Growing a Graph Matching from a Handful of Seeds

Ehsan Kazemi
School of Computer and Communication Sciences
EPFL
ehsan.kazemi@epfl.ch

S. Hamed Hassani
Department of Computer Science
ETHZ
hamed@inf.ethz.ch

Matthias Grossglauser
School of Computer and Communication Sciences
EPFL
matthias.grossglauser@epfl.ch

ABSTRACT

In many graph–mining problems, two networks from different domains have to be matched. In the absence of reliable node attributes, graph matching has to rely on only the link structures of the two networks, which amounts to a generalization of the classic graph isomorphism problem. Graph matching has applications in social–network reconciliation and de-anonymization, protein–network alignment in biology, and computer vision.

The most scalable graph–matching approaches use ideas from percolation theory, where a matched node pair "infects" neighbouring pairs as additional potential matches. This class of matching algorithm requires an initial seed set of known matches to start the percolation. The size and correctness of the matching is very sensitive to the size of the seed set.

In this paper, we give a new graph–matching algorithm that can operate with a much smaller seed set than previous approaches, with only a small increase in matching errors. We characterize a phase transition in matching performance as a function of the seed set size, using a random bigraph model and ideas from bootstrap percolation theory. We also show the excellent performance in matching several real large-scale social networks, using only a handful of seeds.

1. INTRODUCTION

Graph matching refers to the problem of identifying a bijection between the (full or partial) vertex sets of two graphs. Finding such a matching is particularly important and challenging when only the structures of the two graphs are available, i.e., the two graphs can be considered *unlabelled*. Obviously, the availability of node or edge attributes can only make the problem easier.

Formally speaking, the graph matching problem can be stated as follows: We are given two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, where some pairs of vertices $[i, j] \in V_1 \times V_2$ correspond to some unique underlying entity (e.g., a person).

Proceedings of the VLDB Endowment, Vol. 8, No. 10

In general, not all vertices in $V_{1,2}$ have a counterpart in the other graph.

The purpose of graph matching is to find the corresponding vertex pairs in $V_1 \times V_2$, based on the topologies of the two networks. As an example, consider G_1 to be the network of users in Twitter, and G_2 the network formed by the contact relationships of Flicker users. The sets V_1 and V_2 are only partially overlapping in general, because some users have an account on one but not the other service. The goal is to find the bijection between those users who have accounts on both Twitter and Flicker (users in $V_0 = V_1 \cap V_2$), based on the structural similarities of the two networks [19].

One line of work in this area begins with the assumption that there is side information in the form of a *seed set* of "pre–matched" node pairs, i.e., it assumes a (small) subset of nodes across the two graphs are identified a priori. The matching is generated incrementally, starting from the seed pairs and percolating to other node pairs; for this reason, we refer to this class of algorithms as Percolation Graph Matching (PGM) methods.

The pioneering work by Narayanan and Shmatikov [19] was based on a seed-based heuristic PGM algorithm, which succeeded in de-anonymizing social networks with millions of nodes. They empirically observed a strong sensitivity of their algorithm to the seed set size: if the seed set was too small, the percolation did not take off; when the seed set size was increased, there was an abrupt change to a supercritical regime, where the algorithm succeeded in de-anonymizing a large fraction of the network. In [30], the existence of such a phase transition in the seed set size was proven for a random–bigraph network model. A similar model was analyzed in [14] and extended to power–law graphs, under the assumption that seeds are dense (i.e., a constant fraction of nodes are seeds).

These PGM approaches have a basic feature in common: they build the matching incrementally between nodes of the two graphs. In every step, the set of node pairs matched so far are used as evidence to match an additional node pair, if possible. The evidence for deciding which pair to match can take different forms, but it is obtained locally within the two graphs. For example, in [30], the rule is extremely simple: (i) every seed pair is considered matched; (ii) a node pair is matched if it has at least r *already matched* neighbours[1] and i, j are not part of another matched pair already. The recursive application of rule (ii) can, under some conditions, match all the nodes.

[1]Two pairs $[i, j]$ and $[i', j']$ are called neighbours if there is an edge (i, i') in E_1, and an edge (j, j') in E_2.

In this paper, we give a new PGM algorithm. The distinguishing feature of this algorithm is that it requires a dramatically smaller number of seeds. Of course, there is a price to pay for this: whereas [30] proves that with high probability, the algorithm matches *every* node pair (i.e., zero errors) this performance criterion has to be slightly relaxed. Specifically, we will be content with a vanishing *fraction of errors* (w.h.p.). In summary, we manage to trade off a very significant reduction in the seed set size with a fairly benign increase in the error rate.

The reason why this works is quite subtle: For a PGM algorithm to succeed, two conditions have to be satisfied. First, the algorithm has to percolate: if at some point, there is not enough evidence to match a new pair, the algorithm stops. If this happens before a significant portion of the nodes have been matched, the algorithm has failed. Second, if the algorithm does percolate, it has to percolate correctly. If at some point, there is stronger evidence to match an incorrect pair than any correct pair, then the algorithm makes an error. Furthermore, this incorrect match may percolate to other incorrect pairs in future steps, thus potentially leading to a cascade of errors.

Clearly, there is a tradeoff between these two conditions. This tradeoff can be controlled by the strength of the evidence required before we decide to match a pair. For example, consider the parameter r above: if r is chosen quite high ($r = 5$, say), then percolation might easily stop early; however, a high r makes errors less likely; for $r = 2$, the algorithm percolates easily, but might often match wrong pairs.

In this paper, we control this tradeoff in a different way, by decoupling the decision to match a pair from its ability to infect other pairs. We refer to a tentative pair that is not yet matched as a *candidate pair*. Essentially, a candidate pair does provide evidence for other pairs, thereby fueling the percolation process, but this pair is not yet matched. It is not a priori obvious that this decoupling is a good idea; showing this is the key theoretical contribution of this paper. The reason is not obvious, and has to do with the way the evidence for correct and wrong pairs percolates. Basically, correct pairs tend to infect a small number of neighbouring correct pairs, each with relatively high probability; wrong pairs tend to infect only other wrong pairs, but crucially this effect is uniform over all wrong pairs and gets "diluted".

This observation leads to an algorithm that is *highly robust* to wrong candidate pairs. We prove that under a wide range of network parameters, with high probability this algorithm will percolate, generating a large number of *wrong candidate pairs* along the way. However, the majority of *matched pairs* are correct.

Our Contributions: In summary, our contributions are as follows:

- We develop a new graph–matching algorithm called `ExpandWhenStuck`. The distinguishing feature of this algorithm is that it requires a dramatically smaller number of seeds in comparison to state of the art algorithms [14, 30]. It is able to match real social networks with over a million nodes as well as various types of random graphs (for example, Barabási–Albert [4], Chung–Lu [6] and Erdös–Rényi [8] graphs) using only a handful of seeds (see Section 6).
- We analyze the performance of a simplified version of `ExpandWhenStuck` over an Erdös-Rényi random bigraph model with partial–overlapping vertex sets. The simplification needed to make the analysis tractable concerns the generation of candidate pairs: while `ExpandWhenStuck` dynamically percolates from unmatched candidate pairs whenever necessary, we can rigorously analyze only a slightly more restrictive setting, where this occurs only once at the outset. Specifically, the algorithm `ExpandOnce` expands the seed set into a larger set that includes many wrong pairs; a second algorithm `NoisySeeds` then percolates from this latter set in a manner similar to [30]. In Section 4, (i) we demonstrate a phase transition in the number of required seeds, as a function of the network size, overlap between the two graphs, and structural similarity, and (ii) we prove that the algorithm is robust to partial node–overlap. More precisely, we prove that it naturally filters out the nodes without counterparts in the other graph, and correctly matches the rest;

The remainder of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we explain our proposed PGM algorithm. In Section 4, we prove a performance guarantee for the algorithm of Section 3. In Section 5, using the ideas from previous sections, we present a heuristic algorithm whose performance is better in practice. In Section 6, we report the simulation results of our algorithms over real and random graphs. We compare our proposed algorithms with two state–of–the-art graph matching algorithms [14, 30] over several real graphs. In Section 7, we conclude the paper.

2. RELATED WORK

Graph matching is a generalization of the classic graph isomorphism problem where the vertex sets and edge sets of the two graphs are not exactly the same. Even though the graph isomorphism problem is considered very hard in general [9], for some instances of this problem there are efficient polynomial-time algorithms [2].

Graph matching is an important function in several scientific disciplines: In social networking, graph matching can break anonymization schemes if an adversary has a (noisy and incomplete) version of the network available as side information [3, 18, 19, 22, 29]; it also allows for the reconciliation of networks from different domains [14, 30]. In biology, the alignment of protein interaction networks can reveal functional equivalences of proteins in different species [13, 25]. In computer vision, an image–segment graph can serve as a fingerprint of a scene; graph matching can be used to find similar images in a database [7, 26, 28].

A class of graph–matching algorithms uses semantic information (e.g., name, location and image of users) for deanonymization of social networks [16, 20]. The authors in [17], introduce a similarity–flooding algorithm that matches nodes based on the spread of similarities in the network. Several machine–learning models are developed to match graphs based on the collected features about the nodes [1, 7, 20]. It has been shown that structural similarity is the most important feature in the graph–matching process [10] and structure–based algorithms are more accurate and scalable [3, 14, 30].

The analysis of iterative matching algorithms on large graphs using tools from percolation theory and random graphs has, of course, a rich history in the literature. For example, there is an important body of work on the design and analysis of gossip algorithms, whose purpose is to deliver a

message to the whole network as efficiently as possible [12, 24, 27].

In PGM algorithms, initial seeds play an important role. The seed pairs can be obtained in several ways, depending on the scenario: For example, some users of two different social networks might elect to make their identities public, which provides a set of known matches. Alternatively, methods have been proposed in the literature to identify plausible seed pairs based on structural graph features [3, 21] or manually through visual inspection [19].

3. ALGORITHMS

In this section, we define and explain the `ExpandOnce` matching algorithm, which performs one round of expansion of the initial seed set. This helps the percolation process overcome the bottleneck due to a small seed set size. This algorithm is kept deliberately simple for mathematical tractability. A more practical but heuristic algorithm based on the key ideas developed here will be presented in Section 5.

