Optimizing Network Attacks by Artificial Bee Colony

Manuel Lozano[a], Carlos García-Martínez[b], Francisco J. Rodríguez[c], Humberto M. Trujillo[d]

[a] *Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain*
[b] *Computing and Numerical Analysis Department, University of Córdoba, Córdoba, Spain*
[c] *Department of Computer Science, University of Extremadura, Mérida, Spain*
[d] *Department of Methodology of Behavioral Sciences, University of Granada, Granada, Spain*

Abstract

Over the past few years, the task of conceiving effective attacks to complex networks has arisen as an optimization problem. Attacks are modelled as the process of removing a number k of vertices, from the graph that represents the network, and the goal is to maximise or minimise the value of a predefined metric over the graph. In this work, we present an optimization problem that concerns the selection of nodes to be removed to minimise the maximum *betweenness centrality* value of the residual graph. This metric evaluates the participation of the nodes in the communications through the shortest paths of the network.

To address the problem we propose an artificial bee colony algorithm, which is a swarm intelligence approach inspired in the foraging behaviour of honeybees. In this framework, bees produce new candidate solutions for the problem by exploring the vicinity of previous ones, called food sources. The proposed method exploits useful problem knowledge in this neighbourhood exploration by considering the partial destruction and heuristic reconstruction of selected solutions. The performance of the method, with respect to other models from the literature that can be adapted to face this problem, such as sequential centrality-based attacks, module-based attacks, a genetic algorithm, a simulated annealing approach, and a variable neighbourhood search, is empirically shown.

☆E-mail addresses: lozano@decsai.ugr.es (M. Lozano), cgarcia@uco.es (C. García-Martínez), fjrodriguez@unex.es (F.J. Rodríguez), humberto@ugr.es (H.M. Trujillo).

Keywords:
Artificial bee colony, betweenness centrality, network attacks, combinatorial optimization.

1. Introduction

Network theory and its applications arise in a variety of scientific fields (such as physics, engineering, sociology, psychology, criminology, epidemiology, biology, and many others), due to the inherent network's ability to logically represent important relationships (edges) between structural elements (nodes) of complex systems. The optimization of procedures for efficiently breaking complex networks is attracting much attention from a practical point of view (Deng et al., 2016). For instance, these methods may help intelligence agencies to cripple jihadist terrorist networks (Arulselvan et al., 2009), or they may be useful to design vaccination strategies to restrain the spread of pandemic diseases (Ventresca and Aleman, 2014; García-Martínez et al., 2015). In terms of network attacks, the challenge is to find a subset of nodes or edges whose removal would cause great damage to the network. An efficient and doable way of attacking a graph consists in the deletion of vertices according to their importance in the structural functioning of the network, the so called *centrality-based attacks* (Crucitti et al., 2004; Iyer et al., 2013).

Centrality indices are fundamental metrics for network analysis. They refer to how important a vertex is within a network. Some of them, e.g., degree centrality, reflect local properties of the structural elements, while others, like *betweenness centrality* (BC), give information about the global relevance on the network structure, since they are based on shortest path computations. Specifically, BC quantifies the importance of a vertex based on its occurrence in shortest paths between all the pairs of vertices of the graph (Anthonisse, 1971; Freeman, 1977). This measure is useful to identify critical nodes with respect to information transmission between each pair of nodes in the network. Nodes that act as bridges between groups of nodes will have a high BC because communication between pairs of nodes in different groups must go through the bridge. Precisely, BC has been applied in a previous research work to identify key players in social networks (Krebs, 2002). In general, key players are those nodes in the network that control the information flow, they are the most popular, and may have some sort of influence on other nodes.

In spite of the relevance of the BC metric in network analysis (Borgatti, 2006; Hewett, 2011), few works use it to design effective network attacks (Gunasekara et al., 2015). Instead, most of the literature focuses on maximally fragmenting the network, based on the concept of *critical nodes*, into several connected components (Arulselvan et al., 2009; Ventresca and Aleman, 2015a; Veremyev et al., 2015). These problems are named *critical node detection problems* (CNP). This scientific lack may be due to the fact that computing the BC metric is notoriously expensive (the best known algorithm, presented by Brandes (2001), runs in $O(nm)$).

In this work, we formulate the *Min-Max BC optimization problem* as a CNP case where the objective is to minimize the maximum BC value of the graph after node removal. The aim is to degrade the network structure in such a way that the new key players in the residual graph have the minimum possible influence on the information flow. In other words, the goal is to avoid the apparition of strong *BC leaders* in the residual network. Two determinant facts motivated us to tackle this difficult and appealing problem:

1. The recent development of the *artificial bee colony* (ABC) algorithm (Karaboga and Basturk, 2007), which is a novel metaheuristic that has been successfully used to solve a wide spectrum of NP-hard optimization problems (Bansal et al., 2013; Bolaji et al., 2013; Karaboga et al., 2012a; Rodriguez et al., 2013b). ABC has demonstrated a superior performance against other classical methods, in terms of both quality of the solutions and processing time (Karaboga and Basturk, 2008).
2. The apparition of *BC update procedures* that are able to recompute the BC values of a graph in response to modifications of its structure (edge/node insertions and edge/node deletions) faster than calculating them from scratch (Goel et al., 2015; Kas et al., 2014; Lee et al., 2012, 2016).

As a result, we have developed a competent ABC algorithm aimed at obtaining high quality solutions for the Min-Max BC problem. Its key component is an effective neighbourhood operator that benefits from the use of a BC update algorithm to accelerate the ABC convergence towards promising solutions.

The rest of this paper is organized as follows. Section 2 provides an overview of the literature on strategies to attack complex networks. Section 3 presents the Min-Max BC problem, a new CNP case based on BC that, to the

best of our knowledge, has not yet been defined in the literature. Section 4 details our ABC algorithm for the Min-Max BC problem. Section 5 reports the results of the experiments that analyse our ABC algorithm in different contexts. Section 6 presents the conclusions of the work.

2. The Literature on Network Attacks

Most the the research on network attacks is based on the idea of *critical nodes*, which allows the characterization of vulnerability and robustness of a given network with respect to node removals, caused by adversarial offence, random failures, or natural disasters. This class of problems, CNP, has extensively been studied in the last decade (Walteros and Pardalos, 2012), and different cases have been analysed according to the particular interests. Arulselvan et al. (2009) and Pullan (2015) focused on the minimization of the total number of pairs of connected vertices. Shen et al. (2012) aimed at maximising the number of connected components and minimising the size of the largest one. Ortiz-Arroyo (2010) worked on the maximisation of the graph information entropy. Veremyev et al. (2015) analysed the minimisation of a distance-based connectivity measure such as graph efficiency, *Harary* index, characteristic path length, and residual closeness. Gunasekara et al. (2015) also addressed multi-objective CNP cases that emphasized the maximization of the average eigenvector centrality and the distance between critical nodes.

However, most of the attention in the CNP literature has been focused on the particular case defined by Arulselvan et al. (2009), where the optimal attack maximally fragments the network and simultaneously minimizes the variance among the number of vertices in the resulting connected components. That is, the residual network contains a relatively large set of connected components, each with a similar number of vertices (Ventresca and Aleman, 2015a). This CNP instance will be referenced as CNP-A. Arulselvan et al. (2009) presented an integer linear programming (ILP) model and a heuristic approach based on a greedy algorithm coupled with a local search-phase for the CNP-A. Later, the NP-complete nature of this problem (Arulselvan et al., 2009) promoted the application of metaheuristics to obtain near optimal solutions within reasonable computational times: Ventresca (2012) proposed a population-based incremental learning and a simulated annealing model, Pullan (2015) designed a multi-start greedy algorithm, and Aringhieri et al. (2015) presented a variable neighbourhood search approach.

Centrality-based attacks (Crucitti et al., 2004; Iyer et al., 2013) are another alternative to address CNPs, which target the vertices to be removed according to a given centrality measure and one of the following strategies:

- In *simultaneous* targeted attacks, the centrality measure is calculated for all the vertices in the network, and those k with the highest values are removed at once.
- In the *sequential* targeted attacks, only the vertex with the highest centrality measure is removed at a time, and the process is repeated k times. Given that each removal probably modifies the centrality values of the remaining vertices, the metric is computed once for the initial graph and again after every removal for the remaining vertices.

Iyer et al. (2013) investigated the effect of centrality-based attacks with different removal schemes and centrality measures, such as degree, BC (defined in Section 3), closeness, and eigenvector, on a wide range of networks. They found that the sequential removal of the vertex with highest BC was the most effective method to degrade the network structure. This conclusion was also supported by Ventresca and Aleman (2015b), who analysed the effects according to six centrality metrics. The sequential BC-based attack (SBA), whose pseudocode is depicted in Fig. 1, will be considered as baseline for our proposed approach.

Input: $G(V, E)$, k
Output: S
1 $S \leftarrow \emptyset$;
2 Compute $bc(G, v)$ for every $v \in V$;
3 **while** $|S| \neq k$ **do**
4 $\quad v_{max} \leftarrow \underset{v \in V \setminus S}{\operatorname{argmax}} bc(G^S, v)$;
5 $\quad S \leftarrow S \cup \{v_{max}\}$;
6 $\quad$ Compute $bc(G^S, v)$ for every $v \in V \setminus S$;
7 **end**

Figure 1: Pseudocode of SBA

3. The Min-Max BC Problem

Given an undirected and unweighted graph $G(V, E)$, where V is the set of nodes (vertices; and $|V| = n$) and E is the set of edges ($|E| = m$), the BC measure of a vertex $v \in V$ quantifies its relevance as the fraction of shortest paths that contain v (Anthonisse, 1971; Freeman, 1977):

$$bc(G, v) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

where σ_{st} is the number of shortest paths from vertex s to vertex t and $\sigma_{st}(v)$ is the number of those that contain v.

We define the Min-Max BC optimization problem as the task of determining the k vertices to be removed to minimise the maximal BC value of any vertex of the residual graph. Although the Min-Max BC problem is closely related to BC-based attacks and the SBA method (Section 2), their goals were oriented towards the network dismantling and it is explicitly presented here as an optimisation problem. Formally, let $S \subseteq V$ be a subset of k nodes to be removed from G and denote the corresponding graph by $G^S = G(V \setminus S, \{(i, j) \in E | i, j \in V \setminus S\})$, that is, the subgraph of G induced by the set of remaining nodes $V \setminus S$. The Min-Max BC problem consists of minimizing the maximal BC value of any vertex in G^S over the set of all possible subsets of V with cardinality k:

$$\textbf{Min-Max BC(G(V,E))} \equiv \min_{S \subseteq V, |S|=k} BC(G^S),$$

$$\text{with } BC(G^S) = \max_{r \in V \setminus S} bc(G^S, r).$$

Notice that, regarding the most studied CNP, we should not expect a method for the CNP-A to perform well on the Min-Max BC case. We illustrate this fact with the example shown in Fig. 2, which corresponds to a graph generated by the Erdös-Rényi model with $n = 30$ (Fig. 2.a). For this graph, we solved the ILP model of Arulselvan et al. (2009) with IBM ILOG CPLEX V12.5.1 to obtain an optimal solution for the CNP-A with $k = 9$, and then, we computed the BC values of the nodes in the residual graph (Fig. 2.b). We also run the ABC algorithm proposed in this paper on the same graph, which addresses the Min-Max BC problem (the corresponding residual graph is shown in Figure 2.c).

We may see that ABC reduces the maximal BC value more than the

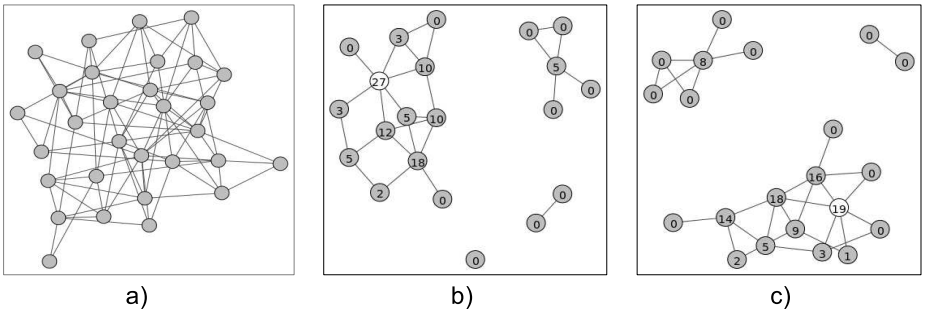

Figure 2: Erdös-Rényi graph (a) and residual graphs by CPLEX (b) and ABC (c)

optimal solution for the ILP model (the maximal BC for ABC is 19 and it is 27 for the ILP model). Since the ILP model pursued achieving maximal network disconnectivity (in fact, it was able to break the graph into more components than ABC), the evaluation of the reached solution from the point of view of the Min-Max BC problem shows acceptable quality. However, it is not as good as the one provided by ABC. Thus, this example justifies the need for optimizers specifically designed to face the requirements of the Min-Max BC problem.

