A General Evolutionary Framework for different classes of Critical Node Problems

R. Aringhieri, A. Grosso, P. Hosteins and R. Scatamacchia.

A general Evolutionary Framework for different classes of Critical Node Problems.

Engineering Applications of Artificial Intelligence, 55: 128–145, 2016. Available online 7 July 2016.

DOI: 10.1016/j.engappai.2016.06.010

A General Evolutionary Framework for different classes of Critical Node Problems

Roberto Aringhieri[a], Andrea Grosso[a], Pierre Hosteins[a,*], Rosario Scatamacchia[b]

[a] *Università degli Studi di Torino*
Dipartimento di Informatica
Corso Svizzera, 185 - 10149 Torino, Italy
[b] *Politecnico di Torino*
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24 - 10129 Torino, Italy

Abstract

We design a flexible Evolutionary Framework for solving several classes of the Critical Node Problem (CNP), i.e. the maximal fragmentation of a graph through node deletion, given a measure of connectivity. The algorithm uses greedy rules in order to lead the search towards good quality solutions during reproduction and mutation phases. Such rules, which are only partially reported in the literature, are generalised and adapted to the six different formulations of the CNP considered along the paper. The link between solutions of different CNP formulations is investigated, both quantitatively and qualitatively. Furthermore, we provide a comparison with best known results when those are available in literature, that confirms the good overall quality of our solutions.

Keywords: Evolutionary algorithm, Critical Node Problem, graph fragmentation, greedy rules, connectivity measures.

*Corresponding author
Email addresses: roberto.aringhieri@unito.it (Roberto Aringhieri), andrea.grosso@unito.it (Andrea Grosso), hosteins@di.unito.it (Pierre Hosteins), rosario.scatamacchia@polito.it (Rosario Scatamacchia)

1. Introduction

The Critical Node Problem (CNP) is a class of Interdiction Network Problems (Wollmer, 1964; Wood, 1993) that focuses on maximally fragmenting a graph $G(V, E)$ by deleting a set $S \subset V$ of its nodes (and all incident edges on such nodes). This problem is of interest in a wide range of possible situations, including the identification of key players in a social network (Borgatti, 2006), transportation networks' vulnerability (Jenelius et al., 2006), power grid construction and vulnerability (Salmerón et al., 2004), homeland security (Brown et al., 2006), telecommunications (Alevras et al., 1997) or epidemic control (Zhou et al., 2006) and immunisation strategies (Arulselvan et al., 2009; Cohen et al., 2003; Ventresca, 2012). A possible application to computational biology, through the example of protein-protein interaction networks, has been suggested in Boginski and Commander (2009).

Each domain of application usually defines a specific version of the problem through the use of a particular connectivity measure. Moreover, solving real graphs with up to thousands of nodes often calls for the use of an efficient heuristic algorithm. The contribution of the approach advocated here is twofold: on one hand, it provides a global and flexible framework that allows us to deal with different fragmentation measures. On the other hand, it can find good quality solutions with limited costs in terms of algorithmic implementation and computational effort. To the best of the authors' knowledge, this is the first attempt to develop a general tool for tackling different classes of the CNP.

We will represent a solution by the set of its deleted nodes S. The degree of fragmentation of the induced graph $G[V \setminus S]$ needs to be measured by a given connectivity metric. We will consider only undirected graphs and we denote the set of maximal connected components as $\mathcal{H}$ and the cardinality of the said components as $|h|$ for $h \in \mathcal{H}$.

Many connectivity measures can be devised according to the type of application desired. We will concentrate on the measures that take into account the number of remaining connected components and their cardinality after the deletion of set S, which is generally enough to determine which nodes are still able to interact in the remaining network. These measures are defined as (i) pair-wise connectivity, i.e. the number of pair of nodes connected by a path inside the graph, (ii) the size of the largest connected component and (iii) the number of connected components. The value of these three measures for a solution set S will be expressed, respectively, through the

following mathematical functions:

$$f(S) = |\{i, j \in V \setminus S : i \text{ and } j \text{ connected by a path in } G[V \setminus S]\}|, \quad (1)$$
$$C(S) = \max\{|h|, h \in \mathcal{H}(G[V \setminus S])\}, \quad (2)$$
$$H(S) = |\mathcal{H}(G[V \setminus S])|. \quad (3)$$