We also introduce a model to generate correlated graphs with partially overlapping vertex sets. Based on this model, we provide an intuitive justification for the performance of our approach. A rigorous analysis of our algorithm is then provided in Section 4.

3.1 Notation

Let us introduce the necessary notation. For the graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ let $V_0 = V_1 \cap V_2$. We assume $n_1 = |V_1|, n_2 = |V_2|$ and $n_0 = |V_0|$. Also, $d_{1,i}$ and $d_{2,j}$ denote degrees of nodes i and j in graphs G_1 and G_2, respectively. Let $(i, i') \in E_{1,2}$ represent an edge between two nodes i, i' in graph $G_{1,2}$; and $[i, j]$ represent a *pair* of nodes where $i \in V_1$ and $j \in V_2$. A pair $[i', j'] \in V_1 \times V_2$ is a neighbour of another pair $[i, j]$ if $(i, i') \in E_1$ and $(j, j') \in E_2$. In the description of the matching algorithms, we refer to a pair $[i, j]$ *spreading out marks* as adding one mark to each neighbouring pair of $[i, j]$. The score of a pair is defined as the number of marks it received from other pairs so far.

For convenience of notation, we assume without loss of generality (w.l.o.g.) that the hidden correct mapping between the nodes in V_0 is the identity mapping. Therefore, a pair is correct iff it is of the form $[i, i]$. Let $\Lambda(\mathcal{S})$ denote the number of correct pairs in a set $\mathcal{S}$ of pairs, and let $\Psi(\mathcal{S})$ represent the number of wrong pairs. Also, $V_1(\mathcal{S})$ is the set of nodes from graph G_1 in a set of pairs $\mathcal{S}$, i.e., $V_1(\mathcal{S}) = \{i | \exists j \text{ s.t. } [i, j] \in \mathcal{S}\}$. Similarly, we define $V_2(\mathcal{S}) = \{j | \exists i \text{ s.t. } [i, j] \in \mathcal{S}\}$.

3.2 NoisySeeds Algorithm

In this section, we give a new PGM algorithm called `NoisySeeds`. Before that, for the sake of better illustration, we first explain the algorithm from [30], which in this paper we refer to as `PercolateMatched`.

`PercolateMatched` starts from an initial seed set (a predefined matching) and iteratively matches pairs having at least r matched neighbours. More specifically: (i) Initially we are given as inputs a set of a_0 predefined (and correct) matched pairs called seed set $\mathcal{A} = \mathcal{A}_0$ ($|\mathcal{A}_0| = a_0$), and a fixed threshold r; (ii) at each time step τ, the algorithm picks an unused pair from set $\mathcal{A}$ and spread marks to all of its neighbouring pairs; (iii) as soon as a pair obtains at least r marks, i.e., it is a neighbour of at least r used pairs in the set $\mathcal{A}$, it will be added to the set $\mathcal{A}$; and (iv) the algorithm stops when there exist no further unused pairs in the set $\mathcal{A}$. The `User-Matching` algorithm [14] is similar to `PercolateMatched` with a slight difference: nodes are matched in several rounds based on a simple degree–bucketing method that matches high–degree nodes first.

The success of the `PercolateMatched` algorithm heavily relies on the condition that all the matched pairs (including the initial seeded pairs) are indeed correct pairs. In order for `PercolateMatched` to succeed, this condition then results in some constraints on r, namely $r \geq 4$. Our main contribution is to show that (i) the matching process can be made robust to a large number of wrong pairs in the seed set, provided there are enough correct pairs in the seed set as well; and (ii) $r = 2$ is sufficient to match almost all the nodes correctly.

The `NoisySeeds` algorithm (see Algorithm 1) starts with an initial noisy seed set $\mathcal{A}_0$, i.e., a set with possibly many wrong pairs. First, the marks coming from all the pairs in $\mathcal{A}_0$ are computed at the beginning (lines 1–4) and all these pairs are added to the set of used pairs $\mathcal{Z}$ (line 5). The algorithm proceeds as follows. We consider a set of matched pairs, denoted by $\mathcal{M}$, which is initially empty. If there is any pair with score at least r, then we add this pair to the matched set $\mathcal{M}$. Each time a pair $[i, j] \in \mathcal{M} \setminus \mathcal{Z}$ is chosen randomly, it spreads marks to its neighbouring pairs and is added to $\mathcal{Z}$. As the pair $[i, j]$ is in the matching $\mathcal{M}$, any other pair in the form of $[i, j']$ or $[i', j]$ is not a candidate for matching any longer and is permanently removed from consideration.

The algorithm stops if there is no remaining unused pair with score at least r. Note that since not all the pairs in the noisy set $\mathcal{A}_0$ are necessarily correct, they are not added to the matched set initially, i.e., the matched set is decoupled from the seed set. These pairs are used only for the sake of creating an initial set of marks for different pairs associated with the two graphs.

EXAMPLE 1. *The execution of* `NoisySeeds` *after four iterations (for $r = 2$) is illustrated in Figure 1.* `NoisySeeds` *begins by spreading marks from the initial noisy seed set (dark green and red nodes in Figure 1). Afterwards, all the newly matched pairs (light green and red nodes in Figure 1) are added to the seed set, and the matching process continues by spreading out their marks.*

3.3 The Random Graph Model

A convenient way to evaluate graph–matching algorithms is to analyze their performance over the $G(n, p; t, s)$ model which is a parsimonious model to generate two correlated graphs with partially overlapping vertex sets. This model is a generalization of the model in [22], which generates two correlated graphs with exactly the same vertex sets. The $G(n, p; t, s)$ bigraph model generates two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ as follows: (i) A graph $G(V, E)$ is sampled from an Erdös-Rényi $G(n, p)$ graph [8]. In a $G(n, p)$ graph with n nodes, each of the $\binom{n}{2}$ possible edges occurs independently with probability $0 < p < 1$; (ii) vertex sets $V_{1,2}$ are sampled independently with probability t from V; (iii) edges of graph $G_1(V_1, E_1)$ are sampled from those edges of graph G whose both endpoints are sampled in V_1 by independent edge sampling processes with probability s. The edges of graph $G_2(V_2, E_2)$ are generated similarly. The objective of graph–matching is to identify the hidden mapping between

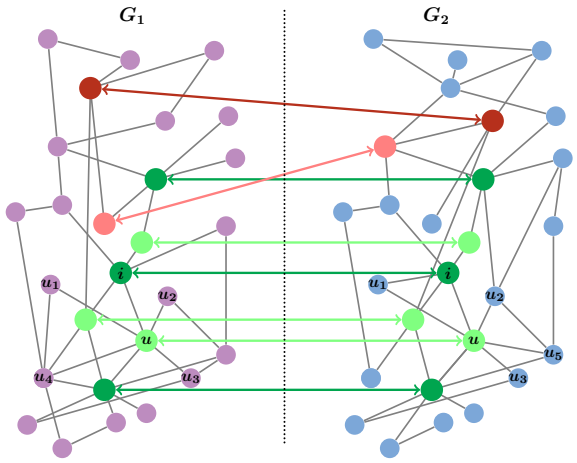

Figure 1: `NoisySeeds` Algorithm: Dark green and dark red nodes correspond to the initial correct and wrong seed pairs, respectively. After the first four iterations (for $r = 2$), light green nodes form correctly matched pairs, and light red nodes form wrongly matched pair (see Example 1).

those nodes which are in the intersection of the vertex sets of the two graphs.

In Section 4, we prove the robustness of `NoisySeeds` under the $G(n, p; t, s)$ model. An intuitive explanation for this robustness is as follows: (i) A correct pair obtains a mark from any other correct pair with probability ps^2. Also, a wrong pair obtains a mark from any other wrong or correct pair with probability $p^2s^2 \ll ps^2$. Thus, the effect of spreading marks from a wrong pair compared to a correct pair is negligible; (ii) Consider a pair which contains a node without any counterpart in the other graph (this is necessarily a wrong pair). This pair obtains $r \geq 2$ marks from any other r pairs with probability at most $O(p^4s^4)$. Therefore, only a small fraction of wrong pairs (in expectation $O(n_1^2n_2^2p^4s^4)$) obtains more than one mark.

3.4 ExpandOnce Algorithm

In this section, we introduce the `ExpandOnce` algorithm, which trades–off a small decrease in precision relative to `PercolateMatched` with a dramatic reduction in the seed set size. This algorithm accepts as input a seed set $\mathcal{A}_0$ of correct pairs. It expands the seed set $\mathcal{A}_0$ to a larger set of candidate pairs $\mathcal{A}'_0$ of size a'. Then, it runs `NoisySeeds` over the expanded seed set. In other words, in its first step, `ExpandOnce` creates from a small set of correct pairs ($\mathcal{A}_0$) a larger set of candidate pairs $\mathcal{A}'_0$, many of which are wrong in general. In Section 4, we will prove that these wrong pairs in $\mathcal{A}'_0$ have only a negligible effect on the performance of the matching process in `ExpandOnce`. Also, the new correct pairs in $\mathcal{A}'_0$ (rather than the ones from $\mathcal{A}_0$) enable the percolation process to kick-off. As a result, by calling `NoisySeeds` on the expanded seed set $\mathcal{A}'_0$ we obtain a successful matching of the two graphs. In summary, the process of expanding correct pairs to a noisy seed set allows us to successfully match graphs with much fewer initial seeds.[2] Algorithm 2 explains `ExpandOnce` in detail.

[2]Experiments over different types of graphs show that expanding an initial correct seed set $\mathcal{A}_0$ to the noisy seed set $\mathcal{A}'_0$ whose size is of order of $\min(n_1, n_2)$ results in an excellent matching performance.

