An accurate and efficient algorithm to identify malicious nodes of a graph

An accurate and efficient algorithm to identify malicious nodes of a graph

Rossano Gaeta

***Abstract*—The identification of misbehaving elements in a distributed system is an important task in many diverse settings that can be represented as graphs; this problem can be cast as the computation of a subset of the graph nodes by exploiting a pre-determined detection mechanism. In this paper we propose a simple yet accurate algorithm to compute the set of nodes of a graph suspected to be malicious that is based on the so called *comparison detection model*. In this framework, a node can play the role of the comparator for two of its neighbors and can provide a boolean result based on the actual status of both. The algorithm we propose has low computational complexity and linear space complexity; furthermore, it only requires one parameter to trade accuracy against computational cost. We also show it outperforms the state-of-the-art and performs equally very well on both synthetic and real world graphs.**

***Index Terms*—Malicious node identification, comparison detection model, efficient algorithm, mathematical model.**

I. Introduction

The problem of identifying misbehaving elements in a distributed system is important in many diverse settings. Examples are represented by the identification of:

- misbehaving nodes in wireless relay networks, e.g., vehicular delay-tolerant networks [1], [2],
- malicious users in (possibly mobile) social networks [3], [4],
- attackers in distributed storage [5] or collaborative streaming networks [6], and
- faulty or malicious computing nodes in a distributed processing/sensing infrastructure [7].

By abstracting away many details, we observe that a common feature of these settings is that they might all be represented as graphs wherein nodes are components and links exist to represent components that somehow exchange information between them. Once this abstraction is accepted the problem of identifying misbehaving nodes can be cast as the problem of computing a subset of the nodes suspected to be malicious by exploiting a pre-determined detection mechanism.

In this paper we focus on the abstract problem of identifying malicious nodes in a graph. In particular, we do so by relying on the so called *comparison detection model* as proposed in [8], [9] and recently exploited in [10], [11]. According to this detection model, a node can play the role of the comparator whose task is to feed a pair of its neighbors with problems. Nodes under scrutiny by the comparator provide answers to these problems: honest nodes never lie and always provide the same correct answer while malicious nodes always give the wrong answer. Honest and malicious nodes are assumed to always provide different answers to the same set of problems. The comparison result is a boolean value assumed to be **false** if comparator and both compared neighbors are honest; the results is assumed to be **true** if the comparator is honest and at least one neighbor is malicious. Finally, the result of the comparison cannot be predicted if the comparator itself is malicious.

Rossano Gaeta is with Università degli Studi di Torino, Dipartimento di Informatica, Torino, Italia. E-mail: rossano.gaeta@unito.it

Our contribution

We first propose a straightforward algorithm based on the comparison detection model to compute the set of malicious nodes that displays low computational cost and we later refine it to obtain a higher accuracy algorithm with higher computational cost, linear space complexity, and the highest accuracy. We will explore the impact of structural characteristics of graphs on the algorithm performance and we will show that it outperforms the state-of-the-art and performs equally very well on both synthetic and real world graphs.

The rest of the paper is organized as follows: Section II describes the framework we consider and presents algorithms that implement the comparison detection model, Section III illustrates three algorithms we developed to compute the set of suspect nodes of a graph, Section IV presents accuracy and complexity results. Comparison against state-of-the-art approaches is presented in Section V while Section VI discusses related literature. Finally, Section VII summarizes paper contribution and outlines future developments.

To help the reader, Table I summarizes the notation used throughout this paper.

II. System and attack model

SYMBOL	DESCRIPTION
Section II	
$\mathcal{G} = (V, E)$	input graph
$M \subseteq V$	set of actual malicious nodes
$\Gamma(u)$	neighborhood of node u
$M_s \subseteq V$	set of suspect malicious nodes
$A \subseteq_x^{rnd} B$	A random subset of B, $\lvert A\rvert = \min(x, \lvert B\rvert)$
$\rho(c) \subseteq_{n_c}^{rnd} (\Gamma(c) - \{u\})$	subset of neighborhood of c excluding u
$n_c = \lvert\rho(c)\rvert$	cardinality of $\rho(c)$
Section III	
p_{fp}	false positive probability
p_{tp}	true positive probability

TABLE I: Notation

We consider an undirected graph $\mathcal{G} = (V, E)$ whose set of nodes and links are denoted as V and E, respectively. For

each node $u \in V$ we denote as $\Gamma(u)$ its neighborhood, i.e., the set of nodes connected to u by an edge.

The set of nodes V is partitioned in two subsets M and $V - M$; the subgraph induced by M is composed of all malicious nodes and is termed as the *malicious region* while the subgraph induced by $V - M$ is called the *honest region*. An edge connecting the two regions is called *attack edge* and no hypothesis are made on the sparsity of attack edges. Figure 1 depicts a portion of $\mathcal{G}$ wherein malicious nodes are drawn as dashed line circles while honest nodes are represented as continuous line circles.

As an example, in the context of online social networks malicious nodes might be spammers, fake users, and compromised normal users that are maintained by an attacker to perform various malicious activities, e.g., influence elections and financial markets, spreading of spam and misinformation, collecting private user data.

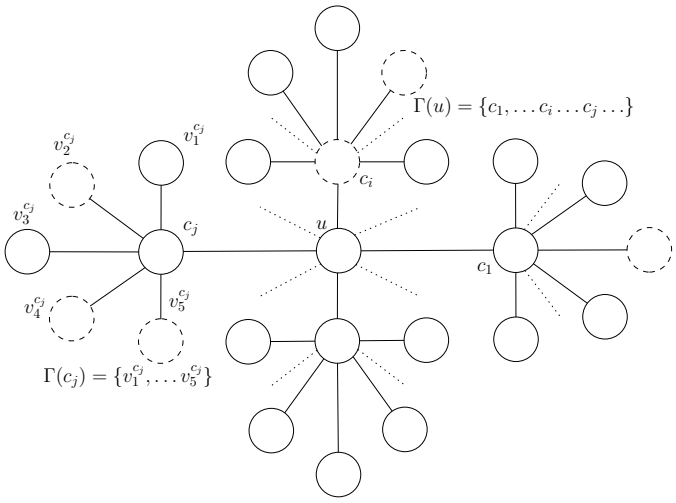

Fig. 1: Part of a graph $\mathcal{G}$: dashed line circles represent malicious nodes, continuous line circles represent honest nodes.

The main goal of the analysis is the computation of the set of suspect nodes $M_s \subseteq V$. To this end, we consider as the only detection mechanism the *comparison detection model* as proposed in [8], [9] and recently exploited in [10], [11]. According to this detection model, a node can play the role of the comparator whose task is to feed a pair of its neighbors with problems. Nodes under scrutiny by the comparator provide answers to these problems: honest nodes never lie and always provide the same correct answer while malicious nodes always give the wrong answer. Honest and malicious nodes are assumed to always provide different answers to the same set of problems. The comparison result is a boolean value assumed to be **false** if comparator and both compared neighbors are honest; the results is assumed to be **true** if the comparator is honest and at least one neighbor is malicious. Finally, the result of the comparison cannot be predicted if the comparator itself is malicious.

Algorithm 1 defines function compare() and it illustrates the computation carried out by a comparator node c that is managing nodes u and v that are both its neighbors, i.e., $u, v \in \Gamma(c)$. We modeled unpredictability of the comparison result for malicious comparators as the outcome of a random experiment whose realization might trigger the logical inversion of the comparison result. Function is_malicious() returns **true** if the argument node is actually malicious and **false** otherwise. Function uniform() returns a random variate from the uniform distribution in the interval $[0, 1]$.

Algorithm 1 compare(c, u, v)

```
r = is_malicious(u) ∨ is_malicious(v)
if is_malicious(c) ∧ uniform() <= 0.5 then
    r = not(r)
end if
return r
```

Algorithm 2 m(u, c, n_c)

```
ρ(c) ⊆^{rnd}_{n_c} (Γ(c) − {u})
r = true;
for v ∈ ρ(c) do
    r = r ∧ compare(c, u, v)
end for
return r
```

Assuming the comparator node c is honest, when the result of the comparison is **true** we only know that at least one between nodes u and v is malicious. More information is required to assess the status of node u. To this end, we build on the comparison detection model and we consider a *malicious indicator function* for node u managed by comparator c defined as $\mathsf{m}(u, c, n_c) = \bigwedge_{v \in \rho(c)} \mathsf{compare}(c, u, v)$ where $\rho(c)$ is a random subset of the neighborhood of comparator c excluding u (the notation $A \subseteq_x^{rnd} B$ is used to describe that A is a random subset of B and that cardinality of A is $|A| = \min(x, |B|)$). The size n_c of $\rho(c)$ is a parameter of Algorithm 2 that describes a straightforward computation of malicious indicator function m().