4. An Artificial Bee Colony Algorithm for the Min-Max BC Problem

The ABC model (Karaboga and Basturk, 2007) maintains a set of food sources, whose positions represent candidate solutions for the addressed problem, and a set of honeybees, which are computational agents that operate on food sources to find new candidate solutions. The nectar amount of a food source corresponds to the quality of its associated solution. Initially, food sources are randomly sampled from the search space and assigned to an employed bee. Then, ABC repeatedly iterates through the following stages until a stopping condition is met:

- *Employed bees phase*: Employed bees explore the neighbourhood of their respective food sources seeking better ones. To do this, they apply a neighbourhood operator that generates a new candidate solution from the previous one. The solution is evaluated and, if it is better than the

current food source, the bee abandons its current position to move to the new food source; otherwise, the bee discards the new candidate solution. After this exploitation process, employed bees return to the hive and share their knowledge, the position of food sources and their nectar amounts, with onlookers bees.

- *Onlooker bees phase*: Onlookers bees wait for employed bees to share their information. Each one probabilistically chooses a food source proportionally according to their nectar amounts and, as employed bees did, try to discover better food sources in the corresponding vicinity (using the neighbourhood operator). If the new food source is more attractive than the previous one, the information of the food source is updated; otherwise, the new food source is discarded.

- *Scout bees phase*: If a food source is explored for a certain number of iterations without any improvement, the food source is abandoned and the associated employed bee becomes a scout. Then, the scout searches for a new food source in a more exploratory way, traditionally, by randomly sampling a new candidate solution from the search space. Then, the scout bee becomes an employed bee again, associated with that food source.

The ABC algorithm was originally designed for continuous optimization problems (Karaboga and Basturk, 2007) and much work has been devoted to the development of improved models for these problems (Akay and Karaboga, 2012; Banitalebi et al., 2015; Maeda and Tsuda, 2015). However, ABC has also been successfully applied to many different fields, such as symbolic regression (Karaboga et al., 2012b), clustering (Karaboga and Ozturk, 2011), binary optimization (Kashan et al., 2012), constrained optimization problems (Karaboga and Akay, 2011), multi-objective optimization problems (Akbari et al., 2012), the maximally diverse grouping problem (Rodriguez et al., 2013b), and the cyclic antibandwidth problem (Lozano et al., 2013). Moreover, ABC algorithms are involved in the solution of many real-world problems such as the composition of medical crews (Delgado-Osuna et al., 2016) and the design of analog filters (Bose et al., 2014) and circular antenna arrays (Bose et al., 2012), among others. The interested reader may refer to Karaboga et al. (2012a) for a comprehensive review of its applications.

Recently, some modifications in the algorithmic components of ABC were presented (Das et al., 2013; Gao et al., 2015; Biswas et al., 2013), which have

proven to improve the performance of the classical version most of the times. For example, Das et al. (2013) propose a new ABC model that focuses on acquiring a better balance between diversification and intensification to successfully deal with numerical optimization problems. The proposed ABC incorporates: (1) a learning mechanism to develop potentially new food sources that are formed by combining the components of the solution vector and those of the best-known food sources, and (2) a new mechanism to modify onlooker bees position that uses a proximity based perturbation scheme and a probabilistic weighted selection strategy.

In the following sections, we describe how the ABC framework is adapted for the Min-Max BC problem. Section 4.1 defines the bees' neighbourhood operator. Section 4.2 depicts the general overview of the proposal. Finally, Section 4.3 gives an account of how move operations are implemented efficiently and highlights the computational complexity issues.

4.1. Destructive-Constructive Neighbourhood Operator

A solution for the Min-Max BC problem is a subset of vertices of an input graph with cardinality k. To explore the vicinity of a solution, we propose a neighbourhood operator (see the detailed pseudocode in Fig. 3) that is to some extent inspired by iterated greedy algorithms (Lozano et al., 2011; Rodriguez et al., 2013a). Specifically, the operator removes a number of random components from the solution and then operates like SBA (Fig. 1) to construct a new complete solution. Notice that destroying S by removing some of the selected nodes, is actually reconstructing the initial graph with these nodes and respective arcs.

Firstly, the neighbourhood operator randomly chooses ω vertices from S (ω is a user parameter dubbed *destruction size*; Line 1 of Fig. 3). Then, it initializes the *neighbour* S' with those vertices in S that have not been selected (Line 2). Finally, it adds nodes to S' using the greedy heuristic (sequential BC-based attack; SBA) one at a time, while its cardinality is less than k (Lines 3-8). Regarding its complexity, the most expensive operation is the BC computation (Lines 3 and 7), which is repeated ω times per vicinity exploration and whose complexity is commented in Section 4.3.

The procedure for determining neighbours in the ABC framework is problem-specific (Sundar and Singh, 2010). Karaboga and Basturk (2007) designed the original ABC algorithm specifically for real parameter optimization. In this case, the neighbour of a particular solution is generated by perturbing one randomly chosen real parameter, while keeping the others

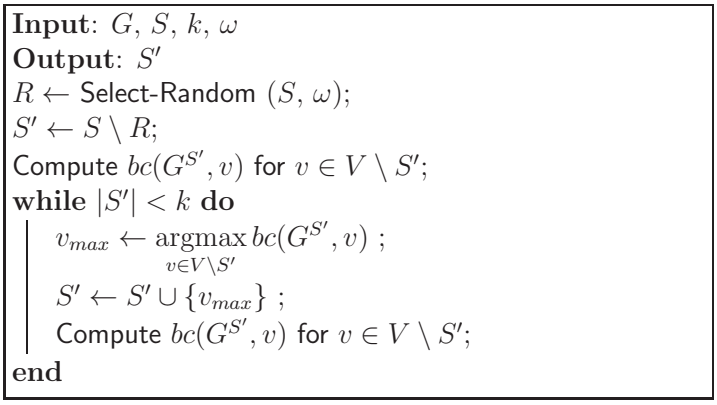

Figure 3: Pseudocode of the neighbourhood operator

unchanged. However, ABC practitioners have devised solution representations and neighbourhood operators that fit well to each tackled optimization problem (Karaboga et al., 2012a). Among the different alternatives, specialised heuristic *destructive-constructive procedures* have been successfully applied in the ABC area (Pan et al., 2011; Rodriguez et al., 2013b; Delgado-Osuna et al., 2016; Tasgetiren et al., 2011), which offer the advantage of incorporating problem knowledge in the creation of neighbours. In fact, the incorporation of problem knowledge is essential to get superior results according to the no free lunch theorems for optimisation (García-Martínez et al., 2012), and the use of heuristic constructive procedures as components of metaheuristics is sometimes necessary to produce practical optimizers for hard optimization problems (Lozano et al., 2016). Ours is another example of a destructive-constructive procedure that exploits some problem knowledge when S' is being re-constructed (Lines 3-8 of Fig. 3). It considers the BC values of the vertices of the current residual graph to select those that will subsequently be removed. Given that these BC values depend on the solution components that remain in the partial solution S', the election of the vertices to be removed may be different, but still heuristically chosen, from those in the original solution S.

Additionally, parameter ω may be tuned to adjust the balance between intensification and diversification promoted by the operator (Blum and Roli, 2003). High values make our operator to diversify the search process by generating solutions from different regions of the search space, i.e., solutions

with different characteristics; whereas low values make S and S' share many components, which promotes intensification.

4.2. General Scheme of the Proposed ABC Algorithm

An outline of our proposal is depicted in Figure 4. The algorithm starts applying SBA (Section 3) to generate an initial feasible solution (Line 1). Then, it generates a set of solutions, the initial food sources, by employing the neighbourhood operator with a diversification setting (function Generate-Neighbouring with a high value for ω, ω_{div}) on the best-so-far solution (Lines 2-5). Then, our ABC iterates through the following steps until a termination criterion is fulfilled:

- *Employed bees phase*: Each employed bee explores the vicinity of its associated food source via the neighbourhood operator with an intensification setting, i.e., with a low value for ω (ω_{int}; Line 8). The new solution is accepted, provided that it is better than the previous food source (Lines 9-11).

- *Onlooker bees phase*: Each onlooker bee chooses a food source by binary tournament (Line 14). This selection mechanism is employed in other ABC applications because of its simplicity (Rodriguez et al., 2013b; Tasgetiren et al., 2011), instead of the original roulette wheel method. Then, onlookers apply the same steps as employed bees, that is, vicinity exploration and selection of the new candidate solution whenever it is better (Lines 15-18).

- *Scout bees phase*: Food sources that have not been improved for a number of *Limit* consecutive iterations are abandoned and replaced with new solutions, which are generated by the neighbourhood operator with a diversification setting on the best-so-far solution (Lines 20-23).

To enhance the exploitation capability of ABC algorithms, it is common to invoke a local improvement procedure on some of the generated solutions (Rodriguez et al., 2013b; Delgado-Osuna et al., 2016; Tasgetiren et al., 2011). Our method applies a local search method after the ABC completion, following the scheme used by Sundar and Singh (2010), as an attempt to further improve the quality of the best found solution (function Local-Search in Line 28). This procedure is based on the *1-interchange move*, which, given a solution S, exchanges a selected vertex $v \in S$ and a not chosen one $v' \in V \setminus S$.

```
Input: G, k, t_max, Limit, NP, ω_div, ω_int
Output: S_b
// Generation of the greedy solution
S_1 ← SBA (G, k);
// Initialization phase
for i = 2 to NP do
    S_b ← Best-So-Far-Solution ();
    S_i ← Generate-Neighbouring (G, S_b, k, ω_div);
end
while computation time t_max not reached do
    // Employed bees phase
    for i = 1 to NP do
        E ← Generate-Neighbouring (G, S_i, k, ω_int);
        if BC(G^E) < BC(G^{S_i}) then
            S_i ← E;
        end
    end
    // Onlooker bees phase
    for i=1 to NP do
        S_j ← Binary-Tournament (S_1, ..., S_NP);
        O ← Generate-Neighbouring (G, S_j, k, ω_int);
        if BC(G^O) < BC(G^{S_j}) then
            S_j ← O;
        end
    end
    // Scout bees phase
    for i=1 to NP do
        if S_i has not change for limit iterations then
            S_b ← Best-So-Far-Solution ();
            S_i ← Generate-Neighbouring(G, S_b, k, ω_div);
        end
    end
end
S_abc ← Best-So-Far-Solution ();
// Local improvement phase
S_b ← Local-Search (S_abc);
```

Figure 4: Pseudocode of the proposed ABC

The result is thus a solution $S' = S \cup \{v'\} \setminus \{v\}$. A first-improvement strategy is considered, i.e., (v, v') pairs are sequentially examined until the process finds a solution better than the current one, when it moves to the new solution. Then the process is repeated until no further improvement is possible, by means of this operator, or when a time limit is reached.

In summary, our ABC algorithm requires setting the value of the following parameters: t_{max} denotes the computation time limit; NP is the number of food sources the algorithm maintains, which is equal to the number of employed and onlooker bees; *Limit* is the number of iterations without improvement before abandoning a food source; and ω_{div} and ω_{int} determine the diversification and intensification settings of the neighbourhood operator, respectively.

4.3. Efficient Implementation and Complexity

Two key procedures of our ABC, the neighbourhood operator (Lines 4, 8, 15, and 23 of Fig. 4) and SBA (Line 1), comprise the following actions over a graph:

1. To calculate BC for all the vertices in the graph (Line 2 in Fig. 1 and 3 in Fig. 3).
2. To find the vertex with the highest BC and remove it from the graph (Line 4 in Fig. 1 and 5 in Fig. 3).
3. To re-calculate BC for all the remaining vertices (Line 6 in Fig. 1 and 7 in Fig. 3).
4. To repeat steps 2 and 3 until k vertices are removed from the graph (ω repetitions in the case of the neighbourhood operator).

Since BC relies heavily on the computation of all the shortest paths, its computation is notoriously expensive (the best known algorithm (Brandes, 2001) runs in $O(nm)$ time for unweighted graphs). This way, the above process becomes infeasible from a practical point of view, even for small graphs. Therefore, it is necessary to incorporate algorithms that provide a faster BC computing.

Several methods have been proposed to update the BC values after edge or node insertions or deletions, rather than computing them from scratch, by exploiting auxiliary data structures. QUBE (Lee et al., 2012) relies on the decomposition of the graph into disjoint minimum union cycles. The algorithm uses this decomposition to identify vertices whose centrality may potentially

change. The authors showed that QUBE could achieve updates several times faster in comparison with Brandes' exact algorithm. Unfortunately, QUBE is limited to the insertion and deletion of non-bridge edges (a *bridge* edge is one whose removal increases the number of connected components).

