentally affecting the performance of the algorithm, they do not present any information about the exact drop in performance. The parallel version splits each iteration into epochs, in which it adds to M every possible candidate pair without spreading out marks, and then spreads out marks simultaneously for the next epoch.

We reproduce the test from the previous section, alongside the results of the parallel version (Figures 20, Figures 19). The parallel version requires a larger seed in order to achieve similar results to the original version. Together with the former findings, we can conclude that EWS has three main difficulties:

1. Significantly lower performances on real-world graphs.
2. Difficulty to handle graphs-overlap lower than 0.7.
3. A simpler paralleled version is needed in order to process large graphs.

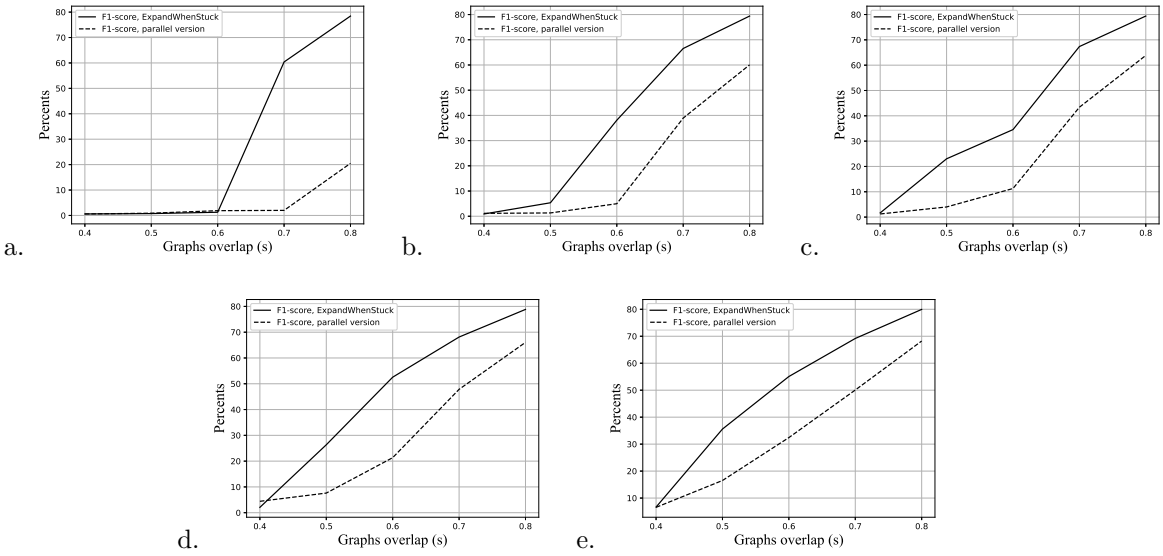


Figure 19: A comparison between EWS and its parallel version: F1-score of both versions over graph 2 as a function of seed size. Sub-figures a-e use increasing s values of $0.4, 0.5, 0.6, 0.7, 0.8$, respectively. One can note a significant drop in performances of the parallel version compared to EWS.

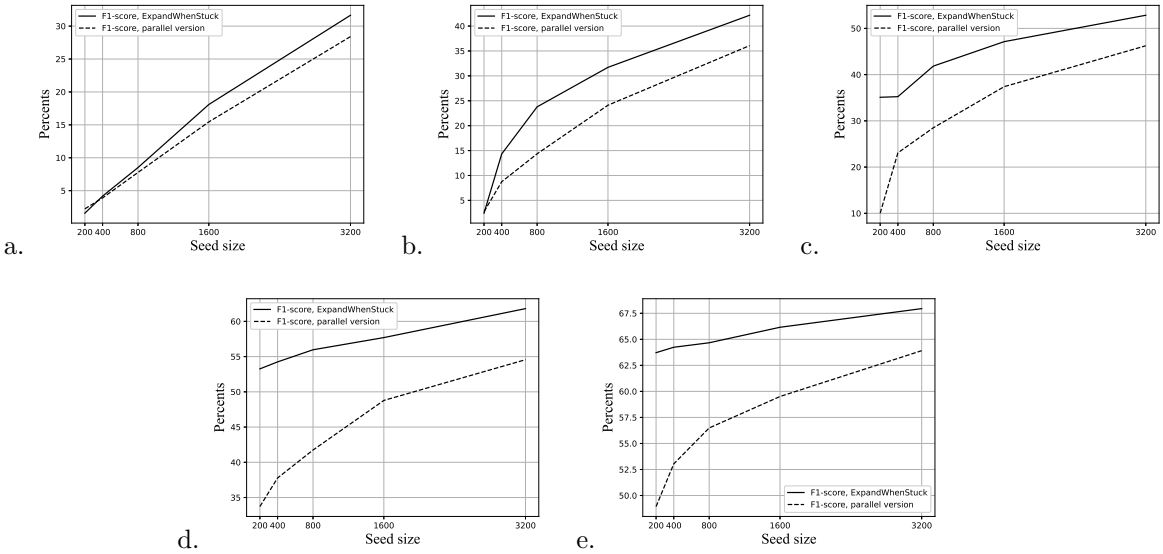


Figure 20: A comparison between EWS and its parallel version: F1-score of both versions over graph 1 as a function of graphs overlap (s). Sub-figures a-e use increasing seed size of $25, 50, 100, 200, 400$, respectively. One can note a significant drop in performances of the parallel version compared to EWS.

4.4 Simplified Version to Second-Run

As suggested in section 3.6, we aim to solve seeded GM by first run ExpandWhenStuck, and then perform a second iteration based on all the information that has been collected. To better justify our choice of $s\bar{o}re_t(p) = max(score_{2,t}(p), score_{1,\infty}(p))$ as the score function of the second iteration, we start with a simplified version in which the score function is only $score_{1,\infty}(p)$). Practically, it means that the second iteration is based on the amount of marks that each pair received during the first iteration alone.

The intuition to keep adding recommendations to mapped pairs and use them on a second run is simple. Since the percolation GM method is based on the assumption that correct pairs tend to receive more marks, we can base on this assumption all along the algorithm, not only until the moment of inserting the pair into M. In Figure 22 we emphasize the advantage of relying upon $score_\infty(p)$ at time t by comparing the number of marks that a pair p has when inserted into M, against the number of marks it has at $t = \infty$.

4.4.1 Proof of improvement

We formally prove the advantage of 'second run' over ExpandWhenStuck. For that purpose, we analyze the algorithm over the input of random graphs generated by $G(n, p, s)$. As argued by [9] when analyzing EWS, the property of generating noisy seed whenever it gets stuck is hard to analyze. For that reason, we analyze a simpler version of that algorithm where an artificial seed is not generated when percolation stops (see Figure 21). We also use a simplified version to Second-Run. Instead of

the function $sc\bar{o}re_t(p)$ defined by the maximum of two functions, in Theorem 1 we only prove the advantage of $score_{1,\infty}(p)$ over $score_{1,t}(p)$ for discovering matching pairs. Intuitively, if $score_{1,\infty}(p)$ proved to be better than $score_{1,t}(p)$, the marks received during the second iteration should be more accurate, improving $score_{2,t}(p)$, therefore improving $sc\bar{o}re_t(p) = max(score_{1,\infty}(p), score_{2,t}(p))$

Simplified ExpandWhenStuck

$A \leftarrow seed$ is the initial set of seed pairs, $M \leftarrow seed$;
$Z \leftarrow \emptyset$ is the set of used pairs
$MarkedPairs \leftarrow \emptyset$ is the set of all marked pairs along with their number of marks
for all pairs $[u, v] \in A$ **do**
    Add the pair [u,v] to Z and add one mark to all of its neighboring pairs;
end for
while there exists an unmatched pair with at least 2 marks in $MarkedPairs$ that does not conflict M **do**
    Among those pairs select the one maximizing $score(p)$;
    Add p=[i,j] to the set M;
    if [i,j] $\notin Z$ **then**
        Add one mark to all of its neighboring pairs and add the pair [u,v] to Z;
    end if
end whilereturn M, $MarkedPairs$;

Figure 21: A simplified version of ExpandWhenStuck. Whenever no pair with two marks exists, the algorithm does not generate an artificial seed and simply stops.

Theorem 1: Given $G_1, G_2 \leftarrow G(n, p, s)$, let $p' = [u, v']$ be a wrong pair inserted into M at time t and let $p = [u, v]$ be a right pair conflicting p'. Assuming at some time $\bar{t} > t$ a correct pair has been inserted into M, the following applies:

$$\mathbb{E}[score_\infty(p) - score_\infty(p')] > \mathbb{E}[score_t(p) - score_t(p')] \quad (8)$$

Since whenever we choose a wrong pair $[u, v']$ before the right pair $[u, v]$, we will eventually avoid the latter insertion of $[u, v]$, since it will conflict M. Theorem 1 suggests that using $score_\infty$ will reduce the probability of a mistake in the next iteration - eventually reducing the number of wrong pairs in M.

Proof of Theorem 1:
Eq. 8 in Theorem 1 is equivalent to:

$$\mathbb{E}[score_\infty(p) - score_t(p)] > \mathbb{E}[score_\infty(p') - score_t(p')] \quad (9)$$

In other words, the expected number of marks that p will get from now on, will be higher than the expected number of marks p' will receive. Let us denote by M_t the map at time t, we define $\Lambda(M_t)$ to be the number of right pairs in M_t and $\Psi(M_t)$ be the number of wrong pairs.
Let $p_{t'} = [\alpha, \beta]$ be a pair inserted to M at time $t' > t$:

- If $p_{t'}$ is a correct pair, α and β represent the same vertex $\gamma \in V$ (u and v represented by the same vertex $w \in V$ as well). $p = [u, v]$ gets one mark if there are edges $(\alpha, u) \in E_1$ and $(\beta, v) \in E_2$. This requires the edge (γ, w) to exist in E (which happens with probability p) and to be sampled in E_1, E_2 (happens with probability s^2). On the other hand, p' gets a mark if $(\alpha, u) \in E_1 and (\beta, v') \in G_2$. This requires two different edges to exist in E (probability of p^2) and to be sampled to E_1 and E_2 accordingly (probability of s^2).
- If $p_{t'}$ is a wrong pair, α and β are represented by the different vertices $\alpha', \beta' \in V$, (note that $p_{t'}$ cannot conflict p'). The pair p gets one mark if there are edges $(\alpha, u) \in E_1$ and $(\beta, v) \in E_2$. This happens with probability $p^2 s^2$. The pair p' gets one mark if there are edges $(\alpha, u) \in E_1$ and $(\beta, v') \in E_2$, which also happens with probability $p^2 s^2$. In fact, it is possible that one of the pairs $p_{t'}$ will have the form $[k, v]$. In that case, p' gets a mark with probability $p^2 s^2$ while p gets none, as $[u, v']$ can never be a neighboring pair of $[k, v']$.

Let us denote $L_t = M_\infty - M_t$, using the analysis above, one can compute:

$$\mathbb{E}[score_\infty(p) - score_t(p)] \geq \Lambda(L_t) * s^2 p + (\Psi(L_t) - 1) * s^2 p^2 \quad (10)$$

$$\mathbb{E}[score_\infty(p') - score_t(p')] = \Lambda(L_t) * s^2 p^2 + \Psi(L_t) * s^2 p^2. \quad (11)$$

Combining Eq. 10 and 11, we obtain:

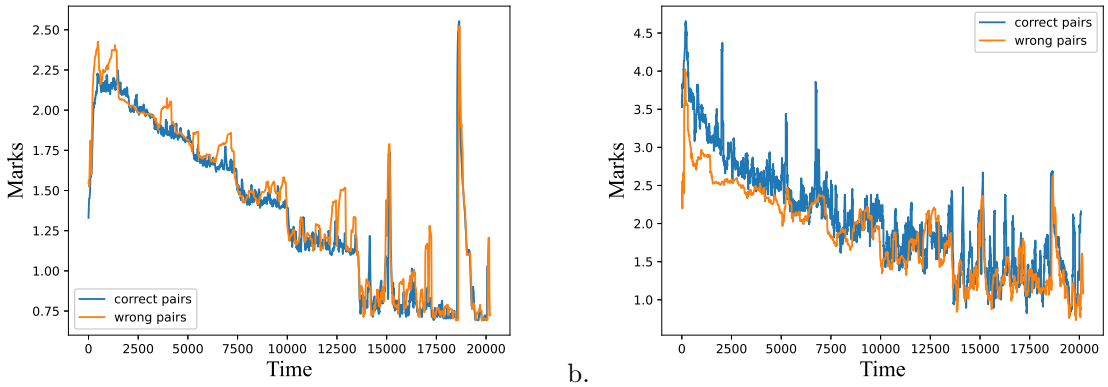
$$\mathbb{E}[score_\infty(p) - score_t(p)] \geq \Lambda(L_t) * s^2 p(1-p) - s^2 p^2 + \mathbb{E}[score_\infty(p') - score_t(p')] \quad (12)$$

To prove Eq. 9, it remains to show that $\Lambda(L_t) * s^2 p(1-p) - s^2 p^2 > 0$. Since we assumed that some correct pair has been inserted to M at time $\bar{t} > t$, we have $\Lambda(L_t) \geq 1$. p is the probability of an edge to exist, and as we only focus on sparse graphs we assume $p << 0.5$, leading to:

$$\begin{aligned} &\Lambda(L_t) * s^2 p(1-p) - s^2 p^2 \\ &= (\Lambda(L_t) - 1) * s^2 p(1-p) + s^2 p(1-p) - s^2 p^2 \\ &= (\Lambda(L_t) - 1) * s^2 p(1-p) + s^2 p(1-2p) > 0 \end{aligned} \quad (13)$$

□

ExpandWhenStuck over Graph 6, s=0.7, $|seed| = 480$



a. b.

Figure 22: On the left, we present the number of marks at the moment of insertion into M during ExpandWhenStuck, Axis X represent time. On the right, are the the same pairs (keeping the same order) but presenting the number of marks they had at the end of the algorithm. The tendency of correct pairs to keep gaining marks after been matched is clear. (Both graphs having been smoothed by sliding window of 50 for clarity.)

4.5 Second-Run's Results

As a reminder, $marks_{2,t}(p)$ refers to the number of marks gained by pair p during iteration 2 until time t. In the Second-Run algorithm we start a second iteration after running ExpandWhenStuck, building a new map M_2 base on the function $s\bar{o}re_t(p) = max(score_{2,t}(p), score_{1,\infty}(p))$. This way we aim to earn from the advantage of both functions: $score_{1,\infty}(p)$ is based on more matched pairs, while $score_{2,t}(p)$ is based on potentially more accurate matches.

We applied both of the algorithms ten times on each graph with a fixed $s = 0.6$, the mean and standard deviation are presented in Table 3 below. Then, we performed a one-side t-test to test the null hypothesis that ExpandWhenStuck distribution's median is higher than ours, the p-values also appear in the table.

		graph1	graph2	graph3	graph4	graph5	graph6
F1-score mean	ExpandWhenStuck	51.82	48.96	46.38	61.32	61.02	38.19
	Second-Run	56.33	50.43	47.45	64.6	61.77	40.62
F1-score std	ExpandWhenStuck	1.44	0.34	0.51	0.5	0.62	1.92
	Second-Run	0.97	0.3	0.39	0.57	0.5	1.86
p-value		$4 * 10^{-7}$	$8 * 10^{-9}$	$6 * 10^{-5}$	$9 * 10^{-11}$	$6 * 10^{-3}$	$7 * 10^{-3}$

Table 3: One-side t-test: We applied ExpandWhenStuck and Second-Run ten times on each graph with a fixed $s = 0.6$. Then a one-side t-test was performed to test the null hypothesis that ExpandWhenStuck distribution's median is higher than ours.

We performed a comparison between ExpandWhenStuck, Simplified-Second-Run (uses only $score_{1,\infty}(p)$) and Second-Run (Figures 23 and 24). The comparison represents precision, recall and F1-score of the algorithm, each figure use two graphs and a fixed graphs-overlap (s). One can notice a consist improvement from ExpandWhenStuck to Second-Run in all indices. The second observation is that while Second-Run has a better recall and F1-score than those of Simplified-Second-Run, they have the same precision values. We suggest here that Simplified-Second-Run has a limited effect on the recall since it does not enlarge the pool of candidate pairs. We thus proposed to enlarge the candidate pairs by spreading out marks during the second iteration, as Second-Run does using $sc\bar{o}re_t(p)$.

