Studia i Materiały Informatyki Stosowanej, Tom 6, Nr 14, 2014
str. 18-26

ALGORITHMS FOR TESTING SECURITY IN GRAPHS*

Arkadiusz Hiler Robert Lewoń Michał Małafiejski
Gdańsk University of Technology
Department of Algorithms and System Modeling
Gabriela Narutowicza 11/12, 80-233 Gdańsk
e-mails: arkadiusz@hiler.pl lewon.robert@gmail.com mima@kaims.pl

Abstract: *In this paper we propose new algorithmic methods giving with a high probability the correct answer to the decision problem of security in graphs. For a given graph G and a subset S of a vertex set of G we have to decide whether S is secure, i.e. every subset X of S fulfils the condition: |N[X] □ S| □ |N[X] \ S|, where N[X] is a closed neighbourhood of X in graph G. We constructed a polynomial time property pseudotester based on the heuristic using simulated annealing and tested it on graphs with induced small subgraphs G[S] being trees or graphs with a bounded degree (by 3 or 4). Our approach is a generalization of the concept of property testers known from the subject literature, but we applied our concepts to the coNP-complete problem.*

KEYWORDS: *Property tester, pseudotester, secure set, security, coNP-completeness*

Algorytmy testujące bezpieczeństwo zbioru wierzchołków grafu

Streszczenie: *W niniejszym artykule przedstawiamy metodę weryfikowania bezpieczeństwa zbioru w grafie, dającą wysokie prawdopodobieństwo poprawnej weryfikacji. Problemem jest określenie, czy dla danego grafu G oraz podzbioru S zbioru wierzchołków tego grafu zbiór S jest bezpieczny, to znaczy każdy jego podzbiór X spełnia warunek: |N[X] ∩ S| ≥ |N[X] \ S|, gdzie N[X] jest domkniętym sąsiedztwem zbioru X w grafie G. Zaprojektowaliśmy pseudotester o wielomianowej złożoności obliczeniowej dla decyzyjnego problemu bezpieczeństwa zbioru w grafie wykorzystując m.in. koncepcję symulowanego wyżarzania. Wykonaliśmy testy dla grafów, w których podgraf indukowany przez zbiór S jest drzewem lub grafem ograniczonego stopnia (przez 3 oraz 4). Z uwagi na coNP-zupełność problemu bezpieczeństwa zaproponowane przez nas podejście jest uogólnieniem koncepcji testowania własności znanej z literatury.*

Streszczenie: *Testowanie własności, pseudotester, zbiór bezpieczny, bezpieczeństwo, coNP-zupełność*

1. INTRODUCTION

The considerations in this paper are motivated by the fact that there exists no non-deterministic exact polynomial time algorithm for testing secure sets in graphs (unless NP = coNP), which was proved in [4]. The problem of graph security was introduced in [3] and is formulated as follows: for a given graph G = (V,E) and a given non-empty subset S V(G) we have to decide whether S is a secure set, i.e. every subset X of S fulfils the condition |N[X] □ S| □ |N[X] \ S|, where N[X] = {v □ V(G): v □ X □□ □w□X {v,w} □ E(G)} is a closed neighbourhood of X in graph G.

For intractable optimization problems one may construct approximation algorithms (or even approximation schemas, e.g. FPTAS), but for decision problems the matter is not so easy, because the only correct answer given by the algorithm may be yes or no. What kind of approximation may we reach in that case? The answer comes with the concept of property testers, which was introduced by Rubinfeld and Sudan in [13], and extended by Goldreich [9], [10] and Raskhodnikova [12].

Property testers (or testers) are □-parametrized algorithms proposed for decision problems which are giving a correct answer with a given probability (which is a constant greater than ½ and independent from □) for a wide subset of all possible inputs. Smaller □□values mean that with the given probability the algorithm can produce the correct answer for a wider subset of inputs.

In this paper we focus on algorithmic approaches to the problem of security in graphs. We propose a new heuristic,

which we called a pseudotester, because we cannot guarantee an independent and uniform distribution of the probability of choosing a valuable witness.

1.1 The model and the problem definition

In this section we introduce all the concepts used in the paper concerning the problem of testing security in graphs.

