Finding the Most Prominent Group in Complex Networks

Article *in* Ai Communications · January 2007

Source: DBLP

CITATIONS
63

READS
434

3 authors:

Rami Puzis
Ben-Gurion University of the Negev
140 PUBLICATIONS **1,531** CITATIONS

SEE PROFILE

Yuval Elovici
Ben-Gurion University of the Negev
482 PUBLICATIONS **12,061** CITATIONS

SEE PROFILE

Shlomi Dolev
Ben-Gurion University of the Negev
487 PUBLICATIONS **8,868** CITATIONS

SEE PROFILE

Finding the Most Prominent Group in Complex Networks *

Rami Puzis [a,**], Yuval Elovici [b] and Shlomi Dolev [c]
[a] *Deutsche Telekom Laboratories at Ben-Gurion University of the Negev*
e-mail: puzis@bgu.ac.il
[b] *Deutsche Telekom Laboratories at Ben-Gurion University of the Negev*
Tel: 972-8-647-7551
e-mail: elovici@inter.net.il
[c] *Department of Computer Science Ben-Gurion University of the Negev*
Tel: 972-8-647-2715
e-mail: dolev@cs.bgu.ac.il

In many applications we are required to locate the most prominent group of vertices in a complex network. Group Betweenness Centrality can be used to evaluate the prominence of a group of vertices. Evaluating the Betweenness of every possible group in order to find the most prominent is not computationally feasible for large networks. In this paper we present two algorithms for finding the most prominent group. The first algorithm is based on heuristic search and the second is based on iterative greedy choice of vertices. The algorithms were evaluated on random and scale-free networks. Empirical evaluation suggests that the greedy algorithm results were negligibly below the optimal result. In addition, both algorithms performed better on scale-free networks: heuristic search was faster and the greedy algorithm produced more accurate results. The greedy algorithm was applied for optimizing deployment of intrusion detection devices on network service provider infrastructure.

Keywords: Complex Networks, Group Betweenness Centrality, Heuristic Search

1. Introduction

Complex networks are used to study the structure and dynamics of complex systems in various disciplines [1]. For example, social networks [2,3], protein interactions networks [4], and computer networks such as the Internet [5,6] are all classified as complex networks. In social networks vertices are usually individuals and edges characterize the relations between them; in computer networks, vertices might be routers connected to each other through communication lines.

Many naturally evolved complex networks are characterized by a Scale-Free (SF) structure [7,8] and in particular the power-law distribution [5] of Connectivity Degree. Scale-free networks have a few vertices with a very high degree of connectivity [9]. While a power-law degree distribution is the main characteristic of scale-free networks, many networks are also characterized by a power-law Betweenness distribution [10].

Identification of the most central group of vertices in a network is an important issue from both theoretical and practical points of view. For example, Ballester et al. state in [11] the importance of finding the key group in a criminal network. Borgatti elaborates in [12] on a Key Player Problem (KPP) that is strongly related to the cohesion of a network. The author defines two problems KPP-Pos and KPP-Neg. The solution of the first problem is a group maximally connected to all other vertices in a graph and the solution of the second is a group maximally disrupting the network. In this paper we define a similar optimization problem: find the group that has maximal potential to control traffic in communication networks. Analogous to [12] we will call this problem KPP-Com.

The potential to control traffic is typically attributed in literature to vertices with high Shortest Path Betweenness Centrality (BC) [13,14,15]. In fact, BC of vertices was used in [16] to define a congestion-free routing strategy. Everett and Borgatti [17] defined Group Betweenness Centrality (GBC) as a natural extension of the Betweenness measure. GBC is used to compute the prominence of groups of vertices in complex networks. Free-

*Supported by Deutsche Telekom.
**Corresponding author: Rami Puzis, puzis@bgu.ac.il

AI Communications
ISSN 0921-7126, IOS Press.

man [13] has defined the Group Betweenness Centralization index as a measure of homogeneity of the members' Betweenness. In this paper we use the GBC definition of Everett and Borgatti and our main goal is to find the group with the highest GBC.

In many applications we are required to locate the most prominent group. One possible application is optimizing deployment of intrusion detection devices in a wide area network. Self-propagating malicious code (computer viruses and worms) poses a significant threat to Internet users. Only some of the users connected to the Internet are protected by updated anti virus tools. One way to prevent users from being infected by such threats is to clean the traffic at the level of Network Service Provider (NSP). The NSP traffic can be monitored and cleaned by a Distributed Network Intrusion Detection System (DNIDS) that may be deployed on the NSP routers/links [18]. It is not realistic to inspect the traffic of all the routers/links of the NSP. Theoretical models suggest protecting the most significant routers/links in order to slow down the threat propagation in the entire network [19]. Previous research has shown that NSP network infrastructure can be regarded as a complex network [6] and it has scale-free properties [5,10]. Thus, the group of routers/links that together have the highest influence on communication in the NSP infrastructure can be located by finding the group with the highest GBC.