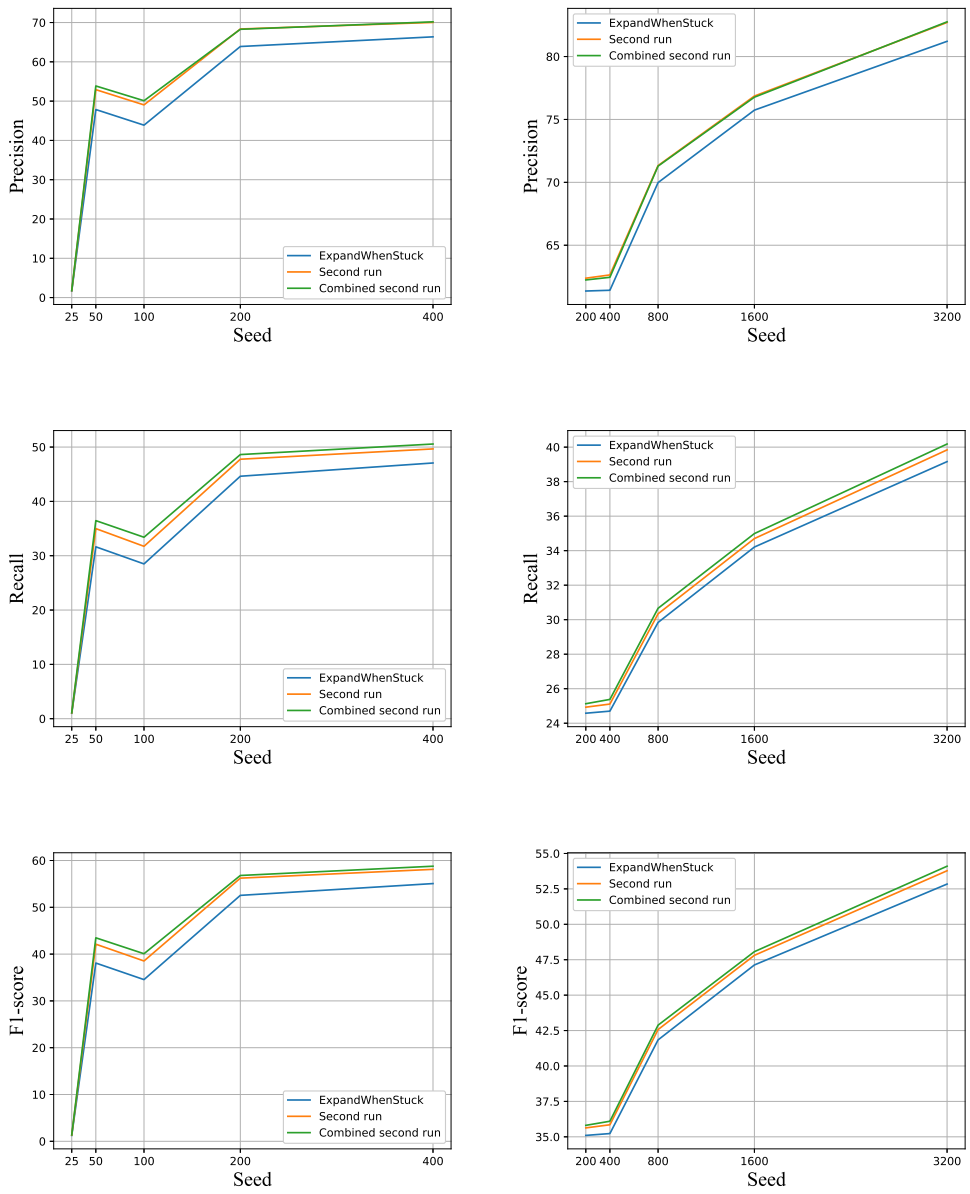


Figure 23: A comparison of precision, recall and F1-score between ExpandWhenStuck (uses $score_{1,t}$), Simplified-Second-Run (uses $score_{0,\infty}$) and Second-Run (uses $s\bar{c}ore_t$). The sub-plots on the left are based on graph 1, and on the right are based on graph 2, both use graphs-overlap (s) of 0.6. Besides the clear improvement over ExpandWhenStuck, Second-Run consistently has higher recall than Simplified-Second-Run.

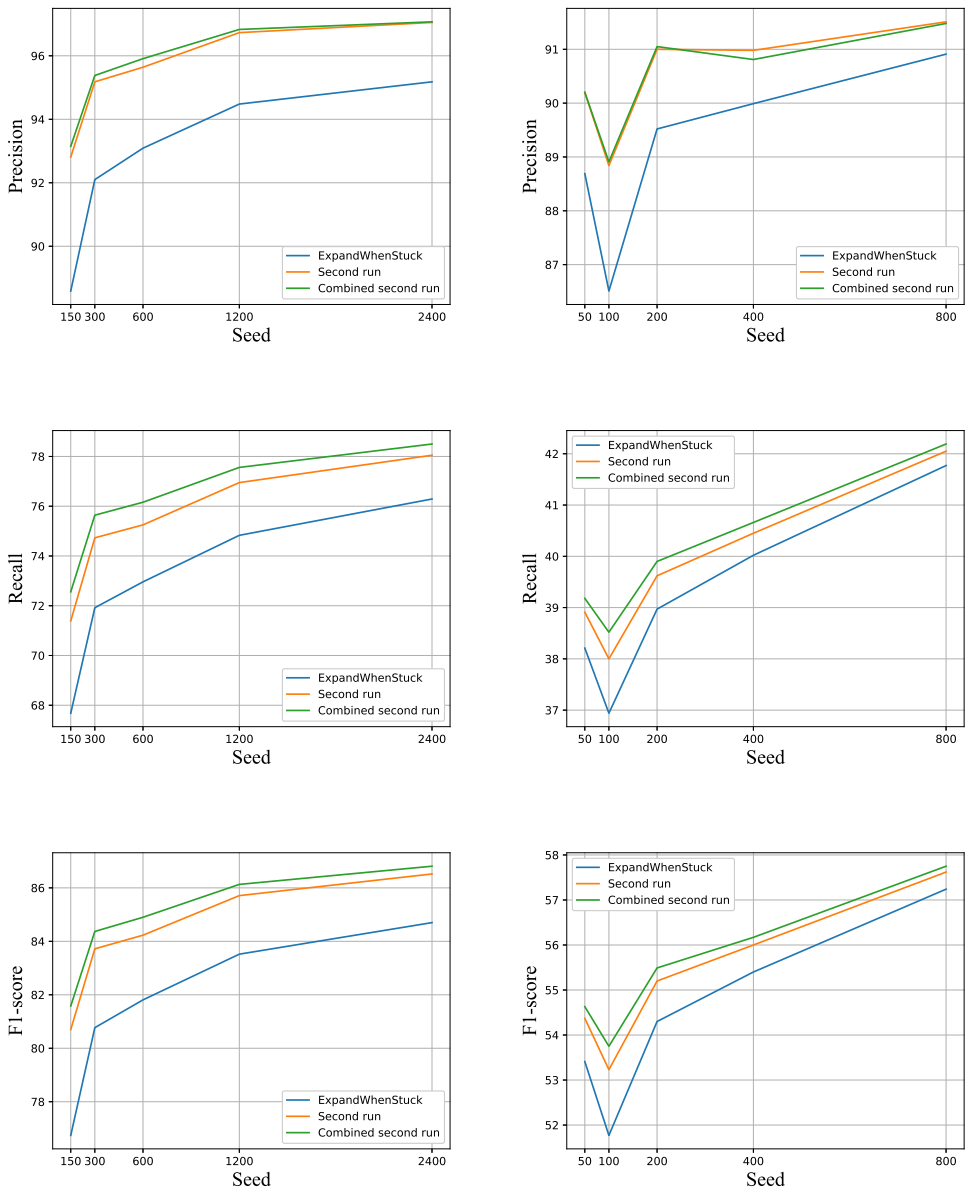


Figure 24: A comparison of precision, recall and F1-score between ExpandWhenStuck (uses $score_{1,t}$), Simplified-Second-Run (uses $score_{0,\infty}$) and Second-Run (uses $s\bar{c}ore_t$). The sub-plots on the left are based on graph 3, and on the right are based on graph 4, both use graphs-overlap (s) of 0.7. Besides the clear improvement over ExpandWhenStuck, Second-Run consistently has higher recall than Simplified-Second-Run.

4.6 Performance of IRMA

The extension from Second-Run to IRMA is very natural. While Second-Run starts a second iteration to build M_2 based on $s\bar{c}ore_{2,t}(p) = max(score_{2,t}(p), score_{1,\infty}(p)$, IRMA builds M_i during the i-th iteration using $s\bar{c}ore_{i,t}(p) = max(score_{i,t}(p), score_{i-1,\infty}(p)$. We already proved the advantage of Second-Run over EWS, so $score_{2,\infty}(p)$ is more accurate than $score_{1,\infty}(p)$, allowing $score_{3,t}(p)$ to be more accurate than $score_{2,t}(p)$ and therefore for $s\bar{c}ore_{3,t}(p)$ to be more accurate than $s\bar{c}ore_{2,t}(p)$. This way we can use induction to argue that M_i is improving from iteration to iteration until convergence.

In Figure 25, each sub-figure consists of several runs of IRMA with a different size of seeds, and with a fixed graph and graphs-overlap (s). For example, in sub-figure a, where we used graph 1 and $s = 0.6$, any seed size is a different run of the algorithm. For any seed size, any line passing above represents another iteration of the same run, gradually converging from EWS to IRMA (the blue line). The first, most noticeable observation is that IRMA indeed gradually repairs the output map of EWS. In fact, the F1-score results are a monotonic non-decreasing function in the number of iterations. Second, the number of iterations is low enough to be negligible from the perspective of run-time (multiplication by small constant). Finally, and most interesting, is that IRMA is less sensitive to the randomness of the process. EWS often achieves lower performances on a bigger seed due to the randomness in the process of generating the graphs and choosing the seed (see sub-figures a,d,e for such cases). Despite accepting EWS's output as an initial map, IRMA exhibits a dependency on the problem properties (as the input graph and the graphs-overlap) rather than the output of EWS, and therefore has much smoother plots.

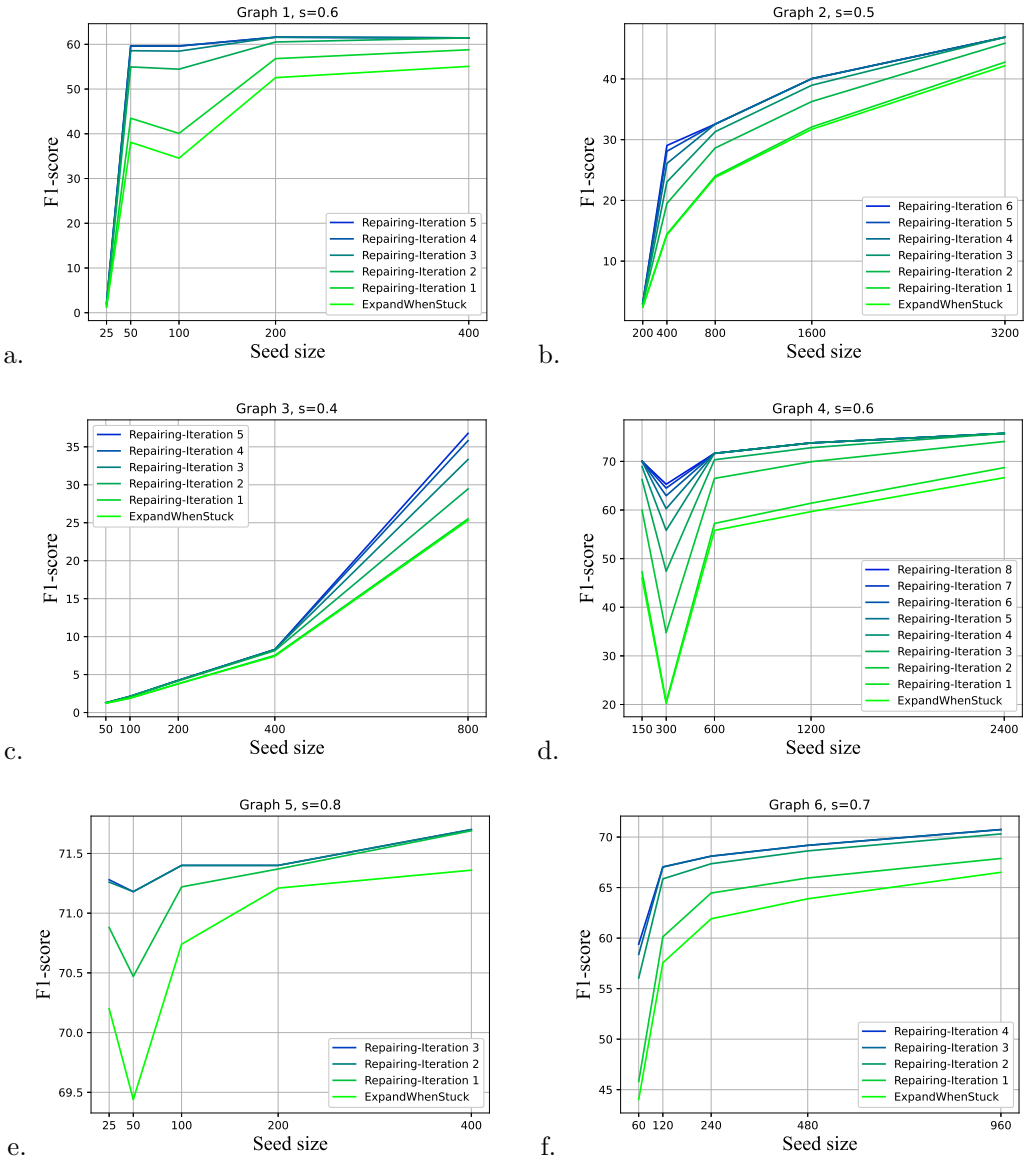


Figure 25: Performances of IRMA: Each sub-figure consists of several runs of IRMA with a different size of seeds with a fixed graph and graphs-overlap (s). We show how the iterations gradually converge from ExpandWhenStuck to IRMA (the blue line).

4.6.1 Stopping condition

In the previous section, we explained why IRMA's iterations improve from M_i to M_{i+1} and supported this claim with relevant figures. We now aim to justify our choice of $weight(M_i)$ as a

quality measure for M_i, and hence as a stopping condition for the algorithm. We recall that $weight(M)$ is defined as $|\{[u,v]|[u,v] \in G_1, [M(u), M(v)] \in G_2\}|$, that is, the number of edges in the common subgraph induced by the M. Since correct pairs share more common neighbors in expectation, we expect better maps to have higher $weight(M)$. In the opposite direction, if $weight(M_i) < (1+\delta)weight(M_{i+1})$, we can assume that M_{i+1} is not significantly better than M_i. We run IRMA and compare $weight(M_i)$ to the F1-score of M_i to test the connection between them (Figure 26). One can note that both indices are converging simultaneously, making $weight(M)$ a perfect halt condition.

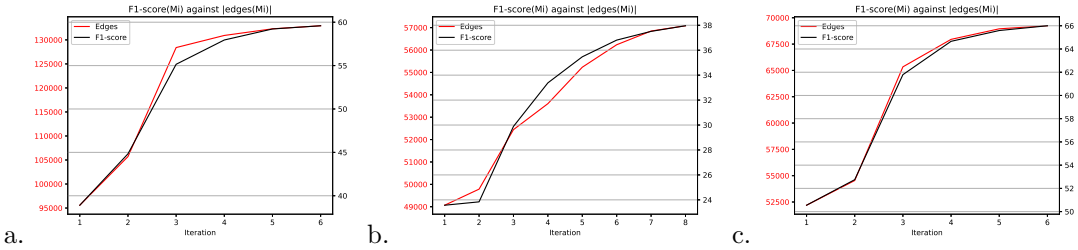


Figure 26: Each sub-figure represents $weight(M)$ and F1-score of a single run of IRMA along the iterations. One can note that both indices are converging simultaneously, making $weight(M)$ a perfect halt condition.

4.6.2 Computational Complexity of IRMA

For a pair $[u,v] \in M$, the number of marks to spread out is $d_1(u) * d_2(v)$ where $d_i(u)$ is the degree of $u \in G_i$. Thus, the number of updates to the priority queue from inserting new pairs to M is $N = \sum_{[u,v] \in M} d_1(u) * d_2(v)$. The cost of each insertion is proportional to the log of the size of the priority-queue, which is bound $O(|V|^2)$, so the cost of each insertion is bound by $O(Log(|V|)$, where $|V|$ is the number of vertices. The total number of updates (N) is between $N = O(|V| * E[d]^2)$ and $N = O(|V| * E[d^2])$, where d is the degree. The first case is for random matching (i.e., $d(u)$ and $d(v)$ are independent), and the second case is for perfect matching $d(u) = d(v)$. In low variance degree distributions, both cases are equivalent. However, in power-low degree distributions, the second case may be much higher than the first. Thus, one can bound the cost of the algorithm by $O(|V| * log(|V|) * E[d^2]$, but often the bound is tighter - $O(|E|^2 * log(|V|)/|V|)$. The Repairing-Iteration has a similar cost, since it is based on the same logic of pulling pairs from the priority queue and spreading out marks. Note that we avoid the problematic case in EWS that the expanded seed may be of $O(|V|^2)$, which adds a V factor to the cost of spreading marks. In their parallel versions, the boundary is similar, but in practice the Repairing-Iteration takes about $30\% - 60\%$ the run time of EWS, and as IRMA runs Repairing-Iteration a few times, it has an overall run time 2-3 times higher than EWS, but of the same order.