Algorithm 1: `NoisySeeds`

Input: $G_1(V_1, E_1), G_2(V_2, E_2)$, noisy seed set $\mathcal{A}_0$ and threshold r
Output: The set of matched pairs $\mathcal{M}$

1 **for** *all pairs* $[i, j] \in \mathcal{A}_0$ **do**
2 add one mark to all the neighbouring pairs of $[i, j]$;
3 **if** *score of a pair* $[i', j'] \geq r$ ***and*** $i' \notin V_1(\mathcal{M})$ ***and*** $j' \notin V_2(\mathcal{M})$ **then**
4 add $[i', j']$ to the set $\mathcal{M}$;
5 $\mathcal{Z} \leftarrow \mathcal{A}_0$;
6 **while** $\mathcal{M} \setminus \mathcal{Z} \neq \emptyset$ **do**
7 randomly choose a pair $[i, j] \in \mathcal{M} \setminus \mathcal{Z}$ and add $[i, j]$ to the set $\mathcal{Z}$;
8 add one mark to all the neighbouring pairs of $[i, j]$;
9 **if** *score of a pair* $[i', j'] \geq r$ ***and*** $i' \notin V_1(\mathcal{M})$ ***and*** $j' \notin V_2(\mathcal{M})$ **then**
10 add $[i', j']$ to the set $\mathcal{M}$;
11 **return** $\mathcal{M}$;

Algorithm 2: `ExpandOnce`

Input: $G_1(V_1, E_1), G_2(V_2, E_2)$, seed set $\mathcal{A}_0$ of correct pairs, integer value $a' \geq 1$ and threshold r
Output: The set of matched pairs $\mathcal{M}$

1 $\mathcal{A}'_0 \leftarrow \mathcal{A}_0$ and $\mathcal{A} \leftarrow \mathcal{A}_0$;
2 **while** $|\mathcal{A}'_0| < a'$ **do**
3 $\mathcal{Z} \leftarrow \emptyset$ and $\mathcal{U} \leftarrow \mathcal{A}'_0$;
4 **for** *all pairs* $[i, j] \in \mathcal{A}$ **do**
5 **for** *all neighbouring pairs* $[i', j']$ *of pair* $[i, j]$ **do**
6 **if** $|\mathcal{A}'_0| < a'$ ***and*** $i' \notin V_1(\mathcal{U})$ ***and*** $j' \notin V_2(\mathcal{U})$ **then**
7 add $[i', j']$ to $\mathcal{A}'_0$ and $\mathcal{Z}$;
8 $\mathcal{A} \leftarrow \mathcal{Z}$;
9 **return** $\mathcal{M} \leftarrow$ `NoisySeeds`$(G_1, G_2, \mathcal{A}'_0, r)$;

4. PERFORMANCE OF MATCHING WITH NOISY SEEDS

In this section, (i) we identify a phase transition in the seed set size of `NoisySeeds` (explained in Algorithm 1); we prove (ii) `NoisySeeds` correctly matches almost all the nodes which are in the intersection of the two graphs and filters–out the nodes without counterparts in the other graph; and (iii) the addition of many wrong pairs to the initial correct seed set $\mathcal{A}_0$ of `NoisySeeds` would have a negligible effect on the performance of this algorithm.

The robustness guarantee for `NoisySeeds` with respect to noisy seed set justifies why `ExpandOnce` (Algorithm 2) requires a small set of initial seeds. Indeed, lines 2 to 8 of `ExpandOnce` turns a small set of clean seeds into a large noisy seed set that contains both correct and wrong pairs. This new set is then fed into `NoisySeeds` as an input and, as this algorithm is robust to wrong pairs, it succeeds with high probability.

Here, for the sake of analysis, we assume that $\mathcal{A}_0$ is a random set in the following sense: each correct pair $[i,i] \in V_0^2$ is placed in $\mathcal{A}_0$ independently, with probability $\frac{c}{n}$. Also, each wrong pair $[i,j]$, $i \neq j \in V_1 \times V_2$, is placed in $\mathcal{A}_0$ independently with probability at most $\frac{w}{n}$. Hence, we expect c correct pairs and at most wn wrong pairs as the initial noisy seed set for `NoisySeeds` algorithm.[3] Throughout this section we assume that the number of nodes n and average degree np tend to infinity. We also assume that the nodes and edge sampling probabilities $0 < t, s \leq 1$ are arbitrary constants. Let $\mathcal{Z}_\tau$ and $\mathcal{M}_\tau$ be respectively the set of used and matched pairs at time step τ of `NoisySeeds`. Also, let $\mathcal{M}^*$ denote the final set of matched pairs from `NoisySeeds`. We now state our main theorem.

We first define two parameters $b_{t,s,r}$ and $a_{t,s,r}$ [11, 30]:

$$b_{t,s,r} = \left[\frac{(r-1)!}{nt^2(ps^2)^r}\right]^{\frac{1}{r-1}} \text{ and } a_{t,s,r} = (1-\frac{1}{r})b_{t,s,r}. \quad (1)$$

THEOREM 1 (ROBUSTNESS OF `NoisySeeds`). *For an arbitrarily small but fixed $\frac{1}{6} > \epsilon > 0$, assume that $n^{-1} \ll p \leq n^{-\frac{5}{6}-\epsilon}$. If all the pairs in the noisy seed set $\mathcal{A}_0$ are chosen uniformly at random, $\mathbb{E}[\Lambda(\mathcal{A}_0)] > (1+\epsilon)a_{t,s,r}$ and $\mathbb{E}[\Psi(\mathcal{A}_0)] \leq wn$ for a constant w, then with high probability the `NoisySeeds` algorithm percolates and the size of its final matching is $nt^2 \pm o(n)$ with $\Lambda(\mathcal{M}^*) = nt^2 \pm o(n)$ and $\Psi(\mathcal{M}^*) = o(n)$.*

The authors in [30] proved that in order to match almost all the nodes correctly under the limited assumptions: (i) the vertex sets of the two graphs $G_{1,2}$ are exactly the same, i.e., $t = 1$, and (ii) no wrong pair in the initial seed set, i.e., $\Psi(\mathcal{A}_0) = 0$, the size of seed set should be at least $a_{1,s,4}$ ($r = 4$). For this special case, Theorem 1 guarantees that a seed set of size $a_{1,s,2}$ ($r = 2$) is enough for matching almost all the nodes correctly with a vanishing fraction of errors. Note further that the ratio $a_{1,s,4}/a_{1,s,2}$ goes to infinity.

Next, we prove Theorem 1 for the case $r = 2$ which needs the least number of seeds, i,e., $a_{t,s,2}$ is the least when $r = 2$. Generalization for values $r > 2$ is straightforward. For ease of notation, we define $a_c = a_{t,s,2} = \frac{1}{2nt^2p^2s^4}$. We first provide a brief sketch for the proof of Theorem 1. The detailed proof is given afterwards.

4.1 Sketch of the Proof

In the beginning of `NoisySeeds` algorithm, all the pairs in the seed set $\mathcal{A}_0$ spread out marks to their neighbouring pairs. At each time step $\tau \geq 1$, one pair from $\mathcal{M}_\tau \setminus \mathcal{Z}_\tau$ is picked and spreads out marks to its neighbouring pairs. It is easy to see that the matching process stops at a time step T^*, where $|\mathcal{M}^*| = |\mathcal{M}_{T^*}| = T^*$ (i.e., T^* is the first time when all the pairs inside $\mathcal{M}_{T^*}$ have already been picked). Note that T^* is at most $\min(n_1, n_2)$, as each node can be matched at most once. In order to prove Theorem 1, we show that with high probability (w.h.p.) $T^* = nt^2 \pm o(n)$, and the number of wrong matched pairs is at most $o(a_c)$. More precisely, the proof can be summarized in the following two steps:

(a) We provide an upper-bound on the number of wrong matched pairs at each step of the algorithm through computing its expected value. Using this upper bound we prove that the effect of wrong pairs is negligible in the final result of `NoisySeeds` (see Lemmas 1-3).

(b) By using step (a) and the results from bootstrap percolation process [30, 11] we prove that w.h.p. the correct pairs percolate as their initial number $\Lambda(\mathcal{A}_0)$ is more than the percolation threshold a_c. Therefore, as a result of percolation of the correct pairs, at time T^* the number of correct matched pairs is $nt^2 \pm o(n)$.

[3]Note that in general the algorithm is robust to a number of additional wrong pairs (in the seed set) which scales with n and p. Here, we have chosen this number to be wn in order to simplify our statements.

4.2 Proof of Theorem 1

Let us first introduce the notations used in this subsection. For an integer ℓ let $\mathcal{P}_{\ell,\tau}$ denote the set of pairs with score ℓ at time step τ; also let $\mathcal{P}_{\geq\ell,\tau}$ be the set of pairs with score at least ℓ at time τ. We let $\mathcal{Z}_\tau$ and $\mathcal{M}_\tau$ be the set of used and matched pairs at time step τ, respectively. Assume the time τ_{cip} corresponds to the completion of the initial phase (cip) of the algorithm, i.e., at the time τ_{cip} all the initial seeds are used for spreading out marks. All the other notations are explained in Section 3.1.

In the beginning of `NoisySeeds` (lines 1 to 4), all the pairs in the seed set $\mathcal{A}_0$ spread out marks to their neighbouring pairs. Afterwards, at each time step $\tau \geq 1$ (lines 6–10 in `NoisySeeds`), one pair from $\mathcal{M}_\tau \setminus \mathcal{Z}_\tau$ is picked and spreads marks to its neighbouring pairs. The matching process stops at a time step T^*, where $|\mathcal{M}^*| = |\mathcal{M}_{T^*}| = T^*$. Note that T^* is at most $\min(n_1, n_2)$, as each node can be matched at most once. In order to prove Theorem 1, we will show that w.h.p. $T^* = n_0 - o(n)$. Using the Chernoff bound, with high probability $n_0 = nt^2 \pm o(n)$. Therefore, we have $T^* = nt^2 \pm o(n)$, and the number of wrong matched pairs is at most $o(a_c)$. More precisely, we bound the number of wrong matched pairs at each step through computing their expected value. Using this upper bound we prove that the effect of wrong pairs is negligible. Also, we prove that the correct pairs percolate as their initial number $\Lambda(\mathcal{A}_0)$ is more than the percolation threshold a_c.

We proceed by computing the expected number of wrong matched pairs at time τ_{cip}.

Lemma 1. $\mathbb{E}\left[\Psi(\mathcal{M}_{\tau_{\text{cip}}})\right] = O(w^2n^4p^4s^4t^2) = o(a_c)$.