III. Identification algorithms

We first propose in Section III-A a straightforward algorithm based on the comparison detection model to compute the set of malicious nodes that displays the least computational cost expressed as the number of calls to function compare(), i.e., Algorithm 1). This simple solution will be refined in Section III-B to obtain a higher accuracy algorithm at the cost of higher computational complexity. Accuracy of both algorithms will be exactly described by mathematical models. Section III-C presents a synthesis of both solutions yielding an algorithm with low computational complexity, linear space complexity, and the highest accuracy.

A. A straightforward solution (sf algorithm)

A straightforward solution to compute the set of suspect nodes M_s is given by Algorithm 3 (that we denote as the *sf algorithm*). It simply considers each node $u \in V$ and computes its status according to Algorithm 2 by means of a randomly chosen neighbor c of node u to act as comparator.

Unfortunately, this simple approach does not yield high accuracy. To show this, we assume we deal with random $d-$regular graphs [12], i.e., a class of random graph models

Algorithm 3 $\mathsf{sf}(\mathcal{G}, n_c)$

```
1: M_s = ∅;
2: for u ∈ V do
3:    c ⊆_1^rnd Γ(u)
4:    if m(u, c, n_c) then
5:       M_s = M_s ∪ {u}
6:    end if
7: end for
8: return M_s
```

wherein all nodes are randomly connected to d others[1]. Under this assumption, we consider all possible cases when each node u (including the comparator c) has d neighbors and a node in V is malicious with probability p_m.

When node u is honest *sf algorithm* can fail when:

- comparator node c is honest. In this case, it could happen that *all nodes* in $\rho(c)$ are malicious ending up in a misidentification of node u as a *false positive* (FP) by function $\mathsf{m}()$. More formally, misidentification of honest node u as FP occurs when c has $n_c \leq y \leq d-1$ malicious neighbors *and* all n_c neighbors randomly chosen by comparator c are malicious. The probability honest node u is misidentified as malicious by a honest comparator is then given by

$$b_{fp}^{hon} = (1-p_m) \sum_{y=n_c}^{d-1} B(y, p_m, d-1) H(n_c, y, d-1-y, n_c),$$

  where factor $(1-p_m)$ accounts for the probability comparator c is honest, $B(y, p_m, d-1)$ is the probability that y out of $d-1$ neighbors are malicious (it is described by the binomial probability distributions with parameters p_m and $d-1$), and $H(n_c, y, d-1-y, n_c)$ is the hypergeometric distribution that describes the probability that in a population composed of y malicious nodes and $d-1-y$ honest nodes exactly n_c malicious nodes are selected when randomly extracting a subset of size n_c.
- comparator c is malicious. In this case, regardless the number of malicious neighbors of comparator c, node u can be misidentified as malicious if c does not invert all of the selected malicious neighbors in $\rho(c)$ and it inverts all of the selected honest neighbors in $\rho(c)$. The probability honest node u is misidentified as malicious by a malicious comparator is then given by

$$b_{fp}^{mal} = p_m \sum_{y=0}^{d-1} B(y, p_m, d-1) \sum_{s=0}^{n_c} \frac{H(s, y, d-1-y, n_c)}{2^{n_c - s} \cdot 2^s} = \frac{p_m}{2^{n_c}}.$$

It follows that, regardless the actual status of comparator c, the above analysis yields the overall probability of misidentification of a honest node u as a false positive by *sf algorithm* as

$$p_{fp} = b_{fp}^{hon} + b_{fp}^{mal}. \tag{1}$$

If node u is malicious *sf algorithm* can accurately identify it when:

[1]Of course, our analysis can be generalized to random graphs whose degree distribution is described by an arbitrary discrete probability distribution $P(d)$.

Algorithm 4 $\mathsf{ex}(\mathcal{G}, n_c)$

```
1: M_s = ∅;
2: for u ∈ V do
3:    γ(u) ⊆_{n_c}^rnd Γ(u)
4:    n_m = 0;
5:    for c ∈ γ(u) do
6:       if m(u, c, n_c) then
7:          n_m = n_m + 1;
8:       end if
9:    end for
10:   if n_m ≥ ⌊n_c/2⌋ + 1 then
11:      M_s = M_s ∪ {u}
12:   end if
13: end for
14: return M_s
```

- comparator c is honest. In this case, node u can never be misidentified as honest since function $\mathsf{m}()$ always returns **true**. It follows that the probability node u is correctly identified as malicious by a honest comparator is simply $b_{tp}^{hon} = 1 - p_m$.
- comparator c is malicious. In this case, a malicious node u is correctly identified as such if c does not invert *all* the comparison results for the selected neighbors in $\rho(c)$. This event occurs with probability $b_{tp}^{mal} = \frac{p_m}{2^{n_c}}$.

It follows that *sf algorithm* yields an overall true positive probability given by

$$p_{tp} = b_{tp}^{hon} + b_{tp}^{mal}. \tag{2}$$

It is easy to observe that the complexity of *sf algorithm* in terms of number of calls to function $\mathsf{compare}()$ is $\mathcal{O}(n_c \cdot |V|)$.

*B. An expensive solution (*ex algorithm*)*

The main issue of *sf algorithm* is that it relies on *only one* randomly chosen neighbor of node u to play the comparator role. A quite simple technique to improve accuracy is to rely *on a set* of comparators and to identify a node based on the outcome of the majority of comparators. Algorithm 4 (that we denote as the *ex algorithm*) implements this solution to compute the set of suspect nodes M_s and requires the number of comparators for identifying node u as a parameter. Accuracy of *ex algorithm* can be exactly described by Equations 3 and 4 that represent the probability the majority of comparators identifies node u as malicious.

$$p_{fp} = \sum_{n_m = \lfloor \frac{n_c}{2} \rfloor + 1}^{n_c} B(n_m, b_{fp}^{hon} + b_{fp}^{mal}, n_c) \tag{3}$$

and that

$$p_{tp} = \sum_{n_m = \lfloor \frac{n_c}{2} \rfloor + 1}^{n_c} B(n_m, b_{tp}^{hon} + b_{tp}^{mal}, n_c) \tag{4}$$

We observe that complexity of *ex algorithm* is $\mathcal{O}(n_c^2 \cdot |V|)$.

*C. An accurate and efficient solution (*ae algorithm*)*

In Section IV we show that *ex algorithm* yields higher accuracy when compared against *sf algorithm*. Nevertheless, it is still based on a *random selection* of comparators and

Algorithm 5 compute_status(u, c, n_c)

```
1: if c == UNDEFINED then
2:    γ(u) ⊆^{rnd}_{n_c} Γ(u)
3:    n_m = 0;
4:    for v ∈ γ(u) do
5:       if m(u, v, n_c) then
6:          n_m = n_m + 1;
7:       end if
8:    end for
9:    return (n_m ≥ ⌊n_c/2⌋ + 1)?MALICIOUS : HONEST
10: else
11:    return (m(u, c, n_c))?MALICIOUS : HONEST
12: end if
```

Algorithm 6 ae$(\mathcal{G}, n_c)$

```
1: n_computed = 0; Q = ∅; M_s = ∅;
2: for u ∈ V do
3:    status[u] = UNDEFINED;
4: end for
5: repeat
6:    for u ∈ V do
7:       if status[u] == UNDEFINED then
8:          n_computed = n_computed + 1
9:          status[u] = compute_status(u, UNDEFINED, n_c)
10:         if status[u] == HONEST then
11:            for v ∈ Γ(u) do
12:               enqueue(v, u, Q)
13:            end for
14:            break
15:         else
16:            M_s = M_s ∪ {u}
17:         end if
18:      end if
19:   end for
20:   while Q ≠ ∅ do
21:      q = dequeue(Q)
22:      if status[q.u] == UNDEFINED then
23:         n_computed = n_computed + 1
24:         status[q.u] = compute_status(q.u, q.c, n_c)
25:         if status[q.u] == HONEST then
26:            for v ∈ Γ(q.u) do
27:               enqueue(v, q.u, Q)
28:            end for
29:         else
30:            M_s = M_s ∪ {q.u}
31:         end if
32:      end if
33:   end while
34: until n_computed < |V|
35: return M_s
```

it does not exploit any of the partial identifications already computed. This observation leads to the highly accurate yet efficient Algorithm 6 (that we denote as the *ae algorithm*).