Park [20] exploited the scale-free structure of computer networks to administer scalable deployment of systems for protection against Distributed Denial of Service (DDoS) attacks and worm containment. The author suggests that deployment based on Vertex Cover can be small enough and provide a high level of protection for the entire network. Small Vertex Cover can be found iteratively by choosing vertices that are connected to the most unprotected peers. In this way a network can be protected from DDoS attacks with only 15% deployment and efficient worm containment can be provided by filters deployed on 4% of the vertices. In contrast to scale-free computer networks, the same deployment sizes provide much less protection when applied to random networks.

In this study we propose two algorithms to solve the KPP-Com problem and we illustrate how the solution of KPP-Com can be used to optimize the deployment of DNIDS in a large network. In the first algorithm we use heuristic search based on Depth First Branch and Bound (DFBnB) when using four different heuristic functions. In the second algorithm we use a greedy approach in order to speedup the search. The algorithms were evaluated on random and scale-free networks for various group sizes. Our evaluation indicates that the greedy algorithm's results are very close to the optimal solution. We also discovered that the heuristic search algorithm performed much faster when applied to scale-free networks compared to random networks. In addition, we found that the quality of solutions produced by the iterative greedy algorithm is higher when the algorithm is applied to scale-free networks compared to random networks. We concluded our evaluation by using the greedy algorithm to optimize the deployment of DNIDS on the infrastructure of NSP.

The rest of the paper is structured as follows: In Section 2 we elaborate on the Group Betweenness Centrality measure; in Section 3 we present our algorithms. In Section 4 we present the evaluations of the algorithms and in Section 5 their application to network security; we conclude the paper in Section 6. Notations and abbreviations used in this paper are summarized in Appendix A.

2. Group Betweenness Centrality

In this study we want to find a group of vertices that has maximal control over the traffic in a communication network $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges. To simplify the discussion we assume that G is a connected, unweighted and undirected graph. However, all proposed methods are also applicable to networks that have no such constraints.

Shortest Path Betweenness is considered to be an accurate estimation of the information flow controlled by a vertex in communication networks that use shortest path routing [15,16]. However, in computer networks traffic may diverge from shortest paths due to load balancing. There are centrality measures that are not designed for shortest path routing. In particular, Flow Betweenness [21] equally respects all paths while Random Walk Betweenness [22] favors shorter paths over the longer ones.

In this study we assume that every two vertices in the network communicate with each other

mainly through shortest paths. Therefore, we are primarily interested in locating the group with maximal Shortest Path Group Betweenness Centrality (BC). We also assume that information sent by $s \in V$ to $t \in V$ can be inspected or modified by s, t or any intermediate vertex that participates in forwarding this information. The KPP-Com problem focuses on finding a group of vertices $S \subseteq V$ of size k that can inspect or modify most of the information flow in the network. Next we will briefly present the definitions of Betweenness Centrality (BC) and Group Betweenness Centrality (GBC).

All shortest path Betweenness measures are defined as a summation over all pairs of vertices in a graph. Let s and t be two vertices in a graph. Let $\sigma_{s,t}$ be the total number of different shortest paths that connect s and t. Let v be a vertex that lies on a shortest path between s and t. We denote the number of shortest paths from s to t that pass through v by $\sigma_{s,t}(v)$. In order to determine how many different paths from s to t traverse v we multiply the number of shortest paths from s to v ($\sigma_{s,v}$) by the number of shortest paths from v to t ($\sigma_{v,t}$).

$$\sigma_{s,t}(v) = \begin{cases} \sigma_{s,v} \cdot \sigma_{v,t} & d(s,t) = d(s,v) + d(v,t) \\ 0 & otherwise \end{cases}$$

where $d(x,y)$ is the distance between vertices x and y.

Betweenness Centrality (BC) of vertex $v \in V$ represents the total influence that v has on communications between all possible pairs of vertices in the graph. The Betweenness Centrality of vertex v is

$$BC(v) = \sum_{s,t \in V | s \neq v \neq t} \left(\frac{\sigma_{s,t}(v)}{\sigma_{s,t}} \right)$$

where the fraction $\frac{\sigma_{s,t}(v)}{\sigma_{s,t}}$ represents the influence of v on the communication between s and t.

In an equivalent definition of BC, the distinction constraint ($s \neq v \neq t$) in the above equation can be replaced with ($s \neq t$) if we define $\sigma_{x,x} = 0$. This definition results in $\sigma_{s,t}(t) = \sigma_{s,t}(s) = 0$.

Note that in the original definition of BC, shortest paths that start or end at v are not included in the computation of $BC(v)$. It is more convenient to think that information originating at some vertex is seen by this vertex and hence, should be accounted for. We want to include shortest paths that start or end at v in the computation of v's Betweenness. For this reason, in this study we define $\sigma_{x,x} = 1$ which results in $\sigma_{s,t}(t) = \sigma_{s,t}(s) = \sigma_{s,t}$. A similar variation of BC was previously mentioned [10]. Defining $\sigma_{x,x} = 1$ results in addition of a constant term $(2 \cdot (n-1))$ to the BC of a single vertex in a connected network.