PROOF. We first recall that the time τ_{cip} corresponds to the completion of the initial phase (cip) of `NoisySeeds` algorithm. We define the random variables $X_{i,j}, i \neq j$ as

$$X_{i,j} = \begin{cases} 1 & \text{if } [i,j] \in \mathcal{P}_{\geq 2,\tau_{\text{cip}}} \\ 0 & \text{o.w.} \end{cases}$$

and $X = \sum_{\forall [i,j], i\neq j} X_{i,j}$. Note that as each node can be matched at most once, X is an upper bound for the total number of wrong matched pairs at time τ_{cip}, i.e., $\Psi(\mathcal{M}_{\tau_{\text{cip}}})$. Therefore, we have

$$\mathbb{E}\left[\Psi(\mathcal{M}_{\tau_{\text{cip}}})\right] \leq \mathbb{E}[X] = \sum_{\forall [i,j], i\neq j} \mathbb{E}[X_{i,j}] \leq n_1 n_2 \mathbb{P}\left[[i,j] \in \mathcal{P}_{\geq 2,\tau_{\text{cip}}}\right]. \quad (2)$$

We will prove that

$$\mathbb{P}\left[[i,j] \in \mathcal{P}_{2,\tau_{\text{cip}}}\right] = O(n^2w^2p^4s^4), \quad (3)$$

and for all $3 \leq r \leq n$

$$\mathbb{P}\left[[i,j] \in \mathcal{P}_{r,\tau_{\text{cip}}}\right] = O(n^rw^rp^{2r}s^{2r}). \quad (4)$$

Based on the assumption on the value of p in the statement of Theorem 1 and (4), we can show that for $r > 3$, $\mathbb{P}\left[[i,j] \in\right.$

$\mathcal{P}_{r,\tau_{\text{cip}}}\big] = O(nw^2p^4s^4)$. Therefore, by using a union bound, the probability that a pair $[i,j], i \neq j$ obtains at least two marks is bounded from above by

$$\mathbb{P}\big[[i,j] \in \mathcal{P}_{\geq 2,\tau_{\text{cip}}}\big] \leq \sum_{r=2}^{n} \mathbb{P}\big[[i,j] \in \mathcal{P}_{r,\tau_{\text{cip}}}\big] = O(n^2w^2p^4s^4),$$

and consequently

$$\mathbb{E}\left[\Psi(\mathcal{M}_{\tau_{\text{cip}}})\right] \leq n_1n_2\mathbb{P}\big[[i,j] \in \mathcal{P}_{\geq 2,\tau_{\text{cip}}}\big] = O(w^2n^4p^4s^4t^2).$$

This proves Lemma 1.

We next prove Equation (3); Equation (4) is proven in a similar way. Consider a pair $[i,j]$, $i \neq j$. The score of this pair is two if there exist two other pairs $[u_1,v_1]$ and $[u_2,v_2]$ in the seed set $\mathcal{A}_0$ such that $(i,u_1),(i,u_2) \in E_1$ and $(j,v_1),(j,v_2) \in E_2$. Note that as a pair $[i,j]$ is added to the matching $\mathcal{M}$, any other pair in the form of $[i,j']$ or $[i',j]$ can not also be in the matching $\mathcal{M}$ and will be discarded to ensure that each node is matched at most once. Hence, for the sake of analysis, we assume all the marks that were previously created from all the pairs that have the form of $[i,j']$ or $[i',j]$ are subtracted.[4]

Let us first assume that $i \notin \{v_1,v_2\}$ and $j \notin \{u_1,u_2\}$. We consider three cases: (i) All the four nodes u_1,u_2,v_1 and v_2 are different: in this case the edges $(i,u_1),(i,u_2) \in E_1$ and $(j,v_1),(j,v_2) \in E_2$ exist independently. Thus the probability that $[i,j]$ obtains two marks from these two pairs is p^4s^4. The number of such pairs $[u_1,v_1],[u_2,v_2]$ is at most $(n_1n_2)^2$, and each such pair is in the seed set with probability $\frac{wn}{n_1n_2}$. Thus the probability that $[i,j]$ obtains two marks from such pairs is bounded from above by $(n_1n_2)^2(\frac{wn}{n_1n_2})^2p^4s^4 = O(w^2n^2p^4s^4)$. (ii) $u_1 \neq v_1$ and $u_2 \neq v_2$, or (iii) Either $u_1 = v_1$, $u_2 = v_2$ or both: Along the same lines as above, it is easy to see the probability that $[i,j]$ obtains two marks is upper bounded by $O(w^2np^3s^3)$.

Now, assume $i \in \{v_1,v_2\}$ or $j \in \{u_1,u_2\}$, similarly as above we upper bound the probability that a pair $[i,j]$ obtains two marks from the pairs $[u_1,v_1]$ and $[u_2,v_2]$. To summarize, by considering all the cases mentioned above, the probability that a pair $[i,j]$ obtains two marks at time τ_{cip} is bounded from above by $O((n_1n_2)^2p^4s^4(\frac{wn}{n_1n_2})^2[1+\frac{1}{np}]) = O(n^2w^2p^4s^4)$. This proves (3). □

The next step is to prove that the number of wrong matched pairs at each time step $1 \leq \tau \leq T^*$ of the matching process is at most $O(w^2n^4p^4s^4) = o(a_c)$. At each time step $\tau \geq 1$, `NoisySeeds` picks a random pair $[i,j] \in \mathcal{M}_\tau \setminus \mathcal{Z}_\tau$ and adds one mark to its neighbouring pairs. It is easy to see that $\Lambda(\mathcal{M}_\tau)$ and $\Psi(\mathcal{M}_\tau)$ are increasing by τ. In Lemma 3 stated below, with using Markov's inequality and the results of Lemmas 1 and 2, we will prove that $\Psi(\mathcal{M}^*) = \Psi(\mathcal{M}_{T^*}) = o(a_c)$. Consequently, from monotonicity of $\Psi(\mathcal{M}_\tau)$ with respect to τ, we conclude that $\Psi(\mathcal{M}_\tau) = o(a_c)$ holds for all $1 \leq \tau \leq T^*$.

Lemma 2. $\mathbb{E}\left[\Psi(\mathcal{M}_{T^*})\right] = O(w^2n^4p^4s^4t^2) = o(a_c)$.

Proof. We define the random variables $X_{i,j}, i \neq j$, as

$$X_{i,j} = \begin{cases} 1 & \text{if } [i,j] \in \mathcal{P}_{\geq 2,T^*} \\ 0 & \text{o.w.} \end{cases}$$

and let $X = \sum_{\forall [i,j], i \neq j} X_{i,j}$. In words, the random variable $X_{i,j}$ indicates whether an individual pair $[i,j]$ can collect at least two marks during the steps of `NoisySeeds`. Of course as each node can be matched at most once, not all pairs in $\mathcal{P}_{\geq 2,T^*}$ will end up in the final match set. Hence we have

$$\mathbb{E}\left[\Psi(\mathcal{M}_{T^*})\right] \leq \sum_{i \neq j} \mathbb{E}\left[X_{i,j}\right] \leq n_1n_2\mathbb{P}\big[[i,j] \in \mathcal{P}_{\geq 2,T^*}\big]. \quad (5)$$

We thus proceed by finding an upper bound for $\mathbb{P}\big[[i,j] \in \mathcal{P}_{\geq 2,T^*}\big]$. Let $\mathcal{P}_{q,\mathcal{M}^*}$ and $\mathcal{P}_{\geq q,\mathcal{M}^*}$ respectively, represent the set of pairs that obtain exactly q and at least q marks from all the T^* matched pairs $\mathcal{M}^* = \mathcal{M}_{T^*}$. Assuming $i \neq j$, the pair $[i,j]$ is in the set $\mathcal{P}_{\geq 2,T^*}$ if one of the three following cases holds (we thus can use the union bound for $\mathbb{P}\big[[i,j] \in \mathcal{P}_{\geq 2,T^*}\big]$):

Case 1. $[i,j] \in \mathcal{P}_{\geq 2,\tau_{\text{cip}}}$, i.e., $[i,j]$ obtains at least two marks from pairs in $\mathcal{A}_0$. This means that the pair $[i,j]$ is added to the set of matched pairs already at time step τ_{cip}. Indeed, for the result of Lemma 1, we have

$$\mathbb{P}\big[[i,j] \in \mathcal{P}_{\geq 2,\tau_{\text{cip}}}\big] = O(w^2n^2p^4s^4)$$

Case 2. $[i,j] \in \mathcal{P}_{\geq 2,\mathcal{M}^*}$, i.e., $[i,j]$ obtains at least two marks from all the matched pairs in $\mathcal{M}_{T^*}$ from time step $\tau = 1$ to $\tau = T^* \leq \min(n_1,n_2) = O(nt)$. To upper bound this probability, we consider two cases: $[i,j] \in \mathcal{P}_{2,\mathcal{M}^*}$, and $[i,j] \in \mathcal{P}_{r,\mathcal{M}^*}$ for $2 < r \leq n$. Let us first find an upper bound for the former. Assume pair $[i,j]$ obtains two marks from the pairs $[u_1,v_1]$ and $[u_2,v_2]$. As each node could be matched at most once, then i, u_1, and u_2 are mutually different. The same is true for j,v_1 and v_2. In this regard, there are three cases:

(i) Either $[u_1,v_1] = [j,i]$ or $[u_2,v_2] = [j,i]$. It is obvious that both cases cannot hold simultaneously. We assume w.l.o.g. that $[u_1,v_1] = [j,i]$, and thus $u_2 \neq j$ and $v_2 \neq i$. In this case each one of the edges $(i,j),(i,u_2),(j,v_2)$ is sampled in the underlying graph G independently with probability p. If an edge exists in the hidden underlying graph G, then it appears with probability s in each one of the sampled graphs G_1 or G_2. Therefore, pair $[i,j]$ obtains two marks with probability p^3s^4. As the pair $[u_1,v_1] = [j,i]$ is fixed, for pair $[u_2,v_2]$ we have at most $T^* - 1 = O(nt)$ choices.