4.7 Expansion to Low Degree Vertices

As explained in the introduction, the Seeded Graph Matching problem is a variant of the Maximum Common Edges Subgraph (MCES) problem. However, while in MCES, one tries to find the common subgraph that contains the maximal possible number of edges, in our problem the target

is to use the edge match as a tool to discover an a priori existing match. Practically, in the MCES it is always beneficial to add more vertices to the map - no matter how wrong their map will be. However, in seeded GM, one tries to maximize both precision and recall. Hence, at some stage, IRMA does not add pairs that with high probability are wrong, since those will lower the precision.

While running both ExpandWhenStuck and the repairing iterations, we use a threshold of two marks to add a candidate pair into M. This threshold has been set as a trade-off to insert only reliable pairs, yet allow percolation to flow. But since we can use the Repairing-Iteration method to filter out wrong matched pairs, it is possible to allow the algorithm to mistake in some cases in return to find new correct pairs. For that reason, we suggest to perform one repairing iteration with a threshold of one mark, after IRMA has been converged. Then, we run regular repairing iterations again to gradually restore the precision.

As explained above in sub-section 3.8.1, now that repairing iterations are used to filter out a massive amount of wrong pairs, $weight(M_i)$ is no longer a good stop condition. We thus examined three possible break conditions to the second stage of IRMA:

1. Let $|M_i|$ be be number of pairs in M_i, where M_i is the set of matched pairs at the end of the i iteration. Similarly, let $Weight(M_i)$ be the total number of edges among members of M_i present in both $G1$ and $G2$. The quality of a bijection can be defined through $Weight(M_i)/|M_i|$.

2. As the target is to scale M down, $|M|$ might indicate if the second stage has ended. We can use the condition $|M_i| \geq (1-\delta) * |M_{i-1}|$ to avoid many redundant iterations.

3. Empirically after a few iterations there is no significant improvement from M_i to M_{i+1}. A simple solution might be to use a constant number of iterations.

We computed the difference in accuracy along snapshots vs these three different measures (Figure 27). Sub-plots a and b represent the difference in precision as a function of the difference in $|weight(M_i)|/|M_i|$ and $|M_i|$, respectively. A clear correlation between the measures can be seen in sub-plots A and B, yet the variance around the origin (i.e., when IRMA converges) is relatively large. Instead, sub-plot C indicates that the most reliable prediction to the convergence of precision is the number of iteration. We thus propose that as the default stopping criteria after the expansion iteration.

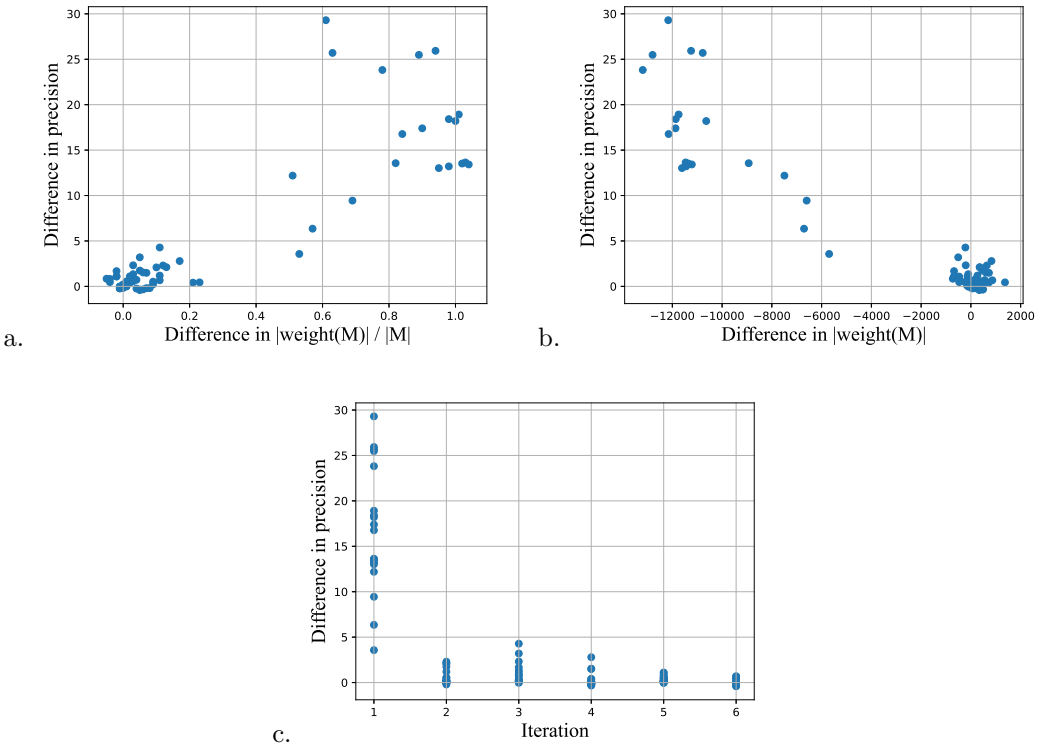


Figure 27: Difference in precision as a function of three indicate. We run IRMA with the exploring iteration multiple times over different graphs, seed-size and graphs-overlap. The scatters indicates that four repairing iterations are enough for the precision to converge.

We used these additional iterations on the runs of IRMA from the previous section and received a small but consistent improvement over IRMA, that is typically bigger on relatively small seeds (except for cases when IRMA failed). We tested the precision, recall and F1-score along IRMA's iterations on different graphs (Figure 28). As follows from Eq. **??**, the precision is always higher than recall, and the relation between them is equal to the ratio between R and M. Both recall and precision are monotonic non decreasing up to the expansion/exploration iteration. The drop in precision together with the pick in recall are caused by the exploration iteration, inserting many pairs with low level of certainty. After that, the Repairing-Iteration method is activated again to restore precision, and successfully does so in a few iterations. Note that while precision is fully restored (and some times even improves), the recall stays higher than before the exploration iteration, as some of the correct new matches lead to an unexplored part of the graphs and gain enough marks for later iterations.

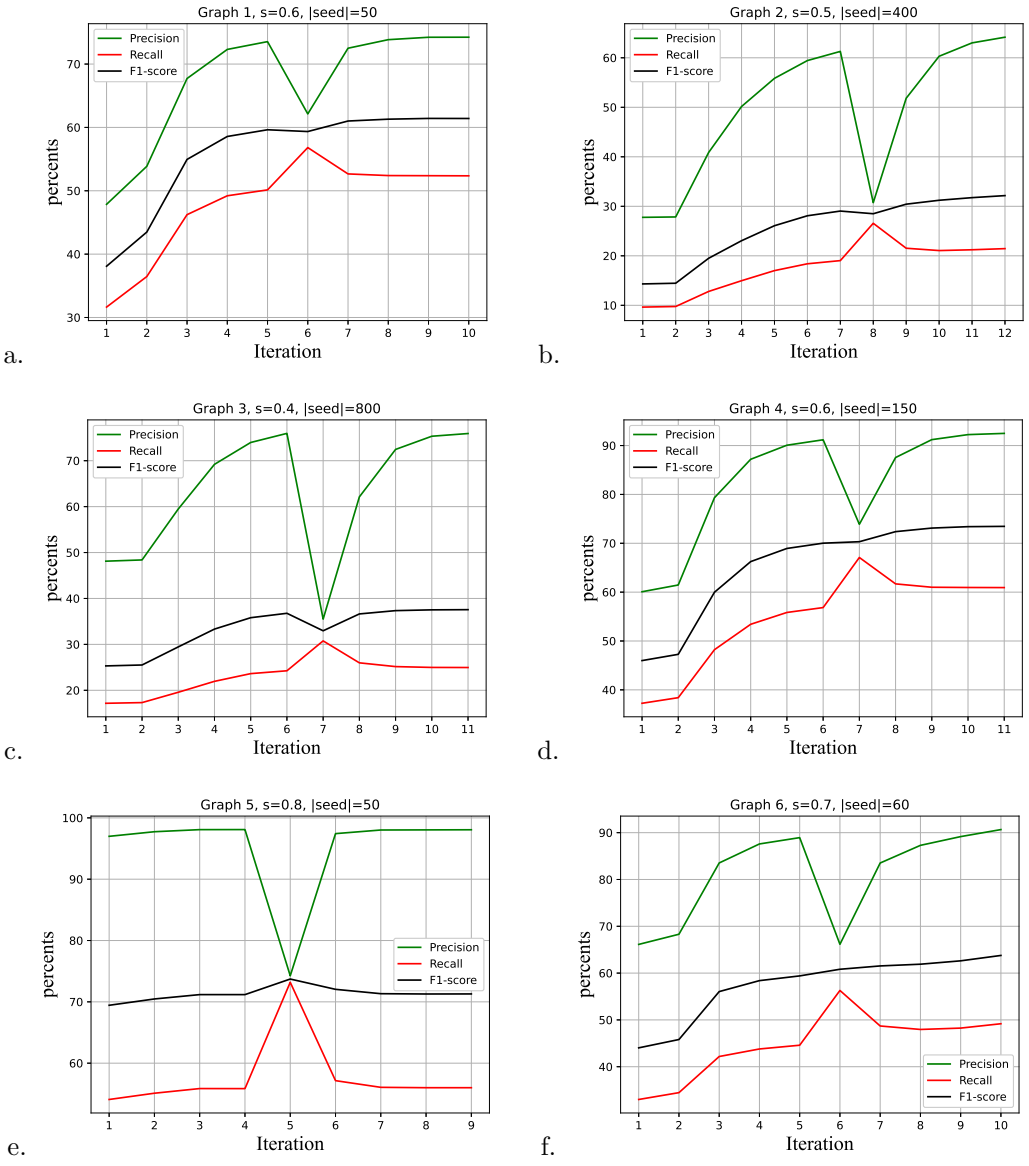


Figure 28: Precision, recall and F1-score during the algorithm's iterations : Each sub-figure consists of a single run of IRMA with the exploring iteration. The drop in precision is caused by the exploring iteration, after which the precision is fully restored while recall stays higher than before.

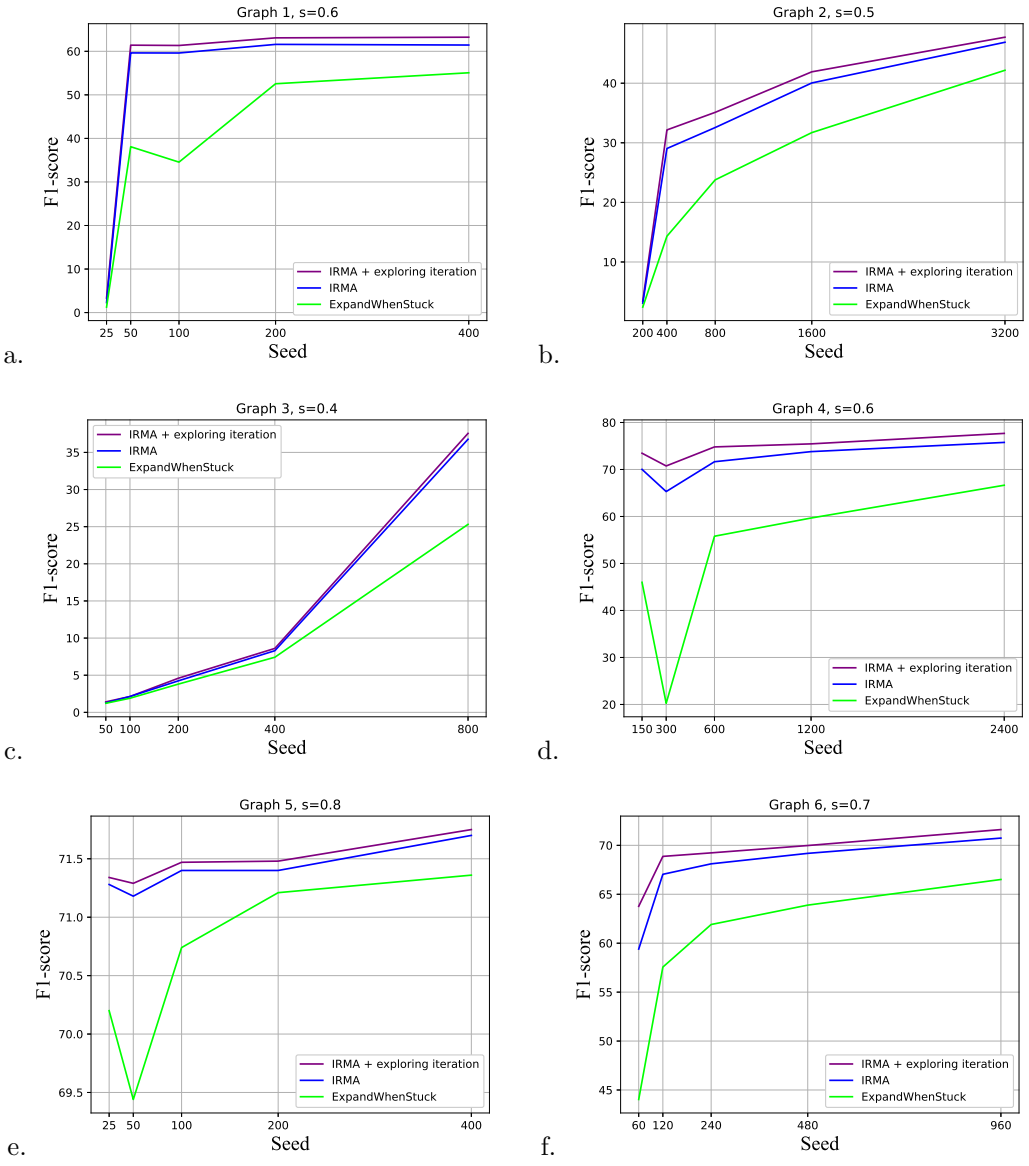


Figure 29: Exploring iteration: We run an exploring iteration (together with repairing iterations) after each run of IRMA. Each sub-figure consists of several runs of the algorithm with a different size of seeds and with a fixed graph and graphs-overlap (s). The exploring iteration consistently improves the performances over IRMA.

4.8 Parallel solution

In section 3.9 we introduced the parallel version to EWS using epochs. The main idea is to split the algorithm into epochs such that marks are spread out only between the epochs, allowing to use parallelism. We therefore also developed a parallel version to Repairing-Iteration in which M is rebuilt based only on the marks of the former iteration. The parallel version of IRMA then starts by running Parallel-EWS and then perform iterations of Parallel-Repairing-Iteration.

A comparison is made in Figure 30 between parallel-EWS, Parallel-IRMA and regular IRMA (both use exploring iteration). Parallel-IRMA has much higher performances than Parallel-EWS. In subfigure e we have an extreme case where Parallel-IRMA with 25 seeds is higher than Parallel-EWS with 400. In many cases Parallel-IRMA has a very close result to standard IRMA, making it an excellent trade-off between run-time and performances. Presenting those results, we recall that (parallel) IRMA has the same run time as (parallel) EWS up to multiplication by a small constant.