Security in graphs. By $SECURE(G,S)$ we denote the answer to the question: is set S secure in graph G? For a given subset $X \subset S$, by $c(X) = |N[X] \cap S| - |N(X) \setminus S|$ we mean the *advantage* of the defending vertices from X over the attacking vertices outside S. Let us define the predicate $SEC(X)$ as equal to $c(X) \geq 0$. Hence, by the definition of a secure set $SECURE(G,S) \Leftrightarrow \forall_{X \subset S}\, SEC(X)$.
By an *attack* on $S=\{v1,\ldots,v_k\}$ we mean a sequence $\{A_1,\ldots,A_k\}$ of mutually disjoint sets such that $A_i \subset N(v_i) \setminus S$ for all $i = 1,\ldots,k$, where $N(v_i) = \{w \in V(G): \{v,w\} \in E(G)\}$ is a neighbourhood of v_i in G. An attack is *maximal* if and only if each vertex from the attacking set (i.e. $N(S) \setminus S$) belongs to the attack (sequence).
By a *defence* of S we mean a sequence $\{D_1,\ldots,D_k\}$ of mutually disjoint sets such that $D_i \subset N[v_i] \cap S$ for all $i = 1,\ldots,k$, where $N[v_i] = N(v_i) \cup \{v_i\}$.
Following [4], we may state that S is secure in G if and only if every attack on S can be *defended*, i.e. for every attack A on S there is a defence D, such that $|D_i| \geq |A_i|$ for all $i = 1,\ldots,k$.

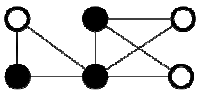

Fig 1: The graph with the secure set (black nodes).

The graph security problem was studied in [3] and [5]. In [4] the author proved the *coNP*-completeness of the *SEC* problem for general graphs, leaving no prospects for any non-deterministic polynomial time algorithm unless *NP=coNP*. In the paper [6] the authors construct a parametrized algorithm with the running time of $O(2^{klog(2k)}n)$, which decides if there exists a secure set of the size less than or equal to k in a given graph G. In [2] we proposed polynomial time algorithms solving the problem $SECURE(G,S)$ for complete k-partite graphs, trees, cacti graphs and subcubic graphs.

Property testing. Following [7], we sketch the concept of property testers for general decision problems, which is a generalization of the concept introduced in [13]. For a given decision problem Π and set of all legal inputs (instances) I we may precisely split I into *yes*-instances (Y) and *no*-instances (N). Now, let DC_ε be a family of subsets of N, such that the following two ε-conditions are met:

- $\forall_{0 < \varepsilon 1 < \varepsilon 2 \leq 1}\, DC_{\varepsilon 1} \subset DC_{\varepsilon 2}$
- $\forall_{i \in I}\, \exists_{0 < \varepsilon \leq 1}\; i \notin DC_\varepsilon$

For a given $p > ½$ we define a *property tester* A_ε as an ε-parametrized algorithm (defined for every $0 < \varepsilon \leq 1$) satisfying the following two *probability conditions*:

1. $Pr[A_\varepsilon(i) = yes \mid i \in Y] \geq p$
2. $Pr[A_\varepsilon(i) = no \mid i \in N_\varepsilon] \geq p$

In the following, we require our property testers to satisfy the probability conditions with $p = 0.95$ (widely used as $p = ⅔$). Note that for any instance from DC_ε the answer given by the algorithm A_ε has no guarantee, thus we call this set a *don't care* set. We may try to define the *approximation property* of testers as follows: the smaller the ε is, the greater the probability that the algorithm produces a correct answer with the probability of at least p. Obviously, the problem is much more complicated, because the key problem is to determine:

- the complexity of the algorithm A_ε in terms of the size of the input and $1/\varepsilon$.
- the relation between DC_ε and ε, i.e. how small ε is required to ensure the correct answer (with the probability of at least p) for at least δ fraction of all possible inputs.