BC of individual vertices can be naturally extended to the Betweenness Centrality of groups of vertices [17]. Let $S \subseteq V$ be a group of vertices. $GBC(S)$ stands for the total fraction of shortest paths between all pairs of vertices that pass through at least one member of the group S. Let $\ddot{\sigma}_{s,t}(S)$ be the number of shortest paths between s and t that traverse at least one member of the group S. The Group Betweenness Centrality of group S is

$$GBC(S) = \sum_{s,t \in V | s \neq t} \left(\frac{\ddot{\sigma}_{s,t}(S)}{\sigma_{s,t}} \right)$$

It can be proved by straightforward reduction from the *Minimal Vertex Cover* problem [23] that finding a group with maximal GBC is NP-hard. Any group that controls all the information flows in the network (in particular, communications of adjacent vertices) must have one vertex on every edge in the network and vise versa. One way to cope with the complexity of the KPP-Com problem is to employ heuristic search. Alternatively, a suboptimal polynomial time algorithm can be used when the network and group become relatively large.

During the search for the group with maximal GBC, many groups are evaluated. The algorithm presented in [24] computes the GBC of a given group of vertices. The complexity of this algorithm is $O(k^3)$, where k is the size of the group. The algorithm requires preprocessing the complexity of which is $O(n^3)$ where n is the size of the network. Good heuristic search decreases the number of groups that have to be evaluated. Nevertheless, the number of groups evaluated during the search is proportional to n^k. Thus, when searching for groups with size three or more, the time spent on preprocessing is negligible compared to the time spent on the search.

3. Solving the KPP-Com

3.1. The search space

In this section we will present a new algorithm for finding the most influential group of vertices

in a network using heuristic search. In order to avoid confusing the search space with the original input graph, we will use the terms "nodes" and "transitions" to refer to parts of the search space, and the terms "vertices" and "edges" to refer to parts of the input graph.

Every node (denoted by C) maintains one group of vertices (denoted by $GM(C)$) and a list of candidate vertices (denoted by $CL(C)$) that may join the group. We will refer to $CL(C) = (c_1, c_2, \ldots)$ as an ordered list of vertices. The order of vertices in $CL(C)$ may vary between the nodes. The methods of ordering vertices in $CL(C)$ will be described in Section 3.3. We will refer to the first vertex in the ordered list $CL(C)$ by the notation v^{best}.

We will refer to the search space as a decision tree. At every node we decide whether v^{best} is included in the most influential group. According to this decision we branch to two sub-trees, one containing all the groups that include v^{best} and the other containing all the groups that do not include it. Thus we assure that all possible solutions can be found during the search.

Every node has two children: the left child is denoted by C^- and the right child is denoted by C^+. $GM(C^+)$ contains all the vertices of $GM(C)$, additionally including v^{best}, and $GM(C^-)$ is equal to $GM(C)$.

$$GM(C^+) = GM(C) \cup \{v^{best}\}$$
$$GM(C^-) = GM(C)$$

The candidates lists of both children contain all the vertices from the candidates list of their father excluding v^{best}.

$$CL(C^+) = CL(C^-) = CL(C) \setminus \{v^{best}\}$$

The root node of the tree represents an empty group of vertices. Its candidates list contains all the vertices in the input graph. Every child of the root holds a candidates list that is missing one vertex and a group that includes one vertex or none. All nodes of the second level hold a candidates list that is missing two vertices and a group that includes zero, one, or two vertices, etc.

Since we are looking for a group of size k, we will refer to nodes with $|GM(C)| = k$ as possible solutions to our problem. We exclude from the search space all sub-trees that do not contain a possible solution. These can be either sub-trees rooted at nodes with $|GM(C)| \geq k$ or sub-trees rooted at nodes with $|CL(C)| < k - |GM(C)|$. The minimal depth of a leaf in the tree is k while the maximal depth of a leaf equals to the size of the input graph.

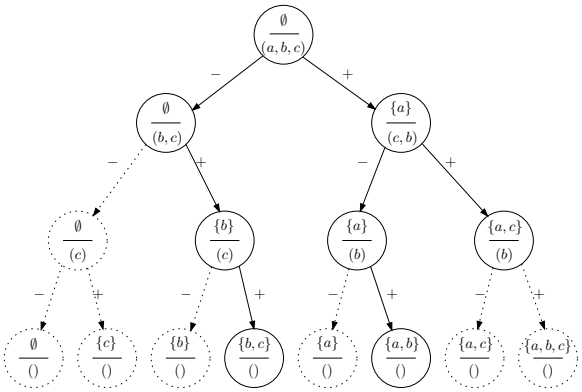

Fig. 1. Tree representing the KPP-Com search space for $n = 3$ and $k = 2$. Dashed lines represent sub-trees that do not contain possible solutions. The set in curly braces $\{\ldots\}$ represents $GM(C)$ and the list in parenthesis $(\ldots)$ represents $CL(C)$.

For example, let a, b, and c be vertices of the input graph ($n = 3$). Assume we are searching for a group of size two ($k = 2$) that has maximal GBC. Figure 1 presents an example of the search space. The search should start at the root node R whose candidates list is (a, b, c) and $a = v^{best}$. We consider all groups where a is a member by moving to the right child (following the "+" transition). R^+ maintains the group $\{a\}$ and a candidates list that includes all vertices of the original graph, excluding the vertex a. It is possible that the order of vertices in $CL(R^+)$ is different from the order of vertices in $CL(R)$. We consider all groups where a is not a member by following the "−" transition from the root node. R^- maintains an empty group and candidates list that is identical to the candidates list of R^+.