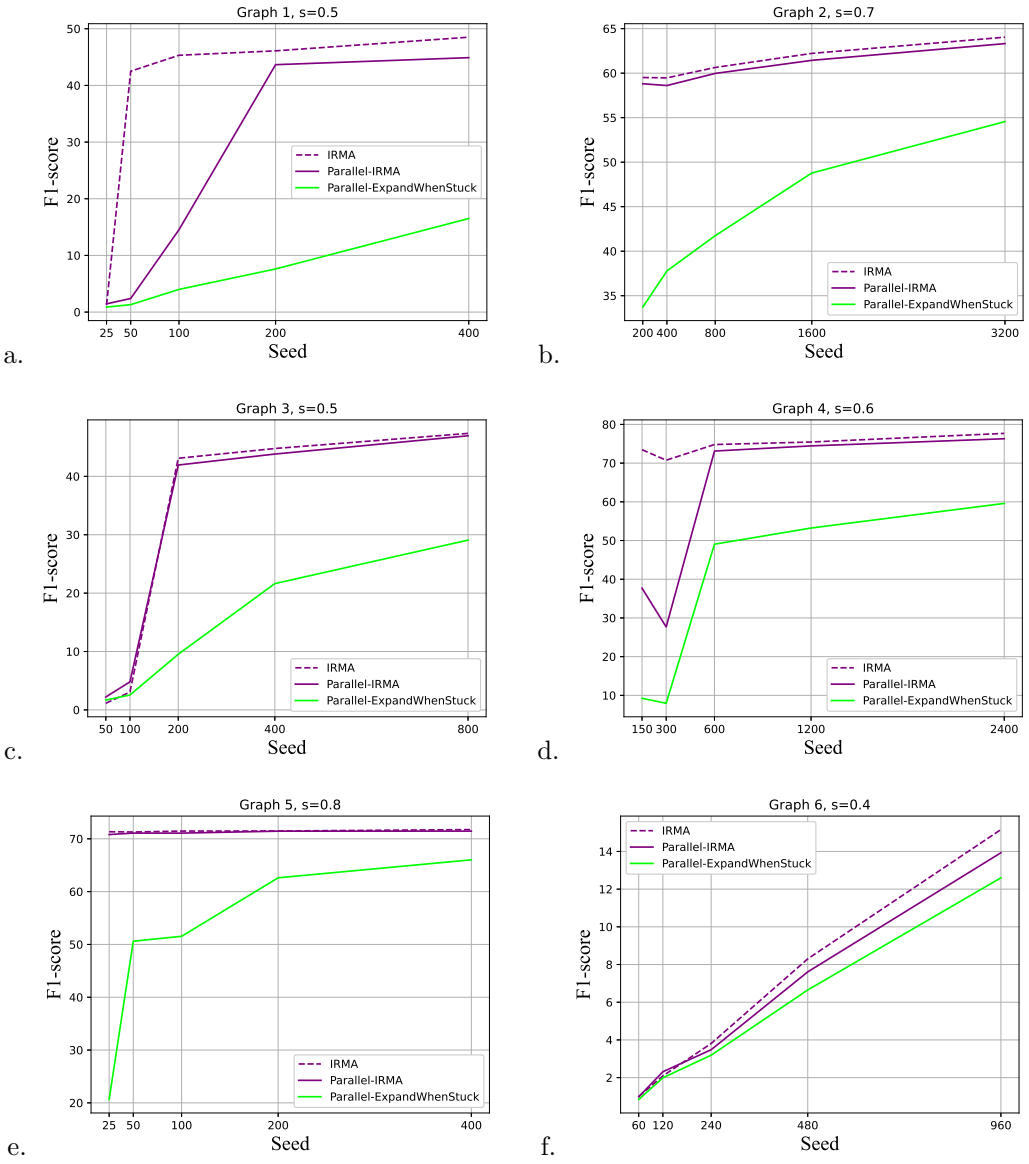


Figure 30: A comparison between parallel-ExpandWhenStuck, Parallel-IRMA and regular IRMA (both use an exploring iteration).

IRMA: Iterative Repair for graph MAtching

Barak Babayov
Bar-Ilan University
Ramat-Gan, Israel
babayov6@gmail.com

Prof. Yoram Louzoun
Bar-Ilan University
Ramat-Gan, Israel
louzouy@math.biu.ac.il

ABSTRACT

The alignment of two similar graphs from different domains is a well studied problem. In many practical usages, there are no reliable labels over the vertices, leaving structural similarity as the only information available to match such a graph. To simplify the matching, one often assumes a small amount of already aligned vertices - called a seed. The current state-of-the-art scalable seeded alignment algorithm is based on percolation. Namely, aligned vertices are used to align their neighbors and gradually percolate in parallel in both graphs. The 'ExpandWhenStuck' algorithm improves former percolation algorithms by generating an inaccurate artificial seed whenever the percolation is stuck, leading to better results using smaller seeds in Erdos Renyi graphs.

However, percolation based graph alignment algorithm are still limited in scale free degree distributions. We here propose 'IRMA' - Iterative Repair for graph MAtching to show that the 'ExpandWhenStuck' can be extended to high performance on real-world graphs with a limited additional computational cost. IRMA starts by creating a primary alignment using 'ExpandWhenStuck', then it iteratively repairs the mistakes in the previous alignment steps.

PVLDB Reference Format:
Barak Babayov and Prof. Yoram Louzoun. IRMA: Iterative Repair for graph MAtching. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://gitlab.com/babayov6/seeded-graph-matching.

1 INTRODUCTION

1.1 Graph Matching

In graph matching (GM), one is given two graphs G_1 and G_2 known to model the same data (i.e., there is an equivalence between the graphs vertices). For example, G_1 may be the friendship graph from the Facebook social network and G_2 the friendship graph from the Twitter social network for the same people. In both cases, the vertices are users and there is an edge between two vertices if the corresponding users are friends in the relevant social network. We assume that a friend in one network has a higher than random probability of being a friend in the second network.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

The goal of GM is to create a bijection $M : V_1 \rightarrow V_2$, such that M maps vertices in V_1 to a vertex in V_2 if they represent the same real-world entities. In the example above, M should connect profiles in Facebook and Twitter that belong to the same person. We note by R the ground-truth, i.e., the set of all pairs of vertices that represent the same entity in both of the graphs. Given a bijection M, if $M(v_1) = v_2$, and $M(v_3) = v_4$, and the edge $(v1, v3) \in E_1$, and $(v2, v4) \in E_2$, the common edge will be defined as a shared edge. The quality measure for the quality of M is usually the *number* of shared edges.

Finding a full bijection is not always optimal, since some vertices may be absent from one of the two graphs. We thus look for a partial bijection: $M : \tilde{V_1} \subset V_1 \rightarrow \tilde{V_2} \subset V_2$, such that the *fraction* of shared edges is maximal. A single edge bijection is obviously a simple solution to that. Thus, a trade-off between the number of shared edges and the fraction of mapped vertices is often required, by adding a constraint on the number of elements in $\tilde{V_1}$.

1.2 Seeded Graph Matching

In the presence of limited initial information on the bijection, one can use seeded GM. In seeded GM, the input contains beyond (G_1, G_2) also a small $seed \subset V_1 \times V_2$, which is a group of vertices pairs known to represent the same real-world entities in G_1 and in G_2. A similar problem emerges when the vertices have additional information named labels or meta-data [16, 19]. For example, the users on Facebook and Twitter may have additional attributes, such as age, address, and user names. It is obvious that a user named "Bob Marley" in Facebook is much more likely to represent the same person as a user with the same name on Twitter than a user named "Will Smith". For that reason, labeled vertices are significantly simplifying the matching problem, and pairs of vertices with highly similar attributes can be used as a seed. Here, we focus on seeded GM based, with no labels over the vertices. We propose an iterative approach to seeded GM.

2 RELATED WORK

2.1 Graph Matching Solvers

GM is used in different disciplines. In social networking, GM can be used for the de-anonymization of data sets from different domains [14, 27]. GM is used on proteins from different species in biology to detect functional equivalences [13, 23], and to discover a resemblance between images in computer vision [5, 25, 26].

The main progress in GM algorithms is based on machine–learning methods, and can be broadly divided into two main categories: profile-based and network-based methods. Profile-based methods rely on the vertices meta-data (e.g., username [28], spatio-temporal

patterns [21], posts [9], or writing style [18], etc.) to link accounts across different sites. Network-based methods rely on the graph topological structure [15, 24, 29]. In general, machine learning based methods are characterized by a high run time, in both training and deployment. Multiple theoretical bounds for a GM solution were proposed [2, 3, 12, 22]. They make use of several parameters such as seed size, degree distribution, graphs-overlap, etc. Polynomial run-time complexity algorithms were developed that in exchange for scalability may achieve a good alignment only under certain assumptions [4, 7, 8, 17].

An important aspect of GM is scaling, since its practical use is often in large graphs. Current state-of-the-art scalable seeded GM methods are based on gradual percolation, starting from the seed and expanding through common neighbors. This class of algorithms is referred to as Percolation Graph Matching (PGM) methods [10, 14, 27]. Despite having in some cases additional information in the form of labels, [10] showed the crucial importance of relying on edges during the process of GM. Both [1] and [11] present an improvement to [27] and are currently state-of-the-art PGM algorithms. Here we focus on [11] and improve it by presenting our Iterative Repair for graph MAtching (IRMA) algorithm.

For convenience, we follow here the notations of ExpandWhenStuck (further denoted EWS) [11] with minor changes (see Table 1 a list of notations).

In the following text, we refer to the input graphs as $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. When we mention a pair $[u, v]$ or p, we refer to a pair of vertices $[u, v] \in V_1 \times V_2$, with no explicit mention. We denote the pairs $[u, v]$, $[u', v']$ as neighboring pairs if $(u, u') \in E_1$ and $(v, v') \in E_2$. Finally, a pair $[u, v]$ conflicts M if it conflicts an existing pair in $[u', v'] \in M$ (i.e., $u = u'$ and $v \neq v'$ or vice versa).

In PGM algorithms, one maintains a score for each candidate pair, and uses the score to gradually build the set of the matched pairs - M. Adding a constant value to the score (also called the mark) of all neighboring pairs of some given pair is called 'spreading out marks'. $marks_t(p)$ of the pair p is defined as the number of marks p received from other pairs until time t. As there is a high chance for two pairs to have the same amount of marks, we define $score_t([u, v])$ to give a priority to pairs with similar degree:

$$score_t([u, v]) = marks_t([u, v]) - \epsilon * |d_{1,u} - d_{2,v}|, \qquad (1)$$

for an infinitesimal $\epsilon > 0$, where $d_{q,v}$ is the degree of a vertex v in graph q. The second term is to solve matches using the degree. In a nutshell, EWS (see code in Figure 1) starts by adding all the seed pairs into M, and spreading out marks to all of their neighbors. Then, at each time step t, EWS chooses a candidate pair $p' = argmax_p(score_t(p))$ that does not conflict M, adds it to M and spreads out marks to all of its neighbors (see Figure 2). When there are no pairs left with more than one mark (line 15 in Figure 1), EWS creates an artificial noisy seed (A), and uses it to further spread out marks (line 6 in Figure 1). A contains all pairs that are: 1) neighbors of matched pairs 2) do not conflict M 3) never have been used to spread out marks. The novelty of EWS is the generation of an artificial seed whenever there are no more pairs with more than one mark. The artificial seed is mostly wrong. Yet, EWS manages to use it to match new correct pairs and continue the percolation.

Pair	A pair $[u, v]$, is $[u, v] \in V_1 \times V_2$. Sometimes we refer to a pair p without specifying that it is in $V_1 \times V_2$.
Neighboring pair	Given the graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the pairs $[u, v]$, $[u', v'] \in V_1 \times V_2$ are neighboring pairs, if there are edges $(u, u') \in E_1$ and $(v, v') \in E_2$.
Spreading out marks	In the description of the matching algorithms, we refer to a pair p spreading out marks as adding one mark to each neighboring pair of p.
$MarkedPairs$	Contains marks-counters for all marked pairs.
$Marks_t(p)$	Number of marks pair p received from other pairs until time t. $score_t(p)$ is defined accordingly (Eq. 1).
$Marks_{i,t}(p)$	Number of marks pair p received from other pairs during the i-th iteration until time t. $score_{i,t}(p)$ is defined accordingly (Eq. 5).
$\bar{Marks}_{i,t}(p)$	$\bar{Marks}_{i,t}(p) = max(marks_{i,t}(p), marks_{i-1,\infty}(p))$. $\bar{score}_{i,t}(p)$ is defined accordingly (Eq. 7).
R	The ground-truth, i.e., the set of all pairs of vertices that represent the same entity in both of the graphs.
$Seed$	$Seed \subset R$ is a small group of pairs given as input and known to contain only correct matches.
A	A set of pairs to spread out marks from. Used in EWS, initialized to $seed$.
Z	A set of pairs that already spread out marks. Used in EWS to prevent repetitions.
M	Set of all pairs matched by the algorithm. We refer to any pair in M as a 'matched pair' and any other pair as a candidate.
Pairs conflict	We say that pairs $[u', v']$, $[u, v] \in V_1 \times V_2$ conflict if $u = u'$ and $v \neq v'$ or vice versa. If a candidate pair p conflicts a matched pair p', we say that p conflicts M.
$Weight(M)$	$\lvert\{(u, v) \lvert (u, v) \in E_1, (M(u), M(v)) \in E_2\}\rvert$
s	$s \in [0, 1]$ - the probability that an edge should be sampled in V_1 or in V_2

Table 1: Main notations

At the end of EWS, the set of mapped pairs M is returned along with $MarkedPairs$ that contains counters of marks for all marked pairs. The last is not needed in EWS, but is used in IRMA. The main contribution of EWS is a dramatic reduction in the size of the required seed set for random $G(n, p)$ networks (graph with n vertices and a probability of p for each edge).

3 METHODS

3.1 Evaluation Methods and Stopping Criteria

We use precision and recall to evaluate the performance of algorithms: (i) Precision refers to the fraction of errors in the set of matched vertices (i.e., pairs in M that are in R), and (ii) Recall is the size of the intersect of M and R out of the size of R as

$$Precision = \frac{\Lambda(M)}{|M|}, Recall = \frac{\Lambda(M)}{|R|}, \qquad (2)$$

ExpandWhenStuck

$A \leftarrow seed$ is the initial set of seed pairs, $M \leftarrow seed$;
$Z \leftarrow \emptyset$ is the set of used pairs
$MarkedPairs \leftarrow \emptyset$ is the set of all marked pairs along with their number of marks
while ($|A| > 0$) **do**
  for all pairs $[u, v] \in A$ **do**
    Add the pair [u,v] to Z and add one mark to all of its neighbouring pairs;
  end for
  while there exists an unmatched pair with at least 2 marks in $MarkedPairs$ that does not conflict M **do**
    among those pairs select the one maximizing $score(p)$;
    Add p=[u,v] to the set M;
    if [u,v] $\notin Z$ **then**
      Add one mark to all of its neighbouring pairs and add the pair [u,v] to Z;
    end if
  end while
  $A \leftarrow$ all neighboring pairs [u,v] of matched pairs M s.t. [u,v]$\notin Z$, $u \notin V_1(M)$ and $v \notin V_2(M)$;
end while
return M, $MarkedPairs$;

Figure 1: ExpandWhenStuck main algorithm

where $\Lambda(M)$ is the number of correct pairs in M and R is the set of all pairs of vertices that represent the same entity in both of the graphs.

To compare the performance of GM algorithms, we also report the F1–score, defined as:

$$F1 - score = 2\frac{precision \times recall}{precision + recall}. \quad (3)$$

The scores above (Recall, Precision, and F1) can only be computed based on known ground truth, yet it is useful to approximate the quality of a solution during the run time, assuming no known ground truth. Namely, given an input to the seeded-GM problem and two possible maps $M, M' : V_1 \rightarrow V_2$, we want to determine which of the two is better. We use $weight(M)$ as a score for the quality of a mapping M:

$$Weight(M) = |\{(u, v)|(u, v) \in E_1, (M(u), M(v)) \in E_2\}|. \quad (4)$$

3.2 The Generation of Correlated Graphs

To examine the performance of seeded GM algorithms, one needs a pair of graphs, where at least a part of the vertices correspond to the same entities. EWS used a simple probabilistic sampling method over a single given graph to create two correlated graphs with different levels of similarity.

We follow the same method. Specifically, given $G = (V, E)$ and s, we generate $G_1 = (V_1, E_1)$, and $G_2 = (V_2, E_2)$ by twice randomly and independently removing edges $e \in E$ with a probability of $1 - s$. We then remove vertices with no edges in G_1 and in G_2 separately. The edge overlap between G_1 andG_2 increases with s. EWS used $s \in [0.7, 0.9]$. Using IRMA, we extend the range to $s \in [0.4, 0.9]$. We name s the 'graphs-overlap'.

3.3 Data Sets

For fully simulated graphs, we use the above sampling method over $G(n, p)$ Erdos-Renyi graphs [6], defined as a graph with n vertices, where every edge of the possible $\binom{n}{2}$ exists with a probability of p. It is common to mark two graphs created by sampling from an Erdos-Renyi graph as $G_1, G_2 = G(n, p, s)$ [20].

To test our algorithm on graphs that better represent real-world data, yet to control their level of similarity, we used sampling over the following real-world graphs (further denoted as graph 1, graph 2, and so on, according to their order here (Table 2)).

Number	Name	Nodes	Edges	Average degree
1	Fb-pages-media	27,900	206,000	14
2	Soc-brightkite	56,700	212,900	7.8
3	Soc-epinions	26,600	100,100	7.9
4	Soc-gemsec-HU	47,500	222,900	9.4
5	Soc-sign-Slashdot081106	77,300	516,600	12.1
6	Deezer_europe_edges	28,300	92,800	6.6

Table 2: Real-world data set graphs.