It remains two other cases: (ii) Either $i = v_1$ and $j = u_2$, or $i = v_2$ and $j = u_1$(but not both), and (iii) $i \notin \{u_1,u_2\}$, and $j \notin \{v_1,v_2\}$. For the sake of space we omit the detailed explanation of these two cases. By using union bound the probability that a pair $[i,j]$ obtains two marks is bounded from above by $\mathbb{P}\left[[i,j] \in \mathcal{P}_{\geq 2,\mathcal{M}^*}\right] = O(n^2p^4s^4t^2)$.

Case 3. $[i,j]$ obtains one mark from the pairs in $\mathcal{A}_0$ and one mark from the matched pairs $\mathcal{M}_{T^*}$, i.e., $[i,j] \in \mathcal{P}_{1,\tau_{\text{cip}}} \cap \mathcal{P}_{1,\mathcal{M}^*}$. By same lines of reasoning as the two other cases we have

$$\mathbb{P}\big[[i,j] \in \mathcal{P}_{1,\mathcal{M}^*}, [i,j] \in \mathcal{P}_{1,\tau_{\text{cip}}}\big] = O(wn^2p^4s^4).$$

To wrap up, we proved $\mathbb{P}\big[[i,j] \in \mathcal{P}_{\geq 2,T^*}\big] = O(wn^2p^4s^4)$.

Finally, as a consequence of (5) we have $\mathbb{E}\big[\Psi(\mathcal{M}_{T^*})\big] = O(w^2n^4p^4s^4t^2)$. This proves Lemma 2. □

Lemma 3. With high probability we have

$$\Psi(\mathcal{M}_\tau) = o(a_c) \text{ and } \Psi(\mathcal{M}_{\tau_{\text{cip}}}) = o(a_c).$$

[4]We observe that in practice, this step only has a small effect on the performance, but is computationally costly.

PROOF. For any $\delta > 0$, by using Markov's inequality and Lemma 2, we have with high probability

$$\mathbb{P}\left[\frac{\Psi(\mathcal{M}_\tau)}{a_c} \geq \delta\right] \leq \delta^{-1}\mathbb{E}\left[\frac{\Psi(\mathcal{M}_\tau)}{a_c}\right] = O(n^{-\epsilon}) = o(1).$$

Similarly, by using Lemma 1, we obtain that with high probability $\Psi(\mathcal{M}_{\tau_{\text{cip}}}) = o(a_c)$. □

In the next step, we use Lemma 3 to prove Theorem 1. We first give a brief overview of bootstrap percolation [11]. Bootstrap percolation is the process of node activation on a $G(n,p)$ random graph [11]. In this process, initially we are given a set $\mathcal{A}(0)$ ($|\mathcal{A}(0)| = a_0$) of active nodes and a threshold $r \geq 2$. A node is activated at time step τ if at least r of its neighbours were activated and used in the previous τ time steps. Let $\mathcal{A}(\tau)$ and $\mathcal{Z}(\tau)$ denote the set of active and used nodes at time step τ. We assume $\mathcal{Z}(0) = \emptyset$. At each time step $\tau \geq 1$, we choose a node u_τ from $\mathcal{A}(\tau-1)\setminus\mathcal{Z}(\tau-1)$ and give each one of its neighbours a mark. We call u_τ a used node and update $\mathcal{Z}(\tau) = \mathcal{Z}(\tau-1) \cup u_\tau$. Assume $\Delta\mathcal{A}(\tau)$ is the set of activated nodes at time step τ and we let $\mathcal{A}(\tau) = \mathcal{A}(\tau-1) \cup \Delta\mathcal{A}(\tau)$. At each step τ (before the process stops) one node is added to the set of used nodes, i.e., $|\mathcal{Z}(\tau)| = \tau$. We define $A_{n,a}(\tau) = |\mathcal{A}(\tau)|$. Also, $T^*_{n,a}$ denote the time step when $A_{n,a}(T^*_{n,a}) = |\mathcal{Z}(T^*_{n,a})| = T^*_{n,a}$. The bootstrap percolation process stops when $\mathcal{A}(\tau) \setminus \mathcal{Z}(\tau) = \emptyset$ or equivalently $A_{n,a}(\tau) \leq \tau$. The phase transition threshold for bootstrap percolation is stated in the following theorem.

THEOREM 2 (THEOREM 3.1 AND LEMMA 8.2 OF [11]). *Assume* $b_{c,r} = n\frac{(pn)^{r-1}}{(r-1)!}e^{-pn}$, $\tau_{c,r} = \left[\frac{(r-1)!}{n(ps^2)^r}\right]^{\frac{1}{r-1}}$ *and* $a_{c,r} = (1-\frac{1}{r})\tau_{c,r}$, *and let* $b^* = b_{c,r}w(n)$, *where* $\omega(n) \to \infty$ *slowly but is otherwise arbitrary. Suppose that* $r \geq 2$ *and* $n^{-1} \ll p \ll n^{-1/r}$. *Then, for any* $a > a_{c,r}$, *w.h.p.* $A_{n,a}(\tau) > \tau$ *for all* $\tau \in [0, n-b^*]$.

Hence the Theorem 2 is valid for any choice of $\omega(n) \to \infty$; it is equivalent to the statement that for all $\tau \in [0, n-O(b_{c,r})]$ w.h.p. $A_{n,a}(\tau) > \tau$ [11]. It is easy to see that $O(b_{c,r}) = o(n)$ [11]. By analogy between graph matching problem over $G(n,p;t,s)$ graphs and the bootstrap percolation process on $G(n_0, ps^2)$ [30], for time steps $\tau \geq 1$ we have

$$\begin{aligned}
&\mathbb{P}\big[\Lambda(\mathcal{M}_\tau) > \tau\big] \\
&\overset{(a)}{\geq} \mathbb{P}\big[A_{n_0-2\Psi(\mathcal{M}_{T^*}),a_0}\big(\tau - 3\Psi(\mathcal{M}_{T^*}) + a_0\big) > \tau + a_0\big] \\
&\overset{(b)}{\geq} \mathbb{P}\big[A_{n_0-3\Psi(\mathcal{M}_{T^*}),a_0}\big(\tau - 3\Psi(\mathcal{M}_{T^*}) + a_0\big) > \tau + a_0\big] \\
&\overset{(c)}{\geq} \mathbb{P}\big[A_{n_0-6\Psi(\mathcal{M}_{T^*}),a_0-3\Psi(\mathcal{M}_{T^*})}\big(\tau - 3\Psi(\mathcal{M}_{T^*}) + a_0\big) \\
&\qquad\qquad > \tau - 3\Psi(\mathcal{M}_{T^*}) + a_0\big] \quad (6)
\end{aligned}$$

The three inequalities follow from the following reasons:

(a) We have a_0 initial correct pairs. For any time step τ we have $\Psi(\mathcal{M}_\tau) \leq \Psi(\mathcal{M}_{T^*}) = o(a_c)$ wrong matched pairs. Each wrong pair $[i,j], i \neq j$, can potentially remove marks produced by the two correct pairs $[i,i]$ and $[j,j]$ from the set of used pairs $\mathcal{Z}_\tau$. We know that among the matched pairs, there are at most $o(a_c)$ wrong pairs. Therefore, we conclude that there are at least $n_0 - 2\Psi(\mathcal{M}_{T^*})$ potential correct pairs that obtain marks from at least $\tau - 3\Psi(\mathcal{M}_{T^*}) + a_0$ correct pairs at time step τ.

(b) As we assume p and the number of initial active nodes are fixed, decreasing the total number of nodes by $\Psi(\mathcal{M}_{T^*})$ would increase the probability of the process stopping.

(c) If we assume at the first $3\Psi(\mathcal{M}_{T^*})$ steps of the bootstrap percolation the chosen nodes from $\mathcal{A}(\tau-1)\setminus\mathcal{Z}(\tau-1)$ do not spread out marks, then the probability of the process stopping would increase.

Note that $\Lambda(\mathcal{A}_0) = Binomial(n_0, \frac{c}{n_0})$ and $c > a_c \to \infty$, therefore, using the Chernoff bound we can conclude that for an arbitrarily small but fixed $\epsilon' > 0$ w.h.p. $a_0 = \Lambda(\mathcal{A}_0) > (1-\epsilon')\mathbb{E}[\Lambda(\mathcal{A}_0)] = (1-\epsilon')c$. Finally, if $a_0 = \Lambda(\mathcal{A}_0) > a_c - 3\Psi(\mathcal{M}_{T^*})$, then from (6) and Theorem 2 we conclude that w.h.p. $T^*_{n_0-6\Psi(\mathcal{M}_{T^*}),a_0} = n_0 - o(n)$. Also, (6) implies that w.h.p. $T^* \geq T^*_{n_0-6\Psi(\mathcal{M}_{T^*}),a_0}$. From Lemma 3 we know that at the time $T^* \leq \min(n_1,n_2)$ the number of wrong matched pairs is upper bounded by $\Psi(\mathcal{A}_{T^*}) = o(a_c)$, and $\Psi(\mathcal{A}_{T^*}) + \Lambda(\mathcal{A}_{T^*}) = T^*$. Thus, w.h.p. $\Lambda(\mathcal{M}_{T^*}) = n_0 - o(n)$ and $\Psi(\mathcal{A}_{T^*}) = o(a_c)$. Note that by using the Chernoff bound w.h.p. we obtain $n_0 = nt^2 \pm o(n)$. This proves Theorem 1.

5. ExpandWhenStuck HEURISTIC

In this section, we introduce a new robust algorithm, called `ExpandWhenStuck`, which is designed based on the robustness ideas developed in the previous sections. This algorithm is able to match real social–networks with over a million nodes using a small number of seeds (e.g., see Figure 8 and Table 1 in Section 6). In comparison with `ExpandOnce`, this algorithm has better performance for both real and random graphs, and its computational complexity is lower. However, we cannot formally characterize its performance. To better illustrate, let us briefly go back to the `PercolateMatched` algorithm described in the beginning of Section 3. In the sub-critical regime of `PercolateMatched`, the number of final matched pairs is at most twice the number of initial seeds [11]. The robustness arguments of Section 4 allow PGM algorithms to be much more aggressive in spreading out marks.