Based on QUBE, Goel et al. (2015) proposed an algorithm that managed two sets of vertices: one for which betweenness scores can be updated and another for which they need to be re-computed. This algorithm speeded up the calculation of BC for real graphs from 7 to 412 times in comparison to Brandes' algorithm. Kas et al. (2014) presented a method that kept auxiliary data structures to represent a directed acyclic subgraph for each vertex of the graph, containing all the edges that belong to, at least, one shortest path from the vertex. Then, shortest paths are updated by running a Dijkstra-like procedure. The proposal empirically achieved a reduction in computation time orders of thousands with regards to Brandes' algorithm. However, this algorithm had a space complexity of $O(n(n+m))$, which becomes prohibitive for large graphs.

Recently, Lee et al. (2016) presented another update algorithm that substantially reduced the number of shortest paths which should be re-computed when a graph is changed. Interestingly, it does not require any auxiliary data structure for updating BC. When an edge is updated, the algorithm identifies a subgraph, called re-calculation subgraph, containing those vertices and edges whose BC will change, and updates the BC of the edges in the re-calculation subgraph without computing all-pairs shortest paths. When a vertex is updated, the algorithm computes the amounts of increases or decreases in BC of the edges in a graph by performing a single-source shortest path computation from the updated vertex. The experimental results showed that the proposed algorithm outperformed QUBE and Brandes' algorithm for all update operations. Specifically, for vertex removals, the algorithm was about 5 times faster than QUBE, and much faster than Brandes' method for vertex insertions.

Given the existence of these BC update methods, we invoke them in the neighbourhood operator and in the SBA algorithm as follows. First, the graph associated with the given solution has already got its BC values computed. This required the invocation of Brandes' algorithm just once per ABC run, in Line 2 of Fig. 1, because the rest of the operations apply on previously evaluated candidate solutions. Then in the case of SBA algorithm, BC values are updated after each vertex removal (SBA removed the vertex with the highest BC value per iteration). And in the case of the neighbourhood

operator, BC values are firstly updated after each vertex insertion, because it starts removing solution components which represents nodes reinserted into the graph; and then it applies SBA with the BC update method.

We should remark that the use of the algorithm of Lee et al. (2016) in the neighbourhood operator is especially important, because it is invoked a large number of times during the ABC search. Particularly, given that BC values have to be updated $2 \cdot \omega$ times per neighbourhood exploration and ω_{int} is considered $2 \cdot NP$ times per ABC iteration, then the complexity of an ABC iteration is usually bounded by $4 \cdot NP \cdot \omega_{int}$ times the complexity of the BC update method of Lee et al. (2016). This makes the global complexity of an ABC iteration be orders of $O(NP \cdot \omega_{int} \cdot nm)$ in the worst case (Brandes' method complexity is considered here). The application of the neighbourhood operator with a diversification setting (ω_{div}) is discarded because it happens much less frequently, about $NP/Limit$ times per ABC iteration in average. Nevertheless, it would insert a constant factor, finally discarded in big O notation.

Finally, the evaluation of the solutions produced by the *1-interchange move* also benefits from this faster BC update methods. This move exchanges a vertex that belongs to the solution with another that does not. Then, we apply the algorithm of Lee et al. (2016) to efficiently evaluate such moves, without recomputing the objective function from scratch. Specifically, this interchange is implemented as a node deletion and then a node insertion, so the update method is invoked after each of these operations. Given that the size of the neighbourhood induced by the 1-interchange move is $k \cdot (n - k)$, the final complexity of an iteration is $O(k \cdot (n - k) \cdot nm)$.

5. Computational Experiments

This section describes the computational experiments that we conducted to assess the performance of the ABC algorithm for the Min-Max BC problem. Firstly, we detail the general experimental setup (Section 5.1), then, we analyse the results. Our aim is: (1) to analyse the influence of the main parameters associated with ABC (Section 5.2); and (2) to compare the results of ABC with other approaches to attack networks (Section 5.3).

5.1. Experimental Setup

The code of all the algorithms has been implemented in C (graph operations were performed using the open source *NetworKit* framework devel-

oped by Staudt et al. (2014)) and the source code has been compiled with gcc 4.6. The experiments were conducted on a computer with a 3.2 GHz Intel® Core™ i7 processor with 12 GB of RAM running Fedora™ Linux V15. Six well-known graph models were taken into account to generate the networks for the experiments:

- *Erdös-Rényi* model (ER) (Erdös and Rényi, 1959). It produces random graphs with a fixed connection probability p_c, which determines whether or not to include the edge e connecting each pair of vertices v, $v' \in V$. Although the ER model often produces graphs that lack common properties observed in real-world networks, it has already been considered in other studies about complex networks (Ventresca, 2012).

- *Clustered random graph* model (CR) (Newman, 2009). It is a simple variation of the Erdös-Rényi model, useful for generating graphs that have distinctive dense areas with sparse connections between them, i.e., *communities*. Nodes are equally distributed over n_s subsets. Then, nodes from the same subset are connected with probability p_{in} and nodes from different subsets with a smaller probability, p_{out}.

- *Random geometric graph* model (RG). A pre-determined number of points are uniformly sampled from the $[0, 1] \times [0, 1]$ space. Then, edges between nodes whose Euclidean distance is inferior or equal to a given radius r, are included in the graph.

- *Barabási-Albert* model (BA) (Barabási and Albert, 1999). It depicts scale-free networks using a preferential attachment mechanism. At each iteration, the process generates a new node v that is connected to m existing nodes in such a way that the more connections a node has, the more likely it is to be selected. That is, nodes with higher degrees have a stronger ability to attract new nodes. This *rich get richer* behaviour results in networks with a power-law degree distribution where a relatively small number of vertices act as hubs with high connectivity. The scale-free property is one of the common properties of real-world complex networks, which can be observed in the Internet, social networks, and airline networks.

- *Watts-Strogatz* model (WS) (Watts and Strogatz, 1998). It starts with a ring of n vertices and connects each vertex in the ring to all of its m_{nn}

nearest neighbours. Then, each edge is considered with probability p_r to rewire one of its endpoints to a randomly chosen node. This model produces networks that exhibit the *small-world property*, inducing a low average geodesic path length between any two vertices and high clustering coefficients. This property was observed in many real-world networks (Ventresca, 2012).

- *Forest fire* model (FF) (Leskovec and Faloutsos, 2007). Like the BA model, this is also a preferential attachment approach. However, this model reproduces heavy tailed degree distributions where the network diameter decreases over time. The FF model seems to be a mix of the ER and BA models (Ventresca, 2012). It was proposed in order to model some real networks such as autonomous systems, patent citations, and affiliation graphs, where they all have the shrinking diameter and densification power law properties.

We created 12 instance sets, each one with 5 artificial graphs of different sizes ($n = \{100, 250, 500, 750, 1000\}$), using ER ($p_c = \{0.05, 0.2\}$), CR (one set with $p_{in} = 0.3$, $p_{out} = 0.1$, and $n_s = 0.3n$ and another with $p_{in} = 0.2$, $p_{out} = 0.05$, and $n_s = 0.3n$), RG ($r = \{0.15, 0.3\}$), BA ($m = \{4, 5\}$), WS ($m_{nn} = \{4, 5\}$ and $p_r = 0.1$), and FF (one set with forward probability of 0.35 and backward probability of 0.32 and another with forward probability of 0.45 and backward probability of 0.35). ER, BA, RG, and WS graphs were generated with *NetworkX* (Hagberg et al., 2008), CR graphs with *NetworKit*, and FF graphs with the *igraph* software package (Csárdi and Nepusz, 2006). To distinguish these graph sets, they will be referred to by their class name and their corresponding parameter values.

5.2. Study of ABC with Different Parameters

In this section, we present a preliminary experiment that was conducted to set the values of the key parameters of ABC: *Limit* and ω_{int} (destruction size of the intensifying neighbourhood operator). We investigated the effects of varying the ω_{int} parameter, rather those of ω_{div}, because ABC invokes the neighbourhood operator with this setting much more frequently. We have built 16 ABC configurations with $Limit = \{0.05n, 0.1n, 0.25n, 0.5n\}$ and $\omega_{int} = \{0.1k, 0.25k, 0.5k, 0.75k\}$ (note that $|S| = k$, where S is the solution to be altered by the operator). Only a few possible combinations are explored, so the performance might be further improved by examining

more combinations/values. The value for ω_{div} is set to $0.75k$ and the number of food sources (NP) to 20, which is widely used in the literature. With the purpose of fine-tuning our proposal and avoiding over-fitting, we have employed only 25 instances of the sets detailed in Section 5.1 (ER-0.2, RG-0.15, BA-5, WS-5, and FF-0.35/0.32) and restricted k to $0.3n$. All the ABC variants were stopped using a time limit of $3.6n$ seconds (a third of this time is reserved for the local search method). Additionally, each algorithm was executed once for each problem instance.

ω_{int}	*Limit*	%D	Av. rank.	%B
$0.5k$	$0.1n$	3.75	5.50	68
$0.5k$	$0.25n$	3.82	6.04	64
$0.25k$	$0.1n$	7.53	6.42	48
$0.25k$	$0.5n$	8.71	6.72	48
$0.5k$	$0.5n$	3.90	6.74	64
$0.25k$	$0.25n$	5.60	7.20	48
$0.1k$	$0.1n$	13.00	7.76	36
$0.1k$	$0.25n$	11.21	8.16	44
$0.1k$	$0.5n$	11.51	8.18	40
$0.75k$	$0.5n$	13.25	9.82	44
$0.75k$	$0.25n$	14.54	10.02	36
$0.75k$	$0.1n$	15.64	10.22	28
$0.1k$	$0.05n$	15.32	10.56	24
$0.25k$	$0.05n$	15.35	10.72	36
$0.50k$	$0.05n$	16.02	10.80	40
$0.75k$	$0.05n$	15.41	11.14	36

Table 1: Results of ABC with different parameter values

We computed the overall best solution value for each problem instance, *BestValue*, obtained by the execution of all the methods under consideration. Then, for each method, we computed the relative deviation between the best solution value and *BestValue* ($\frac{Method-BestValue}{BestValue}$). In Table 1, we report the average of this relative deviation in percentage across all the instances considered in each particular experiment ($\%D$), and the percentage of instances for which the value of the best solution obtained by a given method matches *BestValue* ($\%B$). We also show the averaged *rankings*, computed by the Friedman test (Friedman, 1940), obtained by these algorithms (*Av. rank.*). This measure is obtained by computing, for each instance, the ranking r_a of the observed results for algorithm a, assigning the ranking 1 to the best of them, and to the worst the ranking $|A|$ (where A is the set of algorithms). Then,

an average measure is obtained from the rankings of each algorithm over all the test problems. For example, if a certain algorithm achieves rankings 1, 3, 1, 4, and 2, on five instances, the averaged ranking is $\frac{1+3+1+4+2}{5} = 2.20$. Note that the lower the ranking, the better the algorithm.

Results in Table 1 show that the ω_{int} parameter has an important effect on the quality of the solutions obtained by our ABC. In general, the best algorithms in terms of the three measures use $\omega_{int} = 0.25k$ or $0.5k$. The application of the lowest and highest values for this parameter ($0.1k$ and $0.75k$) degrades all the performance measures. Another important remark is that the lowest value for *Limit* ($0.05n$) has negative effects as well (the four worst ABC variants employ this *Limit* value). We choose $\omega_{int} = 0.5k$ and $Limit = 0.1n$ for all the remaining experiments, because the best ranked configuration employs this setting.

5.3. Comparative Study of Algorithms for the Min-Max BC Problem

In this section, we undertake a comparative analysis between ABC and other algorithms that may be applied to face the Min-Max BC problem from a practical point of view. In particular, we put our proposal against the constructive method SBA (Section 5.3.1), the CNP1 algorithm (Pullan, 2015) (Section 5.3.2), the module-based attack (Requião da Cunha et al., 2015) (Section 5.3.3), and three well-known metaheuristics (Section 5.3.4). All instance sets were assumed for these experiments (a total of 60 test graphs). The raw results of all the methods are provided in Tables A.10-A.12 in Appendix A (best results are boldfaced).

For the comparison between ABC and any another algorithm X, we consider two performance measures, *%Success* (*%S*) and *%Improvement* (*%I*). *%S* represents the percentage of instances for which ABC found strictly better solutions than X in a particular instance set, whereas *%I* is the average in percentage of the relative deviation between the quality of the solution reached by ABC and the one of the solution obtained by X ($\frac{X-ABC}{X}$) across those instances where ABC is the winner. Furthermore, we also analyse these measures for the cases where the performance of X surpasses that of ABC.

5.3.1. ABC versus SBA

SBA is the main method for the Min-Max BC problem so far, given its relevance in the literature (Veremyev et al., 2015). In fact, this procedure is invoked inside ABC to obtain an initial solution. Therefore, we look into whether our ABC algorithm is able to improve the solution provided by this

constructive method more carefully. For these experiments, we grouped the graph instances according to their corresponding model. For each instance group, Table 2 shows the $\%I$ and $\%S$ measures when comparing ABC and SBA with $k = \{0.1n, 0.3n, 0.5n\}$. Note that, since SBA is invoked inside ABC, the reported values for $\%S$ when SBA wins are always equal to zero. The performance measures were also computed for all the tested instances in a single group in the bottom row.