3.2. The search algorithm

We use the Depth First Branch and Bound (DFBnB) algorithm [25] to search the tree for the group with maximal GBC. DFBnB is known to be effective when the depth of the search tree is known, as in our case. DFBnB is similar to DFS but it prunes nodes according to a global bound. We begin the search with a bound equal to zero. During the search the bound is equal to the maximal GBC found so far.

In contrast to the traditional search that is aimed at finding the node with minimal cost, we

are using a utility based approach. We define the utility of the node C as $g(C) = GBC(GM(C))$. A heuristic function $h(C)$ estimates the maximal utility that can be gained by exploring the sub-tree rooted at C. $f(C) = g(C)+h(C)$ is a function that estimates the maximal utility of nodes in C's sub-tree.

The utility of the root node R is equal to zero because $GM(R) = \emptyset$. $h(R)$ is the upper bound on the optimal solution. While searching down the tree, $g(C)$ will grow and $h(C)$ will decrease. When the algorithm finds a possible solution, $h(C)$ is equal to zero and $f(C)$ is equal to $GBC(GM(C))$ where $|GM(C)| = k$.

Pruning decisions made during the search are based on the value of $f(C)$ and the maximal GBC found so far. We can guarantee that the heuristic search will find the optimal solution only if the function $f(C)$ is an upper bound on the maximal GBC that can be found within the sub-tree rooted at C. If this upper bound is below the maximal GBC found so far, the sub-tree is pruned, otherwise it is explored in hope of finding a group with a higher GBC. Heuristic functions used for pruning nodes in the search tree are described in Section 3.4.

When visiting a node, beside the decision whether to prune the current sub-tree, the algorithm should also determine v^{best}. The following section describes two methods of ordering vertices in $CL(C)$ and determining v^{best}.

3.3. Order of vertices in the candidates list

Admissible heuristic functions guarantee that during the search we will find the optimal solution (note that the search may take an exponential time). The order of vertices in $CL(C)$ is important for choosing v^{best} and for computing admissible pruning heuristics. The order of vertices in $CL(C)$ can be determined by either their individual Betweenness or their contribution to Betweenness of $GM(C)$.

Algorithms presented in [26,27,28] compute individual BC for all vertices in the graph with time complexity $O(mn)$. We can sort vertices in $CL(C)$ by their individual BC where the first vertex is the vertex with the highest BC. The computation of BC and the sorting can be done once for the entire search. Sorting vertices by their individual BC will impose the same order on all candidates lists in the tree.

Using individual BC is not the best way to order the vertices $CL(C)$. Assume that there is a vertex in $CL(C)$ with very high BC. Assume also that most of the traffic covered by this vertex is already covered by $GM(C)$. In such case it is misleading to use BC as a measure for the potential of this vertex to help the group in covering more traffic. For this reason we prefer to use the contribution of vertices to GBC of $GM(C)$ rather than their individual Betweenness.

Let $v \in V$ be some vertex in the graph. Let $S \subseteq V$ be a group of vertices. We denote the contribution of the vertex v to the GBC of S by $BC^S(v)$.

$$BC^S(v) = GBC(S \cup \{v\}) - GBC(S)$$

The algorithm presented in [24] is able to compute the contribution of all vertices in the graph with time complexity $O(kn^2)$. Time complexity of updating the contribution of all vertices in the graph when one vertex is added to the group is $O(n^2)$. The actual contribution of vertices in $CL(C)$ depends on the members of $GM(C)$. Therefore, the contribution of vertices should be recalculated each time the search algorithm moves to the right child (by adding v^{best} to $GM(C)$). Recalculating the contribution of vertices in $CL(C)$ may change their order.

3.4. Heuristic evaluation functions

We propose four different heuristic functions for estimating the maximal gain in GBC that can be acquired during exploration of the sub-tree rooted at C. The functions differ in their computation time and accuracy.

The first two functions, h_1 and h_2, assume that the candidates list $CL(C)$ is ordered according to the individual Betweenness of its members.

$$h_1(C) = BC(c_1) \cdot (k - |GM(C)|)$$

In $h_1(C)$ we multiply the BC of the first vertex in the candidates list by the number of vertices that need to be added to $GM(C)$ in order to construct a group of size k. This function is a very rough upper bound. h_1 can be computed at $O(1)$ since the individual Betweenness of all vertices is calculated once prior to the search.

$$h_2(C) = \sum_{i=1}^{k-|GM(C)|} BC(c_i)$$

In $h_2(C)$, we sum the Betweenness of the $k - |GM(C)|$ vertices with maximal individual Betweenness in $CL(C)$. The function $h_2(C)$ is a refinement of $h_1(C)$ because $BC(c_1)$ is higher than or equal to $BC(c_i)$ for any i. Computational complexity of h_2 is $O(k)$.