A main feature of `ExpandWhenStuck` is to expand the seed set by many noisy candidate pairs whenever there are no other unused matched pairs. More precisely, whenever there are no further pairs with score at least two, we add all the unused[5] and unmatched[6] neighbouring pairs of all the matched pairs to the candidate pairs (line 11 in Algorithm 3) and consequently new marks would be spread out. Among these candidate pairs, where a small fraction is correct and most of them are wrong, (i) correct pairs help us to continue the percolation process and match remaining unmatched pairs, and (ii) wrong pairs would have a negligible effect (see Theorem 1).

EXAMPLE 2. *When the percolation graph–matching process is stopped, there is still useful information that can help us match the remaining nodes. Assume, as in Figure 1, there are no unmatched pairs with score at least* $r = 2$. *Node* u *is one of the (correctly) matched nodes. Among all the 16 possible pairs between unmatched neighbours of node* u *(see Figure 2), three of them are correct (light green arrows) and the rest are wrong (light red arrows). Note that all these pairs have one mark. Therefore,* `ExpandWhenStuck` *adds them to the set of candidate pairs. As our algorithm is robust to the*

[5]A pair $[i,j]$ is unused if $[i,j] \notin \mathcal{Z}$.

[6]A pair $[i,j]$ is unmatched if $i \notin V_1(\mathcal{M})$ and $j \notin V_2(\mathcal{M})$.

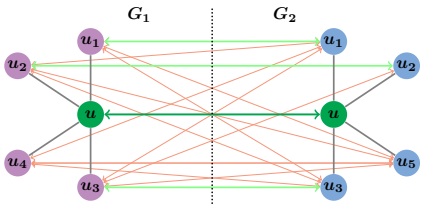

Figure 2: `ExpandWhenStuck` (Algorithm 3): Nodes u_1, u_2, u_3, u_4 and u_5 are unmatched neighbours of node u in the underlying graph G (see Example 2 and Figure 1).

wrong candidate pairs, correct candidate pairs can help us in the matching process.

In addition, to enhance the performance of our algorithm (especially for real graphs), we further make the following modification. At each time step, instead of adding all the candidate pairs with score at least two to the matched set, we choose the one with the highest score among such pairs and add it to the matched set; also, each node is matched at most once. We then proceed with spreading out the marks from this matched pair.

Most of the times (especially in the beginning) there are several pairs with the maximum score. Among all such candidate pairs $[i, j]$, we choose the pair that minimizes the difference in the degrees of nodes $|d_{1,i} - d_{2,j}|$. This can be intuitively justified as $d_{1,i}$ is often closer to $d_{2,j}$ when $[i, j]$ is a correct pair, i.e., $i = j$, than when $i \neq j$. This degree tie–break increases the performance, especially in real graphs, because their degree distributions are often heavily skewed and less concentrated compared to the $G(n, p)$ model. Algorithm 3 explains `ExpandWhenStuck` in detail.

Algorithm 3: `ExpandWhenStuck`

Input: $G_1(V_1, E_1), G_2(V_2, E_2)$, seed set $\mathcal{A}_0$ of correct pairs
Output: The set of matched pairs $\mathcal{M}$

$\mathcal{A} \leftarrow \mathcal{A}_0$ is the initial set of seed pairs, $\mathcal{M} \leftarrow \mathcal{A}_0$;
$\mathcal{Z} \leftarrow \emptyset$ is the set of used pairs;
while $|\mathcal{A}| > 0$ **do**
  for *all pairs* $[i, j] \in \mathcal{A}$ **do**
    add the pair $[i, j]$ to $\mathcal{Z}$ and add one mark to all of its neighbouring pairs;
  while *there exists an unmatched pair with score at least* 2 **do**
    among the pairs with the highest score select the unmatched pair $[i, j]$ with the minimum $|d_{1,i} - d_{2,j}|$;
    add $[i, j]$ to the set $\mathcal{M}$;
    if $[i, j] \notin \mathcal{Z}$ **then**
      add one mark to all of its neighbouring pairs and add the pair $[i, j]$ to $\mathcal{Z}$;
  $\mathcal{A} \leftarrow$ all neighbouring pairs $[i, j]$ of matched pairs $\mathcal{M}$ s.t. $[i, j] \notin \mathcal{Z}$, $i \notin V_1(\mathcal{M})$ and $j \notin V_2(\mathcal{M})$;
return $\mathcal{M}$;

6. SIMULATION RESULTS

In this section, we first demonstrate numerically the phase transitions of `NoisySeeds` given by Theorem 1. We next evaluate through experiments the performance of `ExpandOnce` and `ExpandWhenStuck` over the $G(n, p; t, s)$ model. We show that these two algorithms are able to match graphs with only a handful of seeds. To compare the performance of our algorithm with the other methods in the literature, simulation results for `ExpandWhenStuck` over power–law and preferential attachment random graphs, and real graphs are provided. Finally, we explain the MapReduce implementation of a variant of `ExpandWhenStuck`.

We use precision and recall to evaluate the performance of algorithms: (i) Precision refers to the fraction of errors in the set of matched nodes, and (ii) Recall is the fraction of nodes in the intersection of the two graphs $G_{1,2}$ which are matched correctly. Formally, they are defined as: precision $= \frac{\Lambda(\mathcal{M}^*)}{\Lambda(\mathcal{M}^*)+\Psi(\mathcal{M}^*)}$ and recall $= \frac{\Lambda(\mathcal{M}^*)}{n_{\text{ident}}}$ where n_{ident} is the number of nodes that are present in both graphs $G_{1,2}$ with degrees at least two (for other notations see Section 3.1).

6.1 Experimental Results with Random Graphs

The experiments in this part are performed over two different types of random graphs: (i) Erdös-Rényi graphs, and (ii) scale–free networks. Although the performance of our algorithm is guaranteed for the $G(n, p; t, s)$ model (see Theorem 1 in Section 4), simulation results show the excellent performance of our algorithm versus state-of-the-are graph matching algorithms for all types of studied graphs.

6.1.1 *Erdös-Rényi Random Graphs*

We first demonstrate numerically the phase transitions established in Theorem 1 for `NoisySeeds`. As shown in Theorem 1, for the $G(n, p; t, s)$ model such a transition takes place when the number of correct pairs in the initial seed set passes a certain threshold $a_{t,s,r}$ while there is possibly many wrong pairs in the seed set. In Figure 3, we plot the total number of correct matched pairs versus the normalized number of seeds (i.e., the number of correct seeds divided by $a_{t,s,r}$) for the set of parameters $n = 10^6, p = 20/n$ and different ranges of node and edge sampling probabilities. As can be seen, (i) the phase transitions take place close to the critical values of $a_{t,s,r}$ and (ii) the total number of correctly matched nodes is very close to the expected number of nodes in the intersection of the two vertex sets, i.e., nt^2. Note that in all the cases considered in Figure 3 the fraction of wrongly matched pairs is very small, i.e., the precision is close to one.

We now proceed with simulation results of `ExpandOnce` and `ExpandWhenStuck` to compare their performance over $G(n, p; t, s)$ model (we also compare with `PercolateMatched` [30]). Figures 4 confirms that the `ExpandOnce` algorithm performs surprisingly well and with only a handful of seeds (only 13, 67 and 235 seeds for $s^2 = 0.81, 0.64$ and 0.49, respectively) it can correctly match almost all the nodes in a graph with $n = 10^6$ nodes and average degree 20. Figure 5 shows that when the matching process percolates, the precision is close to one. As a comparison, if we set the minimum threshold $r = 2$, then the `PercolateMatched` algorithm [30] would need at least 1906, 3052 and 5207 seeds for matching $G(n, p; s)$ (equivalent to $t = 1$ in our model) graphs with edge overlap probabilities $s^2 = 0.81, 0.64$ and 0.49, respectively. Also, we observe that `ExpandWhenStuck` needs fewer seeds with respect to the `ExpandOnce` in order to match almost all the nodes correctly. `ExpandWhenStuck` for parameters $t^2 = 1.0$ and $s^2 = 0.81$ with only 8 seeds (i.e., a fraction $8 \cdot 10^{-6}$ of the total number of nodes), matches almost all

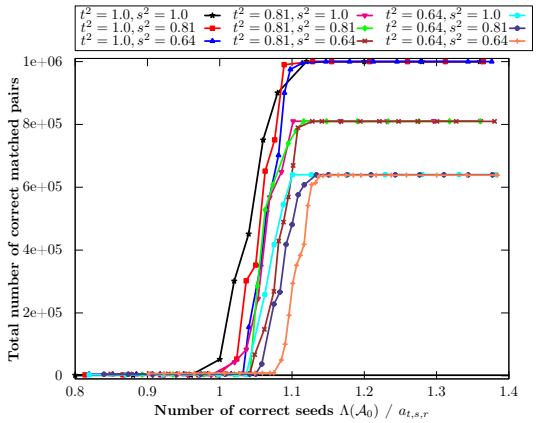

Figure 3: `NoisySeeds` Algorithm: Total number of correct matched pairs vs. number of seeds normalized by $a_{t,s,r}$ for $r = 2$. Simulations are done over $G(n, 20/n; t, s)$ with $n = 10^6$.

the nodes correctly, whereas for the `PercolateMatched` this number is at least 1906 (the threshold for $r = 2$). In other words, in this example, `ExpandWhenStuck` needs 238 times fewer seeds than `PercolateMatched`.

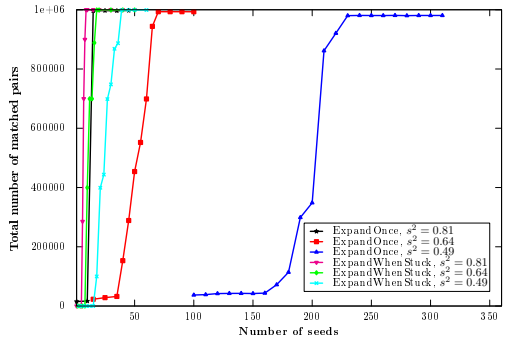

Figure 4: Total number of matched pairs vs. number of seeds. Simulations are done over $G(n, 20/n; t, s)$ with $t = 1$ and $n = 10^6$.