After an initialization phase (lines 1 through 4), the algorithm scans nodes whose status is still to be determined to identify them (for loops in lines 6 through 19). For these nodes no previous knowledge can be exploited hence their status is computed by Algorithm 5 based on a set of comparators as in *ex algorithm*. If a node u is identified as malicious it is inserted in output set M_s otherwise the scanning interrupts and u is used as a *trusted* comparator for all its neighbors in $\Gamma(u)$. All nodes that can be identified by a trusted comparator are inserted in a queue $\mathcal{Q}$ and analyzed in a breadth−first fashion (lines 20 through 33). Their status is computed by Algorithm 5 by simply evaluating function m() with a trusted comparator. As soon as queue $\mathcal{Q}$ empties scanning is resumed for nodes still to be identified. The algorithm terminates when all nodes of graph $\mathcal{G}$ have been considered.

Computational complexity of *ae algorithm* is more difficult to characterize and lies between $\mathcal{O}(n_c \cdot |V|)$ and $\mathcal{O}(n_c^2 \cdot |V|)$. This is because function compute_status() (Algorithm 5) acts like *sf algorithm* if it is invoked with a defined comparator node c while it acts like *ex algorithm*, otherwise. As for space complexity, *ae algorithm* requires linear space to store the results of status computation, i.e., the *status[]* array and the queue $\mathcal{Q}$.

IV. Evaluation

In this section we present results we obtain from running algorithms presented in Section III on both synthetic and real world graphs. In particular, in Section IV-A we validate Equations 1, 2, 3, and 4 by comparing their predictions against results obtained from experiments on random $d-$regular graphs [12]. Section IV-B compares accuracy and computational costs of *sf algorithm*, *ex algorithm*, and *ae algorithm* while Section IV-C analyzes the impact of structural characteristics of synthetic random graphs on the accuracy of *ae algorithm*. Finally, in Section IV-D we show performance of *ae algorithm* on real world graphs taken from [13].

To evaluate the accuracy of identification algorithms we define the:

- true positive probability (denoted as *detection rate* in [11]) as $p_{tp} = \frac{|M_s \cap M|}{|M|}$, and
- false positive probability (denoted as *false positive rate* in [11]) as $p_{fp} = \frac{|M_s - M_s \cap M|}{|V - M|}$.

When dealing with synthetic graphs, we analyzed graphs whose size is $|V| = 10,000$ and we computed average p_{tp} and p_{fp} by considering 500 different realizations of topologies and 20 different assignment of malicious nodes for each topology. The set of malicious nodes M is synthesized by selecting a random subset of elements in V whose size is equal to $p_m \cdot |V|$, i.e., $M \subseteq^{rnd}_{p_m \cdot |V|} V$.

All algorithms have been developed by using the C programming language that exploit the *igraph* library [14] to support graph creation, import, and manipulations. All experiments have been run on a Intel i9-9900K CPU based PC equipped with 64GB RAM. A repository on Github is available for software download at https://github.com/rossano-gaeta/malicious-identification.

A. Validation results

The first set of results we present aims to validate Equations 1, 2, 3, and 4. To this end, we consider random graphs whose nodes all share the same degree d, i.e., random $d-$regular graphs. Figure 2 depicts p_{tp} (left graph) and p_{fp} (right graph) obtained from *sf algorithm* and *ex algorithm* for different values of d and p_m. To avoid cluttering the graphs, we only show results for $n_c = \frac{d+2}{2}$. It can be noted that mathematical models for p_{tp} and p_{fp} exactly predict the output of *sf algorithm* and *ex algorithm* for any choice of d and p_m. The same accuracy is obtained for all values $1 \leq n_c \leq d$.

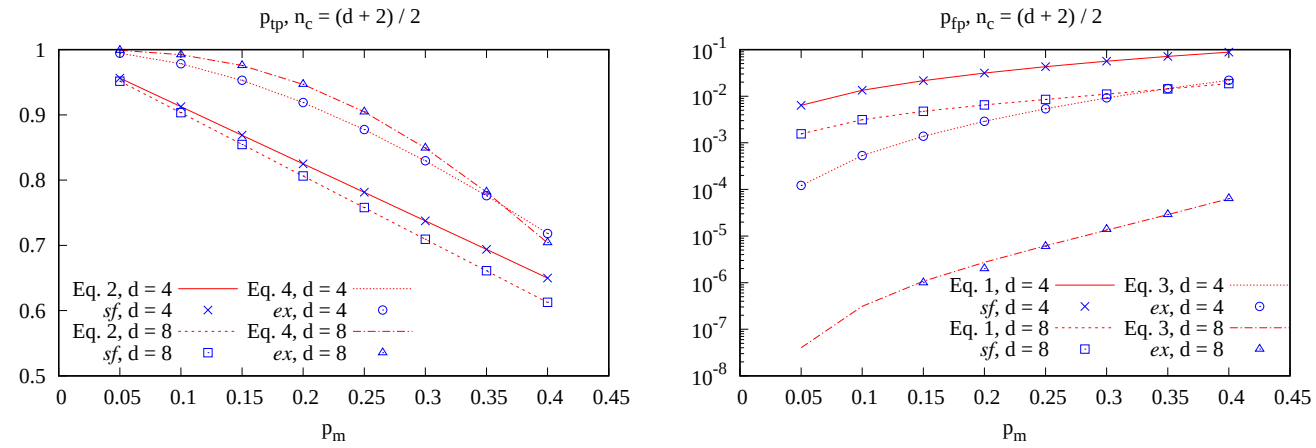

Fig. 2: True positive (left graph) and false positive (right graph) probabilities comparing *sf algorithm* and *ex algorithm* against predictions from Equations 1, 2, 3, and 4 for $d = 4, 8$, and $n_c = \frac{d+2}{2}$ as a function of p_m.

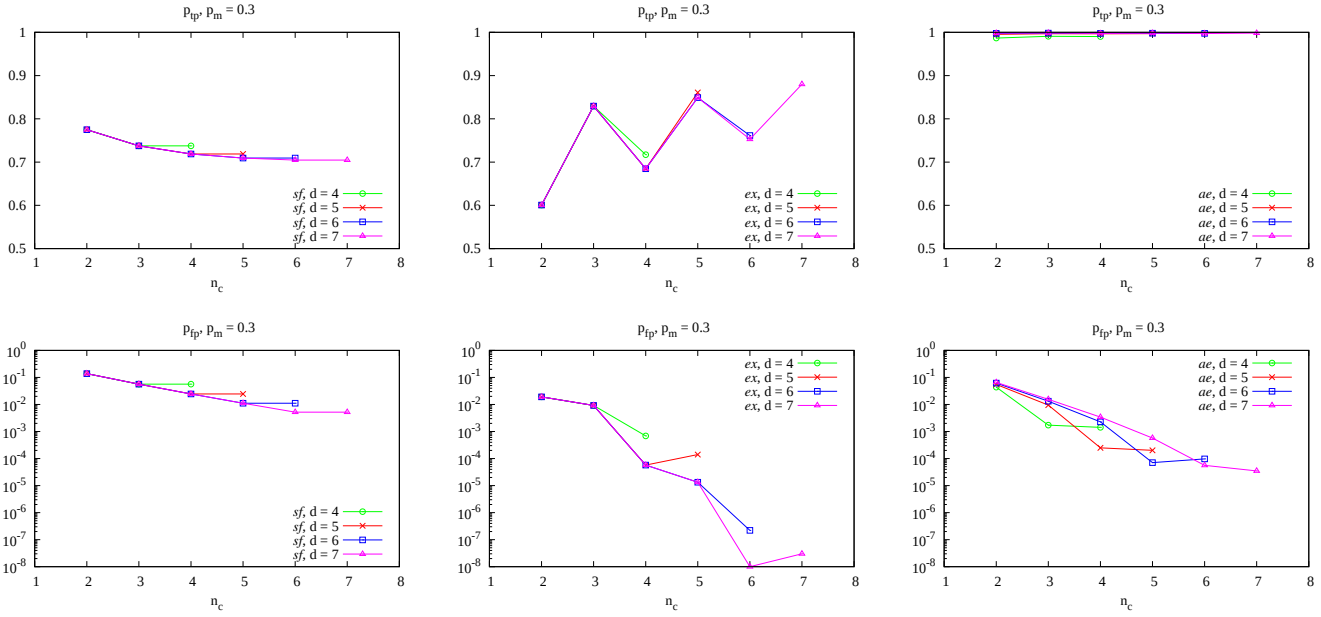

Fig. 3: True positive (top row) and false positive (bottom row) probabilities for *sf algorithm* (left column), *ex algorithm* (middle column), and *ae algorithm* (right column) when $p_m = 0.3$ as a function of n_c.