The above model was widely discussed in [7], where the authors constructed two testers for the security problem (see section 1.2 for more details).
Following [12] we introduce the concept of instances being far (precisely ε-far) from satisfying the given property, formally: an instance $i \in I$ is ε-far from Y if and only if $i \in N_\varepsilon$. Property testers are therefore algorithms that distinguish inputs with a given property from those that are ε-far from satisfying the property. A useful tool for ensuring the probability conditions is *witness lemma*. Consider the problem Π and an input instance $i \in I$ for this problem. By a

witness we mean i^* which is a partial input of i (or a data structure generated based on the input, mostly with a significantly reduced size of the input). By a *witness testimony* we mean an answer *yes* or *no* given by the witness i^* such that if $i \in Y$, then the testimony of the witness i^* is exactly *yes*. Otherwise, if $i \in N$, then the testimony of witness i^* is undefined (equal to *yes* or *no*). The witness i^* constructed for the input $i \in N$ is *valuable* if and only if its testimony is *no*. If a single test (choosing of a witness) catches a valuable witness with the probability of at least p, then random $s = \lceil 3/p \rceil$ independent tests (choosing witnesses uniformly) catch the witness with the probability of at least *95*% (*witness lemma*).By the above, we proposed a testing model that contains a definition of ε-far, a definition of a witness and its testimony, an algorithm *choosing* a witness (independently and uniformly) and an algorithm that produces a testimony for a given witness. Following [7], we can sketch the model and the algorithm rating criteria as follows: practical and intuitive understanding of ε-far (i.e. how valuable the information that input is ε-far from satisfying a given property is to us), the relationship between ε and p_s (the probability of catching a valuable witness by the algorithm), the complexity of the algorithm (in terms of $1/\varepsilon$ and the input size), the relationship between DC_ε and ε.

Testing secure sets. Let $i = (G,S)$ be an input instance. For a given $0 < \varepsilon \leq 1$ we define N_ε as follows:

$$i \in N_\varepsilon \Leftrightarrow min\{|X|: X \subset S \wedge SEC(X) = no\} \leq |S| (1 - \varepsilon)$$

It is easy to verify that the two ε-conditions defined in section 1.1 hold. We build our heuristic taking into account this definition of ε-far and our goal is to verify experimentally the probability conditions from section 1.1. Thus, our heuristic can be called a *pseudotester*.

1.2 Previous results and our contribution

In [7] we proposed two testers based on attacking sets and defending sets, respectively. Let us briefly discuss both of them.

The model based on attacking sets. We take into account all the possible maximal attacks and require that a given fraction of them must be defendable. Formally, an input instance (the pair G and S, or equivalently S and $A = N(G) \setminus S$ forming the graph) is ε-far from being secure if and only if ε fraction of maximal attacks cannot be defended. Every maximal attack is a witness and to check its testimony we can easily apply the maximum matching algorithm presented in [1]. Moreover, we have to ensure that our method guarantees a uniform and independent choosing of a witness.

Algorithm 1.1 Attack based tester.

```
Repeat ⌊3/ε⌋ times
   Choose maximal attack A independently
    and uniformly at random.
   If A cannot be defended then
      Return no
   Else
      Return yes
   End If
End Repeat
```

Obviously, if S is secure in G, the answer is *yes*. Observe that if an input instance is ε-far from being secure, the probability of choosing indefensible attack is at least ε, thus by the *witness lemma* finding a witness in $3/\varepsilon$ tests is at least *0.95*. The detailed analysis can be found in [7].

The model based on defending sets. This model results from the definition of a secure set. We take into account all the possible subsets of S and require that a given fraction of them must satisfy the *SEC* condition. Formally, an input instance is ε-far from being secure if and only if the *SEC* condition is not satisfied for ε fraction of subsets of S. Every subset $X \subset S$ is a witness and its testimony is $SEC(X)$. We have to choose subsets of S uniformly and independently at random.

Algorithm 1.2 Subset based tester.

```
Repeat ⌊3/ε⌋ times
   Choose subset X ⊂ S uniformly and
    independently at random
   If SEC(X) = no then
      Return no
   Else
      Return yes
   End If
End Repeat
```

As previously, every secure set will be correctly determined and by the witness lemma (which is applicable for that tester) we have that finding a witness in $3/\varepsilon$ tests is at least *0.95*. The detailed analysis can be found in [7].

In this paper we are dealing with testing the security of subsets of G inducing trees and graphs with a bounded degree ($\Delta \leq 3$ or $\Delta \leq 4$). Thus we are extending our results from [2] and [7]. We proposed a very promising testing

heuristic, and our goal was to verify the probability conditions for small graphs (exhaustively generated). We were interested in counting how many sets S (exhaustively generated for small $|S|$) with random attacking sets were *no*-instances and the heuristic gave *yes*. In our heuristic we construct a candidate set that is close to being insecure, performing gluing and splitting operations, which give less false positive results than random algorithms presented in [7].