	$k = 0.1n$				$k = 0.3n$				$k = 0.5n$			
	ABC wins		*SBA wins*		*ABC wins*		*SBA wins*		*ABC wins*		*SBA wins*	
Inst. Set	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S
ER	7.5	100	-	0	20.5	100	-	0	26.2	90	-	0
CR	9	100	-	0	21.6	100	-	0	39.6	100	-	0
RG	57.4	100	-	0	45	100	-	0	61.1	90	-	0
FF	37	90	-	0	67.1	70	-	0	58.7	40	-	0
BA	17.5	100	-	0	47.6	100	-	0	50	20	-	0
WS	66.8	100	-	0	21.7	70	-	0	92.9	70	-	0
All inst.	32.5	98.3	-	0	36.4	90	-	0	52.8	68.3	-	0

Table 2: ABC vs. SBA

We point out the following remarks from Table 2. Taking into account all the instances, ABC was able to get better results than those of SBA in 98.3% of the cases for $k = 0.1n$, 90% for $k = 0.3n$, and 68.3% for $k = 0.5n$, with improvements greater than 30% for all the k values. Therefore, we may conclude that the ABC framework allowed us to conceive an effective optimizer for the Min-Max BC problem, which acceptably surpasses the most popular algorithm in the literature that may be employed as a solver for our problem. We should point out that the low values for $\%S$ for the FF and BA graphs ($k = 0.5n$) are due to the fact that SBA was able to completely break many of these networks. It achieved zero as the maximal BC value, which is an unbeatable situation (see the raw results of these methods in Table A.12).

Next, some graphs are shown in Fig. 5 to illustrate the type of solutions returned by SBA and ABC. This is done by presenting the graphs before and after the vertices removal produced by these algorithms. For the sake of saving space, we restrict the plots to one WS-5 instance with $k = 0.2n$ (Fig. 5.a) and one BA-5 instance with $k = 0.3n$ (Fig. 5.b), both graphs having 50 vertices. They are denoted as WS50 and BA50, respectively. The corresponding residual graphs derived from the ABC solution are shown in Figures 5.e and 5.f. Those obtained by SBA are outlined in Figures 5.c and

5.d. Firstly, we may observe that ABC is the winner for the two graphs analysed (for BA50, ABC achieved a maximal BC of 60 and SBA 158, whereas for WA50, ABC obtained 22 and SBA 83). Comparing the residual graphs induced by the two algorithms, we notice that those from ABC show two determinant properties that favour the reduction of the maximal BC:

- *Residual graphs are more fragmented.* For WS50, ABC was able to yield four components (Fig. 5.f) and SBA forced the apparition of just three (Fig. 5.d). For BA50, ABC returned a graph with six components (Fig. 5.e) and SBA a graph with three (Fig. 5.d). The separation among components isolates groups of nodes from others, limiting the number of paths that may pass through relevant vertices. This way, their BC values decrease and, consequently, the maximal one.

- *Residual graphs are denser and have lower diameters.* The residual graphs of ABC have more edges than those of SBA (for WS50, 56 against 52, and for BA50, 58 against 50). The difference in diameter is very clear as well; especially in the case of BA50 (Figures 5.e and 5.c). In this case, ABC yields a residual graph with a diameter of 6 and SBA one of 11. Note that several geodesic paths with long lengths are evident in Figure 5.c for the case of SBA, which propitiated the apparition of nodes with too high BC values. On the other hand, ABC has discovered that gaining density and reducing diameter are adequate to cut down the maximal BC.

5.3.2. ABC versus a Metaheuristic for the CNP-A

We saw in Section 3 that the optimal solution for the CNP-A does not have to produce an accurate solution for the Min-Max BC problem. However, we might intuit that high-quality solutions for the CNP-A problem might show promising features as solutions for our problem, since the minimization of the total number of connected nodes in the graph promotes a decrease in the BC value of many of them, probably affecting the maximal one. Thus, investigating the behaviour of algorithms for the CNP-A as solvers for the Min-Max BC problem deserves attention.

We have implemented a state-of-the-art metaheuristic for the CNP-A, called CNP1, which was proposed by Pullan (2015). CNP1 is a multi-start greedy algorithm that maintains a current solution S (a subset of k vertices from V) and iterates through the main loop that involves the following operations:

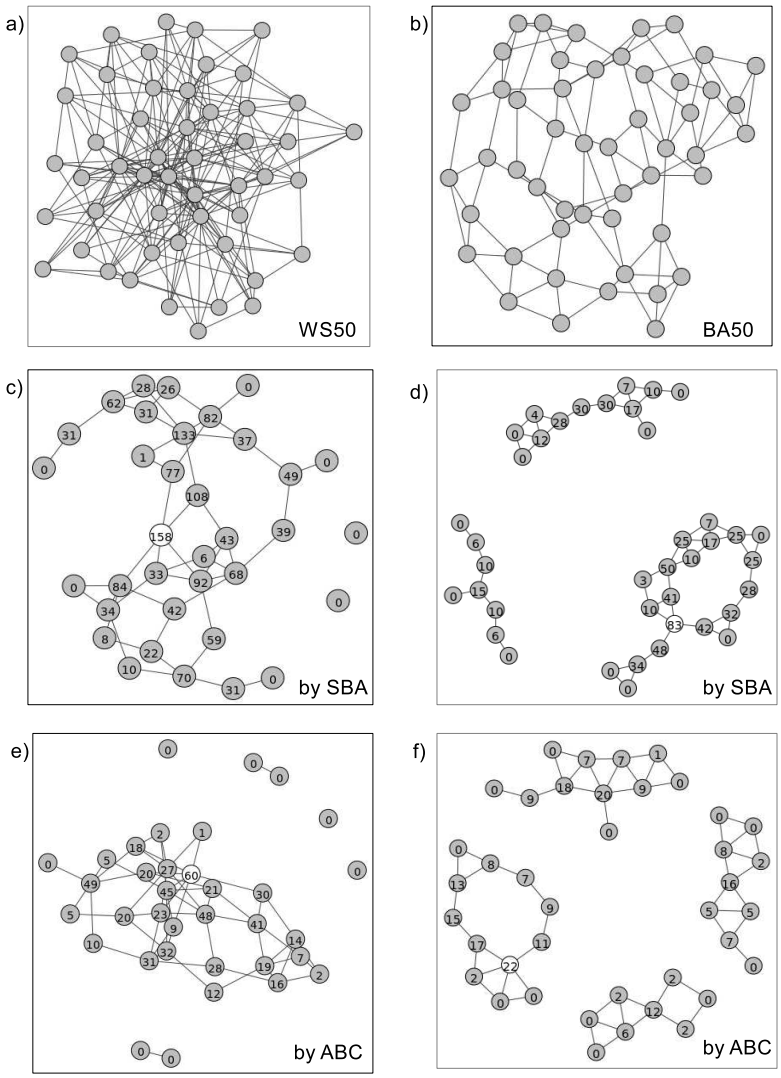

Figure 5: ABC vs. SBA on WS50 and BA50

1. Firstly, it constructs S' by adding a node from G^S to S (notice that S' is not a valid solution for the problem, because it removes $k+1$ nodes from G). The node is randomly selected from those components whose size is greater than the average between those of the largest and the smallest components ($\frac{|C_{largest}|+|C_{smallest}|}{2}$).
2. Then, the node whose removal from S' gives the minimum increase in the CNP-A objective function is extracted from this set (this way, S' turns into a valid solution). Later, S' replaces S only if it provides a better objective function value.

Furthermore, to prevent search stagnation, a partial restart is performed periodically by generating a new initial solution S. The initial solution is created by invoking the first step described above k times. CNP1 finishes when a termination condition is met (e.g., maximum number of iterations or computation time allowed). The best solution generated during the iterative process is kept as the overall result.

We have applied CNP1 to the 60 test graphs and used the same CPU time limit as for ABC ($3.6n$ seconds). The solutions returned by the CNP1 algorithm for the CNP-A problem were evaluated according to the maximal BC of the residual graphs. Table 3 summarizes the comparison between these results and those achieved by our ABC.

	$k = 0.1n$				$k = 0.3n$				$k = 0.5n$			
	ABC wins		*CNP1 wins*		*ABC wins*		*CNP1 wins*		*ABC wins*		*CNP1 wins*	
Inst. Set	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S
ER	27.2	90	4.1	10	45.2	90	0.3	10	48	100	-	0
CR	26.9	100	-	0	43.5	90	7.6	10	49.5	100	-	0
RG	58.7	30	38.4	70	81.8	100	-	0	96.9	100	-	0
FF	59.8	80	8.4	20	72	70	22.6	30	91.7	100	-	0
BA	71	100	-	0	89.3	100	-	0	100	100	-	0
WS	81	100	-	0	98.2	100	-	0	100	100	-	0
All inst.	53.8	83.3	28.9	16.7	72.7	91.7	15.2	8.3	81	100	-	0

Table 3: ABC vs. CNP1

The global statistics in Table 3 (bottom row) reveal that the performance of ABC (in terms of both *%S* and *%I*) surpasses that of CNP1 in terms of the maximal BC. This is especially noteworthy with respect to the results for $k = 0.3n$ and $k = 0.5n$. The algorithmic components of ABC adequately suit the characteristics of the Min-Max BC problem, showing a profitable

problem-knowledge exploitation with regards to a competitive method for a similar problem. CNP1 attempts to fragment the graph into as many components (with similar sizes) as possible, without paying attention to the BC values of the nodes in the remaining graph. Interestingly, this way of acting has clearly been advantageous only when dealing with RG graphs with $k = 0.1n$. In this case, CNP1 was able to reach high-quality solutions that improved those obtained by ABC.

5.3.3. ABC versus MBA

Module-based attack (MBA) (Requião da Cunha et al., 2015) is another appealing contemporary algorithm that can be employed to solve the Min-Max BC problem. It follows two essential strategies:

- A *community detection* algorithm should be applied to identify topological communities or modular structures by which the network can be represented; then, only the nodes that participate of inter-community links are removed in descending order of their BC. The authors argued that since the concentration of links within the modules is greater than the concentration of links connecting them, those nodes that connect different modules (bridge nodes) are the appropriate candidates to be removed in order to effectively detach the communities of a network.

- At each step, the attack should be focused on the remaining largest connected component in order to speed up the fragmentation.

In Figure 6, we sketch the algorithm for the Min-Max BC problem based on these ideas. After the application of the community detection algorithm (Step 3), a set B with the bridge nodes is built (Line 4). Then, the algorithm iterates by obtaining a set $\mathcal{P}$ with subsets of vertices representing connected components of the current graph (Line 6). If B is not empty, the largest component having vertices from this set is chosen (Line 8) and the one with the highest BC value is definitively selected as the vertex to be removed (Line 9). Note that once a node from a link between two communities is deleted, its counterpart should be skipped from B (there is no need to remove it), unless it also participates in other inter-community connections (Requião da Cunha et al., 2015). The procedure Redundant-Bridge-Nodes (Line 10) is employed to identify these redundant nodes. When B becomes empty, the node with the highest BC from the largest component is always the one to be attacked (Lines 13-14). Finally, we should remark that MBA is a very quick

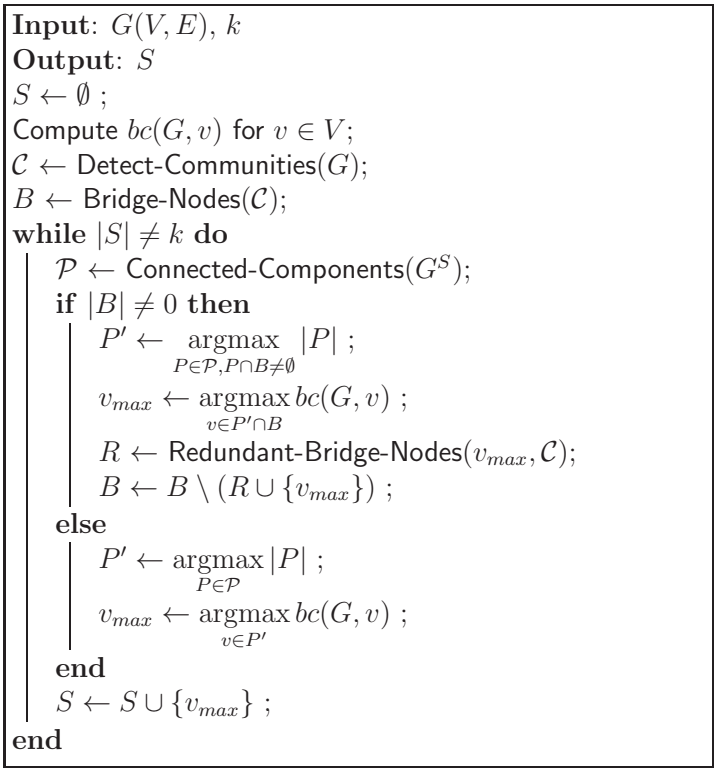

Figure 6: Pseudocode of MBA

procedure, since it follows the strategy of simultaneous attacks (BC values are computed only once in Line 2).