Pair-wise connectivity $f(S)$ can alternatively be expressed in terms of the cardinality of the maximal connected components: $f(S) = \sum_{h \in \mathcal{H}} \frac{|h|(|h|-1)}{2}$. Even though these measures are all different and can lead to very different optimal solutions, as explicitly demonstrated in Shen and Smith (2012), they are not generally unrelated. For example the ideal situation for minimising the pair-wise connectivity is to obtain the largest number of connected components $H(S)$ with the smallest possible variance in their cardinality. This implies a minimisation of the size of the largest component. In practice, this means that disrupting pair-wise connectivity $f(S)$ is a tradeoff between minimising the cardinality of the largest component $C(S)$ and maximising the number of connected components $H(S)$. As the nodes are removed or disabled, we do not count them as single components. An example of the fragmentation of a small graph is provided in Fig. 1: after the removal of two nodes (number 1 and 2), the graph is split into two connected components of five nodes each. This solution corresponds to the optimal solution when trying to either minimise $f(S)$ and $C(S)$ or maximise $H(S)$ by removing at most two nodes from the graph, with corresponding values: $f(\{1, 2\}) = 20$, $C(\{1, 2\}) = 5$ and $H(\{1, 2\}) = 2$.

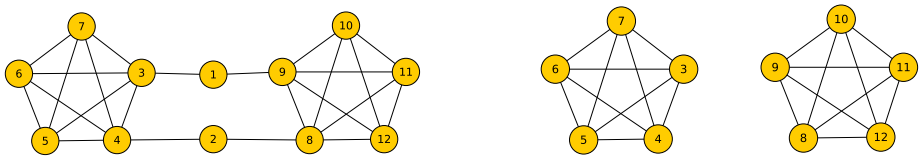

Figure 1: Example of a small graph (on the left) fragmented into two connected components (on the right) after the removal of nodes 1 and 2.

Given a connectivity measure, a CNP solution is defined by the set of deleted nodes and the value of the connectivity metric for the resulting graph. Depending on the problem at hand, the selection of the nodes can be performed using two complementary approaches:

- the budget constrained formulation: minimise/maximise the connectivity under a budget limitation over S ($|S| \leq K$);
- the connectivity constrained formulation: minimise the number of nodes deleted ($|S|$) in order to to meet a threshold connectivity value.

For the sake of clarity, we will refer to the problems with the different connectivity measures $f(S)$, $C(S)$ and $H(S)$ as CNP1, CNP2 and CNP3, respectively. For each problem, we consider the two variants of the CNP that arise taking into account both the budget ("a") and connectivity ("b") constrained formulations, that is

- CNP1a: minimise $f(S)$ (pair-wise connectivity) subject to $|S| \leq K$.
- CNP1b: minimise $|S|$ such that $f(S) \leq P$.
- CNP2a: minimise $C(S)$ (cardinality of the largest connected component of $G[V \setminus S]$) subject to $|S| \leq K$.
- CNP2b: minimise $|S|$ such that $C(S) \leq L$ (L denotes the cardinality parameter in accordance with notations in Boginski and Commander (2009); Arulselvan et al. (2011); Veremyev et al. (2014a)).
- CNP3a: maximise $H(S)$ (number of connected components of $G[V \setminus S]$) subject to $|S| \leq K$.
- CNP3b: minimise $|S|$ such that $H(S) \geq N$.

In this paper we will consider the 6 different types of the CNP problem accordingly to the above taxonomy. Handling each of these formulations through the use of a single algorithmic framework is not straightforward. For instance, the VNS algorithm provided in Aringhieri et al. (2016b) for CNP1a, which provides good results compared to other heuristics for that problem, is hard to generalise even to the CNP1b. One main reason is the fact that finding feasible solutions for "b" types of the CNP is potentially very difficult, posing a relevant challenge for implementing the classical shaking procedures in a VNS framework and in general for the exploration of the solution space. Another important difficulty concerns the application of local search approaches. In order to improve the objective value of an instance of CNP1b, a local search procedure should involve a swap of a node from $V \setminus S$ with at least two nodes from S, which would increase the complexity of a

move by a factor $K/2$ compared to the "a" version (more details about local search procedures for the CNP are provided in Section 3.5). Furthermore, the development of efficient neighbourhoods is also challenging, as discussed in Aringhieri et al. (2016b).