(1) Fb-pages-media - Data collected about Facebook pages (November 2017). These datasets represent verified Facebook page graphs of different categories. Vertices represent the pages and edges are mutual likes among them (http://networkrepository.com/fb-pages-media.php).
(2) Soc-brightkite - Brightkite is a location-based social networking service provider where users shared their locations by checking-in. The dataset contains all links among users (http://networkrepository.com/soc-brightkite.php).
(3) Soc-epinions - Controversial Users Demand Local Trust Metrics: An Experimental Study on epinions.com Community (http://networkrepository.com/soc-epinions.php).
(4) Soc-gemsec-HU - The data was collected from the music streaming service Deezer (November 2017). These datasets represent friendship graphs of users from 3 European countries. Vertices represent the users and edges are the mutual friendships. We re-indexed the vertices in order to achieve a certain level of anonymity. The edge files contain the edges - vertices are indexed from 0. The json files contain the genre preferences of users - each key is a user id, the genres loved are given as lists. Genre notations are consistent across users. In each dataset users could like 84 distinct genres. Liked genre lists were compiled based on the liked song lists. The countries included are Romania, Croatia and Hungary (http://networkrepository.com/soc-gemsec-HU.php).
(5) Soc-sign-Slashdot081106 - Slashdot Zoo signed social network from November 6 2008. It is noteworthy that this graph was also used in [11] (http://networkrepository.com/soc-sign-Slashdot081106.php).

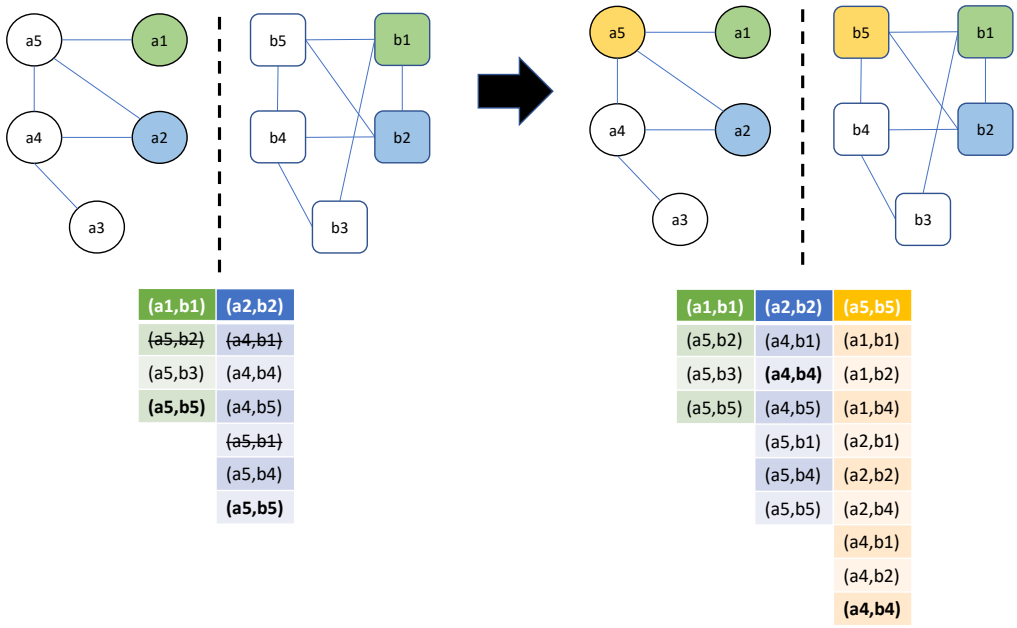


Figure 2: ExpandWhendStuck example. Given the seed $\{[a1, b1], [a2, b2]\}$ on the left, $[a1, b1]$ spread out marks to the Cartesian product $\{a5\} * \{b2, b3, b5\}$ (green column) and $[a2, b2]$ spread out marks to $\{a4, a5\} * \{b1, b4, b5\}$ (blue column). Marked pairs conflicting the seed have been crossed out. $[a5, b5]$ is marked in bold for having the most marks. On the right, $[a5, b5]$ is marked in yellow for getting into M and marks have been spread out to $\{a1, a2, a4\} * \{b1, b2, b4\}$ (yellow column). We marked $[a4, b4]$ in bold for being the next pair to be inserted into M.

(6) Deezer_europe_edges - A social network of Deezer users which was collected from the public API in March 2020. Vertices are Deezer users from European countries and edges are mutual follower relationships between them. The vertex features are extracted based on the artists liked by the users. The task related to the graph is binary node classification - one has to predict the gender of users. This target feature was derived from the name field for each user (http://snap.stanford.edu/data/feather-deezer-social.html).

3.4 Iterative Approach

The EWS algorithm has a greedy property. At step t, it adds to M the pair maximizing $score_t(p)$. Assume that at time t the algorithm chooses some wrong pair $[u', v]$ that obeys $score_t([u', v]) > score_t([u, v])$ for the correct pair $[u, v]$. Let us further assume that $[u, v]$ will receive many marks from neighbors later in the algorithm, such that $score_\infty([u', v]) < score_\infty([u, v])$. Such a scenario is highly likely, since we argue that correct pairs tend to reach a high score. Yet, $[u, v]$ will never get into M as it conflicts with another pair in it $([u', v])$.

We use the accumulation of scores $(score_\infty([u', v]) < score_\infty([u, v]))$ to improve the matching. We suggest an iterative improvement of the match, using the final score of the previous iteration. $mark_{i,t}(p)$ is defined as the number of marks gained by pair p during the i-th iteration until time t; $score_{i,t}(p)$ is defined accordingly as

$$score_{i,t}([u, v]) = marks_{i,t}([u, v]) - \epsilon * |d_{1,u} - d_{2,v}|. \quad (5)$$

Based on these scores, one can establish:

$$\bar{mark}_{i,t}(p) = max(mark_{i,t}(p), mark_{i-1,\infty}(p)), \quad (6)$$

and, following Eq. 1

$$\bar{score}_t([u, v]) = \bar{mark}_{i,t}([u, v]) - \epsilon * |d_{1,u} - d_{2,v}|. \quad (7)$$

We present the pseudo code for the Repairing-Iteration and IRMA in Figures 3 and 4, respectively. IRMA starts with the initialization of M to be the seed set, then it performs a standard EWS iteration. In the following iterations, at time t, while there is a candidate pair with $\bar{mark}_t(p) > 1$, IRMA adds to M the pair p maximizing $\bar{score}_t(p)$ among those that do not conflict M, and spread marks out of it.

IRMA stops the iterations when the mapping quality stops increasing. Formally, the i-th iteration starts by initializing $M = seed$, then,

at time t, it adds to M the candidate pair obeying:

$$p = argmax_p\{s\bar{c}ore_{i,t}(p)\}, \tag{8}$$

and spread marks out of it - updating $mark_{i,t}$. The iteration ends when no candidate pair p, that does not conflicts M, satisfies

$$m\bar{a}rk_{i,t}(p) > 1. \tag{9}$$

Since we do not know the real mapping, we use $weight(M)$ to estimate the quality of the score at the current iteration (section 3.1). We compute $weight(M_i)$ $\forall i$ where M_i is the matching at the end of the i-th iteration. Whenever $weight(M_i) \leq weight(M_{i-1})$, the algorithm stops and returns M_{i-1}. In practice, to avoid many redundant iterations with a limited increase in $weight(M)$, the algorithm stops when $weight(M) \leq (1+\delta) * weight(M_{i-1})$, where δ was empirically set to 0.01. Note that this does not ensure convergence of the mapping, only of its score.

Repairing Iteration

1: $MarkedPairs_i$ is an input of all marks from the previous iteration
2: $MarkedPairs_{i+1} \leftarrow \emptyset$, $M \leftarrow A_0$;
3: Spread out marks from all pairs in M
4: **while** there exists an unmatched pair with at least 2 marks in $MarkedPairs_i$ or $MarkedPairs_{i+1}$ that does not conflict M **do**
5: Among those pairs select the pair $p = argmax_p\{s\bar{c}ore_t(p)\}$;
6: Add p to the set M;
7: Add one mark in $MarkedPairs_{i+1}$ to all of its neighboring pairs and add the p to Z
8: **end while**
9: **return** $M, MarkedPairs_{i+1}$

Figure 3: Repairing Iteration receive the marks from the previous iteration along with the seed as input and builds a map taking into consideration the marks gained at the previous and current iteration.

IRMA (Iterative Repair for graph MAtching)

1: $M, MarkedPairs \leftarrow ExpandWhenStuck(A_0)$
2: $M_0 = \emptyset$
3: **while** $weight(M) > (1+\delta) * weight(M_0)$ **do**
4: $M_0 = M$
5: $M, MarkedPairs \leftarrow RepairingIteration(MarkedPairs)$
6: **end while**
7: **return** M

Figure 4: IRMA builds a primarily map using ExpandWhenStuck and repeatedly improves it by running 'Repairing Iteration'. It uses $weight(M)$ as an indication for convergence by the stop condition that appears in line 3.

3.5 Expansion Boost

In the first part of IRMA, each iteration is built of the main while-loop with a break condition of having no candidate pair, with at least two marks, that does not conflict M. The threshold of marks has been determined to 2 as a trade-off between the precision and the recall. If one sets the threshold to a high value, M will only contain pairs with high confidence, yet the percolation will stop early - leading to high precision but low recall. Similarly, setting the threshold to one will increase recall at the expense of precision.

Following the IRMA iterations, we suggest performing a "noisy iteration" with a threshold of 1 of choosing the next candidate pair after IRMA has converged. This leads to a drop in precision, but an increase in the recall. Then, we run regular repairing iterations again to gradually restore the precision. The idea is that while wrong pairs could not comply with the next iterations' threshold of 2 marks, correct pairs might lead to unexplored areas of the graphs. Such pairs gain marks by their newly revealed neighbors, *a-posteriori* justifying their insertion to M.

3.5.1 Break Condition. Since the iterations performed after the 'noisy iteration' are meant to filter out pairs with lower certainty, we can no longer expect $|weight(M)|$ to increase between iterations. In practice, the second stage of IRMA (after the expansion boost), restores the precision within a few iterations and rapidly stops improving M_i. We thus empirically set the IRMA to always stop after four repairing iterations, after the expansion boost.

3.6 Parallel Version

In order to develop a GPU version of IRMA, we propose a parallel version. The bottleneck of EWS is in spreading out marks from $[u, v]$, which costs $deg_1(u) * deg_2(v)$ updates to the queue of marks. Ideally, we would like to perform multiple mark spreading steps in parallel. However, this is not immediately possible since the pair chosen at time t depends on the marks that have been spread out earlier, including those of time $t-1$. Section 6.3 in [11] presents a paralleled version where the main loop has been split into epochs. This version of EWS starts by spreading out marks from the seed, then at each epoch the algorithm greedily takes all possible pairs from the queue one by one - without spreading any mark. When the queue is eventually empty, it simultaneously spread marks from all pairs selected at the current epoch, creating the queue to the next epoch. This method has the advantage of being extremely fast, allowing input graphs with millions of vertices, and has been argued not to fundamentally affect the performance of the algorithm.

We used a similar logic to parallelize iterations as follows (see Figure 5 for pseudo code). The i-th iteration gets $queue_{i-1}$ as input and starts by greedily adding all possible pairs from the queue into M_i one by one - without spreading any mark. Then, when the queue is eventually empty, we simultaneously spread out marks from all pairs in M_i creating $queue_i$. Iteration i returns M_i and $queue_i$.

Parallel Repairing Iteration

1: $MarkedPairs_{i-1}$ is an input of all marks from the previous iteration.
2: $MarkedPairs_i \leftarrow \emptyset$, $M_i \leftarrow A_0$;
3: **while** there exists an unmatched pair with at least 2 marks in $MarkedPairs_{i-1}$ that does not conflict M_i **do**
4: Among those pairs select the pair $p = argmax_p\{score_{i-1,\infty}(p)\}$;
5: Add p to the set M_i;
6: **end while**
7: $MarkedPairs_i \leftarrow$ marks that has been spread out from all M_i in parallel
8: **return** $M_i, MarkedPairs_i$

Figure 5: In order to allow spreading out marks in parallel, we no longer base over the marks of current iterations. It enables us to perform this stage simultaneously at the end of the iteration (line 7).

3.7 Experiments Setup

In section 3.3, we presented the variety of the data we used throughout this paper. In addition to multiple different graphs and graphs-overlap (s) values, we run all of our tests with different seed sizes that increment exponentially.

In general, experiments with a random factor have to be performed several times to indicate a trend. In our case, the sampling of the input graphs is based on randomness, as well as on choosing the seeds. For example, high degree pairs in the seed are much more effective to boost the process of percolation. However, the IRMA algorithm includes EWS as a first step, enabling it to examine the ability of IRMA to improve the output of EWS with a significantly lower factor of randomness. Thus, whenever comparing the results of IRMA to EWS, we are using the results of EWS as they obtained in the first stage of IRMA. For extra caution, in our main experiments (Figure 10) we repeat each experiment five times and present the average results.

4 RESULTS

4.1 EWS on Real-World Graphs

IRMA is an improvement of EWS [11]. We first tested the accuracy of EWS on a set of real-world graphs (see section 3.3 for detailed information on the selected graphs). We use each of those graphs to create two partially overlapping graphs by twice removing edges from the same graph. The fraction of edges from the original graph (G) maintained in each of the partially overlapping graphs G_1, G_2 is denoted s. We use a value of s in the range of $[0.4, 0.8]$.

We computed the precision, recall and F1-score of EWS over all the graphs with graphs overlap (s) value of $0.5, 0.6, 0.7$ as a function of seed size (Figure 6 a-c, respectively for graph 2, Other graphs are similar). All accuracy measures are highly sensitive to s. The reason is that a correct pair $[u, v] \in G_1 \times G_2$, corresponding to vertex $w \in G$ with degree d, has an expected number of $s^2 d$ common neighbors, which also is approximately the number of marks it will get. In random $G(n, p)$ graphs, most vertices have similar degree (a normal distribution of degrees), and if $s^2 * E(d) > 2$, EWS typically works. More precisely, [11] present a simplified version to the EWS that is much easier to analyze and compute the threshold seed size in order for the algorithm to correctly match $G(n, p, s)$ graphs. Under several assumptions they found an upper bound of $O(\frac{1}{np^4s^4})$.

However, in scale free degree distributions, the vast majority of vertices obey $d < E(d)$. As such, low graphs-overlap values prevent most vertices from gaining enough marks. The precision (and also recall and F1) is typically a monotonic non-decreasing function of the seed size, with no clear threshold (Fig. 6). A consistent large difference between recall and precision can be observed. Eq. **??** shows that the ratio between the precision and the recall as defined here are equal to the ratio between $|M|$ and $|G_1 \cap G_2|$- $\frac{recall}{precision} = \frac{|M|}{|R|}$ (defined to be the vertices contained in both graphs), i.e., the algorithm struggle to percolate through the entire graphs. Since real-world graphs have a power-law degree distribution, there are many vertices with a low degree that will never collect enough marks. As a result, M often contains only part of the possible pairs to match.

4.2 Parallel Version of EWS

Although [11] reports that their parallel version can be run on graphs with millions of vertices without fundamentally affecting the performance of the algorithm, they do not present any information about the exact drop in performance. The parallel version splits each iteration into epochs, in which it adds to M every possible candidate pair without spreading out marks, and then spreads out marks simultaneously for the next epoch.

We reproduced the test from the previous section, alongside the results of the parallel version (Figure 7 for graph 2. Other graphs are similar). The parallel version requires a larger seed in order to achieve similar results to the original version. Together with the former findings, we can conclude that while EWS has an excellent accuracy in $G(n, p)$, it has two main limitations:

(1) Significantly lower performances on real-world graphs than in $G(N, p)$, and difficulty to handle graphs-overlap lower than 0.7 in real-world networks.
(2) A large difference in real-world networks between the parallel and sequential versions.

We here propose that an iterative version of EWS can reduce these limitations.