In Table 4, we show the results of the experimental comparison between ABC and MBA. MBA invoked the community detection by label propagation (Raghavan et al., 2007) as implemented in the *NetworKit* framework (Staudt et al., 2014). According to the results in Table 4, ABC outperforms its competitor on the set formed by all the instances with impressive $\%I$ and $\%S$ values, especially when $k = 0.3n$ and $k = 0.5n$. In general, this superiority is evident in all instance sets except in the case of the RG graphs ($k = 0.1n$), where MBA performs better than ABC for 60% of the instances. In RG graphs, nodes are only connected to nearby nodes, which promotes the formation of clusters with few connections to the rest of the graph. Pre-

	$k = 0.1n$				$k = 0.3n$				$k = 0.5n$			
	ABC wins		*MBA wins*		*ABC wins*		*MBA wins*		*ABC wins*		*MBA wins*	
Inst. Set	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S
ER	16.8	100	-	0	33.1	100	-	0	43.2	100	-	0
CR	17.1	90	-	0	32.9	100	-	0	44.8	100	-	0
BA	47	100	-	0	71.2	100	-	0	99.5	100	-	0
FF	59.1	90	10.6	10	60.2	80	15.4	20	85.8	90	-	0
RG	49.3	40	39.1	60	81.6	100	-	0	97.7	100	-	0
WS	80.2	100	-	0	99.3	100	-	0	100	100	-	0
All inst.	44.7	86.7	35	11.7	63.2	96.7	15.4	3.3	78.4	98.3	-	0

Table 4: ABC vs. MBA

cisely, MBA was specifically designed to perform well in this scenario, which explains its advantage for this type of networks.

5.3.4. Comparison with Other Metaheuristics

We have implemented three other metaheuristic approaches that may be conceived for this problem, a steady-state genetic algorithm (SGA), a simulated annealing algorithm (SA), and a variable neighbourhood search (VNS). All these optimizers assume the same solution encoding as ABC and are given the same amount of computational time as that given to our ABC. Their features are described below.

SGA is a steady-state genetic algorithm (Lozano et al., 2008) that starts with an initial set of random solutions forming a population of so-called chromosomes of size $N_p = 60$. Later, the population is subject to an evolutionary loop that adopts the following operations:

1. Select two parents from the population using the binary tournament selection mechanism. This selection technique is widely used in genetic algorithms due to its simplicity and ability to escape from local optima. It selects the fittest chromosome between two that are randomly picked out from the population.
2. Create an offspring applying the crossover and mutation operators presented by Wolters (2015). The crossover operator generates an offspring by first pooling the unique vertices of the two parent solutions, and then sampling uniformly and randomly a set of k unique vertices from this pool. Next, the genes of the offspring are mutated with probability $p_m = 0.01$ by means of the 1-interchange move.

3. Select an individual from the population and decide if this individual will be replaced by the offspring. For this decision, we consider the replace worst strategy, which replaces the worst individual in the population only if the new individual is better.

SA (Aarts and Korst, 1989; Kirkpatrick et al., 1983; Suman and Kumar, 2006) is said to be the oldest metaheuristic. Its main idea is to scape from local optima by accepting worse solutions according to a decreasing probability. Our implementation applies the logistic acceptance criterion and the 1-interchange move. During the run, it applies 1000 geometric cooling steps, evenly spaced, with a cooling factor that makes the temperature be equal to 0 in the last stage (the smallest positive double value of the programming language, actually). The initial temperature was set according to the logistic rule, an initial probability of keeping the best solution set to 0.49, and an estimation of the fitness difference between two neighbouring solutions calculated as the average of the fitness difference between 10 random solutions and one of their neighbours each.

VNS (Mladenović and Hansen, 1997) is a metaheuristic that systematically exploits the idea of neighbourhood change (from a given set of neighbourhood structures), both in the descent to local optima and in the escape from the valleys which contain them. We have designed a simple VNS algorithm to tackle the Min-Max BC problem that mainly consists of the following two phases, which work on the current solution S_c:

1. *Generation phase*: Firstly, the SBA method is executed to generate an initial solution.
2. *Shaking phase*: This method applies the destructive-constructive neighbourhood operator on S_c (Section 4.1). In order to get a reactive behaviour, the parameter ω is adjusted dynamically depending on the quality of the solutions obtained. At the beginning of the run, and every time the best-known solution is improved, ω is set to $0.1k$; otherwise, ω is increased by $0.1k$. When ω exceeds ω_{max}, ω gets back to $0.1k$. We set ω_{max} to $0.75k$.

In the standard VNS model, the current solution is additionally refined by a local search method. However, we have opted for a *reduced* VNS algorithm, which is a simplified VNS variant where local search (the most time-consuming part of VNS) is removed to improve its search efficiency (Hansen

et al., 2010). We should point out that this approach constitutes an alternative way of exploiting the neighborhood structure that becomes the fundamental component of ABC and, in this way, it may be useful to access the performance of this operator in a different optimization procedure for our problem.

In SA and VNS, the evaluation of the solutions is carried out taking advantage of the implementation issues detailed in Section 4.3. However, in SGA, the objective function is computed by calling Brandes' algorithm. Tables 5, 6, and 7 outline the results for the comparison between ABC and SGA, SA, and VNS, respectively. We highlight the following observations concerning these tables.

1. Results in Table 5 clearly depict that ABC exhibits superior performance compared to the genetic algorithm, SGA. Specifically, its $\%S$ and $\%I$ values are clearly much better.
2. The results of ABC overcome those of SA on the specially structured BA and WS graphs (Table 6). However, although ABC achieves a slight advantage on the ER and CR instance sets when $k = 0.1n$, SA clearly beats our algorithm when $k = 0.3n$ and $k = 0.5n$. It seems that the unstructured topologies of these graphs render the problem hard, so that the work of the 1-interchange move in the SA framework became more profitable than those of the more specialised constructive-destructive neighbourhood operator in ABC. On the other hand, when the tackled graph presents some kind of structure (case of BA and WS), the heuristic decisions taken by this operator were adequate to allow our ABC to get superior performance by an impressive margin.
3. The most significant remark from Table 7 is that ABC achieves better performance than VNS on the ER, CR and RG instances. In addition, it reaches better solutions for BA graphs when $k = 0.1n$ and $k = 0.3n$. Although both algorithms employ the same neighbourhood operator, ABC incorporate two differentiated features that may be responsible of this meaningful advantage: (1) it is a population-based metaheuristic, and (2) it embeds a local search method based on the 1-interchange operator. VNS only provides solutions of superior quality for WS instances when $k = 0.1n$.

	$k = 0.1n$				$k = 0.3n$				$k = 0.5n$			
	ABC wins		*SGA wins*		*ABC wins*		*SGA wins*		*ABC wins*		*SGA wins*	
Inst. Set	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S
ER	6.0	100	-	0	16.7	100	-	0	20.9	90	6.1	10
CR	8.1	70	2.8	30	15.2	80	6.4	20	25.7	100	-	0
RG	19.4	60	1.1	40	65.7	90	4.2	10	92.7	100	-	0
FF	62.7	80	11.2	20	56.9	90	4.3	10	77.9	100	-	0
BA	51.0	100	-	0	69.6	100	-	0	99.8	100	-	0
WS	59.3	100	-	0	94.5	100	-	0	99.4	100	-	0
All inst.	36.0	85	3.9	15	54.2	93.3	5.3	6.7	70.2	98.3	6.1	1.7

Table 5: ABC vs. SGA

	$k = 0.1n$				$k = 0.3n$				$k = 0.5n$			
	ABC wins		*SA wins*		*ABC wins*		*SA wins*		*ABC wins*		*SA wins*	
Inst. Set	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S
ER	3.2	60	2.0	40	28.9	30	9.2	70	50.3	20	15.7	80
CR	3.1	60	4.7	40	5.8	20	11.3	80	100.0	10	12.9	90
RG	33.1	30	5.4	70	71.1	80	12.9	20	87.9	90	-	0
FF	74.2	50	15.6	50	72.6	50	24.7	50	89.8	60	75.6	30
BA	49.5	100	-	0	60.1	100	-	0	99.7	100	-	0
WS	59.3	100	-	0	92.2	100	-	0	96.2	100	-	0
All inst.	39.9	66.7	7.1	33.3	67.2	63.3	13.8	36.7	91.8	63.3	23.4	33.3

Table 6: ABC vs. SA

5.3.5. Comparative Analysis using Wilcoxon's Test

The aim of this section is to assess whether the performance differences previously observed between ABC and its competitors are statistically significant. We undertake this study by means of the Wilcoxon matched-pairs signed ranks test (Garcia et al., 2009), which allows the results of two algorithms to be compared. In statistical terms, this test answers the following question: Do the two samples represent two different populations? Table 8 summarizes the results of this procedure for a level of significance $\alpha = 0.05$, where the values of R^+ (associated with ABC) and R^- (associated with the corresponding competitor) of the test are specified. If R^- is smaller than both R^+ and the critical value, ABC is statistically better than the other algorithm (represented with the sign + in the column named *Dif?*); if R^+ is smaller than both R^- and the critical value, our algorithm is statistically worse than its competitor (represented with the sign −); if neither R^+ nor R^- is smaller than the critical value, the test does not find statistical differ-

	$k = 0.1n$				$k = 0.3n$				$k = 0.5n$			
	ABC wins		*VNS wins*		*ABC wins*		*VNS wins*		*ABC wins*		*VNS wins*	
Inst. Set	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S	%I	%S
ER	7.1	90	0.6	10	12.0	80	7.2	20	19.0	90	-	0
CR	9.5	90	0.0	10	11.7	90	1.4	10	26.4	90	-	0
RG	42.4	90	12.7	10	10.7	60	9.2	40	37.7	70	5.5	20
FF	27.9	70	3.0	10	72.6	50	-	0	71.5	20	25.7	20
BA	15.8	100	-	0	26.6	70	23.8	30	-	0	50	10
WS	7.2	10	14.5	90	12.7	20	18.8	40	-	0	100	20
All inst.	19.8	75	11.3	21.7	22.7	61.7	14.2	23.3	30.2	45.0	44.6	11.7

Table 7: ABC vs. VNS

ences among them (represented with the sign $\sim$). This non-parametric test was applied considering the results on two groups of graphs: (1) *Random graphs*, formed by the ER, CR, and RG instances and (2) *Structured graphs*, composed of the FF, BA, and WS instances.

		$k = 0.1n$			$k = 0.3n$			$k = 0.5n$		
	Alg.	R^+	R^-	Dif?	R^+	R^-	Dif?	R^+	R^-	Dif?
Random Graphs	CNP1	465	0	+	465	0	+	465	0	+
	MBA	465	0	+	465	0	+	465	0	+
	SBA	465	0	+	465	0	+	463	1.5	+
	VNS	433	32	+	378	87	+	416	18.5	+
	SGA	384	81	+	449	16	+	460	5	+
	SA	207	258	$\sim$	241	224	$\sim$	226	209	$\sim$
Structured Graphs	CNP1	465	0	+	465	0	+	465	0	+
	MBA	465	0	+	465	0	+	435	0	+
	SBA	435	0	+	454.5	10.5	+	367	68	+
	VNS	314.5	150.5	$\sim$	308	127	+	183.5	251.5	$\sim$
	SGA	455	10	+	463	2	+	465	0	+
	SA	428	37	+	407	58	+	378	57	+

Table 8: Comparison of ABC and the other algorithms (Wilcoxons test with p-value = 0.05 and critical value = 137).

The Wilcoxon test reveals that: (1) ABC has the upper hand in the statistical comparison over CNP1, SBA, MBA, and SGA for all k values, (2) our algorithm statistically outperforms SA in the case of structured graphs and there are not significant differences between them for random graphs, (3) an opposite scenario seems to occur with regard to the comparison with VNS; ABC is statistically superior to VNS on the random graphs and no

differences were found on the structured graphs for $k = 0.1n$ and $k = 0.5n$ (our algorithm attains significantly better results, as well, for $k = 0.3n$).

To sum up, this study reveals that ABC arises as a very attractive alternative to other approaches that might be applicable to the Min-Max BC problem. However, we believe that there is still room for improvement. Specifically, as future work, we will try to hybridize our ABC with SA, which provided outstanding performance on less structured graphs. In fact, the hybridization of metaheuristics is currently a prospective research area for finding more effective search algorithms (Rodriguez et al., 2012). Our idea is to suitably combine complementary algorithm concepts to provide hybrid approaches with a better performance than that obtained by ABC and SA separately.