The next two functions, h_3 and h_4, assume that the candidates list $CL(C)$ is ordered according to the contribution of its vertices to the GBC of $GM(C)$.

$$h_3(C) = BC^{GM(C)}(c_1) \cdot (k - |GM(C)|)$$

The function $h_3(C)$ is similar to the first function, but in this case we locate the vertex in $CL(C)$ with the highest contribution to the GBC of the group $GM(C)$ and not with the highest individual Betweenness. Since the contribution of a vertex is always smaller than its individual Betweenness, h_3 is a refinement of h_1. We should keep in mind that h_3 includes the computation of the contributions. The computational complexity of h_3 is $O(1)$ when performing the "$-$" transitions. When performing the "$+$" transitions there are two options: if $|GM(C)| + 1 < \sqrt{n}$ then the computational complexity of h_3 is $O(k^2 n)$ otherwise it is $O(n^2)$.

$$h_4(C) = \sum_{i=1}^{k-|GM(C)|} BC^{GM(C)}(c_i)$$

The function $h_4(C)$ is similar to h_2, but in this case we order the vertices in $CL(C)$ by their contribution to the GBC of $GM(C)$. This function is more accurate than all the other functions, since it is a refinement of both h_2 and h_3. Computational complexity of h_4 is $O(k)$ when performing the "$-$" transitions. When performing the "$+$" transitions there are two options: if $|GM(C)| + 1 < \sqrt{n}$ then the computational complexity of h_4 is $O(k^2 n)$, otherwise it is $O(n^2)$.

The above four heuristic functions can be used for pruning sub-trees. If $f(C)$ is below the maximal GBC found so far, the sub-tree rooted at C will be pruned. Otherwise the algorithm will visit the two children of C by first following the "$+$" transition and afterwards the "$-$" transition. v^{best} is chosen from the candidates list $CL(C)$ according to the heuristic evaluation function we use. For h_1 and h_2 v^{best} is a candidate with highest Betweenness centrality. For h_3 and h_4 v^{best} is a candidate with highest contribution to GBC of the group represented by the current node.

3.5. Admissibility of pruning heuristics

We will prove that all of the heuristic functions described in the previous subsection are admissible. Since we are working with utility, we will show that the heuristic function f overestimates the GBC of any group in the sub-tree. We will show that $h_2(C)$ is always lower than or equal to $h_1(C)$ and that $h_4(C)$ is lower than or equal to $h_2(C)$ and $h_3(C)$. In light of this partial order, it is enough to show that $f(C) = g(C) + h_4(C)$ overestimates the maximal GBC that can be found within the sub-tree rooted at C.

Let $\{BC(v)\}_{v \in CL(C)}$ be a sequence of Betweenness values. Let $\{BC^{GM(C)}(v)\}_{v \in CL(C)}$ be a sequence of contributions to the Group Betweenness of $GM(C)$. Let $l = k - |GM(C)|$ be the number of vertices that are still missing in the group represented by the node C. Contribution of a vertex to the GBC of some group is always lower than or equal to its individual Betweenness [24]. Summation of the l highest contributions is lower than or equal to the summation of the l highest Betweenness values of distinct vertices in $CL(C)$, thus $h_2(C) \geq h_4(C)$. h_2 and h_4 are summations of the l highest elements in $\{BC(v)\}_{v \in CL(C)}$ and $\{BC^{GM(C)}(v)\}_{v \in CL(C)}$, respectively. h_1 and h_3 are l times the highest element in the respective lists, therefore, $h_1(C) \geq h_2(C)$ and $h_3(C) \geq h_4(C)$.

The following lemma is based on the concepts described in [24].

Lemma 3.1 *h_4 is an admissible heuristic function.*

Proof: To prove that h_4 is admissible, we need to prove that for any descendant C' of C:

$$g(C') - g(C) \leq h_4(C)$$

Let $q = |GM(C')| - |GM(C)|$, let $S = \{s_1, \ldots, s_q\}$ be a subset of $CL(C)$ such that $GM(C') = GM(C) \cup S$, and let $c_1, \ldots, c_{|CL(C)|}$ be the elements of $CL(C)$ in order of decreasing $BC^{GM(C)}(c_i)$. The following five steps (explained below) complete the proof:

$$g(C') - g(C) = \sum_{i=1}^{q} BC^{GM(C) \cup \{s_1, \ldots, s_{i-1}\}}(s_i) \quad (1)$$

$$\leq \sum_{i=1}^{q} BC^{GM(C)}(s_i) \quad (2)$$

$$\leq \sum_{i=1}^{q} BC^{GM(C)}(c_i) \quad (3)$$

$$\leq \sum_{i=1}^{k-|GM(C)|} BC^{GM(C)}(c_i) \quad (4)$$

$$= h_4(C) \quad (5)$$

Steps (1) follows from the definition of g. Step (2) follows from the fact that the joint contribution of all vertices in S is smaller than or equal to the sum of their individual contributions. For any two vertices $s_i, s_j \in S$ the contribution of s_i to the GBC of $GM(C)$ is greater than or equal to its contribution to the GBC of $GM(C) \cup \{s_j\}$ because some of the shortest paths contributed by s_i to $GM(C)$ may also pass through s_j. This implies that for any superset S' of $GM(C)$ (in particular for $S' = GM(C) \cup \{s_1, \ldots, s_{i-1}\}$) it holds that:

$$BC^{GM(C)}(s_i) \geq BC^{S'}(s_i)$$

Step (3) follows from the fact that the c_i are ordered from high to low $BC^{GM(C)}$. Hence, for any set $\{s_1, \ldots, s_l\}$ it holds that:

$$\sum_{i=1}^{l} BC^{GM(C)}(s_i) \leq \sum_{i=1}^{l} BC^{GM(C)}(c_i)$$

Step (4) follows from $q < k - |GM(C)|$ and the positiveness of BC values. Step (5) follows from the definition of h_4. ■