4.3 IRMA Algorithm

As suggested in section 3.4, IRMA is based on the iterative Graph Matching problem, we first run EWS, and then perform repairing iterations. In each iteration, IRMA uses the marks received at the end of the former iteration to build a new and better map (Figures 3 and 4). The intuition to keep adding recommendations to mapped pairs and use them on the next iteration is simple. Correct pairs share more common neighbors than the wrong ones in the average case, therefore correct pairs are more likely to gain

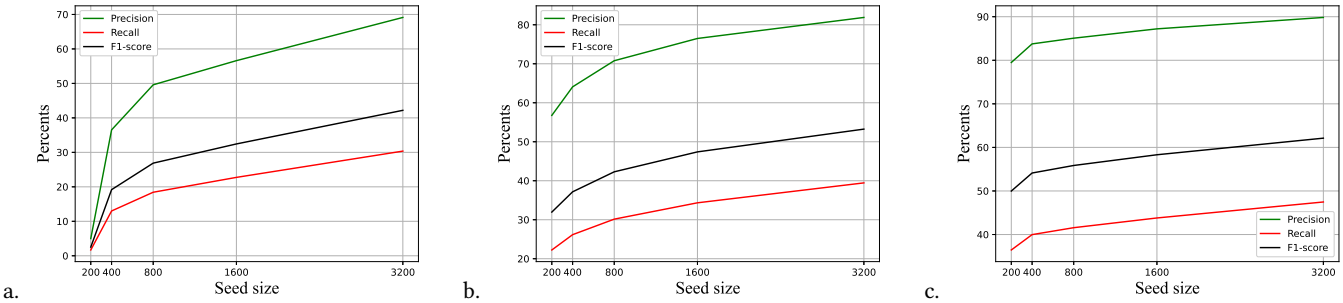


Figure 6: ExpandWhenStuck algorithm: Precision, recall and F1-score over graph 2 as a function of seed size. Sub-figures a-c use increasing s values of $0.5, 0.6, 0.7$, respectively. Even with large seed size, performances are significantly worse than the ones originally reported in $G(n, p)$ Erdos-Renyi graphs. In addition, there is a consistent large gap between recall and precision.

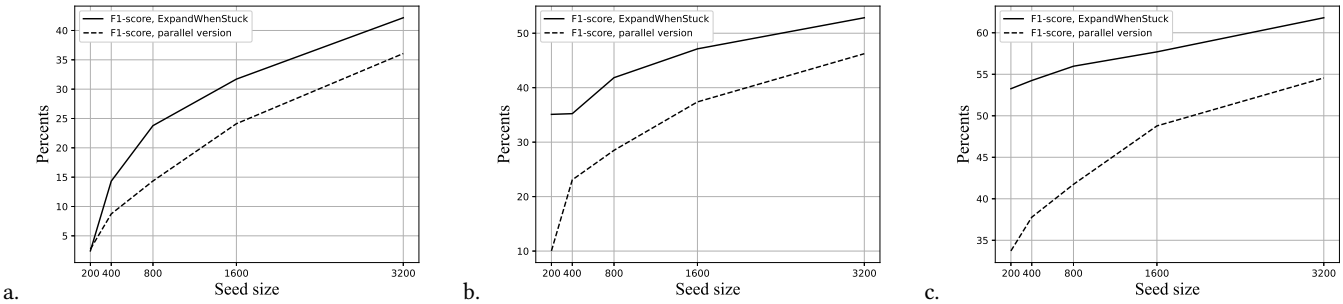


Figure 7: A comparison between EWS and its parallel version: F1-score of both versions over graph 2 as a function of seed size. Sub-figures a-c use increasing graphs overlap (s) of $0.5, 0.6, 0.7$, respectively. One can note a significant drop in performances of the parallel version compared to EWS.

more recommendations, even if they initially contradict another assignment.

4.4 IRMA improves accuracy along iterations

As a reminder, $score_t(p)$ is a score function defined by $marks_t(p)$ (the number of marks p gained until time t) and uses the difference in the vertices degree to break ties (Eq. 1). In Figure 9, we emphasize the advantage of relying upon $score_\infty(p)$ at time t by comparing the number of marks that a pair p has when inserted into M, against the number of marks it has at $t = \infty$.

4.4.1 Proof of improvement. We demonstrate the advantage of IRMA over EWS by showing that performing even one parallel iteration of 'Iterative Repair' lowers the probability of mapping wrong pairs when running over $G(n, p, s)$ graphs. $Score_{i,t}(p)$ is based on $marks_{i,t}(p)$ (the number of marks p gained during the i-th iteration until time t) see Eq. 5. Intuitively, if $score_{1,\infty}(p)$ is better than $score_{1,t}(p)$, the marks received during the second iteration should be more accurate than in the first iteration, improving $score_{2,t}(p)$, therefore improving $\bar{score}_t(p) = max(score_{1,\infty}(p), score_{2,t}(p))$. As argued by [11] when analyzing their algorithm, EWS has a complex property of generating noisy seed whenever it gets stuck, which is hard to analyze. For that reason, we analyze a simpler version of that algorithm where an artificial seed is not generated when percolation stops (see Figure 8).

Simplified ExpandWhenStuck

$A \leftarrow seed$ is the initial set of seed pairs, $M \leftarrow seed$;
$Z \leftarrow \emptyset$ is the set of used pairs
$MarkedPairs \leftarrow \emptyset$ is the set of all marked pairs along with their number of marks
for all pairs $[u, v] \in A$ **do**
    Add the pair [u,v] to Z and add one mark to all of its neighboring pairs;
end for
while there exists an unmatched pair with at least 2 marks in $MarkedPairs$ that does not conflict M **do**
    Among those pairs select the one maximizing $score(p)$;
    Add $p = [u, v]$ to the set M;
    if [u,v] $\notin Z$ **then**
        Add one mark to all of its neighboring pairs and add the pair $[u, v] to Z$; **end if**
return M, $MarkedPairs$;

Figure 8: A simplified version of ExpandWhenStuck. Whenever no pair with two marks exist, the algorithm does not generate an artificial seed and simply stops.

Theorem 1: Given $G_1, G_2 \leftarrow G(n, p, s)$, let $p' = [u, v']$ be a wrong pair inserted into M at time t and let $p = [u, v]$ be a right pair conflicting p'. Assuming at some time $\bar{t} > t$ a correct pair has been inserted into M, the following applies:

$$\mathbb{E}[score_\infty(p) - score_\infty(p')] > \mathbb{E}[score_t(p) - score_t(p')] \quad (10)$$

Since whenever we choose a wrong pair $[u, v']$ before the right pair $[u, v]$, we will eventually avoid the latter insertion of $[u, v]$, since it will conflict M. Theorem 1 suggests that using $score_\infty$ will reduce the probability of a mistake in the next iteration - eventually reducing the number of wrong pairs in M.

Proof of Theorem 1:
Eq. 10 in Theorem 1 is equivalent to:

$$\mathbb{E}[score_\infty(p) - score_t(p)] > \mathbb{E}[score_\infty(p') - score_t(p')] \quad (11)$$

In other words, the expected number of marks that p will get from now on, will be higher than the expected number of marks p' will receive. Let us denote by M_t the map at time t, we define $\Lambda(M_t)$ to be the number of right pairs in M_t and $\Psi(M_t)$ be the number of wrong pairs.
Let $p_{t'} = [\alpha, \beta]$ be a pair inserted to M at time $t' > t$:

- If $p_{t'}$ is a correct pair, α and β represent the same vertex $\gamma \in V$ (u and v represented by the same vertex $w \in V$ as well). $p = [u, v]$ gets one mark if there are edges $(\alpha, u) \in E_1$ and $(\beta, v) \in E_2$. This requires the edge (γ, w) to exist in E (which happens with probability p) and to be sampled in E_1, E_2 (happens with probability s^2). On the other hand, p' gets a mark if $(\alpha, u) \in E_1 and (\beta, v') \in G_2$. This requires two different edges to exist in E (probability of p^2) and to be sampled to E_1 and E_2 accordingly (probability of s^2).
- If $p_{t'}$ is a wrong pair, α and β are represented by the different vertices $\alpha', \beta' \in V$, (note that $p_{t'}$ cannot conflict p'). The pair p gets one mark if there are edges $(\alpha, u) \in E_1$ and $(\beta, v) \in E_2$. This happens with probability $p^2 s^2$. The pair p' gets one mark if there are edges $(\alpha, u) \in E_1$ and $(\beta, v') \in E_2$, which also happens with probability $p^2 s^2$. In fact, it is possible that one of the pairs $p_{t'}$ will have the form $[k, v]$. In that case, p' gets a mark with probability $p^2 s^2$ while p gets none, as $[u, v']$ can never be a neighboring pair of $[k, v']$.

Let us denote $L_t = M_\infty - M_t$, using the analysis above, one can compute:

$$\mathbb{E}[score_\infty(p) - score_t(p)] \geq \Lambda(L_t) * s^2 p + (\Psi(L_t) - 1) * s^2 p^2 \quad (12)$$

$$\mathbb{E}[score_\infty(p') - score_t(p')] = \Lambda(L_t) * s^2 p^2 + \Psi(L_t) * s^2 p^2. \quad (13)$$

Combining Eq. 12 and 13, we obtain:

$$\mathbb{E}[score_\infty(p) - score_t(p)] \geq \Lambda(L_t) * s^2 p(1-p) - s^2 p^2 + \mathbb{E}[score_\infty(p') - score_t(p')] \quad (14)$$

To prove Eq. 11, it remains to show that $\Lambda(L_t) * s^2 p(1-p) - s^2 p^2 > 0$. Since we assumed that some correct pair has been inserted to M at time $\bar{t} > t$, we have $\Lambda(L_t) \geq 1$. p is the probability of an edge to exist, and as we only focus on sparse graphs we assume $p << 0.5$, leading to:

$$\begin{aligned} &\Lambda(L_t) * s^2 p(1-p) - s^2 p^2 \\ &\quad = (\Lambda(L_t) - 1) * s^2 p(1-p) + s^2 p(1-p) - s^2 p^2 \\ &\quad = (\Lambda(L_t) - 1) * s^2 p(1-p) + s^2 p(1-2p) > 0 \end{aligned} \quad (15)$$

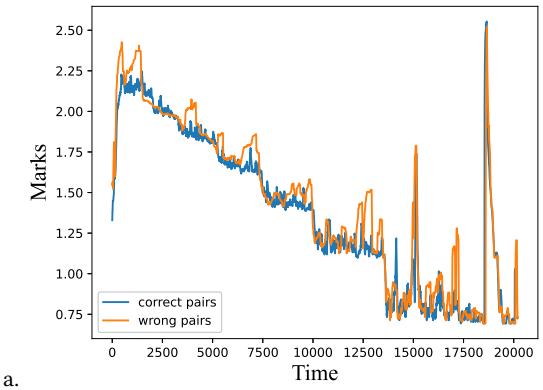
□

IRMA builds M_i during the i-th iteration using $s\bar{c}ore_{i,t}(p) = max(score_{i,t}(p), score_{i-1,\infty}(p)$. IRMA uses the advantage of each iteration over the previous one. We have shown the expected advantage of the first Repairing-Iteration over EWS, so $score_{2,\infty}(p)$ is more accurate than $score_{1,\infty}(p)$ on average. The proof above did not use any assumption on the way the previous marks were produced. Thus, the same logic applies to all following iterations. Thus, $s\bar{c}ore_{3,t}(p)$ is expected to be more accurate than $s\bar{c}ore_{2,t}(p)$, and the same holds for all following iterations.

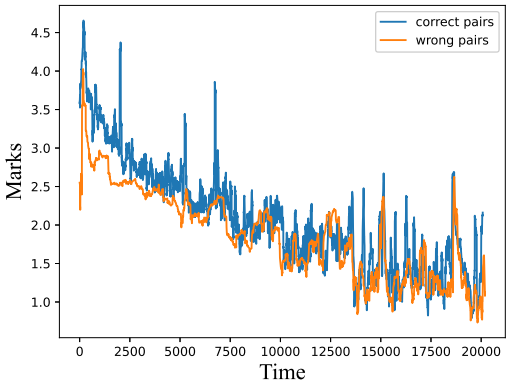
To test the improvement in accuracy, we computed the F1 score for all the real-world networks above (Figure 10). Each sub-figure consists of several runs of IRMA with different sizes of seeds, with a fixed graph and graphs-overlap (s). Each value is the average of 5 repetitions. The iterations are colored from green to blue. The lowest values are always of EWS, and the results keeps improving until they converge.

The first, most noticeable observation is that IRMA indeed gradually repairs the output map of EWS. In fact, the F1-score results are a monotonic non-decreasing function of the iterations. Moreover, the convergence is very rapid, and number of iterations is rarely more than 5 or 6 until convergence. Finally, the most interesting thing is that IRMA is less sensitive to the randomness of the process. EWS often achieves lower performances on a bigger seed due to the randomness in the process of generating the graphs and choosing

ExpandWhenStuck over Graph 6, $s = 0.7$, $|seed| = 480$



a.



b.

Figure 9: In the top plot, we present the number of marks at the moment of insertion into M during EWS, The X axis represents time (i.e., steps of additions of pairs). In the bottom plot, the same pairs are presented (keeping the same order) but the Y axis is the number of marks they had at the end of the algorithm. The tendency of correct pairs to keep gaining marks after being matched can be clearly seen (Blue line is higher than Orange line in bottom plot and lower in top plot). Both graphs have been smoothed by a sliding window of 50 for clarity.

the seed (see sub-figures a,d,e for such cases). Despite accepting EWS's output as an initial map, IRMA exhibits a dependency on the problem properties (as the input graph and the graphs-overlap) rather than the output of EWS, and therefore has much smoother plots. Importantly, the recall (and thus the F1 value) of IRMA, as presented here, does not rise to 1. Instead, it rises to the maximal number of vertices that can have at least two marks. However, it does so even for small seeds.

4.4.2 Stopping condition. While the convergence of IRMA in F1 is clear from Fig 10, in real life scenarios, the real map is not known, and F1 cannot be used as a stopping condition. We thus propose to use $weight(M_i)$ (See Eq. 11) as a quality measure for M_i, and hence as a stopping condition for the algorithm. We recall that $weight(M)$ is defined as $|\{[u,v]|[u,v] \in G_1, [M(u), M(v)] \in G_2\}|$ i.e., is the number of edges in the common subgraph induced by the M. Since correct pairs share more common neighbors in expectation, we expect better maps to have higher $weight(M)$. Similarly, if $weight(M_i) < (1+\delta)weight(M_{i+1})$, one can assume that M_{i+1} is not significantly better than M_i. We run IRMA and compared $weight(M_i)$ to the F1-score of M_i to test the connection between them (Figure 11). One can clearly observe that both indices are converging simultaneously and have very similar dynamics (albeit different values as expressed by the different y axes in Figure 11), suggesting that $weight(M)$ can serve as a proxy for the $F1$ score.

4.4.3 Computational Complexity of IRMA. For a pair $[u,v] \in M$, the number of marks to spread out is $d_1(u) * d_2(v)$ where $d_i(u)$ is the degree of $u \in G_i$. Thus, the number of updates to the priority queue from inserting new pairs to M is $N = \sum_{[u,v] \in M} d_1(u) * d_2(v)$. The cost of each insertion is proportional to the log of the size of the priority-queue, which is bound $O(|V|^2)$, so the cost of each insertion is bound by $O(Log(|V|)$, where $|V|$ is the number of vertices. The total number of updates (N) is between $N = O(|V| * E[d]^2)$ and $N = O(|V| * E[d^2])$, where d is the degree. The first case is for random matching (i.e., $d(u)$ and $d(v)$ are independent), and the second case is for perfect matching $d(u) = d(v)$. In low variance degree distributions, both cases are equivalent. However, in power-low degree distributions, the second case may be much higher than the first. Thus, one can bound the cost of the algorithm by $O(|V| * log(|V|) * E[d^2]$, but often the bound is tighter - $O(|E|^2 * log(|V|)/|V|)$. The Repairing-Iteration has a similar cost, since it is based on the same logic of pulling pairs from the priority queue and spreading out marks. Note that we avoid the problematic case in EWS that the expanded seed may be of $O(|V|^2)$, which adds a V factor to the cost of spreading marks. In their parallel versions, the boundary is similar, but in practice the Repairing-Iteration takes about $30\% - 60\%$ the run time of EWS, and as IRMA runs Repairing-Iteration a few times, it has an overall run time 2-3 times higher than EWS, but of the same order.

4.5 Expansion to Low Degree Vertices

As explained in the introduction, the Seeded Graph Matching problem is a variant of the Maximum Common Edges Subgraph (MCES) problem. However, while in MCES, one tries to find the common subgraph that contains the maximal possible number of edges, in our problem the target is to use the edge match as a tool to discover an a priori existing match. Practically, in the MCES it is always beneficial to add more vertices to the map - no matter how wrong their map will be. However, in seeded GM, one tries to maximize both precision and recall. Hence, at some stage, IRMA does not add pairs that with high probability are wrong, since those will lower the precision.