B. Comparison results

A careful observation of validation results presented in Figure 2 also suggests that:

- values of p_{fp} decrease for both algorithms as d increases;
- values of p_{tp} also decrease as d increases but only for *sf algorithm*;
- the behavior of p_{tp} as a function of d is not easily assessable for *ex algorithm*. Indeed, for small values of p_m the higher d the higher p_{tp} while for $p_m > 0.35$ smaller values of d yield higher accuracy.

To clarify the last point, Figure 3 depicts results for p_{tp} (top row) and p_{fp} (bottom row) for *sf algorithm* (left column), *ex algorithm* (middle column), and *ae algorithm* (right column) when $p_m = 0.3$ as a function of n_c and d. It can be noted that:

- for *sf algorithm* optimal p_{tp} is obtained when $n_c = 2$ for any value of d while optimal p_{fp} can be attained whenever $n_c = d$;
- for *ex algorithm* and fixed d, both p_{tp} and p_{fp} do not show monotonic trends with respect to n_c. In particular, optimal values of p_{tp} are obtained when $n_c = d$ if d is odd and for $n_c = d - 1$ if d is even. The opposite is valid if we focus on p_{fp} whose optimality is reached for $n_c = d - 1$ if d is odd and for $n_c = d$ if d is even;
- for *ae algorithm* $n_c = d$ yields optimal accuracy for both p_{tp} and p_{fp}.

Non-monotonicity of accuracy with respect to n_c for *ex algorithm* (the saw-shaped curves in Figure 3, middle graphs)

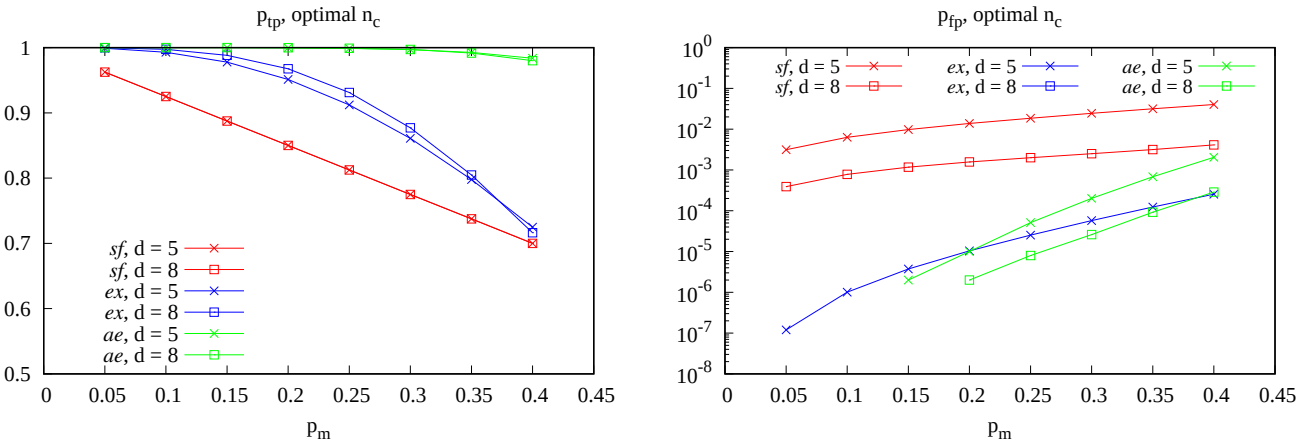

Fig. 4: True positive (left graph) and false positive (right graph) probabilities comparing *sf algorithm*, *ex algorithm*, and *ae algorithm* for $d = 5, 8$ as a function of p_m and optimal n_c.

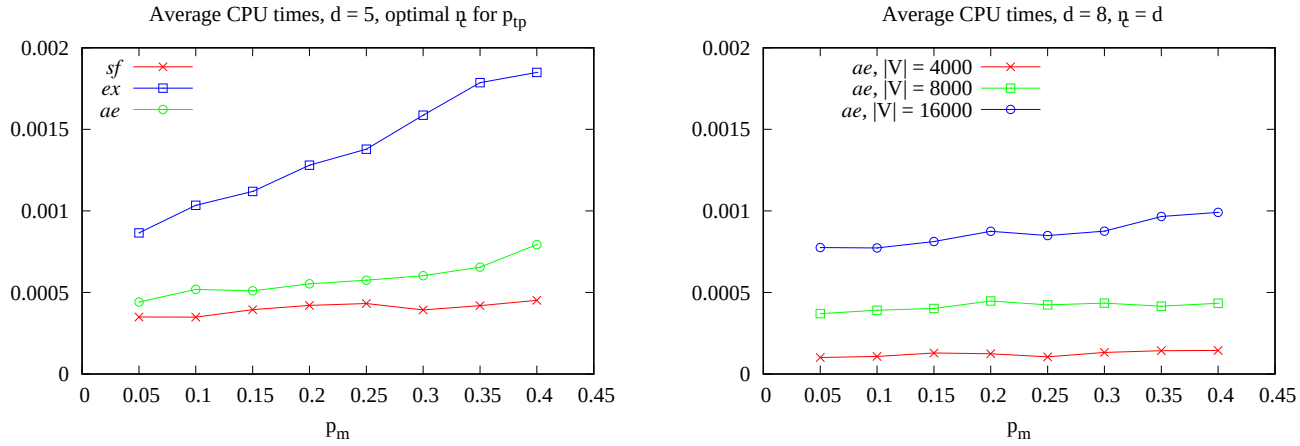

Fig. 5: Average CPU times (in seconds). Left graph shows a comparison among *sf algorithm*, *ex algorithm* , and *ae algorithm* for $d = 5$ and optimal n_c for p_{tp} as a function of p_m. Right graph depicts same results for *ae algorithm* for $d = 8$ and optimal n_c ($n_c = d$) as a function of p_m for different graph sizes $|V|$.

is due to the majority rule implemented to determine the status of a node. In Algorithm 4, line 10 the comparison is performed by using the floor function on $\frac{n_c}{2}$ and the same is described by Equations 3 and 4.

Analysis of results of Figure 3 also suggests that optimal accuracy of identification algorithms can differ greatly. To this end, we compare them by using for each the value of n_c that yields optimal p_{tp} and p_{fp}. Figure 4 depicts p_{tp} (left graph) and p_{fp} (right graph) comparing *sf algorithm*, *ex algorithm*, and *ae algorithm* for $d = 5, 8$ as a function of p_m and optimal n_c for both accuracy indexes.

It can be noted that *sf algorithm* always performs the worst with respect to both p_{tp} and p_{fp}. On the contrary, *ae algorithm* always yields higher accuracy with respect to p_{tp}. Remarkably, it is able to correctly identify up to 99% of actual malicious nodes even when p_m is *very* high.

As far as p_{fp} is concerned, superiority of one algorithm over the other depends on both node's degree d and overall fraction of malicious nodes p_m. In particular, for small degree nodes *ae algorithm* yields better results with respect to *ex algorithm* for $0 < p_m \leq 0.2$. For $p_m > 0.2$ *ex algorithm* yields lower p_{fp} values. For higher degree nodes *ex algorithm* always proves to be the best choice with $p_{fp} < 10^{-8}$ for all values of p_m we considered.

Nevertheless, better performance of an algorithm from the accuracy point of view should be carefully scrutinized in view of the required computational cost. Figure 5 (left-graph) presents the average cpu times (in seconds) to compare *sf algorithm*, *ex algorithm*, and *ae algorithm* for $d = 5$ and optimal n_c for p_{tp} as a function of p_m. As expected, the computational effort for *sf algorithm* is the lowest since optimal p_{tp} is obtained for $n_c = 2$ while *ex algorithm* requires the highest effort. Actual CPU times are also affected by the values of p_m; indeed, high values require more frequent execution of the logical inversion in line 3 of Algorithm 1. It can be noted that computational cost of *ae algorithm* is only slightly higher than the least expensive *sf algorithm*.