We will demonstrate how our Evolutionary Framework (EF) can tackle any of the six problems above by using tailored reproduction and mutation operators capable of repairing the solutions through appropriate greedy rules (preliminary results of such a framework can be found in Aringhieri et al. (2016a)). Such rules can effectively guide the search through the solution space, in particular when they are properly combined as pointed out by the previous work of Addis et al. (2016).

Based on the considerations above, the aim of this work is to provide a simple and easy to implement algorithmic framework that can tackle many different versions of the CNP by embedding suitable and efficient greedy rules.

	type "a"	type "b"
CNP1	Greedy algorithms (Arulselvan et al. (2009), Ventresca and Aleman (2015a), Addis et al. (2016)) Simulated Annealing and PBIL (Ventresca, 2012) VNS and ILS (Aringhieri et al., 2016b)	Approximation algorithm (Dinh et al., 2010)
CNP2	-	Greedy and Genetic algorithm (Boginski and Commander (2009), Arulselvan et al. (2011))
CNP3	-	-

Table 1: Heuristic algorithms from the literature for the six types of the CNP considered in this work.

Table 1 reports the main heuristic algorithms in the literature for the different types of the CNP considered. CNP1a has gained more attention, while there exists a gap in the literature for the other five versions. We further extend the analysis of the CNP to these versions and propose a set of benchmark results which may constitute an interesting basis for comparison for future algorithms.

The paper is organized as follows. Section 2 introduces the greedy rules adopted as well as some greedy algorithms that will be used for comparison. Section 3 describes a general evolutionary algorithm for the different types of the CNP as defined above, embedding the greedy rules defined in Section 2 within the tailored reproduction and mutation operators. Section 4 discusses

the results of the evolutionary algorithm over a set of benchmark instances and investigates the correlation between solutions of the different types of the CNP. Finally Section 5 provides conclusions and remarks. The remainder of this section gives a brief overview of the existing literature on the CNP.

A pseudo-approximation algorithm is proposed in Dinh et al. (2010) to select a set of nodes S whose deletion will lower pair-wise connectivity under a certain threshold (CNP1b). The minimisation of pair-wise connectivity through the deletion of K nodes (CNP1a) is investigated in numerous works. Its NP-completeness is proved in Arulselvan et al. (2009), Di Summa et al. (2011) and Addis et al. (2013) while Arulselvan et al. (2009) also proposes a *greedy algorithm* and an ILP formulation (with $\mathcal{O}(|V|^3)$ constraints). A generally more efficient ILP formulation with a potentially non polynomial number of constraints is presented in Di Summa et al. (2012) while the more recent work of Veremyev et al. (2014a) proposes an alternative more compact formulation with only $\mathcal{O}(|V|^2)$ constraints. Several heuristic algorithms exist, based on the use of greedy rules (Ventresca and Aleman, 2015a; Addis et al., 2016) with interesting results or metaheuristic methods, such as Simulated Annealing and Population Based Incremental Learning (Ventresca, 2012) or Iterated Local Search and Variable Neighbourhood Search (Aringhieri et al., 2015, 2016b). Approximation algorithms have also been proposed (Ventresca and Aleman, 2014a,b) but with limited applicability since they are based on an ILP formulation with $\mathcal{O}(|V|^3)$ constraints. Polynomiality of the CNP1a over trees is established in Di Summa et al. (2011) and extended to graphs with bounded tree-width (and to the CNP2a and the CNP3a) in Addis et al. (2013).

The CNP2b has been introduced in Boginski and Commander (2009) and Arulselvan et al. (2011): it seeks the smallest set S inducing a graph $G[V \setminus S]$ whose largest component is smaller than a given threshold L ($C(S) \leq L$). An ILP formulation is given along with a *greedy algorithm* very similar to the one of Arulselvan et al. (2009) and a genetic algorithm. A more compact linear model is proposed in Veremyev et al. (2014a) and polynomiality of the CNP2b over trees and proper interval graphs is established in Lalou et al. (2015). The works of Shen and Smith (2012) and Shen et al. (2012) study the versions where a set of nodes with maximum cardinality $|S| \leq K$ is deleted to respectively minimise the size of the largest connected component $C(S)$ (CNP2a) or maximise the number of connected components $H(S)$ (CNP3a). Exact algorithms are proposed for both versions, including dynamic programming approaches as well as ILP models. Contrary to the version that consid-

ers pair-wise connectivity minimisation, few efficient heuristics to tackle real world graph instances have been developed in the literature for other connectivity measures. We note that a very general mathematical model for dealing with Critical Node-Edge Deletion Problems has been proposed in Veremyev et al. (2014b), providing a linear formulation for any connectivity measure that uses the number of connected components and their size, with a limited number of variables and constraints ($\mathcal{O}(|V|^2)$). As a side note, it can be observed that formulations CNP2*b* and CNP3*b* have strong ties to the Vertex Separator Problem (Balas and de Souza, 2005), which in its simplest form seeks to find the smallest possible set of nodes whose removal fragments the graph in two balanced components. Recent examples of exact and heuristic approaches for the Vertex Separator Problem can be found in Cavalcante and de Souza (2011) and Sánchez-Oro et al. (2016).