5.3.6. Global Summary

In this section, we perform a global comparison of all the algorithms presented in this work. In order to do this, we analyse, for each method, two performance measures that were described in Section 5.2, the mean ranking computed according to Friedman's test (Friedman, 1940) and *%Best*. Table 9 displays the corresponding results on the random graphs and the structured graphs considering all k values. The algorithms were ranked according to their *Av. rank.* values.

Random Graphs			*Structured Graphs*		
Alg.	Av. rank.	%B	Alg.	Av. rank.	%B
ABC	1.833	34.4	ABC	1.783	62.2
SA	2.206	54.4	VNS	2.244	61.1
VNS	3.156	14.4	SBA	3.361	23.3
SGA	4.044	1.1	SA	3.578	15.6
SBA	4.372	2.2	SGA	4.606	0
MBA	6.078	0	MBA	6.039	1.1
CNP1	6.311	0	CNP1	6.389	0

Table 9: Comparison among all the algorithms

Based on the performance values that are shown in Table 9, we can make the following remarks: (1) ABC is the best ranked algorithm for the two experimental scenarios, proving to be very robust; (2) SA reveals satisfactory potential for the group of random graphs (it obtains the best *%B* performance in this case); (3) the algorithms exploiting heuristic knowledge about the problem (SBA, VNS, and ABC) get the best rankings for the structured

graphs (SA falls behind them, obtaining a poor $\%B$ value); (4) the greedy heuristic algorithms, SBA and MBA, are surpassed in both scenarios by several metaheuristic approaches; and (5) just as we expected, CNP1 gets the worst ranking values.

6. Conclusions

We have proposed an ABC algorithm to tackle an instance of CNP whose objective concerns the minimization of the maximal BC value in the residual graph. Our algorithm applies a destructive–constructive neighbourhood operator for generating new candidate solutions when bees explore the vicinity of food sources. One of its most essential characteristics involves the incorporation of a BC update algorithm that makes recomputing BC values efficient. The designed ABC approach, tested on 60 graphs, has proven to be very competitive with respect to a baseline attack algorithm, a state-of-the-art metaheuristic for the CNP-A, a recent attack method that embeds a community detection algorithm, and three approaches designed to face the problem and based on three well-known metaheuristics. Therefore, we may conclude that this metaheuristic is a tool of choice for this problem.

We believe that the work presented in this paper is a significant contribution because it represents the meeting point between three prospective research lines: CNP (Pullan, 2015; Ventresca and Aleman, 2015a; Veremyev et al., 2015), BC (Goel et al., 2015; Kourtellis et al., 2015; Lee et al., 2016; Riondato and Kornaropoulos, 2015), and ABC (Bansal et al., 2013; Bolaji et al., 2013; Karaboga et al., 2012a). Therefore, we will intend to explore other interesting avenues of research, such as the adaptation of the proposed approach for its application on large networks with many thousands of nodes. In this case, we shall explore the incorporation, in our ABC framework, of approximate algorithms that estimate inexact but accurate BC values and reduce the computational effort further (Kourtellis et al., 2015; Riondato and Kornaropoulos, 2015; Yoshida, 2014). Furthermore, we will try to improve our ABC proposal exploring two different avenues: (1) the incorporation of innovative ABC components appeared recently in the literature (Das et al., 2013; Biswas et al., 2013) and (2) the implementation on parallel hardware, following the indications given by Parpinelli et al. (2011). Finally, we will study the application of the proposed algorithm as a tool to deal with appealing real-world applications such as to break up jihadist terrorist networks and to enhance lifetime of wireless networks (Kundu et al., 2015).

Acknowledgements

This work has been financed/supported by the Spanish Ministry of Economy and Competitiveness and European Regional Development Fund (SMEC / ERDF) within the framework of the research project that is referenced by DER2015-63857-R and by the project 18/16 CEMIX UGR-MADOC.

References

Aarts, E., Korst, J., 1989. Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing. John Wiley & Sons, Inc., New York, NY, USA.

Akay, B., Karaboga, D., 2012. A modified artificial bee colony algorithm for real-parameter optimization. Information Sciences 192, 120–142.

Akbari, R., Hedayatzadeh, R., Ziarati, K., Hassanizadeh, B., 2012. A multi-objective artificial bee colony algorithm. Swarm and Evolutionary Computation 2, 39–52.

Anthonisse, J. M., 1971. The rush in a directed graph. Tech. Rep. BN 9/71, Stichting Mathematisch Centrum, Amsterdam.

Aringhieri, R., Grosso, A., Hosteins, P., Scatamacchia, R., 2015. VNS solutions for the critical node problem. Electronic Notes in Discrete Mathematics 47, 37–44.

Arulselvan, A., Commander, C. W., Elefteriadou, L., Pardalos, P. M., 2009. Detecting critical nodes in sparse graphs. Computers & Operations Research 36 (7), 2193–2200.

Banitalebi, A., Aziz, M. I. A., Bahar, A., Aziz, Z. A., 2015. Enhanced compact artificial bee colony. Information Sciences 298, 491–511.

Bansal, J. C., Sharma, H., Jadon, S. S., 2013. Artificial bee colony algorithm: a survey. International Journal of Advanced Intelligence Paradigms 5 (1-2), 123–159.

Barabási, A.-L., Albert, R., 1999. Emergence of scaling in random networks. Science 286 (5439), 509–512.

Biswas, S., Kundu, S., Das, S., Vasilakos, A., 2013. Information sharing in bee colony for detecting multiple niches in non-stationary environments. In: Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation. GECCO '13 Companion. pp. 1–2.

Blum, C., Roli, A., 2003. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys 35 (3), 268–308.

Bolaji, A., Khader, A., Al-Betar, M., 2013. Artificial bee colony algorithm, its variants and applications: a survey. Journal of Theoretical and Applied Information Technology 47 (2), 434–459.

Borgatti, S. P., 2006. Identifying sets of key players in a social network. Computational & Mathematical Organization Theory 12 (1), 21–34.

Bose, D., Biswas, S., Vasilakos, A. V., Laha, S., 2014. Optimal filter design using an improved artificial bee colony algorithm. Information Sciences 281, 443–461.

Bose, D., Kundu, S., Biswas, S., Das, S., 2012. Circular antenna array design using novel perturbation based artificial bee colony algorithm. In: Panigrahi, B. K., Das, S., Suganthan, P. N., Nanda, P. K. (Eds.), Swarm, Evolutionary, and Memetic Computing: Third International Conference, SEMCCO 2012, Bhubaneswar, India, December 20-22, 2012. Proceedings. Springer Berlin Heidelberg, pp. 459–466.

Brandes, U., 2001. A faster algorithm for betweenness centrality. Journal of Mathematical Sociology 25, 163–177.

Crucitti, P., Latora, V., Marchiori, M., Rapisarda, A., 2004. Error and attack tolerance of complex networks. Physica A: Statistical Mechanics and its Applications 340 (1), 388–394.

Csárdi, G., Nepusz, T., 2006. The igraph software package for complex network research. InterJournal Complex Systems 1695, 1–9.

Das, S., Biswas, S., Kundu, S., 2013. Synergizing fitness learning with proximity-based food source selection in artificial bee colony algorithm for numerical optimization. Applied Soft Computing 13 (12), 4676–4694.

Delgado-Osuna, J. A., Lozano, M., García-Martínez, 2016. An alternative artificial bee colony algorithm with destructive-constructive neighbourhood operator for the problem of composing medical crews. Information Sciences 326, 215–226.

Deng, Y., Wu, J., Tan, Y.-j., 2016. Optimal attack strategy of complex networks based on tabu search. Physica A: Statistical Mechanics and its Applications 442, 74–81.

Erdös, P., Rényi, A., 1959. On random graphs. Publicationes Mathematicae 6, 290–297.

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

Friedman, M., 1940. A comparison of alternative tests of significance for the problem of m rankings. The Annals of Mathematical Statistics 11 (1), 86–92.

Gao, W., Chan, F. T., Huang, L., Liu, S., 2015. Bare bones artificial bee colony algorithm with parameter adaptation and fitness-based neighborhood. Information Sciences 316, 180–200.

Garcia, S., Molina, D., Lozano, M., Herrera, F., 2009. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. Journal of Heuristics 15 (6), 617–644.

García-Martínez, C., Blum, C., Rodriguez, F., Lozano, M., 2015. The firefighter problem: Empirical results on random graphs. Computers and Operations Research 60, 55–66.

García-Martínez, C., Rodríguez, F. J., Lozano, M., 2012. Arbitrary function optimisation with metaheuristics. Soft Computing 16 (12), 2115–2133.

Goel, K., Singh, R. R., Iyengar, S., Gupta, S., 2015. A faster algorithm to update betweenness centrality after node alteration. Internet Mathematics 11 (4-5), 403–420.

Gunasekara, R. C., Mehrotra, K., Mohan, C. K., 2015. Multi-objective optimization to identify key players in large social networks. Social Network Analysis and Mining 5 (1), 1–20.

Hagberg, A. A., Schult, D. A., Swart, P. J., 2008. Exploring network structure, dynamics, and function using NetworkX. In: Proceedings of the 7th Python in Science Conference (SciPy2008). Pasadena, CA USA, pp. 11–15.

Hansen, P., Mladenović, N., Moreno Pérez, J. A., 2010. Variable neighbourhood search: methods and applications. Annals of Operations Research 175 (1), 367–407.

Hewett, R., 2011. Toward identification of key breakers in social cyber-physical networks. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. pp. 2731–2736.

Iyer, S., Timothy, K., Bala, S., Zhen, W., 04 2013. Attack robustness and centrality of complex networks. PLoS ONE 8 (4), e59613.

Karaboga, D., Akay, B., 2011. A modified Artificial Bee Colony (ABC) algorithm for constrained optimization problems. Applied Soft Computing 11 (3), 3021–3031.

Karaboga, D., Basturk, B., 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization 39 (3), 459–471.

Karaboga, D., Basturk, B., 2008. On the performance of artificial bee colony (ABC) algorithm. Applied Soft Computing 8 (1), 687–697.

Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N., 2012a. A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artificial Intelligence Review 42 (1), 21–57.

Karaboga, D., Ozturk, C., 2011. A novel clustering approach: Artificial Bee Colony (ABC) algorithm. Applied Soft Computing 11 (1), 652–657.

Karaboga, D., Ozturk, C., Karaboga, N., Gorkemli, B., 2012b. Artificial bee colony programming for symbolic regression. Information Sciences 209, 1–15.

Kas, M., Carley, K. M., Carley, L. R., 2014. An incremental algorithm for updating betweenness centrality and k-betweenness centrality and its performance on realistic dynamic social network data. Social Network Analysis and Mining 4 (1), 1–23.

Kashan, M. H., Nahavandi, N., Kashan, A. H., 2012. DisABC: A new artificial bee colony algorithm for binary optimization. Applied Soft Computing 12 (1), 342–352.

Kirkpatrick, S., Gelatt, C., Vecchi, M., 1983. Optimization by simulated annealing. Science 220 (4598), 671–680.

Kourtellis, N., Morales, G. D. F., Bonchi, F., 2015. Scalable online betweenness centrality in evolving graphs. IEEE Transactions on Knowledge and Data Engineering 27 (9), 2494–2506.

Krebs, V., 2002. Uncloaking terrorist networks. First Monday 7 (4).

Kundu, S., Das, S., Vasilakos, A., Biswas, S., 2015. A modified differential evolution-based combined routing and sleep scheduling scheme for lifetime maximization of wireless sensor networks. Soft Computing 19, 637–659.

Lee, J., Lee, M.-J., Park, J. Y., Choi, R. H., Chung, C.-W., 2012. QUBE: a Quick algorithm for Updating BEtweenness centrality. In: 21st International World Wide Web Conference (WWW2012). pp. 351–360.

Lee, M.-J., Choi, S., Chung, C.-W., 2016. Efficient algorithms for updating betweenness centrality in fully dynamic graphs. Information Sciences 326, 278–296.

Leskovec, J., Faloutsos, C., 2007. Scalable Modeling of Real Graphs Using Kronecker Multiplication. In: Proceedings of the 24th International Conference on Machine Learning. ICML '07. ACM, New York, NY, USA, pp. 497–504.

Lozano, M., Duarte, A., Gortázar, F., Martí, R., 2013. A hybrid metaheuristic for the cyclic antibandwidth problem. Knowledge-Based Systems 54, 103–113.

Lozano, M., Herrera, F., Cano, J., 2008. Replacement strategies to preserve useful diversity in steady-state genetic algorithms. Information Sciences 178 (23), 4421–4433.

Lozano, M., Molina, D., García-Martínez, C., 2011. Iterated greedy for the maximum diversity problem. European Journal of Operational Research 214 (1), 31–38.

Lozano, M., Rodriguez, F., Peralta, D., García-Martínez, C., 2016. Randomized greedy multi-start algorithm for the minimum common integer partition problem. Engineering Applications of Artificial Intelligence 50, 226–235.