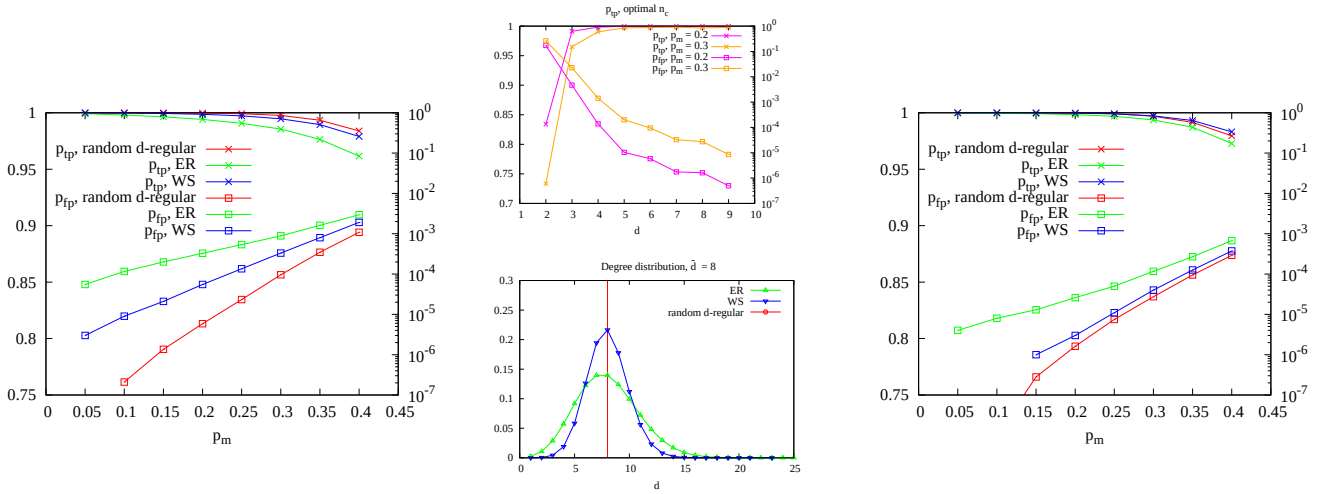

Fig. 6: True positive and false positive probabilities for *op algorithm* comparing random graphs with different degree distributions and same average $\overline{d}$ for $\overline{d} = 6$ (left column) and $\overline{d} = 8$ (right column) as a function of p_m. Middle column, top graph shows performance on random d-regular graphs as a function of d. Middle column, bottom graph shows all degree distributions.

For this reason, in the sequel we will only consider the *ae algorithm* with optimal n_c, i.e., $n_c = d$. This also means that *ae algorithm* with optimal n_c is actually parameterless; indeed, parameter n_c can be dropped from *ae algorithm* and from Algorithms 2 and 5 by:

- removing it from the parameter list of each algorithm,
- replacing operator $\subseteq_{n_c}^{rnd}$ with $=$ in Algorithms 2 and 5, and
- replacing n_c with $|\Gamma(u)|$ in Algorithm 5.

In the sequel, we shall term the *ae algorithm* with optimal n_c as the *op algorithm*.

C. *Results for* op algorithm *and structural characteristics*

In this section we further explore accuracy of *op algorithm* to analyze the impact of structural characteristics of graphs. To this end, we evaluated its accuracy over Erdős-Rényi (ER), Watts-Strogatz (WS), and random d-regular graphs [15] sharing the same average degree $\overline{d}$. For the ER graphs we adopted the $G(n, p)$ model where p is set to yield the desired average degree $\overline{d}$ while for the WS graphs we used the unidimensional model with degree $\overline{d}$ and rewiring probability equal to 0.25. Figure 6 depicts results comparing random graphs with different degree distributions and same average $\overline{d}$ for $\overline{d} = 6$ (left column) and $\overline{d} = 8$ (right column) as a function of p_m. It can be noted that *op algorithm* yields the lowest accuracy when run on ER random graphs. To find an explanation in Figure 6 (middle column, top graph) we show how p_{tp} and p_{fp} behave on random d-regular graphs as a function of d. The higher d the higher the accuracy for both indexes. Figure 6 (middle column, bottom graph) shows the degree distribution of these three classes of random graphs; it can be noted that the left tail for ER random graphs is "heavier", i.e., the fraction of low degree nodes is higher with respect to both WS and random d-regular graphs. The influence of low degree nodes on the accuracy of *op algorithm* can also be noticed by comparing results for $\overline{d} = 6$ (left column) and $\overline{d} = 8$ (right column); higher $\overline{d}$ yields higher accuracy.

We further investigated the accuracy of *op algorithm* by considering ER random graphs whose average is $\overline{d} = 8$ and whose clustering coefficient (CC) can be tuned according to method presented in [16]. Figure 7 depicts p_{tp} and p_{fp} comparing ER random graphs with different clustering coefficients and same average degree $\overline{d} = 8$ as a function of p_m. Note that for $p_m \leq 0.25$ index p_{tp} is not affected by CC while for higher fraction of malicious nodes higher clustering is beneficial to accuracy. On the contrary, higher clustering is detrimental to index p_{fp} whose values increase as CC increase for all values of p_m.

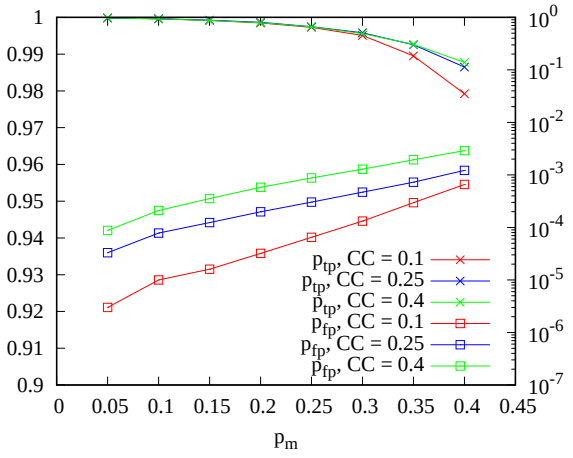

Fig. 7: True positive (left graph) and false positive (right graph) probabilities for *op algorithm* comparing ER random graphs with different clustering coefficients and same average degree $\overline{d} = 8$ as a function of p_m.

D. Results for op algorithm on real world graphs

DATASET	$\|V\|$	$\|E\|$	$\overline{d}$
Facebook	22,470	171,002	15.2
Twitch	168,114	6,797,557	80.8
Pokec	1,632,803	22,301,964	27.3

TABLE II: Dataset description for the evaluation of *op algorithm*.

To prove that accuracy of the *op algorithm* is very high also on real world graphs, Figure 8 shows p_{tp} and p_{fp} as a function of p_m for Pokec, Twitch, and Facebook crawled graphs retrieved from [13] and whose statistics are summarized in Table II. It can be noted that p_{tp} is very close to 1 even for very high values of p_m. At the same time, p_{fp} keeps very low for both curves for all values of p_m in the range we considered.

As a final remark, in [11] it is pointed out that when a node is surrounded by malicious neighbors its identification as either honest or malicious is unreliable with high probability. We could redefine the accuracy of the identification algorithms by considering only the subset of nodes whose neighborhood includes at least one honest node; we call these nodes as *structurally identifiable*. In this case, Figure 9 shows that p_{tp} is virtually equal to 1 for all p_m values in the range we considered and for all real world crawled graphs we analyzed.

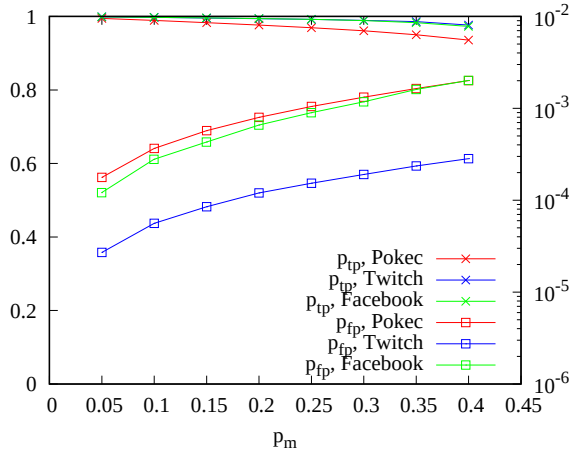

Fig. 8: Accuracy of the *op algorithm* for Pokec, Twitch, and Facebook crawled graphs retrieved from [13].

V. COMPARISON AGAINST STATE-OF-THE-ART

In this section we compare performance of the *op algorithm* against state-of-the-art approaches. In particular, in Section V-A we consider the work in [11] that exploits Hamiltonian cycle decomposition of a graph. This work is the closest to ours and the one that inspired the current paper. We show that the *op algorithm* outperforms it, i.e., it yields a false positive probability that is at least one order of magnitude lower and a true positive probability that keeps very close to 1 for a wide range of values of the fraction of malicious nodes. Since in [11] the authors showed their method is superior to

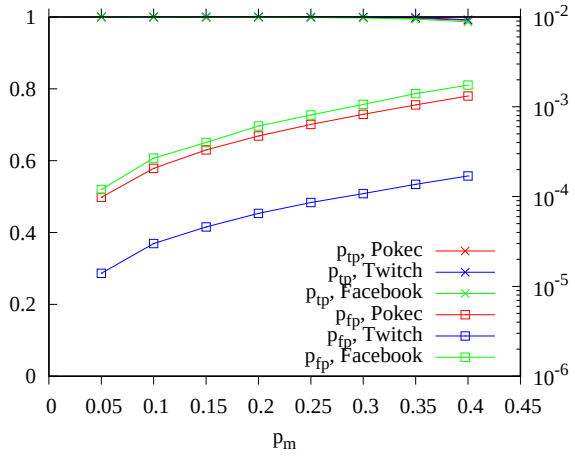

Fig. 9: Accuracy of the *op algorithm* for Pokec, Twitch, and Facebook crawled graphs retrieved from [13] for structurally identifiable nodes.

trust−based approaches in [17], [18] and machine learning based approaches in [19], [20] this means that the *op algorithm* can be considered as a better approach with respect to them, as well.