To summarize, the literature does provide heuristic algorithms able to tackle real graphs with up to thousands of nodes but only for two out of the six CNP types defined above (CNP1*a*, CNP2*b*). At the same time, the applicability of the ILP models is in general limited to small graphs. Although we will focus in this paper on the three connectivity metrics detailed above, we remark that many alternative ways to quantify a graph's fragmentation can be used, for example: the network's diameter (Albert et al., 2000), single/multiple-commodity maximum flow or the shortest path between given source-sink node pairs (Grubesic and Murray, 2006; Matisziw and Murray, 2009; Cormican et al., 1998; Lim and Smith, 2007; Veremyev et al., 2015).

2. Greedy Rules and Algorithms

The evolutionary framework we propose relies on the use of suitable greedy rules with the aim of providing an efficient and flexible tool for dealing with the different types of CNP. These greedy rules are embedded in the initialization, reproduction and mutation phases of the evolutionary algorithm. For each type of CNP, we will discuss two complementary types of greedy rules that allow us to generate heuristic solutions quickly. These rules are based on moving a node from the set S of deleted nodes to the remaining graph $V \setminus S$, or conversely, from $V \setminus S$ to S, depending on the characteristics of the current solution. Depending on the CNP version one wants to tackle, a set of nodes is selected to maximise or minimise a relevant connectivity criterion and a full greedy procedure is devised using one of these rules for

computing a feasible solution. We will adopt here the two greedy approaches defined in Addis et al. (2016) (referenced there as *Greedy1* and *Greedy2*) as a basis.

For each CNPnx problem, we propose a greedy rule which identifies nodes that can be deleted from S while minimally worsening the objective function — called GR1(S, n, x) — and a second rule that identifies nodes that can be added to S while maximally improving the objective function — called GR2(S, n, x). It often happens that a rule is the same for $x = a$ and $x = b$, in which case we do not explicitly define it for case $x = b$. Based on these rules we can devise two types of *greedy algorithm* for each version of the CNP. These algorithms will be referred to as $G_1^{(nx)}$ and $G_2^{(nx)}$ and will have the basic structures shown in Algorithms 1, 2, 3 and 4.

Algorithm 1: $G_1^{(na)}$

Data: Graph: G, K
Result: S
$S :=$ Vertex Cover(G);
while $|S| > K$ **do**
  $B := \{\text{GR1}(S, n, a)\}$;
  $S := S \setminus \{\text{Select}(B)\}$;

Algorithm 2: $G_2^{(na)}$

Data: Graph: G, K
Result: S
$S := \{\emptyset\}$;
while $|S| < K$ **do**
  $B := \{\text{GR2}(S, n, a)\}$;
  $S := S \cup \{\text{Select}(B)\}$;

Algorithm 3: $G_1^{(nb)}$

Data: Graph: G, connectivity parameter P, L or N
Result: S
$S =$ Vertex Cover(G);
$B := \{\text{GR1}(S, n, b)\}$;
$u :=$ Select(B);
while $S \setminus \{u\}$ *satisfies the connectivity constraint* **do**
  $S := S \setminus \{u\}$;
  $B := \{\text{GR1}(S, n, b)\}$;
  $u :=$ Select(B);

The philosophy of Algorithms 1 and 3 is to start from a relaxed solution where the residual graph that has no induced edges — no pairwise communication, a maximum component size of 1, and $|V| - |S|$ components — and