3.6. Suboptimal polynomial time greedy algorithm for KPP-Com

A naïve greedy algorithm for locating an influential group of vertices can choose k vertices with the highest individual Betweenness. We will denote this algorithm as $TopK$. k vertices with highest individual Betweenness can be selected after single execution of the algorithm that computes BC for all vertices in the graph [28,26]. Therefore, the complexity of the $TopK$ algorithm is $O(nm)$.

We hypothesize that the first solution encountered during DFBnB is close to the optimal solution. Thus, we define a simple iterative algorithm that at every step during the search chooses the next best candidate and stops after choosing k vertices. We will denote this algorithm as $ItrK$. The algorithm chooses the next best candidate $v \in CL(C)$ according to its contribution to GBC of the group represented by the current node C ($BC^{GM(C)}(v)$). The complexity of the $ItrK$ algorithm is $O(n^3)$ and therefore it is significantly faster than the exponential time heuristic search (for all heuristic functions).

4. Evaluating the algorithms' performance

In order to evaluate the algorithms we implemented them in Python and executed the implementation on a PC P-Duo 3GHz machine with 2GB memory. Our first goal was to compare the accuracy of the greedy algorithms ($TopK$ and $ItrK$) to DFBnB based on four different heuristic functions. We conducted the comparison on random networks (with equal probability to connect every two vertices) and preferential-attachment [7] networks. All algorithms were used to find the most prominent groups of sizes $1, 2, \ldots, 10$ on networks of sizes $10, 20, \ldots, 500$. The networks we considered were sparse, with the number of edges equal to 1.3 times the number of vertices. We have evaluated five random and five preferential-attachment networks for each network size. Every execution of the algorithms was bounded by ten minutes, therefore, larger networks were evaluated for smaller groups only. The experiments have shown similar behavior of the evaluated algorithms for all network sizes. Here we choose to present the results of the evaluations performed on networks of size 100.

Figures 2 and 3 compare the GBC of groups found by $TopK$, $ItrK$, and Heuristic Search algorithms for random and preferential-attachment networks, respectively. Similarly the running time of various GBC maximization strategies is presented in Figures 4 and 5. The results of the evaluation experiments suggest that results of the $ItrK$ algorithm are very close to the optimal solution (see Figures 2 and 3) while its computation time (see Figures 4 and 5) is low. The evaluation results show that heuristic search works faster on scale-free networks (compare the plots in Figures 4 and 5).

We compared the computation time of the $ItrK$ algorithm, $TopK$ and the heuristic search for four heuristic functions. $TopK$ and $ItrK$ are much faster than heuristic search. It is easy to see that as the heuristic function is more accurate (h_1 least accurate, h_4 most accurate), the computation time decreases even though the complexity of

the heuristic function is increasing. Table 4 compares the results and the execution time of various algorithms.

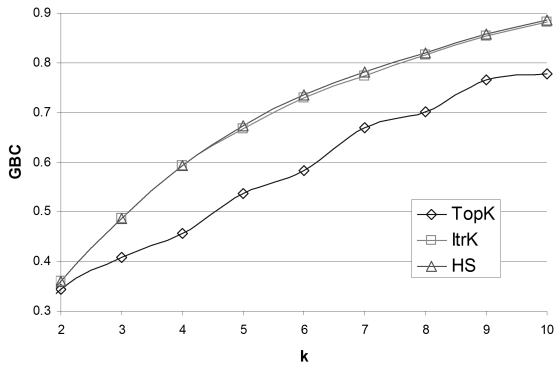

Fig. 2. GBC for random network.

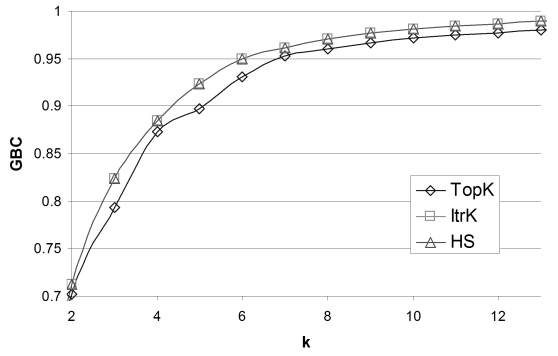

Fig. 3. GBC for scale-free network.

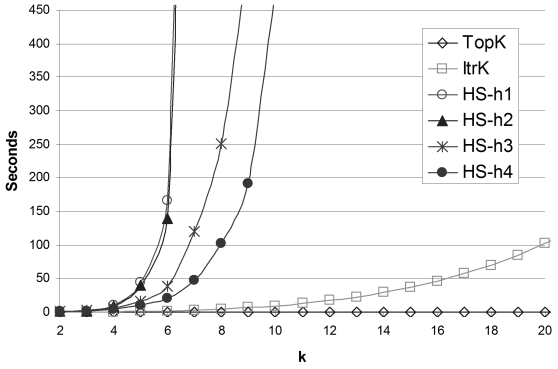

Fig. 4. GBC computation time for random network.