Maeda, M., Tsuda, S., 2015. Reduction of artificial bee colony algorithm for global optimization. Neurocomputing 148, 70–74.

Mladenović, N., Hansen, P., Nov. 1997. Variable neighborhood search. Computers and Operations Research 24 (11), 1097–1100.

Newman, M. E. J., 2009. Random graphs with clustering. Physical Review Letters 103, 058701.

Ortiz-Arroyo, D., 2010. Computational Social Network Analysis: Trends, Tools and Research Advances. Springer London, London, Ch. Discovering sets of key players in social networks, pp. 27–47.

Pan, Q.-K., Tasgetiren, M. F., Suganthan, P., Chua, T., 2011. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Information Sciences 181 (12), 2455–2468.

Parpinelli, R. S., Benitez, C. M. V., Lopes, H. S., 2011. Parallel approaches for the artificial bee colony algorithm. In: Handbook of Swarm Intelligence: Concepts, Principles and Applications. Springer Berlin Heidelberg, pp. 329–345.

Pullan, W., 2015. Heuristic identification of critical nodes in sparse real-world graphs. Journal of Heuristics 21 (5), 577–598.

Raghavan, U. N., Albert, R., Kumara, S., 2007. Near linear time algorithm to detect community structures in large-scale networks. Physical Review E 76 (3).

Requião da Cunha, B., Carlos, G.-A. J., Sebastián, G., 11 2015. Fast fragmentation of networks using module-based attacks. PLoS ONE 10 (11), e0142824.

Riondato, M., Kornaropoulos, E. M., 2015. Fast approximation of betweenness centrality through sampling. Data Mining and Knowledge Discovery 30 (2), 438–475.

Rodriguez, F., Lozano, M., Blum, C., García-Martínez, C., 2013a. An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. Computers and Operations Research 40 (7), 1829–1841.

Rodriguez, F. J., García-Martínez, C., Lozano, M., 2012. Hybrid metaheuristics based on evolutionary algorithms and simulated annealing: taxonomy, comparison, and synergy test. IEEE Transactions on Evolutionary Computation 16 (6), 787–800.

Rodriguez, F. J., Lozano, M., García-Martínez, C., González-Barrera, J. D., 2013b. An artificial bee colony algorithm for the maximally diverse grouping problem. Information Sciences 230, 183–196.

Shen, S., Smith, J. C., Goli, R., 2012. Exact interdiction models and algorithms for disconnecting networks via node deletions. Discrete Optimization 9 (3), 172–188.

Staudt, C., Sazonovs, A., Meyerhenke, H., 2014. NetworKit: an interactive tool suite for high-performance network analysis. CoRR abs/1403.3005. URL `http://arxiv.org/abs/1403.3005`

Suman, B., Kumar, P., 2006. A survey of simulated annealing as a tool for single and multiobjective optimization. Journal of the Operational Research Society 57 (10), 1143–1160.

Sundar, S., Singh, A., 2010. A swarm intelligence approach to the quadratic minimum spanning tree problem. Information Sciences 180 (17), 3182–3191.

Tasgetiren, M. F., Pan, Q.-K., Suganthan, P., Chen, A. H.-L., 2011. A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. Information Sciences 181 (16), 3459–3475.

Ventresca, M., 2012. Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem. Computers & Operations Research 39 (11), 2763–2775.

Ventresca, M., Aleman, D., 2014. A derandomized approximation algorithm for the critical node detection problem. Computers & Operations Research 43, 261–270.

Ventresca, M., Aleman, D., 2015a. Efficiently identifying critical nodes in large complex networks. Computational Social Networks 2 (1), 1–16.

Ventresca, M., Aleman, D., 2015b. Network robustness versus multi-strategy sequential attack. Journal of Complex Networks 3 (1), 126–146.

Veremyev, A., Prokopyev, O. A., Pasiliao, E. L., 2015. Critical nodes for distance-based connectivity and related problems in graphs. Networks 66 (3), 170–195.

Walteros, J. L., Pardalos, P. M., 2012. Applications of Mathematics and Informatics in Military Science. Springer New York, New York, NY, Ch. Selected topics in critical element detection, pp. 9–26.

Watts, D. J., Strogatz, S. H., 1998. Collective dynamics of small-world networks. Nature 393 (6684), 409–10.

Wolters, M. A., 2015. A genetic algorithm for selection of fixed-size subsets with application to design problems. Journal of Statistical Software, Code Snippets 68 (1), 1–18.

Yoshida, Y., 2014. Almost Linear-time Algorithms for Adaptive Betweenness Centrality Using Hypergraph Sketches. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '14. ACM, New York, NY, USA, pp. 1416–1425.

Appendix A. Detailed Results

Insts.	n	m	*SBA*	*CNPI*	*MBA*	*SGA*	*SA*	*VNS*	*ABC*
ER-0.05	100	220	627.5	536.6	749.2	417.6	419.2	559.5	**397.5**
ER-0.05	250	1497	800.1	1864.2	902.1	744.4	**663.6**	783.6	684.5
ER-0.05	500	6199	1046.1	1439.8	1282.1	1086.1	1023.5	1046.0	**1013.0**
ER-0.05	750	14182	1277.1	1679.0	1600.3	1361.8	**1225.3**	1277.1	1248.5
ER-0.05	1000	25134	1497.3	2219.1	1782.1	1610.7	1493.8	1489.2	**1477.2**
ER-0.2	100	976	130.8	139.5	139.8	114.2	**112.5**	130.8	114.0
ER-0.2	250	6291	248.5	332.4	299.9	254.0	**242.7**	248.5	247.3
ER-0.2	500	24966	486.2	649.4	575.3	493.5	478.1	482.0	**471.9**
ER-0.2	750	56064	685.1	879.8	806.6	733.7	704.4	**678.7**	683.0
ER-0.2	1000	100292	880.0	1127.3	1067.0	971.2	944.9	879.2	**876.3**
CR-0.2/0.05	100	250	510.3	566.4	637.7	**379.0**	391.6	510.3	379.4
CR-0.2/0.05	250	1605	725.9	1004.0	840.6	628.7	**605.4**	725.9	682.7
CR-0.2/0.05	500	6409	1060.8	1567.1	1178.4	1004.9	**919.0**	1032.2	937.4
CR-0.2/0.05	750	14116	1272.8	2216.8	1590.8	1354.6	**1205.2**	1267.7	1244.4
CR-0.2/0.05	1000	25485	1438.9	2035.2	1881.6	1607.2	1474.2	1434.0	**1430.2**
CR-0.3/0.1	100	547	231.7	244.3	231.7	176.8	**172.9**	231.7	177.3
CR-0.3/0.1	250	3214	410.4	559.2	446.4	367.3	363.5	410.4	**351.0**
CR-0.3/0.1	500	12439	634.0	1015.8	779.4	649.8	611.9	630.4	**608.3**
CR-0.3/0.1	750	28456	858.8	1197.9	1017.6	918.8	861.4	850.8	**841.8**
CR-0.3/0.1	1000	50593	1093.8	1475.3	1358.8	1231.4	1161.0	**1090.5**	1090.7
RG-0.15	100	361	391.2	769.4	1790.0	572.2	522.9	391.2	**308.7**
RG-0.15	250	1922	4549.7	5374.0	5634.6	2803.8	2958.6	1964.7	**1287.8**
RG-0.15	500	7583	8170.0	8894.7	7248.8	5110.6	**4984.0**	5922.4	5211.7
RG-0.15	750	16832	36845.6	10873.1	9439.0	7402.3	7260.3	**6488.0**	7433.9
RG-0.15	1000	30864	57206.5	13778.9	13468.7	9831.0	**8940.2**	25353.5	10017.1
RG-0.3	100	956	1147.2	604.8	1029.7	415.8	404.8	1054.0	**397.7**
RG-0.3	250	6989	1414.3	1141.9	1144.6	714.0	**692.2**	1240.5	716.2
RG-0.3	500	26714	4114.9	2376.2	2472.7	1662.5	**1560.5**	3138.3	1625.2
RG-0.3	750	60147	5280.8	3766.5	3380.8	2658.2	**2351.5**	4838.2	2506.5
RG-0.3	1000	108433	6892.4	3979.2	4414.2	3209.7	**2874.6**	6466.2	3087.4
FF-0.35/0.32	100	203	**42.0**	77.0	190.0	294.5	163.1	**42.0**	**42.0**
FF-0.35/0.32	250	642	164.1	685.1	1066.8	1037.2	169.4	147.0	**134.6**
FF-0.35/0.32	500	1618	1614.7	32609.9	36658.6	12524.2	5449.1	**659.3**	**659.3**
FF-0.35/0.32	750	2403	913.7	25985.4	41441.1	16038.2	6214.2	**615.0**	634.2
FF-0.35/0.32	1000	2758	552.0	151030.0	198661.0	57594.9	27380.4	421.2	**390.1**
FF-0.45/0.35	100	1630	777.8	659.8	831.9	316.4	**312.3**	764.1	335.4
FF-0.45/0.35	250	12875	1234.5	1196.8	1591.5	720.6	**627.5**	1234.5	663.8
FF-0.45/0.35	500	53618	4094.8	4127.2	4149.3	2562.1	**1650.5**	3528.5	2026.5
FF-0.45/0.35	750	119604	4580.1	7122.7	6195.9	3769.7	**3078.2**	4580.1	4532.4
FF-0.45/0.35	1000	195466	12357.0	12080.1	9831.5	8116.3	**6194.2**	10995.1	7288.9
BA-4	100	384	1429.1	1909.4	1675.9	895.9	912.1	1429.1	**774.9**
BA-4	250	984	2881.4	6737.9	4411.5	3642.8	3546.0	2881.4	**2600.9**
BA-4	500	1984	7930.4	35396.0	13259.0	18240.9	16863.4	7352.9	**6549.2**
BA-4	750	2984	11788.3	94856.7	35977.1	44696.3	38077.8	10969.8	**10625.7**
BA-4	1000	3984	18828.7	150876.0	45311.6	51782.4	73341.6	18828.7	**18001.5**
BA-5	100	475	840.0	1544.5	938.8	908.7	811.3	840.0	**530.4**
BA-5	250	1225	2481.9	9812.6	4797.4	3003.8	2782.4	2481.9	**1993.5**
BA-5	500	2475	6564.4	29556.6	14093.6	10643.7	11019.0	6564.4	**6272.4**
BA-5	750	3725	9702.1	82545.7	24198.1	27675.5	25140.6	9266.7	**8153.4**
BA-5	1000	4975	14319.8	96156.4	54986.7	52380.3	43490.8	14319.8	**12795.9**
WS-4	100	200	4214.0	2316.4	3212.7	1125.1	1141.8	404.5	**375.4**
WS-4	250	500	2147.6	6167.9	7045.1	3070.9	3538.8	**1233.7**	1415.5
WS-4	500	1000	17303.0	19545.8	14712.7	9525.1	10802.2	**3327.6**	3425.2
WS-4	750	1500	49712.0	31454.8	27842.5	18157.9	15603.0	**3928.2**	5893.1
WS-4	1000	2000	40663.6	47474.3	46022.5	28423.8	25733.1	**10395.9**	10541.8
WS-5	100	200	828.7	1625.2	2413.7	707.0	709.8	**483.3**	502.8
WS-5	250	500	6387.7	6621.1	6335.8	3201.2	3446.8	**1276.2**	1552.4
WS-5	500	1000	13475.4	24030.2	16693.0	10997.1	10443.5	**3740.0**	4673.3
WS-5	750	1500	21225.4	34222.4	31282.4	22044.2	20846.5	**4971.9**	5339.7
WS-5	1000	2000	18897.9	56587.7	45200.3	28758.3	27326.1	**7128.0**	10344.0