Algorithm 4: $G_2^{(nb)}$

Data: Graph: G, connectivity parameter P, L or N
Result: S

$S := \{\emptyset\}$;
$B := \{\text{GR2}(S, n, b)\}$;
$v := \text{Select}(B)$;
while $S \cup \{v\}$ *violates the connectivity constraint* **do**
    $S := S \cup \{v\}$;
    $B := \text{GR2}(S, n, b)$;
    $v := \text{Select}(B)$;
$B := \{\text{GR1}(S, n, b)\}$;
$u := \text{Select}(B)$;
while $S \setminus \{u\}$ *satisfies the connectivity constraint* **do**
    $S := S \setminus \{u\}$;
    $B := \text{GR1}(S, n, b)$;
    $u := \text{Select}(B)$;

then moving nodes from S back to $V \setminus S$ so as to keep the connectivity as close to optimality as possible. While Arulselvan et al. (2009), Boginski and Commander (2009) and Arulselvan et al. (2011) use a deterministic vertex cover solution S that is defined from an input node $i \in V$, we use a heuristic procedure that shuffles the nodes at each run of the algorithm as in Addis et al. (2016). Likewise, we adopt a different strategy for breaking ties. The nodes with the best possible impact identified at each iteration are stored in the set B. While the previous approaches select the first node in the set, we break ties by selecting at random a node inside B through the function *Select()*, as was proposed in Addis et al. (2016).

The philosophy of Algorithms 2 and 4 is to start from the set of nodes V and move nodes to set S, optimising the connectivity as much as possible at each step. Note that for $G_2^{(nb)}$, we have to satisfy a constraint on the connectivity measure itself: it is possible that in trying to cope with such a constraint we end up moving nodes to S that reveal themselves as unnecessary choices once we have reached feasibility, this is why it is wise to use rule $\text{GR1}(S, n, b)$ at the end of the process (i.e. when the solution is feasible) to reduce $|S|$ as much as possible. The necessity of this additional phase is

illustrated on the small graph introduced in Fig. 1: Consider the case of the CNP3b version with $N = 2$. Since no single node is an articulation point (i.e. a node whose removal from the graph splits it in two or more connected subcomponents), too many nodes could be blindly removed at random over the whole set $V \setminus S$ until the number of connected components is at least two. Nevertheless it is easy to see that just deleting nodes 1 and 2 would provide an optimal solution. Consider in Fig. 2 a situation where GR2$(S, 3, b)$ chooses at random nodes 7, 10 and 1 before finally selecting node 2. The use of GR1$(S, 3, b)$ allows us to reintroduce nodes 7 and 10 in the graph and to reduce $|S|$ from 4 to 2.

Figure 2: Illustrative example for the CNP3b (with $N = 2$) where using GR2$(S, 3, b)$ in the graph in Fig. 1 deletes unnecessary nodes, namely nodes 7 and 10 (on the left). The sequential use of GR1$(S, 3, b)$ right after reintroduces nodes 7 and 10 in the graph thus yielding an optimal solution (on the right).

The remainder of the section is devoted to a detailed description of the greedy rules considered. We adopt the following notation, referring to the subgraph $G[V \setminus S]$ induced by a partial solution S. We refer to a connected component of a graph by considering only its node set $h \in \mathcal{H}$. For a node $t \in V \setminus S$, we denote by $h(t)$ the connected component of subgraph $G[V \setminus S]$ that includes node t; we denote by $\chi(t)$ the collection of connected components in which $h(t)$ decomposes if node t is deleted. For a node $t \in S$, we denote by $h'(t)$ the connected component to which t belongs in subgraph $G[(V \setminus S) \cup \{t\}]$, i.e. after node t has been added back to the graph; for a node $t \in S$, we denote by $\chi'(t)$ the set of connected components of $G[V \setminus S]$ that merge into a single component $h'(t)$ when node t is added back to the graph.

2.1. Greedy rules for the CNP1

We adopt here the greedy rule GR1$(S, 1, a)$ proposed in Arulselvan et al. (2009) and Addis et al. (2016):

$$\text{GR1}(S, 1, a) := \arg\min\{f(S \setminus \{t\}) - f(S) \colon t \in S\}. \tag{4}$$

An efficient implementation of Greedy Rule 1 looks at all neighbours of the nodes in S in order to determine the connected components that can be joined together and the size of the new component. If the connected components of the residual graph are stored, this requires at most $\mathcal{O}(|E|)$ operations; after the best node has been chosen and added back to $V \setminus S$, $G[V \setminus S]$ is explored to update the connected components. The complexity of identifying and/or merging components does not exceed $\mathcal{O}(|V| + |E|)$.