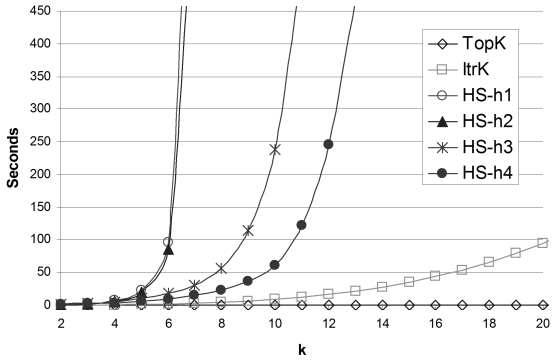

Fig. 5. GBC computation time for scale-free network.

5. Applying the greedy algorithms to optimize the deployment of DNIDS

One way to prevent users from being infected by computer viruses, worms, and Trojan horses is to clean the traffic at the NSP level (edge routers). The NSP traffic can be monitored and cleaned by Distributed Network Intrusion Detection System (DNIDS) that may be deployed on the NSP routers/links [18,20]. It is not realistic to inspect the traffic of all the routers/links of the NSP. Theoretical models suggest protecting the most significant routers/links in order to slow down threat propagation in the entire network [19]. We used the greedy algorithm to locate routers/links that together have the highest influence on communication in the NSP infrastructure. The NSP network included 26 core routers, 70 edge routers, 204 customer routers, and 451 links.

We assumed that five local area networks are connected to each customer router. Then, we analyzed the threat propagation in the NSP infrastructure assuming that a DNIDS is monitoring the traffic of the influential group of routers/links. The analysis was performed using a network simulator that was developed for this purpose and whose main goal is to assist in finding the appropriate size and the location of a DNIDS deployment. We repeated the analysis for various group sizes while measuring the number of infected local area networks. We repeated the simulation for deployment based on $ItrK$, $TopK$, and random deployment. The simulation results (see Figure 6) illustrate the superiority of deployment located by the iterative greedy algorithm ($ItrK$).

Table 1
Statistics of a typical execution of various algorithms when searching for a group of size 6 in a network of size 100.

Network Model	Algorithm	Result (GBC)	Total time (sec.)	Visited Nodes
Preferential-attachment	$TopK$	0.904	0.015	1
Preferential-attachment	$ItrK$	0.941	1.937	6
Preferential-attachment	$HS - h_1$	0.941	96.058	1520
Preferential-attachment	$HS - h_2$	0.941	84.84	1338
Preferential-attachment	$HS - h_3$	0.941	18.608	90
Preferential-attachment	$HS - h_4$	0.941	8.891	42
Random	$TopK$	0.67	0.016	1
Random	$ItrK$	0.721	1.922	6
Random	$HS - h_1$	0.733	166.195	3108
Random	$HS - h_2$	0.733	139.822	2606
Random	$HS - h_3$	0.733	38.967	204
Random	$HS - h_4$	0.733	20.092	104

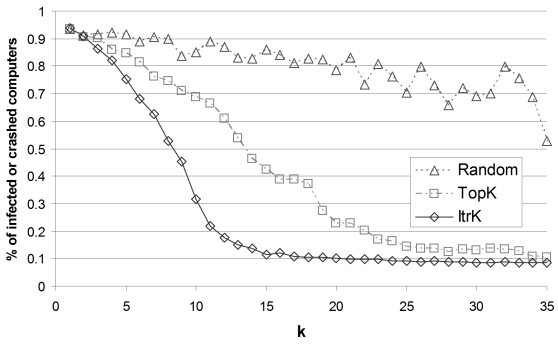

Fig. 6. Infected or crashed computers as a function of deployment size.

6. Conclusions

In this paper we proposed two algorithms for finding the group with the highest GBC. The first algorithm is based on heuristic search and the second is based on a greedy choice. For the heuristic search algorithm we proposed four pruning heuristics and two heuristics for guiding the search. The algorithms were evaluated on random and scale-free networks. Empirical evaluation suggests that results of the $ItrK$ algorithm were negligibly below the optimal results that can be found by exponential heuristic search. In addition, both algorithms performed better on scale-free networks: heuristic search was faster and the greedy algorithm produced more accurate results. The greedy, $ItrK$, algorithm was applied for optimizing deployment of intrusion detection devices on a network that resembles the network service provider infrastructure.

References

[1] S. H. Strogatz. Exploring complex networks. *Nature*, **410**:268–276, March 2001.

[2] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*. Cambridge University Press, 1994.

[3] J. Scott. *Social Network Analysis: A Handbook*. Sage Publications, London, 2000.

[4] P. Bork, L. J. Jensen, C. von Mering, A. K. Ramani, I. Lee, and E. M. Marcotte. Protein interaction networks from yeast to human. *Curr. Opin. Struct. Biol.*, **14**(3):292–299, Jun. 2004.

[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM Comput. Comm. Rev.*, **29**(4):251–262, 1999.