Table A.10: Results of the algorithms for $k = 0.1n$

Insts.	n	m	*SBA*	*CNPI*	*MBA*	*SGA*	*SA*	*VNS*	*ABC*
ER-0.05	100	220	262.0	619.4	983.5	295.1	246.3	**36.0**	42.0
ER-0.05	250	1497	925.3	1395.8	876.8	558.6	**435.6**	752.6	491.9
ER-0.05	500	6199	947.1	1451.6	1215.3	876.1	**687.2**	930.7	797.0
ER-0.05	750	14182	1135.4	1697.5	1483.8	1145.4	**890.4**	1114.4	1061.2
ER-0.05	1000	25134	1286.0	2281.7	1681.1	1362.6	**1107.2**	1275.1	1215.1
ER-0.2	100	976	121.2	114.8	154.4	90.7	**78.8**	118.2	86.7
ER-0.2	250	6291	206.2	295.8	252.6	200.4	**176.8**	197.2	185.2
ER-0.2	500	24966	378.8	593.7	522.5	397.5	**355.8**	373.2	357.6
ER-0.2	750	56064	524.6	865.0	773.8	574.1	520.0	**518.5**	518.7
ER-0.2	1000	100292	685.8	1095.0	1007.5	756.9	704.3	683.7	**679.9**
CR-0.2/0.05	100	250	499.7	536.1	712.2	268.9	165.5	**149.9**	152.0
CR-0.2/0.05	250	1605	697.0	896.4	834.6	503.3	**407.9**	648.2	504.4
CR-0.2/0.05	500	6409	973.8	1387.1	1143.4	876.1	**672.4**	908.9	754.1
CR-0.2/0.05	750	14116	1096.0	2105.4	1460.0	1153.1	**877.7**	1073.6	1001.0
CR-0.2/0.05	1000	25485	1273.4	1983.4	1701.5	1374.2	**1082.2**	1256.4	1188.2
CR-0.3/0.1	100	547	311.5	210.3	315.6	144.4	**122.7**	234.1	165.4
CR-0.3/0.1	250	3214	357.6	466.3	448.6	285.0	**256.3**	346.4	271.4
CR-0.3/0.1	500	12439	512.2	1002.7	715.7	536.3	**450.2**	493.9	480.7
CR-0.3/0.1	750	28456	681.8	1207.1	993.1	733.5	**652.5**	667.8	664.4
CR-0.3/0.1	1000	50593	843.1	1505.5	1304.2	953.7	862.0	833.1	**833.0**
RG-0.15	100	361	10.2	435.5	87.7	19.9	19.1	7.0	**6.0**
RG-0.15	250	1922	85.0	4346.4	6731.9	1802.2	1612.8	**61.4**	65.4
RG-0.15	500	7583	783.1	9589.2	11885.8	3240.7	3451.7	245.5	**201.1**
RG-0.15	750	16832	4955.8	10597.6	11698.9	5100.7	4217.1	564.0	**523.1**
RG-0.15	1000	30864	26189.3	14003.6	15380.8	6636.3	6136.9	1243.1	**1190.8**
RG-0.3	100	956	35.5	631.6	101.4	235.0	238.4	34.3	**27.9**
RG-0.3	250	6989	1087.8	1040.7	1328.1	427.0	**410.3**	414.3	445.5
RG-0.3	500	26714	898.0	2245.1	3602.9	1041.4	987.8	**449.8**	532.6
RG-0.3	750	60147	1796.4	3756.6	3530.8	1737.8	**1424.3**	1591.7	1736.3
RG-0.3	1000	108433	1822.0	3937.3	5167.8	2301.4	1923.4	1822.0	**1784.3**
FF-0.35/0.32	100	203	**2.0**	6.0	4.0	6.0	4.0	**2.0**	**2.0**
FF-0.35/0.32	250	642	**4.0**	316.7	539.7	63.0	8.0	**4.0**	**4.0**
FF-0.35/0.32	500	1618	4.0	14205.8	25902.9	3305.9	16.0	**2.0**	**2.0**
FF-0.35/0.32	750	2403	**2.0**	22426.3	22353.9	5407.4	14.0	**2.0**	**2.0**
FF-0.35/0.32	1000	2758	4.0	110595.0	162119.0	15020.6	20.0	**2.0**	**2.0**
FF-0.45/0.35	100	1630	554.1	364.4	490.1	115.3	**115.0**	450.6	120.5
FF-0.45/0.35	250	12875	1618.1	1111.3	1823.9	362.2	**208.1**	1618.1	265.7
FF-0.45/0.35	500	53618	5191.2	4256.3	4062.4	1285.4	**629.5**	5191.2	1262.0
FF-0.45/0.35	750	119604	5423.8	7113.9	5931.6	2184.6	**1522.2**	5184.3	1868.9
FF-0.45/0.35	1000	195466	11413.8	12919.8	10374.1	4195.2	**2720.8**	11413.8	3812.0
BA-4	100	384	272.0	1244.5	1368.0	496.1	411.8	86.0	**56.0**
BA-4	250	984	406.3	6945.2	3207.2	1870.8	1402.2	324.7	**308.0**
BA-4	500	1984	576.0	33031.3	13829.8	7126.3	4404.3	**218.0**	378.0
BA-4	750	2984	473.0	93812.5	20827.5	15329.3	13151.3	218.0	**214.0**
BA-4	1000	3984	346.0	161175.0	33607.9	24100.5	27930.7	**266.0**	294.0
BA-5	100	475	413.8	960.8	697.2	324.8	332.9	305.1	**259.8**
BA-5	250	1225	6926.0	9519.8	2829.3	1812.5	1388.1	2019.9	**1240.5**
BA-5	500	2475	5515.3	27255.6	8634.5	6659.3	4436.2	**3035.4**	3772.9
BA-5	750	3725	14472.3	80680.4	17713.9	16419.6	9897.8	9445.6	**6040.4**
BA-5	1000	4975	18283.0	92681.6	32236.4	28279.7	19636.9	16752.6	**7521.8**
WS-4	100	200	**5.3**	30.7	70.0	12.0	12.0	**5.3**	**5.3**
WS-4	250	500	11.7	6233.5	10701.2	1941.9	1285.0	**10.0**	**10.0**
WS-4	500	1000	12.0	21067.5	23086.3	6699.8	736.7	**10.0**	12.0
WS-4	750	1500	12.0	35557.8	47199.6	12383.1	2028.8	9.0	**8.0**
WS-4	1000	2000	15.3	42949.3	71638.9	23811.0	3894.9	**12.0**	**12.0**
WS-5	100	200	16.2	228.2	770.0	126.7	42.0	**8.0**	11.0
WS-5	250	500	12.0	4940.2	17904.8	1622.5	455.5	9.3	**8.0**
WS-5	500	1000	11.7	23205.8	31455.9	5309.7	759.3	**8.0**	10.0
WS-5	750	1500	**10.3**	32910.8	51790.1	13023.6	1764.3	**10.3**	**10.3**
WS-5	1000	2000	12.0	61587.5	51855.5	23844.9	2107.5	**10.3**	11.7

Table A.11: Results of the algorithms for $k = 0.3n$

Insts.	n	m	*SBA*	*CNP1*	*MBA*	*SGA*	*SA*	*VNS*	*ABC*
ER-0.05	100	220	**0.0**	10.0	4.0	30.0	12.0	**0.0**	**0.0**
ER-0.05	250	1497	1484.8	1307.9	960.8	417.4	**310.7**	744.0	444.3
ER-0.05	500	6199	978.1	1479.5	1182.0	768.4	**478.9**	901.4	608.0
ER-0.05	750	14182	1071.6	1659.1	1418.2	985.6	**638.9**	1042.7	833.6
ER-0.05	1000	25134	1175.9	2143.7	1711.9	1155.4	**787.5**	1169.7	991.8
ER-0.2	100	976	153.4	128.6	187.5	65.5	**55.7**	110.6	63.4
ER-0.2	250	6291	173.6	264.8	266.6	147.2	**128.0**	160.9	141.8
ER-0.2	500	24966	285.6	586.9	507.6	289.0	**249.6**	278.0	266.5
ER-0.2	750	56064	392.4	864.4	723.0	425.3	**370.4**	388.3	378.1
ER-0.2	1000	100292	508.6	1094.7	993.0	561.6	497.0	503.2	**494.5**
CR-0.2/0.05	100	250	2.0	16.0	46.0	156.0	20.0	**0.0**	**0.0**
CR-0.2/0.05	250	1605	783.8	783.4	1016.0	460.4	**275.6**	620.5	299.8
CR-0.2/0.05	500	6409	911.8	1449.9	1074.7	726.6	**477.1**	871.5	624.6
CR-0.2/0.05	750	14116	1080.2	2036.0	1449.8	960.3	**626.9**	1041.3	845.3
CR-0.2/0.05	1000	25485	1191.2	1991.3	1614.1	1098.4	**783.7**	1149.0	983.1
CR-0.3/0.1	100	547	342.7	217.9	450.1	117.5	**83.8**	180.9	90.3
CR-0.3/0.1	250	3214	401.7	460.1	451.1	239.3	**177.3**	339.9	195.4
CR-0.3/0.1	500	12439	471.6	992.2	690.0	429.6	**318.1**	448.2	355.7
CR-0.3/0.1	750	28456	538.8	1165.1	968.3	556.5	**458.8**	532.2	494.9
CR-0.3/0.1	1000	50593	658.6	1496.2	1236.8	733.5	**597.0**	644.6	619.2
RG-0.15	100	361	**0.0**	16.0	8.0	10.0	**0.0**	**0.0**	**0.0**
RG-0.15	250	1922	16.8	3798.9	7340.6	995.1	414.1	4.0	**3.4**
RG-0.15	500	7583	63.1	9670.6	14938.3	2332.1	2045.2	19.5	**14.3**
RG-0.15	750	16832	114.8	11250.0	14073.2	3697.7	2445.6	68.7	**48.9**
RG-0.15	1000	30864	162.1	14089.9	16213.5	4607.8	3552.2	**98.5**	106.1
RG-0.3	100	956	24.3	515.6	60.0	127.0	24.7	6.2	**0.7**
RG-0.3	250	6989	134.7	908.5	2004.2	301.5	210.3	38.7	**15.8**
RG-0.3	500	26714	154.4	2178.9	3388.3	684.6	491.0	36.7	**26.1**
RG-0.3	750	60147	398.1	3536.7	4223.7	1140.3	734.6	357.9	**301.6**
RG-0.3	1000	108433	544.5	3946.1	6513.6	1527.0	1089.7	**482.6**	501.5
FF-0.35/0.32	100	203	**0.0**	2.0	**0.0**	2.0	**0.0**	**0.0**	**0.0**
FF-0.35/0.32	250	642	**0.0**	77.3	73.0	31.4	2.0	**0.0**	**0.0**
FF-0.35/0.32	500	1618	**0.0**	6760.4	10743.3	947.5	4.0	**0.0**	**0.0**
FF-0.35/0.32	750	2403	**0.0**	14490.8	17685.0	1340.3	6.0	**0.0**	**0.0**
FF-0.35/0.32	1000	2758	**0.0**	128216.0	88842.7	626.7	10.0	**0.0**	**0.0**
FF-0.45/0.35	100	1630	**4.0**	285.2	12.0	71.0	10.9	**4.0**	**4.0**
FF-0.45/0.35	250	12875	649.9	1921.5	1393.2	198.5	**3.6**	168.8	178.0
FF-0.45/0.35	500	53618	1621.2	3600.2	3458.4	682.7	**117.8**	1563.7	422.5
FF-0.45/0.35	750	119604	5018.2	7391.6	5844.7	1249.0	**288.7**	2216.2	666.2
FF-0.45/0.35	1000	195466	238.0	12879.3	10279.0	2161.8	965.8	**126.0**	234.0
BA-4	100	384	**0.0**	10.0	2.0	244.8	151.8	**0.0**	**0.0**
BA-4	250	984	**0.0**	7085.8	2948.7	1178.8	623.8	**0.0**	**0.0**
BA-4	500	1984	**0.0**	31590.3	9124.5	3969.3	2265.2	**0.0**	**0.0**
BA-4	750	2984	**0.0**	96303.5	20020.9	8317.4	4308.6	**0.0**	**0.0**
BA-4	1000	3984	**0.0**	144535.0	32658.2	14639.1	5924.0	**0.0**	**0.0**
BA-5	100	475	8.0	715.4	40.0	217.0	154.3	**2.0**	4.0
BA-5	250	1225	4.0	7720.8	3187.2	1131.4	621.1	**2.0**	**2.0**
BA-5	500	2475	**2.0**	25267.3	9104.5	3717.5	1657.9	**2.0**	**2.0**
BA-5	750	3725	**2.0**	81924.6	12555.3	8520.6	3677.3	**2.0**	**2.0**
BA-5	1000	4975	**2.0**	90017.7	31140.6	15473.7	6984.7	**2.0**	**2.0**
WS-4	100	200	1.0	72.0	6.0	6.0	6.0	**0.0**	**0.0**
WS-4	250	500	1.0	3944.1	4752.8	16.0	4.0	**0.0**	**0.0**
WS-4	500	1000	2.0	19255.9	41862.3	50.0	12.0	**0.0**	**0.0**
WS-4	750	1500	1.0	30757.8	57718.7	56.0	12.0	**0.0**	**0.0**
WS-4	1000	2000	2.0	43955.1	53824.2	48.0	12.0	**1.0**	**1.0**
WS-5	100	200	2.0	8.0	8.0	8.0	1.0	**0.0**	**0.0**
WS-5	250	500	2.0	3517.8	566.4	30.0	4.0	**0.0**	**0.0**
WS-5	500	1000	1.0	20139.6	46769.3	64.0	8.0	**0.0**	1.0
WS-5	750	1500	**1.0**	37272.8	50403.6	70.0	12.0	**1.0**	**1.0**
WS-5	1000	2000	1.0	63466.4	84441.4	118.0	12.0	**0.0**	1.0

Table A.12: Results of the algorithms for $k = 0.5n$