2. SECURITY PROBLEM PSEUDOTESTER

In this section we will present our main result: the heuristic algorithm with the use of an improved model based on defending sets (i.e. a valuable witness is a subset X of S with $SEC(X)=no$).

2.1 Model

We are given a graph $G = (V,E)$ and a nonempty subset S of V, and an attacking set $A = N(S) \setminus S$. Our goal is to determine $SECURE(G,S)$. Let us recall that $c(X) = |N[X] \cap S| - |N(X) \setminus S|$ and define $d(X) = |X|$. The higher the $c(X)$ is, the stronger the set X, hence we want to minimize the function c. Minimizing the second function d prevent us from terminating the algorithm too fast, with the cluster size equal to $|S|$.

The definition of ε-far. Following section 1.1, let $i = (G,S)$ be an input instance. For a given $0 < \varepsilon \leq 1$ we recall that:

$$i \in N_\varepsilon \Leftrightarrow min\{|X|: X \subset S \wedge SEC(X) = no\} \leq |S| (1 - \varepsilon)$$

2.2 Algorithm

Our goal is to find a valuable witness for set S in graph G. In the first step, we split vertex set V into mutually disjoint clusters $\{c_1, c_2,...,c_s\}$. To find the witness, we glue together the clusters chosen on the basis of reasonable criteria. Based on the idea of simulated annealing, we can also sometimes split certain clusters into smaller clusters (e.g. singletons) with some decreasing probability. The algorithm consists of a number of rounds, where each round is either a gluing or a splitting cluster. If we find a valuable witness, we stop the algorithm (the answer is *no*).

Remark 1. We require the instances to meet the following conditions:

1. S is safe: $SEC(S) = yes$,
2. all singletons are safe: $SEC(\{s\})$ for all $s \in S$,
3. no useless attackers: $N[S] = V$,
4. no useless defenders: $N[N[A]] = V$.

Let us briefly discuss these conditions to justify this selection. First of all, we can verify these requirements in polynomial time. The first and the second condition leave us with nontrivial instances. The third and the last one account for all the vertices significant to the answer.

Initial clusters. The first phase of the heuristic is to establish the initial partition of the vertex set into clusters. In the general approach we split the vertex set into random disjoint clusters, which guarantees that we can always find a witness. In our tests we split the vertex set into clusters of single elements (*singletons*), because we wanted to eliminate unexpected random successes when testing the heuristic.

Gluing clusters. In order to glue the clusters together, we use the criteria c and d, as previously defined. We calculate the value of $c(C_i \cup C_j)$ for all possible $i \neq j$ and find pairs with the minimum value. Of course, a negative value of c means that some X does not satisfy the SEC condition, therefore S is not a secure set. The algorithm objective is to construct a valuable witness, so the minimization of c is well founded. The second criterion d is used in the case of ambiguity of the c criterion. This will lead to a creation of small clusters. In case of further ambiguity, we choose a cluster at random.

Splitting clusters. Our idea is to sometimes split clusters instead of gluing them. First of all, it ensures that any witness can be reached by such an algorithm (assuming that starting clusters were randomly generated), and secondly, we improve the probability of finding a valuable witness. Yet to ensure that our algorithm fulfils the halting property (no clusters to glue) we opt for the decreasing probability of the cluster splitting by means of splitting with the use of the simulated annealing concept. We introduce two functions to define the probability of the cluster splitting. We call them *outer* and *inner probability* functions. To define these functions, we use the average cluster size $EC(r)$ in round r and $bs(r)$ as the total number of splitting rounds before round r. The outer probability function $g: \boldsymbol{R} \rightarrow [0,1]$ has a real argument, which is $bs(r)$. We require it to be a slowly decreasing function tending to zero to achieve the halting property. Our proposal is:

$$g(x) = max\{0, 3/2 - e^{x/h}\},$$

where h is constant or instance dependent.