Furthermore, in Section V-B we compare the performance of the *op algorithm* against those of *Sybilscar* as proposed in [21]. This approach has been developed with the specific goal of detecting of malicious nodes (termed as sybil nodes) in online social networks. We show that the *op algorithm* outperforms it as soon as the homophily assumption of the graph under study is relaxed. Since in [21] the authors showed their method is superior to [22], [23] this means that the *op algorithm* can be considered as a better approach with respect to these proposals, as well.

A. Hamiltonian cycle decomposition: a brief description

The work in [11] starts from a graph with S nodes, randomly selects $s < S$ nodes, considers a set of n common features, and forms $N = 2^n$ groups of nodes used to represent subsets of nodes that share the same feature values. Relations among groups are then represented by a n-dimensional hypercube Q_n with N nodes with the (very strong) assumption that an entire group is either honest or malicious. The analysis is carried out by:

1) obtaining $2^{n-\lceil \log_2(n+1) \rceil}$ cycles whose length is $\lceil \log_2(n+1) \rceil$ in a Hamiltonian cycle decomposition of Q_n;
2) computing the results of all comparisons performed by each node on its neighbors in each cycle of the Hamiltonian cycle decomposition to obtain a so called *syndrome*;
3) exploiting properties of cycle decompositions and syndromes proved for hypercube Q_n to devise an identification algorithm.

Results comparison: To perform a comparison, as in [11] we consider the graph crawled from the Pokec social network. Figure 10 shows p_{tp} and p_{fp} as a function of p_m. The inset

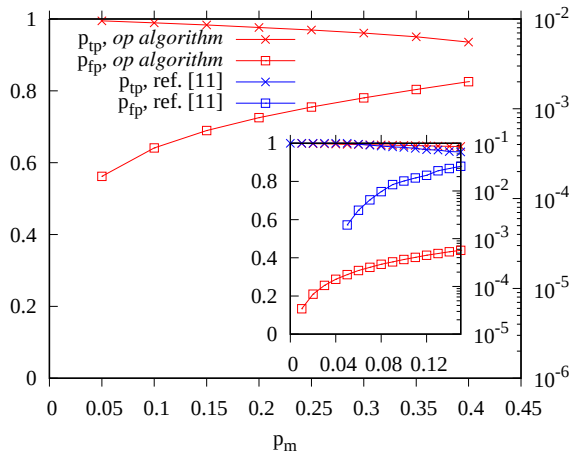

Fig. 10: True positive and false positive probabilities for the *op algorithm* (comparison against Figure 15 and Figure 16 in [11]) as a function of p_m.

zooms in the range $[0, 0.15]$ that is the one considered in Figures 15 and 16 in [11]. It can be noted that both proposals yield p_{tp} values (denoted as *detection rate* in [11]) that keep close to 1 for p_m up to 0.15 although for $p_m = 0.15$ the *op algorithm* yields $p_{tp} = 0.983$ while the approach in [11] yields $p_{tp} = 0.954$. Unfortunately, it is not possible to assess the accuracy of the technique proposed in [11] for higher values of p_m. As for p_{fp} (denoted as *false positive rate* in [11]), the *op algorithm* yields values for $p_m = 0.15$ that are at least one order of magnitude lower than those depicted in Figure 16 in [11].

Computational complexity of [11] depends on the number of nodes N. In particular, obtaining a Hamiltonian cycle decomposition of Q_n and computing the results of all comparisons (a syndrome) are both $\mathcal{O}(N)$. The last third step in our brief summary, i.e., the amount of comparisons to exploit properties of cycle decompositions and syndromes of Q_n is $\mathcal{O}(N+2\log_2 N)$. Furthermore, [11] requires a long pre-processing whose complexity adds to the identification algorithm. Also, computational complexity of both pre-processing and identification depends on the chosen number n of common features: the higher n, the larger $N = 2^n$ hence the complexity. It is worth noting that this complexity is necessary only provide information on $s < S$ nodes of an original graph and only in an aggregated fashion since identification is performed on a group level. An additional algorithm would be needed to analyze a single group to identify malicious nodes of the original graph increasing the overall complexity. Finally, at least $\frac{S}{s}$ runs of the algorithm would be necessary to obtain information on the whole original graph.

Accuracy of [11] also depends on the set of n common features but no sensitivity analysis is provided with respect to this performance index. To increase accuracy for the subset of s nodes, the authors of [11] suggest to consider several different Hamiltonian cycle decompositions of the original hypercube Q_n and to run multiple rounds of their algorithm to exploit different realizations of comparisons by malicious nodes. Unfortunately, no evidence is provided that this approach would work and no details are offered on how to deal with multiple realizations of their identification algorithm. Of course, such a strategy would increase the overall computational complexity.

Accuracy of the *op algorithm* is higher with respect to [11] because comparison results obtained by malicious nodes are taken into account the lowest number of times. Indeed, in [11] the second step requires all N nodes (including malicious nodes) to compare their two neighbors in their cycle to obtain a syndrome. Results obtained from malicious nodes cannot be trusted entirely (on average, half of the comparisons performed by malicious nodes are incorrect). In the *op algorithm*:

- on the one hand, more resources are spent in the initial steps to determine the status of a node (function compute_status() is invoked with parameter c equal to UNDEFINED). In this case, the *op algorithm* acts like the *ex algorithm* whose values of $1 - p_{fp}$ are the highest possible as witnessed by results in Figure 2 right graph. This means that once a node is identified as honest it is truly so with the highest possible probability. Therefore, it is immediately used as a trusted comparator for all its neighbors (**break** instruction in line 14 of Algorithm 6).
- on the other hand, Figure 2 left graph also shows that the probability to let a malicious node serve as comparator, i.e., $1 - p_{tp}$, is the lowest possible.

The *op algorithm* then reduces the negative impact of malicious nodes on the overall accuracy. Figure 2 also suggests that the higher the node degree d the higher the overall accuracy. One of the future development of this work could be the analysis of improvements that can be obtained by having a degree−guided choice to analyze nodes.

B. Sybilscar: a brief description

Sybilscar was proposed with the goal of unifying previous approaches that were based on loop Belief Propagation or random walks by defining an algorithm that could work by exploiting a local rule to update the *posterior* probability of a node to be malicious. Therefore, an original rule that combines neighbor influences with *prior* knowledge is defined and it is iteratively applied to every node to compute the posterior probabilities of being malicious. At the core of this and other approaches lies the assumption that the honest regions and the malicious region (termed as the *benign region* and the *sybil region* in [21], respectively) are sparsely connected, i.e., edges connecting the two regions are few with respect to the number of edges connecting nodes within the same region. The sparsity of attack edges is called *homophily* in [21] to mean that two connected nodes are either both honest or both malicious with high probability.

Results comparison: To compare performance we reproduced results in [21] by:

- using the software the authors made available at https://github.com/binghuiwang/sybildetection;
- considering the same real world graphs whose statistics are summarized in Table III;
- adopting the same modeling approach wherein for each network the graph to analyze is obtained by duplicating

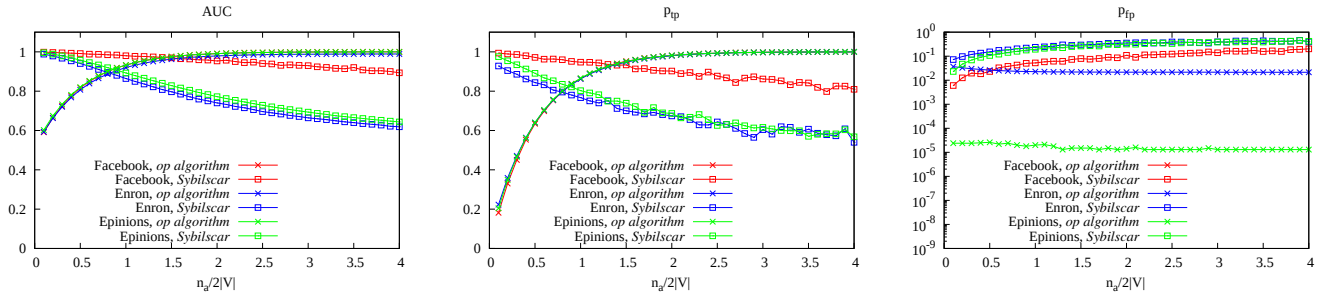

Fig. 11: AUC (left graph), true positive (middle graph), and false positive (right graph) probabilities for *op algorithm* compared against Sybilscar as n_a increases.