The second greedy rule, say $\mathrm{GR2}(S, 1, a)$, for this problem is logically defined in a symmetric way:

$$\mathrm{GR2}(S, 1, a) := \arg\max\{f(S) - f(S \cup \{t\}) \colon t \in V \setminus S\}. \tag{5}$$

Greedy Rule 2 can be implemented through a Depth-First Search (DFS) exploration for each connected component, using rules to track articulation points and the impact of their removal on the graph. Therefore, it has an overall complexity of $\mathcal{O}(|V| + |E|)$.

When we want to tackle the CNP1*b*, where a limit on the maximum pairwise connectivity is imposed, the same greedy rules as (4) and (5) can be used. By using (4) for $G_1^{(1b)}$, we ensure that each node reintroduced will raise the connectivity by the smallest possible amount, which leaves the potential for more nodes to be deleted from S until we reach the maximum allowed connectivity value P. Conversely, by using (5) for $G_2^{(1b)}$, we try to lower the connectivity as much as possible with each deleted node and reach the value $f(S) \leq P$ with the lowest possible value for $|S|$.

2.2. Greedy rules for the CNP2

There are important differences between the CNP1 and the CNP2: while moving a node from S to $V \setminus S$, or vice versa, always has an impact on the pairwise connectivity objective ($f(S) - f(S \cup \{v\}) > 0$ and $f(S \setminus \{u\}) - f(S) > 0$ for any $v \in V \setminus S$ or $u \in S$ respectively), it is no longer the case when one considers the maximum cardinality of the connected components. In fact, it is very difficult to recognize what is going to be the largest component in the final solution by considering one node at a time, especially when a very large set S is considered. Consequently, one will have to make many choices whose impact is not quantifiable at the moment they are made. For example the application of the greedy approach described by Boginski and Commander (2009) and Arulselvan et al. (2011) for the CNP2*b* blindly moves nodes from a vertex cover S of $G(V)$ back to the graph $G[V \setminus S]$ until it is no longer

possible to satisfy the constraint on the maximum cardinality of connected components. This strategy can be improved by considering an additional criterion.

For the CNP2a, the logic is similar to the CNP1a: a node t is candidate to be moved from S to $V \setminus S$ if it will belong to the connected component of $G' = G[(V \setminus S) \cup \{t\}]$ that has minimum size.

$$\mathrm{GR1}(S, 2, a) := \arg\min\{|h'(t)| \colon t \in S\} \tag{6}$$

A "symmetric" greedy rule that chooses candidate nodes to be moved from $V \setminus S$ into S can be devised as follows. We consider only nodes belonging to connected components of $G[V \setminus S]$ with maximum cardinality $C(S)$, and try to minimize the size of the largest resulting component in $\chi(t)$.

$$\mathrm{GR2}(S, 2, a) := \arg\min \Big\{ \max\{|\omega| \colon \omega \in \chi(t)\} \colon |h(t)| = C(S),\, t \in V \setminus S \Big\}. \tag{7}$$

When considering the CNP2b, the solutions have to satisfy a constraint L on the size of each connected component in the residual graph. The greedy rule (6) can be used equally to define $\mathrm{GR1}(S, 2, b)$: as each node which is removed from S will be integrated into the smallest possible component, we will avoid the premature reappearance of a component which is much larger than the others. When such a component exists, it will more frequently contain a neighbour of any given node $u \in S$: therefore, it could link to other components more easily and violate the maximum cardinality constraint. The greedy process could then stop when $|S|$ is still much larger than the optimal value.

Adapting greedy rule $\mathrm{GR2}(S, 2, a)$ to the connectivity constrained case is more involved. Deleting a node t belonging to a component $h(t)$ having maximum size in $G[V \setminus S]$ is an obvious greedy choice, but we do not want such a choice to be completely blind with respect to the constraint on the component size. We formulate the rule $\mathrm{GR2}(S, 2, b)$ so that it considers the decrease of the size of the components:

$$\mathrm{GR2}(S, 2, b) := \arg\max \Big\{ |h(t)| - \max\{|\omega| \colon \omega \in \chi(t)\} \colon t \in V \setminus S \Big\}, \tag{8}$$

and we break ties with the use of a secondary objective, trying to maximise the number of components among those in the collection $\chi(t)$ which will satisfy the size constraint. This secondary objective can be written as: $\max\{|\{\omega \in \chi(t) \colon |\omega| \leq L\}|\}$ where the argument t runs over nodes in $V \setminus S$ which provide a maximal objective as defined in Eq. (8).