6.1.2 *Scale–Free Random Graphs*

We evaluate `ExpandWhenStuck` over scale–free random graphs as better representative of real–world (e.g., social and biological) networks. Note that as Erdös-Rényi graphs contain less structural information (for example, degree distribution is concentrated around the mean and a low clustering coefficient) it is harder to match them. Also, simulation results confirm that matching scale–free networks is an easier task.

First, we apply `ExpandWhenStuck` algorithm to the Chung–Lu graphs [6] (a variant of power–law random graphs). In these graphs, the degree distribution of nodes follows a power–law distribution, i.e., the proportion of nodes of degree d scales like $d^{-\beta}$. In this model, the probability of having an edge between two nodes i and j with degrees d_i and d_j, which is independent of all the other edges in the graph, is proportional to $d_i d_j$. We generate two graphs $G_{1,2}$ through node–sampling with probability t and edge–sampling with probability s over a Chung–Lu graph. In Figure 6, for example, we observe that with only 20 seeds `ExpandWhenStuck` matches almost all the nodes for fairly small node and edge overlap probabilities being equal to 0.75. In all our experiments, we observe that precision is always better than recall.

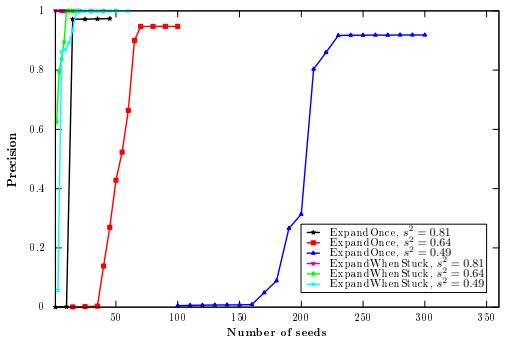

Figure 5: Precision vs. number of seeds. Simulations are done over $G(n, 20/n; t, s)$ with $t = 1$ and $n = 10^6$.

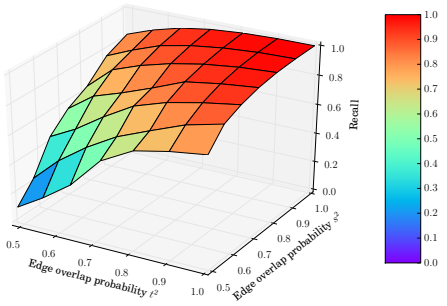

Figure 6: Recall vs. node and edge overlap probabilities (i.e., t^2 and s^2). Number of seeds is 20. Simulations are done over power–law (Chung–Lu) random graphs with $n = 10^5$, $\beta = 2.5$ and average degree 20.

Next, we apply `ExpandWhenStuck` to the preferential attachment random graphs. The Barabási–Albert model [4] is one of the most referred models for social networks. This model generates random scale–free networks in a preferential attachment setting. A Barabási–Albert (BA) random graph is generated as follows [5]: (i) It starts with a a single node with m self–loops; and (ii) each new node is connected to m existing nodes with probabilities proportional to their current degrees. Figure 7 shows the simulation result of `ExpandWhenStuck` over BA random graphs. In these experiments, the underlying graph G is sampled from BA model. The two graphs G_1 and G_2 are generated by independent node and edge sampling processes from graph G.

Our experiments show that choosing seeds among high–degree nodes, instead of picking them randomly, results in better matchings. For example, given only the highest–degree node as seed is enough to match almost all the nodes correctly in Chung–Lu and BA graphs with $n = 10^6$, average degree 20, and sampling probabilities $t^2 = 0.81$ and $s^2 = 0.81$.

6.2 Experimental Results with Real Graphs

In this section, we illustrate the experimental results of `ExpandWhenStuck` algorithm over five different real social networks. The baseline for our comparisons are state–of–the–art graph–matching algorithms: (i) `PercolateMatched` [30] (here, by `PercolateMatched` we mean a deferred version of it which has been reported to have better performance compared to its basic version), and (ii) `User-Matching` [14]. In all our experiments, `PercolateMatched` [30] outperforms

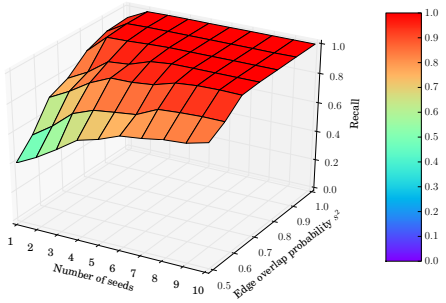

Figure 7: `ExpandWhenStuck` algorithm: Recall vs. edge overlap probabilities (i.e., s^2) and number of seeds. Simulations are done over Barabási–Albert random graphs with edge overlap probability $t^2 = 0.81$, $n = 10^5$ and average degree 20.

`User-Matching` [14]. Therefore, due to space limitation we only plot the results corresponding to `PercolateMatched` algorithm in some figures.

In statistical analysis and machine learning, F_1–score combines both the precision and the recall in one metric to provide an average of them [23]. This measure is defined as

$$F_1\text{–score} = 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \quad (7)$$

The value of F_1–score is between 0 and 1. When F_1–score is close to 1, we can conclude that (i) precision is close to 1, i.e., the fraction of errors in the set of matched pairs is small, and (ii) recall is close to 1, i.e., a large fraction of nodes which are present in the both graphs $G_{1,2}$ are matched correctly. We use this measure to compare the performance of algorithms.

For the first experiment, we choose a very large real graph. We run `ExpandWhenStuck` over Youtube graph with 1134890 nodes and average degree 5.26 [15]. In this graph links correspond to friendships among users. The edge–sampling with probability s generates two graphs $G_{1,2}$. To make a comparison with `User-Matching` [14] and `PercolateMatched` [30] algorithms we choose the node–sampling probability $t = 1.0$. Figure 8 compares our algorithm with the two baseline algorithms. The F_1–scores for `ExpandWhenStuck` is non–zero from the very beginning and with few seeds reach large values. We observe the F_1–scores of `User-Matching` and `PercolateMatched` (for the sampling probabilities that we have considered here) are always around zero.

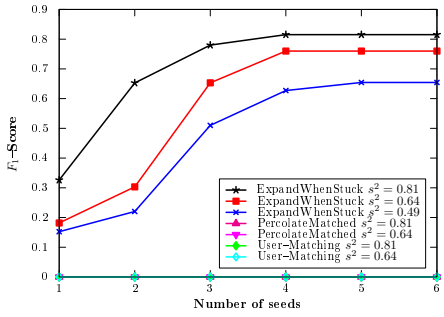

Figure 8: F_1–score (see Equation (7)) vs. number of seeds. Simulations are done over over Youtube graph with 1134890 node.

The second graph matching is done over friendship links on the Slashdot social network [15]. This network has 74118 nodes and average degree 12.13. The two graphs $G_{1,2}$ are generated though node and edge–sampling processes over the Slashdot network. In Figure 9, we observe the F_1–score (see (7)) for different node and edge sampling probabilities when 20 seeds are provided.

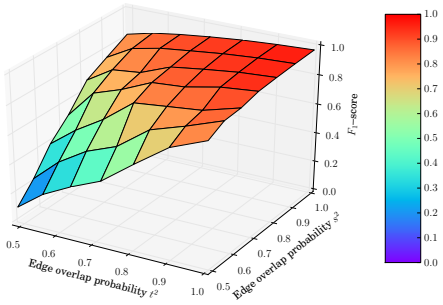

Figure 9: F_1–score vs. node and edge overlap probabilities (i.e., t^2 and s^2). Simulations are done over Slashdot network when the number of seeds is 20.

For the third experiment, we use the most popular online social network in Slovakia called Pokec with 1632803 nodes and average degree 37.50 [15]. Again, the two graphs $G_{1,2}$ are generated though node and edge–sampling processes. The excellent performance of `ExpandWhenStuck` over Pokec social–network is shown in Table 1.

Algorithm \ s^2	0.81	0.64	0.49
`ExpandWhenStuck`	0.99	0.98	0.97
`User-Matching` [14]	0.04	0.02	≈ 0
`PercolateMatched` [30]	0.05	0.02	≈ 0

Table 1: F_1–score (see Equation (7)): Simulations are done over Pokec social network with $n = 1632803$ and $t^2 = 1.0$, when 5 seeds are provided.

In the fourth experiment, we use different snapshots of the e-mail network on EPFL campus [22]. Each snapshot of the network is created by aggregating all the exchanged e-mails in a given time period. Each node corresponds to an account, and undirected edges represent exchanged e-mails between entities. In this experiment, we match two real graphs without any modelling assumptions, i.e., we do not assume any node or edge sampling process. As shown in Figure 10, with only one seed we can match most of the nodes in the EPFL e-mail network. In all snapshots of the EPFL e-mail network, the nodes with the highest degrees are the same. This is because the node degree distributions of real graphs are often heavy–tailed. For this network, the performance of `PercolateMatched` [30] is superior to `User-Matching` [14], and thus for our comparison we have only provided the results corresponding to `PercolateMatched`.

The fifth experiment is done over the Gowalla social network [15]. This dataset contains friendship relations and timestamped check-ins of users to different locations. Using this information, two snapshots of Gowalla network are generated [14]; in the first snapshot, two nodes are connected if they are friends and they check-in to exactly the same location in an even month. The second snapshot is generated similarly by looking at the friendships and check-ins in

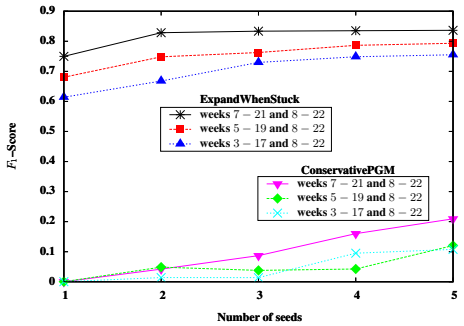

Figure 10: F_1–score (see Equation (7)) vs. number of seeds. Simulations are done over EPFL e-mail network with. Each snapshot of the e-mail network is created by aggregating all the exchanged e-mails in a given time period.

odd months. In this experiment the number of identifiable nodes, which is defined as the total number of nodes that are present in both graphs G_1 and G_2 with degrees greater than five, is 6634. Figure 11 shows the superior performance of `ExpandWhenStuck` versus algorithms from [14, 30]. Note that (i) these two Gowalla graphs are not generated through an edge sampling process, i.e., we match two real graphs without any modelling assumptions and, (ii) as the authors in [14] use check-ins to *approximately* the same locations instead of *exactly* the same locations to generate two Gowalla social graphs our simulation results differ a little bit from their reported results.