While running both EWS and the repairing iterations, we use a threshold of two marks to add a candidate pair into M. This threshold has been set as a trade-off to insert only reliable pairs, yet to allow the percolation to flow. But since the Repairing-Iteration method filters wrong matched pairs, it is possible to allow the algorithm to make mistakes in some cases in return to find new correct pairs. For that reason, we suggest to perform one repairing iteration

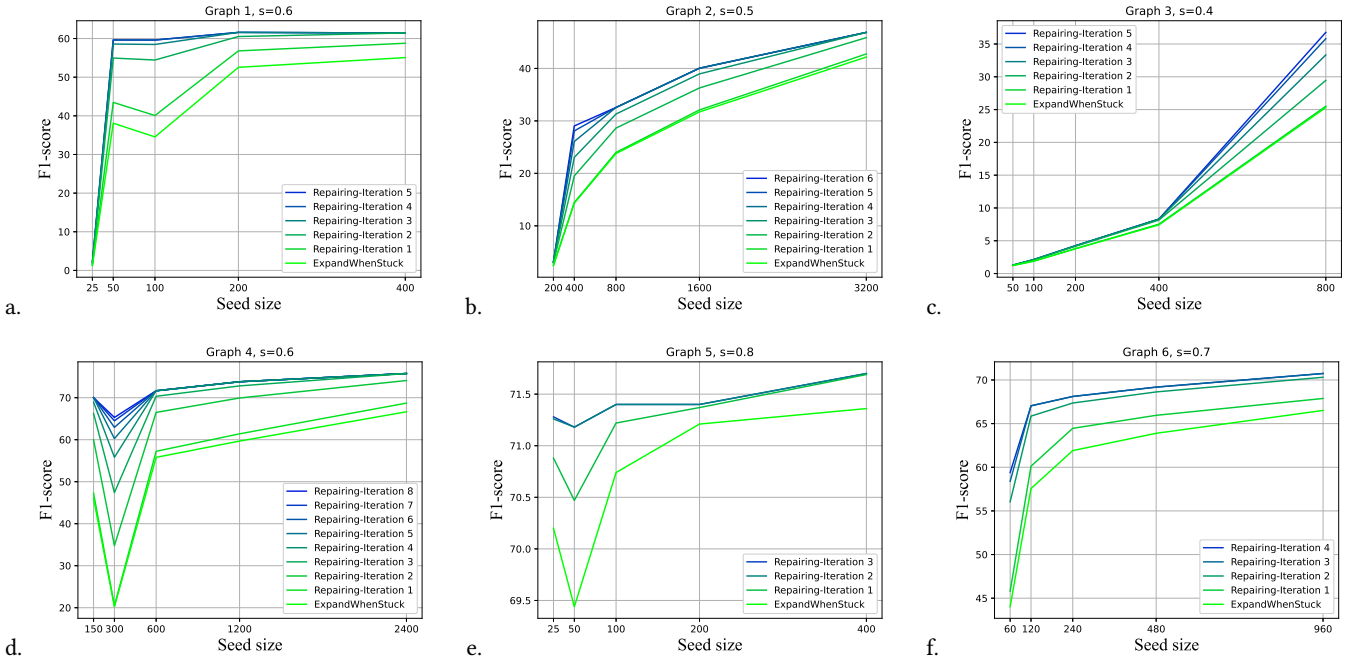


Figure 10: Performances of IRMA: Each sub-figure consists of several runs of IRMA with different sizes of seeds with a fixed graph and graphs-overlap (s). We show how the iterations gradually converge from ExpandWhenStuck to IRMA (the blue line).

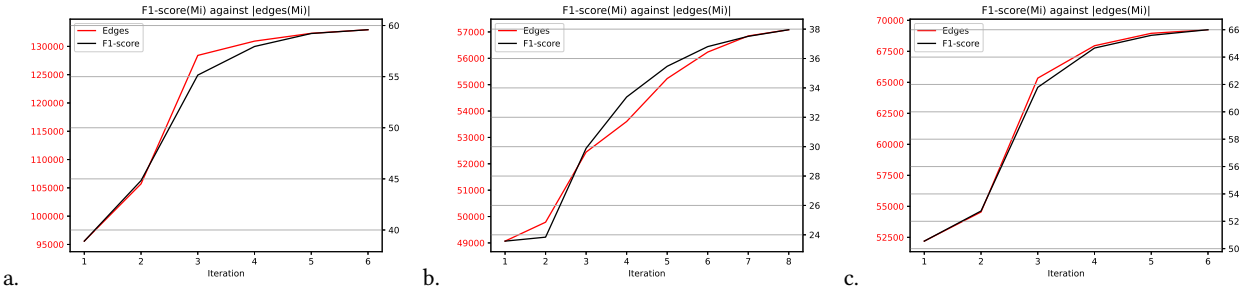


Figure 11: Each sub-figure represents $weight(M)$ and F1-score of a single run of IRMA along the iterations. One can note that both indices converge simultaneously, making $weight(M)$ a perfect halt condition.

with a threshold of one mark, after IRMA has converged. Then, we run regular repairing iterations again to gradually restore the precision.

As explained above in the methods section 3.5.1, since repairing iterations are used to filter out a massive amount of wrong pairs, $weight(M_i)$ is no longer a good condition for testing the convergence. We thus tested three possible break conditions to the second stage of IRMA:

(1) Let $|M_i|$ be be number of pairs in M_i, where M_i is the set of matched pairs at the end of the i iteration. Similarly, let $Weight(M_i)$ be the total number of edges among members of M_i present in both $G1$ and $G2$. The quality of a bijection can be defined through $Weight(M_i)/|M_i|$.
(2) As the target is to scale M down, $|M|$ might indicate if the second stage has ended. We can use the condition $|M_i| \geq (1-\delta) * |M_{i-1}|$ to avoid many redundant iterations.
(3) Empirically after a few iterations there is no significant improvement from M_i to M_{i+1}. A simple solution might be to use a constant number of iterations.

We computed the difference in accuracy along snapshots vs these three different measures (Figure 12). Sub-plots a and b represent the difference in precision as a function of the difference in $|weight(M_i)|/|M_i|$ and $|M_i|$, respectively. A clear correlation between the measures can be seen in sub-plots A and B, yet the variance around the origin

(i.e., when IRMA converges) is relatively large. Instead, sub-plot C indicates that the most reliable prediction to the convergence of precision is the number of iteration. We thus propose that as the default stopping criteria after the expansion iteration.

We used these additional iterations on the runs of IRMA from the previous section and received a small but consistent improvement over IRMA, that is typically bigger on relatively small seeds (except for cases when IRMA failed). We tested the precision, recall and F1-score along IRMA's iterations on different graphs (Figure 13). As follows from Eq. **??**, the precision is always higher than recall, and the relation between them is equal to the ratio between R and M. Both recall and precision are monotonic non decreasing up to the expansion/exploration iteration. The drop in precision together with the pick in recall are caused by the exploration iteration, inserting many pairs with low level of certainty. After that, the Repairing-Iteration method is activated again to restore precision, and successfully does so in a few iterations. Note that while precision is fully restored (and some times even improves), the recall stays higher than before the exploration iteration, as some of the correct new matches lead to an unexplored part of the graphs and gain enough marks for later iterations.

4.6 Parallel Solution

In section 3.6, we introduced the parallel version to EWS. The main idea is to split the algorithm into epochs such that marks are spread out only between the epochs, allowing parallel spreading. Using the same concept, we developed a parallel version to Repairing-Iteration in which M is rebuilt based only on the marks of the former iteration. The parallel version of IRMA starts by running Parallel-EWS and then performs iterations of Parallel-Repairing-Iteration.

When comparing (Fig. 14) the parallel-EWS, Parallel-IRMA and regular IRMA (both use exploring iteration). The parallel-IRMA has much better precision and recall than the Parallel-EWS. In sub-plot E, one can see an extreme case where Parallel-IRMA with a seed of 25 pairs is more accurate than Parallel-EWS with 400 pairs in the seed. Often, the Parallel-IRMA and the standard IRMA have very similar recall and precision, making the parallel IRMA an excellent trade-off between run-time and performances. Note that the parallel IRMA has the same run time as (parallel) EWS up to multiplication by a small constant, and is much faster, yet much more accurate than the non-parallel EWS.

5 DISCUSSION

We have here extended the EWS seeded-GM algorithm to include iterative corrections of the matching. The EWS is greedy in the sense that once a pair has been matched, it cannot be changed. The EWS is based on the spread of marks to common neighbors of previously matched pairs.

We have here shown that real matches tend to accumulate more marks than wrong ones. As such, one can repair at the end of each iteration the paired matches, and repeat the iteration. The resulting iterative algorithm is named IRMA, which has a better accuracy than the current state-of-the-art GM algorithms.

While in $G(n,p)$ graphs existing seeded GM algorithms seems to get a very high accuracy even for a small seed, the same does not hold for scale-free networks, as are most real-world networks. There are two main reasons for the failure of seeded GM in scale free networks: A) pairs of very high degree vertices (that are not real pairs) easily accumulate wrong marks. In a greedy approach, high-degree vertices will receive a lot of marks from common neighbors, since they tend to have a lot of random common neighbors. B) Very low degree vertices have few neighbors, and as such receive very few marks.

IRMA solves the first problem by iteratively fixing errors. However, a simple iterative correction cannot solve the second problem. To address the second problem, an expansion iteration was added, where IRMA accumulates a large number of low-degree pairs with a large fraction of wrong-pairs. This is then followed by multiple correction iterations to improve the accuracy, while maintaining a high recall.

Each iteration of IRMA is a full iteration of EWS, and typical applications require 5-10 iterations. The run time could be expected to be 5-10 time longer than EWS. However, in practice, the gain in accuracy of IRMA is much higher than the accuracy difference between the parallel and iterative versions of EWS. As such IRMA can be used with the parallel version of EWS, and be as fast if not faster than regular EWS with higher precision and recall.

We have tested alternative methods to fix wrong pairs in each iteration, including the analysis of negative marks (marks from contradicting neighbors), or the relation between marks and degree. However, eventually, the simplest approach to accumulate marks, even after a pair was selected, ended up being the one best improving the precision and recall.

The extension of seeded GM algorithms to scale-free network with limited overlap (low values of s) is essential for their application in real-world networks. However, the model we have used here for the generation of partially overlapping graphs (random sampling from a larger common graph), may not represent the real difference between networks. Thus, an important extension of the current method would be to test it on different partially overlapping networks models.

Another important caveat of IRMA is that in the current version it is applied to unweighted undirected graphs. The extension to directed and weighted graphs is straightforward. However, this would add multiple free parameters to the analysis (such as the difference between the in and out degree distributions, and the weight distribution). We have thus preferred to focus on the simpler case, leaving the more general models as future extensions.

REFERENCES

[1] Carla-Fabiana Chiasserini, Michele Garetto, and Emilio Leonardi. Social network de-anonymization under scale-free user relations. *IEEE/ACM Transactions on Networking*, 24(6):3756–3769, 2016.

[2] Daniel Cullina and Negar Kiyavash. Improved achievability and converse bounds for erdos-rényi graph matching. *ACM SIGMETRICS Performance Evaluation Review*, 44(1):63–72, 2016.

[3] Daniel Cullina and Negar Kiyavash. Exact alignment recovery for correlated erd\h {o} sr\'enyi graphs. *arXiv preprint arXiv:1711.06783*, 2017.

[4] Jian Ding, Zongming Ma, Yihong Wu, and Jiaming Xu. Efficient random graph matching via degree profiles. *Probability Theory and Related Fields*, 179(1):29–115, 2021.

[5] Amir Egozi, Yosi Keller, and Hugo Guterman. A probabilistic approach to spectral graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*,

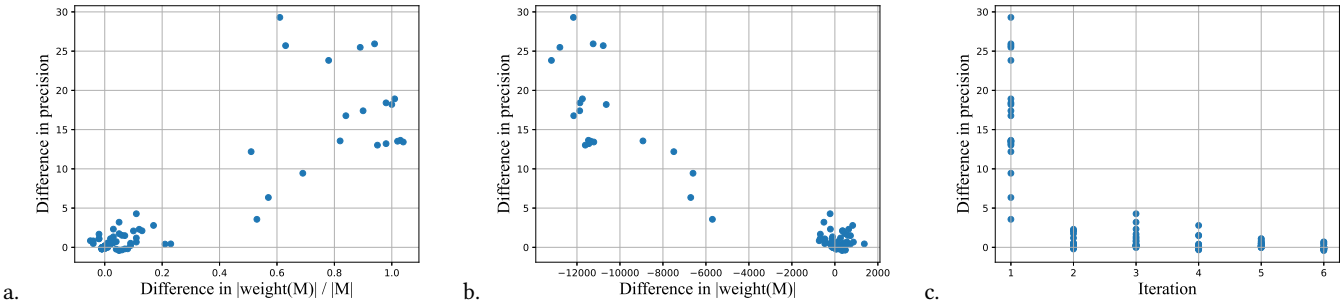


Figure 12: Difference in precision as a function of three indices. We run IRMA with the exploring iteration multiple times over different graphs, seed-size and graphs-overlap. The scatters indicate that four repairing iterations are enough for the precision to converge

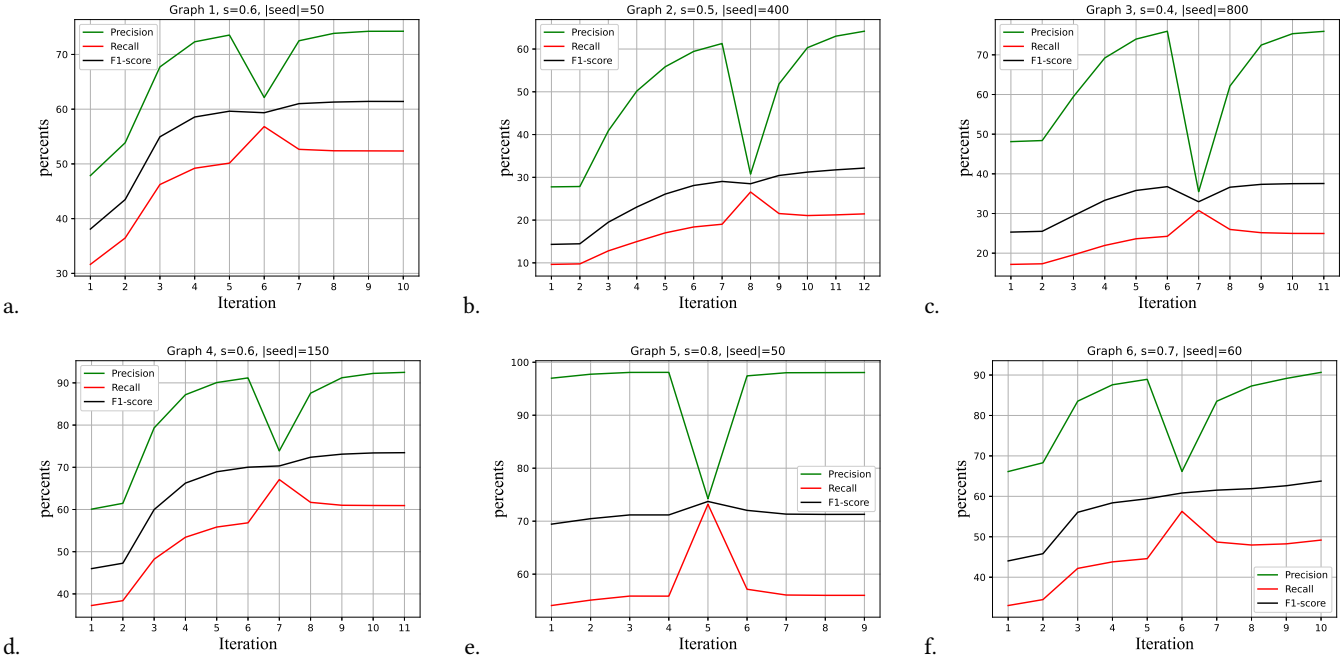


Figure 13: Precision, recall and F1-score during the algorithm's iterations: Each sub-figure consists of a single run of IRMA with the exploring iteration. The drop in precision is caused by the exploring iteration, after which the precision is fully restore while recall stays higher than before.

35(1):18–27, 2012.

[6] P Erdös and A Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[7] Donniell E Fishkind, Sancar Adali, Heather G Patsolic, Lingyao Meng, Digvijay Singh, Vince Lyzinski, and Carey E Priebe. Seeded graph matching. *arXiv preprint arXiv:1209.0367*, 2012.

[8] Donniell E Fishkind, Sancar Adali, Heather G Patsolic, Lingyao Meng, Digvijay Singh, Vince Lyzinski, and Carey E Priebe. Seeded graph matching. *Pattern Recognition*, 87:203–215, 2019.

[9] Oana Goga, Howard Lei, Sree Hari Krishnan Parthasarathi, Gerald Friedland, Robin Sommer, and Renata Teixeira. Exploiting innocuous activity for correlating users across sites. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 447–458, 2013.