2.3. Greedy rules for the CNP3

When trying to reduce a set S with high cardinality from a CNP3 perspective, we wish to maintain the highest possible number of connected components while reducing $|S|$. We introduce a greedy rule that considers a node t as a candidate to be moved from S to $V \setminus S$ when that node is adjacent a minimum collection of connected components $\chi'(t)$ that will merge into a single one. The greedy rule $\mathrm{GR1}(S, 3, a)$ is then formulated as:

$$\mathrm{GR1}(S, 3, a) = \arg\min\{|\chi'(t)| \colon t \in S\} \tag{9}$$

Whenever a tie arises between several nodes, the node or nodes that integrate into the smaller component is chosen, so as to avoid creating large components early in the greedy procedure: this is mainly because a large component will connect more easily to another when other nodes are added afterwards.

One difficulty in dealing with the CNP3 is due to the fact that starting from a vertex cover to obtain a graph with no edge does not automatically provide the most satisfying connectivity. It is possible that another choice for the initial set S could provide a larger number of nodes in $V \setminus S$ and thus a larger number of components. Actually for the CNP3a a minimum vertex cover S with cardinality $|S| \leq K$ would yield indeed an optimal solution, with the optimum number of components equal to $|V| - |S|$. Hence a vertex cover is not *a priori* a bad choice for disconnecting the graph, however finding a minimum vertex cover is very demanding in practice. For the CNP3b, a minimum vertex cover would provide an upper bound on the value of N (i.e. the cardinality of a maximum independent set). At the same time, even a minimum vertex cover is not guaranteed to be optimal for the problem. It could be possible to move back in the graph one or more of its nodes (thus lowering the objective function) without violating the constraint on the number of components. Therefore the choice of the vertex cover as initial set for the greedy procedure that will use Greedy Rule (9) can already limit drastically the quality of the solution. Being aware of that situation, we can anyway devise a greedy procedure as described in Algorithm 1.

The second greedy rule for the CNP3, say $\mathrm{GR2}(3, a)$, would consider moving a node t from $V \setminus S$ to S if such a move maximised the number of components $|\chi(t)|$ obtained from the fragmentation of the component $h(t)$.

$$\mathrm{GR2}(S, 3, a) := \arg\max\{|\chi(t)| \colon t \in V \setminus S\}. \tag{10}$$

Concerning the connectivity constrained version, the CNP3b, we need to deal with a minimum constraint, which is somewhat different from the CNP1b and the CNP2b. This means that for using greedy rule (9) we first need to find a set S such that $H(S) \geq N$, which is non-trivial for large N. However, once such a set S is found, we can use rule GR1$(S, 3, a)$.

3. An Evolutionary Algorithm for the CNP

In this section we present a flexible evolutionary framework that can be applied to any of the CNP types discussed so far. Although Arulselvan et al. (2011) have designed a genetic algorithm to deal with the CNP2b, the features of their algorithm are quite different from the characteristics of our approach. More specifically, we make use of greedy rules for repair operations during reproduction and mutation phases. This is one of the key features of the framework presented in this work as it allows a potential adaptation of the algorithm to many different types of the CNP in a straightforward manner. The evolutionary algorithm is presented in Algorithm 5.

Algorithm 5: Genetic Algorithm

Data: Graph: G; Type of the CNP: n (Connectivity Measure: $f(S)$, $C(S)$ or $H(S)$) and x (type a: Budget Constrained or type b: Connectivity Constrained); Constraint parameter: K, P, L or N; $t_{\max}$; $\mathcal{N}$; α

Result: S^*

$t \longleftarrow 0$;
Initialise$(\mathcal{N}, \mathcal{P}, S^*, \gamma, \pi, \alpha)$;
while $t \leq t_{\max}$ **do**
    $\mathcal{P}' :=$ New_Generation$(\mathcal{N}, \mathcal{P}, \gamma, n, x)$;
    $\mathcal{P}' :=$ Mutate$(\mathcal{P}', \pi, n, x)$;
    $\mathcal{P} :=$ Ordering$(\mathcal{P}, \mathcal{P}', \gamma, S^*, n, x)$;
    $(\gamma, \pi) :=$ Update(γ, π, α);
    $t \longleftarrow$ cpuTime();
$S^* :=$Local_Search(S^*, n, x);

We adopt the standard algorithmic framework of a Genetic Algorithm (GA). First we generate a population of solutions, then we mix them to produce new solutions (reproduction phase) which we randomly perturb (mutation phase). After that we order the old and new solutions according to

a fitness function and eventually we create a new population by eliminating the worst quality solutions. The process is iterated until a time limit is reached. The best solution is returned after the application of a local search procedure.