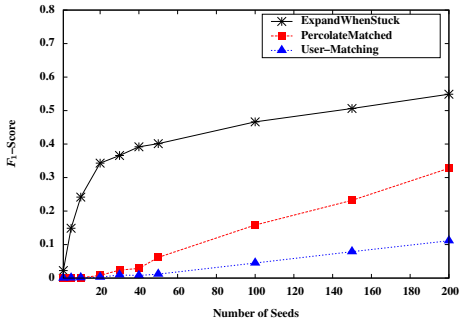

Figure 11: F_1–score (see Equation (7)) vs. number of seeds. Simulations are done over Gowalla social network. The number of identifiable nodes, $n_{\text{ident}} = 6634$, is defined as the total number of nodes that are present in both snapshots with degrees greater than five.

Our experiments show that `ExpandWhenStuck` is indeed robust against low node overlaps between the graphs. For example, in Gowalla (see Figure 11) the overlap between the two graphs is 0.72. Note that, in the intersection of Gowalla graphs, there are many nodes which are present in one of the graphs with degree 1. If we consider only the nodes with degrees more than 1 in both graphs, then the overlap reduces to 0.42. As another example, the overlaps for EPFL e-mail networks (see Figure 10) are between 0.27 to 0.31. Also, for random graphs with low overlaps, increasing the number of seeds (e.g., only 100 seeds for Chung–Lu graphs with $n = 10^6$, $\beta = 2.5$, $t^2 = 0.49$ and $s^2 = 0.49$) results in good (close to 1) recalls and precisions.

6.3 MapReduce implementation

One of the key features of PGM algorithms is their computational simplicity. Nevertheless, for extremely large graphs (100s of millions of nodes or more, say), the computational and storage overhead for a single machine may still be prohibitive. For this reason, we explored the implementation of a parallelized variant of `ExpandWhenStuck` within the MapReduce framework for scalability; we briefly report the main ideas and results here.

The `ExpandWhenStuck` algorithm cannot be readily parallelized, given the explicitly sequential back–and–forth between spreading marks (lines 9–10 in Algorithm 3) and matching new pairs (lines 7–8 in Algorithm 3). However, it turns out that without fundamentally affecting the performance of the algorithm, it is possible to reorder these two operations. We can spread marks from all the eligible pairs first, then perform the matching of new pairs afterwards. More concretely, this approximation of the original `ExpandWhenStuck` algorithm works as follows: (i) We spread marks from the pairs in the seed set $\mathcal{A}$; (ii) we add all the pairs with at least r marks to the matched set $\mathcal{M}$; (iii) we spread marks from all the new matched pairs. The steps (ii) and (iii) are repeated iteratively up to the point that there is no new pair with score at least r; (iv) at the point the percolation process stops a new set of candidate pairs $\mathcal{A}$ is generated from the neighbouring pairs of matched pairs and the graph matching process continues by going to step (i).

In this setting, the process of spreading marks can be done independently for all the pairs. This enables a parallel implementation of the algorithm through four consecutive MapReduce jobs per iteration. Next we sketch the function of each of these MapReduce jobs, without providing a detailed pseudo–code in the interest of space:

- The Mapper in the first job spreads out marks from the pairs in the candidate set $\mathcal{A}$. The output of Reducer in this job is the set of all the pairs with score at least r.
- It is possible for a node to be in several pairs with score above the threshold. The second MapReduce job filters out the nodes that appear in more than one pair with score at least r.
- The output of the second MapReduce job is the newly matched pairs. These pairs are fed to the third MapReduce job to spread their marks and match new pairs. Percolation graph–matching process continues by running the second and third MapReduce jobs iteratively.
- When there is no newly matched pairs, i.e., the percolation process is stuck, the forth MapReduce job is called. This job generates a new set of candidate pairs $\mathcal{A}$. Provided there are enough seeds, a few iteration of these four MapReduce jobs will match almost all the nodes correctly.

Our MapReduce implementation is able to easily match graphs with millions of nodes. For example, by using a Hadoop cluster with 15 nodes, it took less than twenty minutes to match random graphs with 10 million nodes (starting with 18 seeds); in under half an hour, the algorithm matches graphs sampled from LiveJournal and Orkut online social networks [15] with 4,847,571 and 3,072,441 nodes, respectively.

7. CONCLUSION

In this paper, we study the problem of graph matching between two unlabelled graphs when only the structures of the two graphs are available. We characterize the graph–

matching problem for graphs with partial–overlapping vertex sets. We give a new percolation graph matching algorithm. We prove that our algorithm correctly matches the nodes which are in the intersection of the two graphs and filters–out the nodes without counterparts in the other graph. A phase transition in the seed set size of percolation graph matching is formally established. Also, we prove that under a wide range of network parameters, our algorithm is robust against a noisy seed set. As our algorithmic contribution, we achieve a dramatic reduction in the size of the seed set. We also show the excellent performance in matching several large real social networks.

Acknowledgements The authors wish to thank Lyudmila Yartseva for her valuable comments. Seyed Hamed Hassani is supported by ERC Starting Grant under grant number 307036.

8. REFERENCES

[1] F. Abel, N. Henze, E. Herder, and D. Krause. Interweaving Public User Profiles on the Web. In *UMAP, HI, USA*, pages 16–27, September 2010.

[2] L. Babai, P. Erdös, and S. M. Selkow. Random Graph Isomorphism. *SIAM J. Comput.*, 9(3):628–635, 1980.

[3] L. Backstrom, C. Dwork, and J. M. Kleinberg. Wherefore art thou R3579X?: Anonymized Social Networks, Hidden Patterns, and Structural Steganography. *Commun. ACM*, 54(12):133–141, 2011.

[4] A.-L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *science*, 286(5439):509–512, 1999.

[5] B. Bollobás and O. Riordan. The Diameter of a Scale-Free Random Graph. *Combinatorica*, 24(1):5–34, 2004.

[6] F. Chung and L. Lu. Connected Components in Random Graphs with Given Expected Degree Sequences. *Annals of Combinatorics*, 6(2):125–145, 2002.

[7] A. Egozi, Y. Keller, and H. Guterman. A Probabilistic Approach to Spectral Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):18–27, 2013.

[8] P. Erdös and A. Rényi. On random graphs I. *Publ. Math. Debrecen*, 6:290–297, 1959.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[10] K. Henderson, B. Gallagher, L. Li, L. Akoglu, T. Eliassi-Rad, H. Tong, and C. Faloutsos. It's Who You Know: Graph Mining Using Recursive Structural Features. In *SIGKDD, San Diego, CA, USA*, pages 663–671, August 2011.

[11] S. Janson, T. Luczak, T. Turova, and T. Vallier. Bootstrap Percolation On The Random Graph $G_{n,p}$. *The Annals of Applied Probability*, 22(5):1989–2047, 2012.

[12] R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized Rumor Spreading. In *FOCS, Redondo Beach, California, USA*, pages 565–574, November 2000.

[13] G. Klau. A New Graph–Based Method for Pairwise Global Network Alignment. *BMC Bioinformatics*, 10(Suppl 1):S59, 2009.

[14] N. Korula and S. Lattanzi. An Efficient Reconciliation Algorithm for Social Networks. *PVLDB*, 7(5):377–388, 2014.

[15] J. Leskovec. Stanford Network Analysis Project. `http://snap.stanford.edu/index.html`.

[16] A. Malhotra, L. C. Totti, W. M. Jr., P. Kumaraguru, and V. Almeida. Studying User Footprints in Different Online Social Networks. In *ASONAM, Istanbul, Turkey*, pages 1065–1070, August 2012.

[17] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *ICDE, San Jose, CA, USA*, pages 117–128, 2002.

[18] A. Mislove, B. Viswanath, P. K. Gummadi, and P. Druschel. You Are Who You Know: Inferring User Profiles in Online Social Networks. In *WSDM, New York, NY, USA*, pages 251–260, February 2010.

[19] A. Narayanan and V. Shmatikov. De-anonymizing Social Networks. In *S&P, Oakland, California, USA*, pages 173–187, May 2009.

[20] A. Nunes, P. Calado, and B. Martins. Resolving User Identities Over Social Networks Through Supervised Learning and Rich Similarity Features. In *SAC, Riva, Trento, Italy*, pages 728–729, March 2012.

[21] P. Pedarsani, D. R. Figueiredo, and M. Grossglauser. A Bayesian Method for Matching Two Similar Graphs without Seeds. In *Conference on Communication, Control, and Computing, Allerton Park, Monticello, IL, USA*, pages 1598–1607, October 2013.

[22] P. Pedarsani and M. Grossglauser. On the Privacy of Anonymized Networks. In *SIGKDD, San Diego, CA, USA*, pages 1235–1243, August 2011.

[23] D. M. Powers. Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[24] D. Shah. *Gossip Algorithms*. Now Publishers Inc, 2009.

[25] R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proceedings of the National Academy of Sciences*, (35):12763–12768.

[26] L. Torresani, V. Kolmogorov, and C. Rother. Feature Correspondence Via Graph Matching: Models and Global Optimization. In *ECCV, Marseille, France*, pages 596–609, October 2008.

[27] B. Von Bahr and A. Martin-Löf. Threshold Limit Theorems for Some Epidemic Processes. *Advances in Applied Probability*, pages 319–349, 1980.

[28] L. Wiskott, J.-M. Fellous, N. Kuiger, and C. Von Der Malsburg. Face Recognition by Elastic Bunch Braph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):775–779, 1997.

[29] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A practical attack to de-anonymize social network users. In *S&P, Berleley/Oakland, California, USA*, pages 223–238, May 2010.

[30] L. Yartseva and M. Grossglauser. On the performance of percolation graph matching. In *COSN, Boston, MA, USA*, pages 119–130, October 2013.