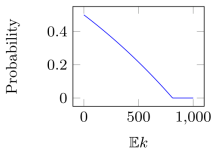

Fig 2: The inner probability function for $h = 5000$

The inner probability function $f: \boldsymbol{R} \rightarrow [0,1]$ also has a real argument, which is $EC(r)$. We require that the function f increases. For a given ε we can define $k_0 = |S|(1 - \varepsilon)$ and require that $f(k_0) = ½$, which means that if the average size of the cluster is greater than k_0, then the cluster is more likely to be split. Moreover, we require that $f(k_0 - p|S|) = q$, where $p, q \in (0,1)$, which defines the slope of the function as near to the point k_0. Our candidate is the following function:

$$f(x) = ½ + 1/\pi \, arctan(c(p,q)\,(x - k_0))$$

where:

$$c(p,q) = - \tan(\pi(q - ½)) / p|S|$$

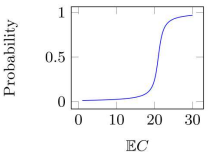

Fig 3: The outer probability function for:

$|S| = 30$, $\varepsilon = 0.3$, $p = 0.1$ and $q = 0.1$

Finally, we obtain the probability of splitting clusters in round r as the product of both inner and outer probability: $f(EC(r))g(bs(r))$. The decision which cluster to choose is also probabilistic and depends on the cluster size. In fact, we are dealing here with the roulette method of the probability distribution. The probability of the selection of the cluster C_i is proportional to the cluster size and equals $|C_i| / \sum_j |C_j|$.

Heuristic and tester pseudocode. We construct the tester *TH* and the heuristic *H*.

Algorithm 2.1 The tester *TH* based on the heuristic $H(\varepsilon, f, g, c, d)$

```
Repeat ⌊3/ε⌋times
   If ¬H(ε,f(ε),g,c,d) then
      Return no
   End If
End Repeat
Return yes
```

Algorithm 2.2 Security problem heuristic schema $H(\varepsilon, f, g, c, d)$

```
Split S into s disjoint clusters C₁,…,Cₛ
bs := 0
Repeat
   EC := average cluster size
   Pick at random Y or N with
    probabilities:  P(Y) = f(EC)g(bs) and
                    P(N) = 1 - P(Y)
   If Y then
      Pick one random cluster Cᵢ with
       probability P(Cᵢ) = |Cᵢ| / ∑ⱼ |Cⱼ|
      Split cluster Cᵢ into singletons.
      bs := bs + 1
   End If
   For every pair of distinct clusters
    Cᵢ,Cⱼ calculate c(Cᵢ ∪ Cⱼ)
   Join both clusters with minimum c, in
    the case of ambiguity use criterion d
Until there is only one cluster or there
   is a cluster with SEC = no
If there is a cluster with SEC = no then
   Return no
Else
   Return yes
End If
```

3. TESTS

We have conducted numerical tests for the tester *TH* in order to verify the probability conditions from section 1.1 for the definition of ε-far given in the same section. We used three virtual machines running in a cloud, each of them had 8 logical computation cores (hardware-backed with Intel Hex-Core CPUs), 160GB SSD and 16GB of RAM. The implementation was done in Python 2.7 and the source code is available on GitHub [8].

3.1 Test parameters

We used five parameters in our tester *TH*, namely:

- ε (as in the definition of ε-far),
- the *inner probability* function $f(\varepsilon, p, q)$,
- the *outer probability* function g,
- the *first merge* criterion c,

- the *second merge* criterion *d*.

We have chosen only one candidate for the outer probability function $g(x) = max\{0, 3/2 - e^{x/1000}\}$. The inner probability is a function *f* with three parameters:

- ε - determines whether *f* reaches ½,
- *q* - the value to be reached,
- *p* - determines whether *f* reaches *q*.

Let us consider the inner probability function $f(x) = 1/2 + 1/\pi \ arctan(c(x - k_0))$, where k_0 is a parameter dependent on the instance and equals $k_0 = |S|(1-\varepsilon)$, and $c = -tan(\pi(q - ½))/(p|S|)$. Our test cases cover four sets of *p*, *q*: (*p*, *q*) = (*0.1, 0.1*), (*p*, *q*) = (*0.1, 0.2*), (*p*, *q*) = (*0.2, 0.1*) and (*p*, *q*) = (*0.2, 0.2*). As mentioned before, we consider criteria *c* and *d* (both intended to be minimized) given by the formulas: $c(X) = |N[X] \cap S| - |N[X] \setminus S|$ and $d(X) = |X|$. The complexity of the heuristic depends on the parameter h of the function g. Taking a constant value of *h* in our tests (*h = 1000*) we guarantee that the simulated annealing process stops after a constant number of steps. Thus the overall complexity is polynomial.