[10] Keith Henderson, Brian Gallagher, Lei Li, Leman Akoglu, Tina Eliassi-Rad, Hanghang Tong, and Christos Faloutsos. It's who you know: graph mining using recursive structural features. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 663–671, 2011.

[11] Ehsan Kazemi, S. Hamed Hassani, and Matthias Grossglauser. Growing a graph matching from a handful of seeds. *Proceedings of the VLDB Endowment*, 8(10):1010–1021, 2015.

[12] Ehsan Kazemi, Lyudmila Yartseva, and Matthias Grossglauser. When can two unlabeled networks be aligned under partial overlap? In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 33–42. IEEE, 2015.

[13] Gunnar W Klau. A new graph-based method for pairwise global network alignment. *BMC Bioinformatics*, 10(1):1–9, 2009.

[14] Nitish Korula and Silvio Lattanzi. An efficient reconciliation algorithm for social networks. *arXiv preprint arXiv:1307.1690*, 2013.

[15] Danai Koutra, Hanghang Tong, and David Lubensky. Big-align: Fast bipartite graph alignment. In *2013 IEEE 13th International Conference on Data Mining*,

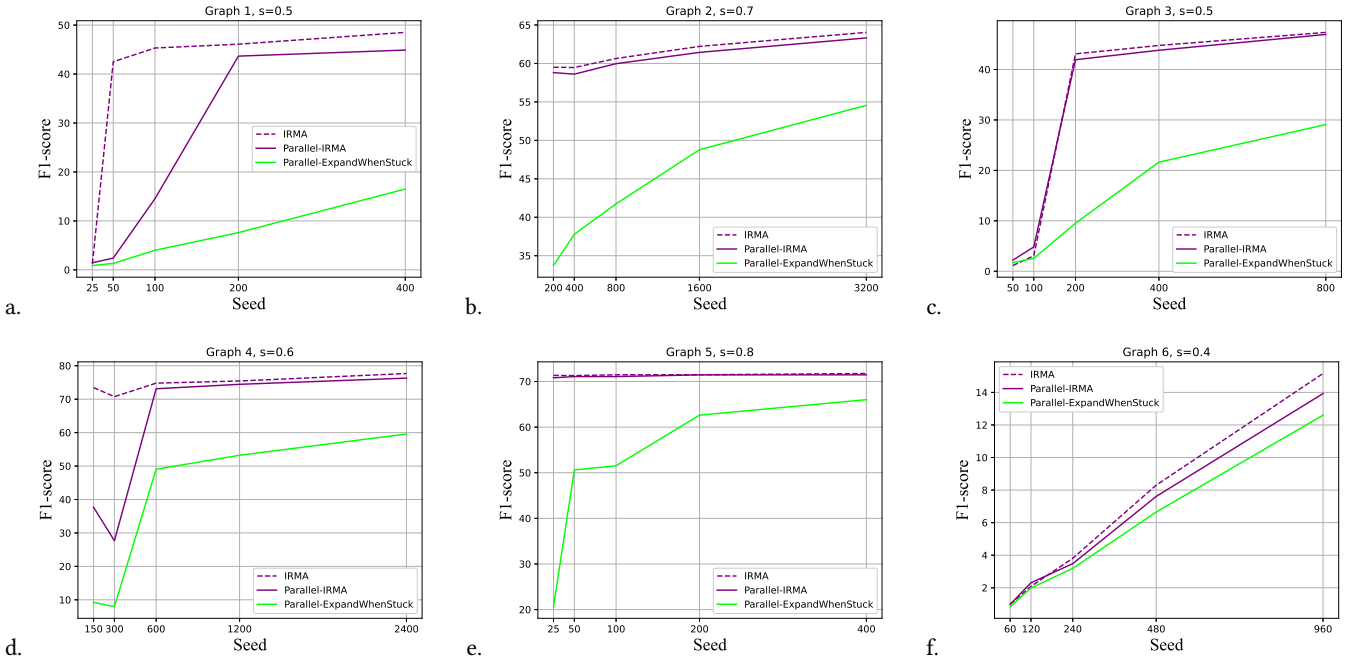


Figure 14: A comparison between parallel-ExpandWhenStuck, Parallel-IRMA and regular IRMA (both use an exploring iteration).

pages 389–398. IEEE, 2013.

[16] Anshu Malhotra, Luam Totti, Wagner Meira Jr, Ponnurangam Kumaraguru, and Virgilio Almeida. Studying user footprints in different online social networks. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1065–1070. IEEE, 2012.

[17] Elchanan Mossel and Jiaming Xu. Seeded graph matching via large neighborhood statistics. *Random Structures & Algorithms*, 57(3):570–611, 2020.

[18] Arvind Narayanan, Hristo Paskov, Neil Zhenqiang Gong, John Bethencourt, Emil Stefanov, Eui Chul Richard Shin, and Dawn Song. On the feasibility of internet-scale author identification. In *2012 IEEE Symposium on Security and Privacy*, pages 300–314. IEEE, 2012.

[19] André Nunes, Pável Calado, and Bruno Martins. Resolving user identities over social networks through supervised learning and rich similarity features. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 728–729, 2012.

[20] Pedram Pedarsani and Matthias Grossglauser. On the privacy of anonymized networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1243, 2011.

[21] Christopher Riederer, Yunsung Kim, Augustin Chaintreau, Nitish Korula, and Silvio Lattanzi. Linking users across domains with location data: Theory and validation. In *Proceedings of the 25th International Conference on World Wide Web*, pages 707–719, 2016.

[22] Farhad Shirani, Siddharth Garg, and Elza Erkip. Seeded graph matching: Efficient algorithms and theoretical guarantees. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 253–257. IEEE, 2017.

[23] Rohit Singh, Jinbo Xu, and Bonnie Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proceedings of the National Academy of Sciences*, 105(35):12763–12768, 2008.

[24] Shulong Tan, Ziyu Guan, Deng Cai, Xuzhen Qin, Jiajun Bu, and Chun Chen. Mapping users across networks by manifold alignment on hypergraph. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[25] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *European Conference on Computer Vision*, pages 596–609. Springer, 2008.

[26] Laurenz Wiskott, Norbert Krüger, N Kuiger, and Christoph Von Der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):775–779, 1997.

[27] Lyudmila Yartseva and Matthias Grossglauser. On the performance of percolation graph matching. In *Proceedings of the First ACM Conference on Online Social Networks*, pages 119–130, 2013.

[28] Reza Zafarani and Huan Liu. Connecting users across social media sites: a behavioral-modeling approach. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 41–49, 2013.

[29] Fan Zhou, Lei Liu, Kunpeng Zhang, Goce Trajcevski, Jin Wu, and Ting Zhong. Deeplink: A deep learning approach for user identity linkage. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1313–1321. IEEE, 2018.

5 Discussion

We have here extended the EWS seeded-GM algorithm to include iterative corrections of the matching. The EWS is greedy in the sense that once a pair has been matched, it cannot be changed. The EWS is based on the spread of marks to common neighbors of previously matched pairs.

We have here shown that real matches tend to accumulate more marks than wrong ones. As such, one can repair at the end of each iteration the paired matches, and repeat the iteration. The resulting iterative algorithm is named IRMA, which has a better accuracy than the current state-of-the-art GM algorithms.

While in $G(n,p)$ graphs existing seeded GM algorithms seems to get a very high accuracy even for a small seed, the same does not hold for scale-free networks, as are most real-world networks. There are two main reasons for the failure of seeded GM in scale free networks: A) pairs of very high degree vertices (that are not real pairs) easily accumulate wrong marks. In a greedy approach, high-degree vertices will receive a lot of marks from common neighbors, since they tend to have a lot of random common neighbors. B) Very low degree vertices have few neighbors, and as such receive very few marks.

IRMA solves the first problem by iteratively fixing errors. However, a simple iterative correction cannot solve the second problem. To address the second problem, an expansion iteration was added, where IRMA accumulates a large number of low-degree pairs with a large fraction of wrong-pairs. This is then followed by multiple correction iterations to improve the accuracy, while maintaining a high recall.

Each iteration of IRMA is a full iteration of EWS, and typical applications require 5-10 iterations. The run time could be expected to be 5-10 time longer than EWS. However, in practice, the gain in accuracy of IRMA is much higher than the accuracy difference between the parallel and iterative versions of EWS. As such IRMA can be used with the parallel version of EWS, and be as fast if not faster than regular EWS with higher precision and recall.

We have tested alternative methods to fix wrong pairs in each iteration, including the analysis of negative marks (marks from contradicting neighbors), or the relation between marks and degree. However, eventually, the simplest approach to accumulate marks, even after a pair was selected, ended up being the one best improving the precision and recall.

The extension of seeded GM algorithms to scale-free network with limited overlap (low values of s) is essential for their application in real-world networks. However, the model we have used here for the generation of partially overlapping graphs (random sampling from a larger common graph), may not represent the real difference between networks. Thus, an important extension of the current method would be to test it on different partially overlapping networks models.

Another important caveat of IRMA is that in the current version it is applied to unweighted undirected graphs. The extension to directed and weighted graphs is straightforward. However, this would add multiple free parameters to the analysis (such as the difference between the in and out degree distributions, and the weight distribution). We have thus preferred to focus on the simpler case, leaving the more general models as future extensions.

References

[1] Yunsheng Bai, Derek Xu, Ken Gu, Xueqing Wu, Agustin Marinovic, Christopher Ro, Yizhou Sun, and Wei Wang. Neural maximum common subgraph detection with guided subgraph extraction. 2019.

[2] Yunsheng Bai, Derek Xu, Alex Wang, Ken Gu, Xueqing Wu, Agustin Marinovic, Christopher Ro, Yizhou Sun, and Wei Wang. Fast detection of maximum common subgraph via deep q-learning. *arXiv preprint arXiv:2002.03129*, 2020.

[3] Carla-Fabiana Chiasserini, Michele Garetto, and Emilio Leonardi. Social network de-anonymization under scale-free user relations. *IEEE/ACM Transactions on Networking*, 24(6):3756–3769, 2016.

[4] Amir Egozi, Yosi Keller, and Hugo Guterman. A probabilistic approach to spectral graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):18–27, 2012.

[5] P Erdös and A Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[6] Keith Henderson, Brian Gallagher, Lei Li, Leman Akoglu, Tina Eliassi-Rad, Hanghang Tong, and Christos Faloutsos. It's who you know: graph mining using recursive structural features. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 663–671, 2011.

[7] Ruth Hoffmann, Ciaran McCreesh, and Craig Reilly. Between subgraph isomorphism and maximum common subgraph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[8] Dixin Luo Hongteng Xu and Lawrence Carin. Scalable gromov-wasserstein learning for graph partitioning and matching. 2019.

[9] Ehsan Kazemi, S. Hamed Hassani, and Matthias Grossglauser. Growing a graph matching from a handful of seeds. *Proceedings of the VLDB Endowment*, 8(10):1010–1021, 2015.

[10] Gunnar W Klau. A new graph-based method for pairwise global network alignment. *BMC Bioinformatics*, 10(1):1–9, 2009.

[11] Nitish Korula and Silvio Lattanzi. An efficient reconciliation algorithm for social networks. *arXiv preprint arXiv:1307.1690*, 2013.

[12] Anshu Malhotra, Luam Totti, Wagner Meira Jr, Ponnurangam Kumaraguru, and Virgilio Almeida. Studying user footprints in different online social networks. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1065–1070. IEEE, 2012.

[13] Ciaran McCreesh, Patrick Prosser, and James Trimble. A partitioning algorithm for maximum common subgraph problems. 2017.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[15] André Nunes, Pável Calado, and Bruno Martins. Resolving user identities over social networks through supervised learning and rich similarity features. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 728–729, 2012.

[16] Pedram Pedarsani and Matthias Grossglauser. On the privacy of anonymized networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1243, 2011.

[17] John W Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-aided Molecular Design*, 16(7):521–533, 2002.

[18] Rohit Singh, Jinbo Xu, and Bonnie Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proceedings of the National Academy of Sciences*, 105(35):12763–12768, 2008.

[19] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *European Conference on Computer Vision*, pages 596–609. Springer, 2008.

[20] Laurenz Wiskott, Norbert Krüger, N Kuiger, and Christoph Von Der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):775–779, 1997.

[21] Lyudmila Yartseva and Matthias Grossglauser. On the performance of percolation graph matching. In *Proceedings of the First ACM Conference on Online Social Networks*, pages 119–130, 2013.

[22] Thomas Dullien Oriol Vinyals Yujia Li, Chenjie Gu and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. *ICML*, 2019.

תקציר

בבעיית ההתאמה של גרפים נתונים שני גרפים כך שיש צורך לבצע התאמה (מיפוי) בין קודקודי הגרפים על סמך המבנה הטופולוגי שלהם. השימושים לבעיה מגוונים מאוד ומגיעים מתחומים של תעשיית התרופות, זיהוי תמונה ועוד. הבעיה יכולה להופיע במגוון אופנים כמו תוויות על קודקודי הגרפים שמספקות מידע נוסף או קלט נוסף המכונה גרעין שמכיל זוגות מותאמים של קדקודים. אלגוריתמים שונים לבעיה מתכתבים עם גרפים בסדרי גודל שונים. כפונקציה בסיבוכיות זמן הריצה שלהם. בעבודה זו, אנו מתמקדים בפתרון פסאודו-לינארי שמסוגל להתמודד עם גרפים של מיליוני קדקודים. אנו מניחים קלט של גרעין ולא עושים שימוש בתוויות על קודקודי הגרפים.

השיטה היעילה והחסכונית ביותר בזמן ריצה להתאמת גרפים היא פעפוע הדרגתי. בשיטה זו אנו משתמשים בזוגות הקדקודים הממופים כדי לצבור ידע על הסביבה שלהם בגרפים וכך ממפים זוג קדקודים נוסף שממוקם בסמוך אליהם. הזוג הממופה משמש להרחבת הידע והסביבה, כך שניתן להוסיף עוד ועוד זוגות באופן חזרתי. למעשה השיטה מתבססת בכל שלב על הידע החלקי שצברנו עד כה במהלך האלגוריתם.

אנו מציגים בעבודה זו את אלגוריתם IRMA (Iterative Repair for graph MAtching). האלגוריתם מתבסס על שיטה של פעפוע, והחידוש המרכזי שלו הוא ביצוע איטרציות חוזרות לתיקון הדרגתי של המיפוי. האלגוריתם מתחיל על ידי הרצה של אלגוריתם התאמת גרפים קיים שמבוסס על שיטת פעפוע. לאחר השלמת בניית המיפוי, אנו משתמשים בידע הכולל שצברנו על מבנה הגרפים במהלך האלגוריתם, כדי ליצור מיפוי חדש לחלוטין. כיוון שהידע עליו התבססנו הינו שלם יותר, אנו מקבלים מיפוי מדויק יותר. כעת ניתן להשתמש במיפוי החדש כדי לקבל תמונת מצב אמינה יותר אודות הגרפים, מה שמאפשר בנייה של מיפוי חדש ומדויק יותר, וחוזר חלילה.

במהלך העבודה הוכחנו כי תוחלת מספר השגיאות שהאלגוריתם מבצע יורדת מאיטרציה לאיטרציה עד לכדי התכנסות. בניסויים שביצענו מצאנו שיפור משמעותי בין המיפוי ההתחלתי של האלגוריתם ועד לפלט הסופי שלאחר התיקונים. מספר האיטרציות הוא קבוע קטן, כך שסיבוכיות זמן הריצה נשארת אסימפטוטית זהה והאלגוריתם מאפשר ריצה על הגרפים הגדולים ביותר. בנוסף, מימשנו גרסה מקבילית של הפתרון המאפשרת שיפור משמעותי בזמן הריצה תמורת פגיעה זניחה באיכות הפתרון.

עבודה זו נעשתה בהדרכתם של פרופ׳ יורם לוזון ופרופ׳ ליעם רודיטי, הפקולטה למדעים מדויקים, אוניברסיטת בר-אילן.

אוניברסיטת בר אילן

תיקון מבוסס חזרות להתאמת גרפים בהינתן גרעין

ברק באביוב

עבודה זו מוגשת כחלק מהדרישות לשם קבלת תואר מוסמך
במחלקה למדעי המחשב של אוניברסיטת בר-אילן

רמת-גן, ישראל התשפ״ב

אוניברסיטת בר אילן

תיקון מבוסס חזרות להתאמת גרפים בהינתן גרעין

ברק באביוב

עבודה זו מוגשת כחלק מהדרישות לשם קבלת תואר מוסמך
במחלקה למדעי המחשב של אוניברסיטת בר-אילן

רמת-גן, ישראל התשפ״ב