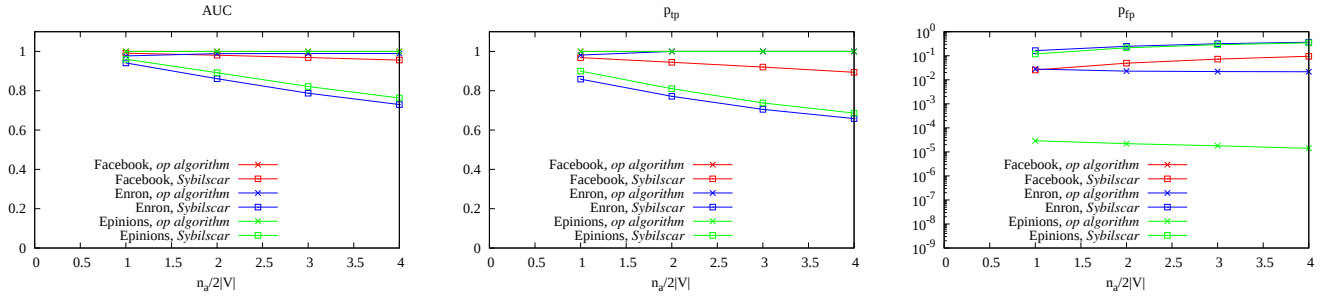

Fig. 12: AUC (left graph), true positive (middle graph), and false positive (right graph) probabilities for *op algorithm* compared against Sybilscar as n_a increases for a different attack model.

it to represent the malicious region and n_a attack edges are added between them;
- selecting 200 nodes uniformly at random and using them as a training dataset with the remaining ones that are used as testing data;
- setting the numerous parameters required by Sybilscar to the same values used in [21];
- adding to the p_{tp} and p_{fp} performance indexes the so called *Area Under the Receiver Operating Characteristic Curve (AUC)* to evaluate ranking accuracy. If we assume nodes are ranked with respect to their posterior probability of being malicious in a descending order then AUC is the probability that a randomly selected malicious node ranks higher than a randomly selected honest node. Clearly, random guessing yields AUC $= 0.5$.

DATASET	$\|V\|$	$\|E\|$	$\overline{d}$
Facebook	4,039	88,234	43.7
Enron	33,696	180,811	10.7
Epinions	75,877	405,739	10.6
Twitter	41,652,230	1,202,513,046	57.7

TABLE III: Dataset description for the comparison against Sybilscar taken from [13].

For a given real world graph, an experiment consists in adding uniformly at random n_a attack edges between the honest and the malicious regions. We performed 30 independent experiment and averaged results that are depicted in Figures 11 and 12. There the AUC (left graph), p_{tp} (middle graph), and p_{fp} (right graph) probabilities are shown for the *op algorithm* compared against Sybilscar as n_a increases. Results are presented as a function of the normalized number of attack edges $\frac{n_a}{2|V|}$ for a fair comparison of results among different networks.

On the one hand, it can be noted that the *op algorithm* always outperforms Sybilscar with respect to p_{fp} (for the Facebook network $p_{fp} = 0$). On the other hand, Sybilscar yields higher AUC and p_{tp} values with respect to the *op algorithm* when the number of attack edges is low with respect to the overall size of the graph to be analyzed. This is not surprising because Sybilscar heavily relies on the fact that the two regions are sparsely connected. This also means that many malicious nodes are not structurally identifiable as defined in Section IV-D yielding poorer performance of the *op algorithm* with respect to p_{tp}. Nevertheless, as the amount of attack edges increases the *op algorithm* outperforms Sybilscar with respect to all performance indexes.

To further prove that Sybilscar can offer better performance only under very special hypothesis, Figure 12 depicts the same comparison under an attack scenario wherein *each node* in the malicious region establishes exactly $1, 2, \ldots$ attack edges with randomly chosen honest nodes in the honest region. This attack model involves the same overall amount of attack edges but with a different distribution among malicious nodes. It can be noted that in this case the *op algorithm* always outperforms Sybilscar for all performance indexes. Also note that the *op algorithm* is able to yield high accuracy when half of the nodes are malicious, i.e., when $p_m = 0.5$.

Finally, the Twitter dataset used in [21] is considered as

retrieved from https://people.duke.edu/~zg70/dataset.html; it includes randomly sampled 100,000 malicious and 10,000,000 honest nodes which are used as ground truth for training and testing. Also in this case we reproduced results presented in [21] by using the same set of values for the parameters required by Sybilscar. To this end, we first considered 500,000 nodes sampled uniformly at random among the ground truth to use them as a training dataset; the rest of honest and malicious nodes are used as testing data. In this case, results reported in Table IV clearly show that (as commented in [21], Section 7.2.2) Sybilscar yields poor results due to both weak homophily and the use of an unbalanced training dataset, i.e., a training dataset wherein honest nodes are *much more* than malicious ones. On the contrary, the *op algorithm* performs very well in this scenario with respect to all performance indexes.

We also considered a balanced training dataset wherein we randomly sampled 50,000 honest nodes and the same amount of malicious nodes from the provided ground truth as in [24] that discusses a preliminary version of Sybilscar. Sybilscar performs much better in this case (as commented in [21], Section 7.2.2) but still much worse than the *op algorithm*.

ALGORITHM	AUC	p_{tp}	p_{fp}
Sybilscar (unbalanced)	0.234	0.004	0.001
Sybilscar (balanced)	0.810	0.995	0.980
op algorithm	0.999	0.999	0.00005

TABLE IV: Comparison between the *op algorithm* and Sybilscar for the Twitter dataset.

VI. RELATED WORKS

The problem of detecting misbehaving elements in a distributed system has received ample attention in the literature. Depending on the context different ad-hoc techniques have been devised.

For instance, survey [1] paints a very useful and comprehensive picture of attacks and countermeasures in wireless relay networks. In this context, techniques proposed in [25], [26], [27], [28], [17], [18], [29], [30], [31] are somehow based on reputation and/or trust.

In the context of online social networks, exploitation of social information to design detection mechanisms has been followed in [32], [33], [34], [19], [35], [36], [37]. Furthermore, several interesting papers have dealt with the problem of detecting fake users (termed as Sybils) by exploiting properties of mixing times of random walk on social graphs with limited attack edges [38], [22], [39] or by defining a semi-supervised learning framework to perform both malicious nodes classification and ranking [23]. Notably, [24] proposes a framework to unify random walk−based methods and loop belief propagation−based methods.

Machine learning, statistics, and cryptography protocols are very important tools of several approaches for malicious behavior detection in other contexts [40], [41], [20], [42], [43], [44], [45], [46].

The work in [11] is the closest to ours and the one that inspired the current paper. It focuses on human contact networks and exploits social features to build a topology structure. It then devises a graph theoretical comparison detection model to compute the set of malicious nodes. In particular, the detection method is a diagnosis process that relies on the Hamiltonian cycle decomposition of hypercubes. The authors showed their method is superior to those in [17], [18], [19], [20]. Our method yields better results with respect to [11], i.e., a false positive probability that is at least one order of magnitude lower and a true positive probability that keeps very close to 1 for a wide range of values of the fraction of malicious nodes. Furthermore, the low computational cost of our proposal makes it feasible to run the algorithm on very large graphs.

VII. CONCLUSIONS AND FUTURE DEVELOPMENTS

In this paper we tackled the problem of computing the set of malicious nodes in a graph. To this end, we relied on the comparison detection model already exploited by previous work on this subject. We first proposed a straightforward algorithm that displays the least computational cost and we later refined it to obtain a higher accuracy algorithm with the highest computational complexity. We finally synthesized them to develop an algorithm with low computational complexity, linear space complexity, and the highest accuracy. We explored the impact of structural characteristics of graphs on the algorithm accuracy and we also showed it outperforms the state-of-the-art. Finally, we showed our proposal performs equally very well on both synthetic and real world graphs.

The current work could be extended by considering/devising other detection models to cope with nodes that could alternate between honest and malicious behaviors. Furthermore, since we showed that both degree distribution and clustering coefficient of the graph can impact the accuracy of the algorithm we plan to further investigate how to exploit them to improve accuracy. In particular, we observed that the *op algorithm* reduces the negative impact of malicious nodes on the overall accuracy as commented in Section V-A. Our analysis suggests to explore a degree−guided choice for node analysis.