3.2 Test cases

Generation method. In the first step, we construct the graph *G*[*S*] (i.e. a graph with the vertex set equal to *S*). We exhaustively generate all connected small graphs of certain classes using the nauty package [11]:

- Trees T_n where $n \leq 16$,
- Subcubic graphs ($\Delta \leq 3$) SC_n where $n \leq 12$,
- Subquadro graphs ($\Delta \leq 4$) SQ_n where $n \leq 9$,

where *n* refers to the number of vertices.

In table 1 we present the number of the generated graphs.

n =	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
T_n	*1*	*1*	*1*	*2*	*3*	*6*	*11*	*23*	*47*	*106*
SC_n	*1*	*1*	*2*	*6*	*10*	*29*	*64*	*194*	*531*	*1733*
SQ_n	*1*	*1*	*2*	*6*	*21*	*78*	*353*	*1929*	*12207*	

n =	***11***	***12***	***13***	***14***	***15***	***16***
T_n	*235*	*551*	*1301*	*3159*	*7741*	*19320*
SC_n	*5524*	*19430*				

Table 1. The total number of the generated graphs.

The generation method for the attacking set. In the second step, we attach attacking vertices *A* to the graph *G*[*S*], thus obtaining the whole graph *G* with $V(G) = S \cup A$. Thus any graph *S* will be a candidate to be secure in graph *G*. Each full test case (i.e. the graph *G*[*S*] and the set *A*) is constructed in the following steps:

- Set the average attacker degree Δ_a,
- Take the connected graph *S* as a base,
- Generate the vertex set *A*, with $|A| = |S|$,
- For each $a \in A$ choose a vertex $s \in S$ from the set $\{s: SEC(\{s\})\}$ independently and uniformly at random and add the edge $\{a,s\}$.
- Repeat the following steps:
  - Take the attacker $a \in A$ independently and uniformly at random.
  - Take a vertex $s \in S$ from the set $\{s: SEC(\{s\})\}$ independently and uniformly at random and add the edge $\{a,s\}$.
- Until the set $\{s \in S: SEC(\{s\})\}$ is empty or the average attacker degree is greater than or equal to Δ_a.
- Verify if the constructed input instance meets the requirements specified in remark 1.

For each generated set *S* construct (if possible) *100* attacking sets.

Test cases choice motivation. As already mentioned, the problem of determining whether a set *S* is secure in graph *G* is *coNP*-complete. However, for trees and subcubic graphs there exist polynomial time algorithms [7]. We test our heuristic on extended but still related graph classes. We do not require the graph *G* to be a tree (or a subcubic graph) but only the induced subgraph *G*[*S*] should be a tree (or a subcubic graph). In addition, we go a bit further and extend the test cases to graphs with $\Delta \leq 4$. We conjecture that the problem for such graphs is *coNP*-complete.

The experiment description. Obviously, our heuristic gives a correct answer for the secure set *S* in graph *G* (no valuable witness). Thus we take into consideration only these instances where *S* is not a secure set. We use an exact brute-force algorithm (of exponential complexity) to verify that such an instance is a *no*-instance. Additionally we find a valuable witness *X* of the smallest cardinality. Finally, we obtain the greatest value of $\varepsilon = 1 - |X|/|S|$ such that the instance is contained in N_ε. Ultimately, we start the heuristic

for graphs (instances) prepared in that manner. Each instance is associated with a proper ε. We demand that the probability of the *yes*-answer given by the ε-heuristic is less than *5*%.

Extracting hard cases. In the third step, due to the limited computational resources, we selected only these graph instances that failed each test (i.e. instance is in *N* and all tests gave the *yes*-answer), and which we suppose are good candidates for breaking the probability conditions of the tester. For each such instance we verified if it is a member of N_ε by running our tester *1000* times. If the instance can be confirmed by the heuristic as belonging to the set *Y* with the probability greater than *0.05*, we call it a *hard instance*.

3.3 Test results

Results for the average attacker degree of *1*. For instances where the average attacker degree equals *1*, our heuristic was almost perfect (over *99.99*% of well recognized cases). We retested all false positive results carefully in order to determine whether they are instances of N_ε. We did not find any hard instances.