[6] S.-H. Yook, H. Jeong, and A.-L. Barabasi. Modeling the internet's large-scale topology. *PNAS*, **99**(21):13382–13386, Oct. 2002.

[7] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, **286**:509–512, 1999.

[8] A.-L. Barabasi, R. Albert, and H. Jeong. Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A*, **281**:69–77, 2000.

[9] B. Bollobas and O. Riordan. Robustness and vulnerability of scale-free random graphs. *Internet Mathematics*, **1**(1):1–35, 2003.

[10] M. Barthélemy. Betweenness centrality in large complex networks. *The European Physical Journal B – Condensed Matter*, **38**(2):163–168, March 2004.

[11] C. Ballester, A. CalvA -Armengol, and Y. Zenou. Who's who in networks. wanted: The key player. *Econometrica*, **74**(5):1403–1417, Sep. 2006.

[12] Stephen P. Borgatti. Identifying sets of key players in a social network. *Comput. Math. Organ. Theory*, **12**(1):21–34, 2006.

[13] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, **40**(1):35–41, 1977.

[14] J. M. Anthonisse. The rush in a directed graph. Technical Report BN 9/71, Stichting Mathematisch Centrum, Amsterdam, 1971.

[15] P. Holme. Congestion and centrality in traffic flow on complex networks. *Advances in Complex Systems*, **6**(2):163–176, 2003.

[16] G. Yan, T. Zhou, B. Hu, Z.-Q. Fu, and B.-H. Wang. Efficient routing on complex networks. *Phys. Rev. E*, **73**:046108, 2006.

[17] M. G. Everett and S. P. Borgatti. The centrality of groups and classes. *Mathematical Sociology*, **23**(3):181–201, 1999.

[18] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the domino overlay system. in: *NDSS*, 2004.

[19] D. H. Zanette and M. Kuperman. Effects of immunization in small-world epidemics. *Physica A*, **309**:445–452, Jun. 2002.

[20] K. Park. Scalable protection against ddos and worm attacks. DARPA ATO FTN project AFRL contract F30602-01-2-0530, Purdue University, West Lafayette, 2004.

[21] L. C. Freeman, S. P. Borgatti, and D. R. White. Centrality in valued graphs: A measure of betweenness based on network flow. *Social Networks*, **13**(2):141–154, Jun. 1991.

[22] M. E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, **27**(1):39–54, Jan. 2005.

[23] R. G. Downey and M. R. Fellows. Parametrized computational feasibility. *Feasible Mathematics II*, pages 219–244. Boston: Birkhäuser, 1995.

[24] R. Puzis, Y. Elovici, and S. Dolev. Fast algorithm for successive group betweenness centrality computation. Technical Report #03-07, Dept. of Computer Science, Ben Gurion University of the Negev, Jan. 2007.

[25] R. E. Korf and W. Zhang. Performance of linear-space search algorithms. *Artificial Intelligence*, **79**(2):241–292, 1995.

[26] M. E. J. Newman. Scientific collaboration networks ii. shortest paths, weighted networks, and centrality. *Phys. Rev. E*, **64**(016132), 2001.

[27] M. E. J. Newman. Erratum: Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. *Phys. Rev. E*, **73**:039906(E), 2006.

[28] U. Brandes. A faster algorithm for betweenness centrality. *Mathematical Sociology*, **25**(2):163–177, 2001.

Appendix

A. Glossary

BC – Shortest Path Betweenness Centrality

GBC – Shortest Path Group Betweenness Centrality

KPP-Com – The problem of finding group of vertices of given size that has the maximal potential to control the traffic flow in communication networks that are based on shortest path routing

TopK – naïve algorithm for GBC maximization that takes k vertices with maximal BC

ItrK – greedy algorithm for GBC maximization that at every step chooses the vertex with maximal contribution to the group of already chosen vertices

HS – Heuristic Search

DFBnB – Depth First Branch and Bound

NSP – Network Service Provider

DNIDS – Distributed Network Intrusion Detection System

G – unweighted and undirected graph

vertex, *edge* – part of the input graph

V – the set of all vertices in G

E – the set of all edges in G

n – the number of vertices

m – the number of edges

$a, b, c, s, t, u, v, x, y$ – vertices

S, S' – group of vertices

k – size of the group of vertices

$d(x, y)$ – the distance between vertices x and y

$\sigma_{s,t}$ – the total number of different shortest paths that connect s and t

$\sigma_{s,t}(v)$ – the number of shortest paths from s to t that pass through v

$\ddot{\sigma}_{s,t}(S)$ – the number of shortest paths between s and t that traverse at least one member of the group S

R, C, C' – nodes in the search tree where R is the root

node, *transition* – part of the search tree

$GM(C)$ – group of vertices maintained by C

$CL(C)$ – candidate list maintained by C

v^{best} – the first vertex in the ordered list $CL(C)$

C^- – left child of C obtained by removing v^{best} from $CL(C)$

C^+ – right child of C obtained by removing v^{best} from $CL(C)$ and adding it to $GM(C)$

$g(C)$ – utility of C

$h(C)$ – heuristic function

$f(C) - g(C) + h(C)$

$BC^S(v)$ – the contribution of the vertex v to the GBC of S