REFERENCES

[1] B. Jedari, F. Xia, and Z. Ning, "A survey on human-centric communications in non-cooperative wireless relay networks," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 914–944, 2018.

[2] J. A. Dias, J. J. Rodrigues, F. Xia, and C. X. Mavromoustakis, "A cooperative watchdog system to detect misbehavior nodes in vehicular delay-tolerant networks," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 12, pp. 7929–7937, 2015.

[3] L. Xu, L. Lin, and S. Wen, "First-priority relation graph-based malicious users detection in mobile social networks," 11 2015, pp. 459–466.

[4] B. Jedari, F. Xia, H. Chen, S. K. Das, A. Tolba, and Z. AL-Makhadmeh, "A social-based watchdog system to detect selfish nodes in opportunistic mobile networks," *Future Generation Computer Systems*, vol. 92, pp. 777–788, 2019.

[5] R. Gaeta, "On the impact of pollution attacks on coding-based distributed storage systems," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 292–302, 2022.

[6] A. Fiandrotti, R. Gaeta, and M. Grangetto, "Securing network coding architectures against pollution attacks with band codes," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 730–742, 2019.

[7] K. Gu, X. Dong, and W. Jia, "Malicious node detection scheme based on correlation of data and network topology in fog computing-based vanets," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 1215–1232, 2022.

[8] M. Malek, "A comparison connection assignment for diagnosis of multiprocessor systems," in *Proceedings of the 7th Annual Symposium on Computer Architecture*, ser. ISCA '80, 1980, p. 31–36.

[9] J. Maeng and M. Malek, "A comparison connection assignment for self-diagnosis of multicomputer systems," in *Proc. 1981 Symp. on Fault Tolerant Comp.*, 1981, pp. 173–175.

[10] L. Lin, L. Xu, D. Wang, and S. Zhou, "The g -good-neighbor conditional diagnosability of arrangement graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 3, pp. 542–548, 2018.

[11] L. Lin, Y. Huang, L. Xu, and S.-Y. Hsieh, "Better adaptive malicious users detection algorithm in human contact networks," *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 2968–2981, 2022.

[12] B. Bollobás, *Random Graphs*, 2nd ed. Cambridge University Press, 2001.

[13] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[14] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, p. 1695, 2006. [Online]. Available: https://igraph.org

[15] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications," *Physical Review E*, vol. 64, no. 2, 2001.

[16] E. Volz, "Random networks with tunable degree distribution and clustering," *Physical Review E*, vol. 70, no. 5, 2004.

[17] H. Zhu, S. Du, Z. Gao, M. Dong, and Z. Cao, "A probabilistic misbehavior detection scheme toward efficient trust establishment in delay-tolerant networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 22–32, 2013.

[18] S. K. Dhurandher, A. Kumar, and M. S. Obaidat, "Cryptography-based misbehavior detection and trust control mechanism for opportunistic network systems," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3191–3202, 2017.

[19] H. Alvari, E. Shaabani, S. Sarkar, G. Beigi, and P. Shakarian, "Less is more: Semi-supervised causal inference for detecting pathogenic users in social media," in *Companion Proceedings of The 2019 World Wide Web Conference*, 2019, pp. 154–161.

[20] A. Abou Daya, M. A. Salahuddin, N. Limam, and R. Boutaba, "Botchase: Graph-based bot detection using machine learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 15–29, 2020.

[21] B. Wang, J. Jia, L. Zhang, and N. Z. Gong, "Structure-based sybil detection in social networks via local rule-based propagation," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 523–537, 2019.

[22] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro, "Aiding the detection of fake accounts in large scale social online services," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 197–210.

[23] N. Z. Gong, M. Frank, and P. Mittal, "Sybilbelief: A semi-supervised learning approach for structure-based sybil detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 6, pp. 976–987, 2014.

[24] B. Wang, L. Zhang, and N. Z. Gong, "Sybilscar: Sybil detection in online social networks via local rule based propagation," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[25] W. R. Pires, T. H. de Paula Figueiredo, H. C. Wong, and A. A. F. Loureiro, "Malicious node detection in wireless sensor networks," in *18th International Parallel and Distributed Processing Symposium, 2004.*

[26] W. Zhang, S. Zhu, J. Tang, and N. Xiong, "A novel trust management scheme based on dempster–shafer evidence theory for malicious nodes detection in wireless sensor networks," *The Journal of Supercomputing*, vol. 74, pp. 1779–1801, 2018.

[27] J. A. Dias, J. J. Rodrigues, F. Xia, and C. X. Mavromoustakis, "A cooperative watchdog system to detect misbehavior nodes in vehicular delay-tolerant networks," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 12, pp. 7929–7937, 2015.

[28] E. Ayday and F. Fekri, "An iterative algorithm for trust management and adversary detection for delay-tolerant networks," *IEEE Transactions on Mobile Computing*, vol. 11, no. 9, pp. 1514–1531, 2011.

[29] M. Nitti, R. Girau, and L. Atzori, "Trustworthiness management in the social internet of things," *IEEE Transactions on knowledge and data engineering*, vol. 26, no. 5, pp. 1253–1266, 2013.

[30] R.-I. Ciobanu, C. Dobre, M. Dascălu, Ş. Trăuşan-Matu, and V. Cristea, "Sense: A collaborative selfish node detection and incentive mechanism for opportunistic networks," *Journal of Network and Computer Applications*, vol. 41, pp. 240–249, 2014.

[31] R. Chen, F. Bao, M. Chang, and J.-H. Cho, "Dynamic trust management for delay tolerant networks and its application to secure routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1200–1210, 2013.

[32] A. Mei, G. Morabito, P. Santi, and J. Stefa, "Social-aware stateless forwarding in pocket switched networks," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 251–255.

[33] B. Jedari, F. Xia, H. Chen, S. K. Das, A. Tolba, and A.-M. Zafer, "A social-based watchdog system to detect selfish nodes in opportunistic mobile networks," *Future Generation Computer Systems*, vol. 92, pp. 777–788, 2019.

[34] G. Yang, S. He, and Z. Shi, "Leveraging crowdsourcing for efficient malicious users detection in large-scale social networks," *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 330–339, 2016.

[35] X. Chen, B. Proulx, X. Gong, and J. Zhang, "Exploiting social ties for cooperative d2d communications: A mobile social networking case," *IEEE/ACM Transactions on Networking*, vol. 23, no. 5, pp. 1471–1484, 2014.

[36] I. Parris and T. Henderson, "Friend or flood? social prevention of flooding attacks in mobile opportunistic networks," in *2014 IEEE 34th international conference on distributed computing systems workshops (ICDCSW)*. IEEE, 2014, pp. 16–21.

[37] K. Gu, X. Dong, and W. Jia, "Malicious node detection scheme based on correlation of data and network topology in fog computing-based vanets," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 1215–1232, 2020.

[38] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, 2006, pp. 267–278.

[39] J. Jia, B. Wang, and N. Z. Gong, "Random walk based fake account detection in online social networks," in *2017 47th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2017, pp. 273–284.

[40] S. Dvorak, P. Prochazka, and L. Bajer, "Gnn-based malicious network entities identification in large-scale network data," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–4.

[41] M. Aravind, V. Sujadevi, M. R. Krishnan, P. S. AU, S. Pal, A. Vazhayil, G. Sridharan, and P. Poornachandran, "Malicious node identification for DNS data using graph convolutional networks," in *2022 IEEE 7th International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, vol. 7. IEEE, 2022, pp. 104–109.

[42] T. N. D. Pham and C. K. Yeo, "Detecting colluding blackhole and greyhole attacks in delay tolerant networks," *IEEE Transactions on Mobile Computing*, vol. 15, no. 5, pp. 1116–1129, 2015.

[43] Z. Cui, Y. Zhao, Y. Cao, X. Cai, W. Zhang, and J. Chen, "Malicious code detection under 5g hetnets based on a multi-objective rbm model," *IEEE Network*, vol. 35, no. 2, pp. 82–87, 2021.

[44] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Madness: A multi-layer anomaly detection framework for complex dynamic systems," *IEEE Transactions on Dependable and Secure computing*, vol. 18, no. 2, pp. 796–809, 2019.

[45] X. Lin, "Lsr: Mitigating zero-day sybil vulnerability in privacy-preserving vehicular peer-to-peer networks," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 9, pp. 237–246, 2013.

[46] B. Wang, J. Jia, and N. Z. Gong, "Graph-based security and privacy analytics via collective classification with joint weight learning and propagation," in *ISOC Network and Distributed System Security Symposium (NDSS)*, 2019.