Results for the average attacker degree of *2*. For instances where the average attacker degree equals *2*, we did not find any hard instances where *S* is a tree. The results for subcubic and subquadro ($\Delta \leq 4$) instances are presented below in table 2.

n =	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
SC_n	*1*	*1*	*0.5*	*0.1*	*0.2*	*0.3*	*0.5*	*0.4*	*0.3*	*0.16*	*0.08*
SQ_n	*0*	*0*	*0*	*0*	*0.2*	*0.8*	*1.3*	*1.6*			

Table 2. The fraction of instances suspected to be hard. The values represent the percentage of the suspected instances.

After having found the instances suspected to be hard, we conducted additional tests in order to confirm that they actually are hard instances. The overall results for the instances confirmed to be hard are summarized in table 3. We skip the columns which both values equal *0*%.

n =	***6***	***7***	***8***	***9***	***10***	***11***	***12***
SC_n	*0.14*	*0.2*	*0.41*	*0.23*	*0.18*	*0.08*	*0.04*
SQ_n	*0.13*	*0.61*	*0.93*	*1.12*			

Table 3. The fraction of instances confirmed to be hard. The values represent the percentage of hard instances.

Example of an instance confirmed to be hard. The graph presented in figure 3 is a hard instance for our heuristic. The set *S* is depicted as filled nodes whereas its indefensible subset is illustrated as four filled nodes with an envelope. Any set of the cardinality less than *4* fulfils the *SEC* condition, thus it is an ε-far instance for $\varepsilon = 1 - 4/6 = 1/3$. Note that from the very beginning we expected the *1*/*3*-heuristic to be an effective algorithm for that instance.

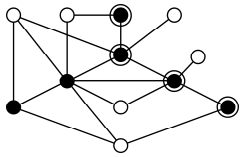

Fig 4: Hard case for the heuristic
($|S| = 6$, $|A| = 6$, the size of the smallest unsecure set: *4*).

Conclusions and further research. Our tests showed that the proposed construction of the heuristic looks very promising. We suspect that its high efficiency can be increased by the modification of the coefficients used in the algorithm. We definitely hope to resolve the two cases when $G[S]$ is a subcubic graph or a tree. We plan to continue our research to give strong evidence to the conjecture that the security problem for these classes is in *NP*, or even in *P*. To improve the heuristic we consider alternative functions for the inner and outer probability functions. Moreover, we plan further tests of the heuristic for large sparse random graphs with a well estimated ε-far (by attaching some small hard instances). One of the results of the computations are hard instances which can be attached to the much bigger graph. Described technique will be helpful with the creation of large instances with a well estimated ε-far.

References

1. Blukis T. „Secure sets in graphs” (in Polish), MSc Thesis at Gdańsk University of Technology, 2012.
2. Blukis T., Lewoń R., Małafiejski M., „Efficient algorithms for graph security testing”, IX International Colloquium on Graphs and Optimization (Sirmione, Italy), 2014 (prepared to submit to special issue of Discrete Applied Mathematics).
3. Brigham R., Dutton R., Hadetniemi S., „Security in graphs”, Discrete Applied Mathematics 2007, vol. 155, pp. 1708-1714.
4. Dutton R., „Secure set algorithms and complexity”, Congressus Numerantium 2006, vol. 180, pp. 115-121.

5. Dutton R., Lee R., Brigham R., „Bounds on a graph's security number", Discrete Applied Mathematics 2008, vol. 156, pp. 695-704.
6. Dutton R., Enciso R., „Parameterized complexity of secure sets", Congressus Numerantium 2008, vol. 189, pp 161-168.
7. Gieniusz T., Lewoń R., Małafiejski M., „Graph security testing", Journal of Applied Computer Science 2014, vol. 22, no. 2 (in press).
8. GitHub repository, http://github.com/ivyl/graph-security
9. Goldreich O., Ron D., „A Sublinear Bipartitness Tester for Bounded Degree Graphs", Combinatorica 1999, vol. 19, pp. 335-373.
10. Goldreich O., Goldwasser S., Ron D., „Property testing and its connection to learning and approximation", Journal of the ACM 1998, vol. 45, pp. 653-750.
11. McKay B., nauty Software Program, Version 2.5, http://cs.anu.edu.au/˜bdm/nauty/, 2013.
12. Raskhodnikova S., „Property Testing: Theory and Applications", PhD Thesis at Massachusetts Institute of Technology, 2003.
13. Rubinfeld R., Sudan M., „Robust characterizations of polynomials with applications to program testing", SIAM Journal on Computing 1996, vol. 25, no. 2, pp. 252-271