The notation must be understood as follows: $\mathcal{P}$ and $\mathcal{P}'$ are populations of $\mathcal{N}$ individual solutions of the CNP; individual solutions are represented by the set S of deleted nodes, thus $\mathcal{P}^{(\prime)} = \{S_i^{(\prime)} : i \in \{1, ..., \mathcal{N}\}\}$; a parameter γ is used in the fitness function and its value is set by an input parameter α, while the parameter π refers to the probability of mutation for each newly created solution.

The steps of the Evolutionary Algorithm proposed are described in detail in the following subsections.

3.1. Initialisation

In order to evaluate the solutions we introduce a fitness function:

$$F(S, \gamma, S^*, n, x) = z(S, n, x) + \gamma\, \Sigma(S, S^*).$$

The function $\Sigma(S, S^*)$ computes the number of nodes in S that are also present in the best known solution S^*, while $z(S, n, x)$ represents the objective function of the problem. Using $F(S, \gamma, S^*, n, x)$ – instead of the objective function $z(S, n, x)$ – should maintain some diversity among solutions by penalising those that are too close to the best one, therefore boosting solutions that depart from the best one while maintaining a competitive objective function. With the CNP3a, which is a maximisation problem, we replace function Σ with the Hamming distance between S and the best solution S^*, i.e. the number of nodes in S not present in S^*, so that solutions with a high $F(S, \gamma, S^*, 3, a)$ are favoured.

The parameter γ sets the respective weight of the two terms in the fitness function. It is initialised at each generation as:

$$\gamma = \alpha\, z(S^*, n, x) / \langle \Sigma(S, S^*) \rangle_{\mathcal{P}}, \tag{11}$$

where the value α quantifies the importance of each criterion over the other. The notation $\langle \bullet \rangle_{\mathcal{P}}$ means that the average of a quantity has been computed over the solutions of population $\mathcal{P}$.

Parameter π is initially set at $\pi_{\min}$, that represents the minimal probability of mutation for an individual solution. The value of π will be then updated

according to the discovery of a new best solution at each new generation (see Section 3.4).

The initial solutions in $\mathcal{P}$ are obtained by the *greedy algorithm* $G_1^{(nx)}$ described in the previous section. Even though this would seem to drastically reduce the variety of the initial population, initial numerical tests on the CNP1a instances indicate that the quality of the solutions is improved. In any case, a degree of variety among the greedy solutions is provided by the randomly broken ties in the selection of the nodes and by the choice of an initial vertex cover of the graph. However, given that for very large graphs (specifically when $|V| + |E|$ becomes large), generating the whole initial population through the use of greedy $G_1^{(nx)}$ can be extremely time consuming, we devote only a fraction $\mathcal{I}$ of the running time to this operation. Beyond that time, we create the remaining solutions (up to $\mathcal{N}$) by deleting nodes at random in the graph until we satisfy the particular constraint of the CNP considered.

3.2. Reproduction

In order to produce a child S' from a pair of parents S_1 and S_2, we need to state how the information from S_1 and S_2 will be merged into a single solution. This is usually done by first merging S_1 and S_2 inside a single set $S' := S_1 \cup S_2$. For the CNPna where $n \in \{1, 2, 3\}$, it usually leads to a solution with too large a set $|S'| \geq K$. For the CNP1b and the CNP2b, it leads to a solution with a lower connectivity measure $f(S)$ or $C(S)$. The situation is a bit more complicated for the CNP3b: for example, a node that has no neighbours in $V \setminus S_1$ might be part of S_2 and a component might be deleted in the merge, resulting in an overall lower number of connected components. Should the solution obtained be infeasible, it is then discarded during the selection process. We can then apply the greedy rule that moves nodes from S' to the remaining graph $V \setminus S'$. This is what is called GR1(S, n, x) in accordance with the notation in Section 2. As outlined above, a node is selected at random between all the best nodes found.

From the pseudocode in Algorithm 6, it is evident that we have chosen a democratic strategy for reproduction, in the sense that the best individuals are not favoured compared to the worst ones when choosing the parents. This slows down the uniformisation of the population but guarantees a better exploration of the solution space. Such a choice is motivated theoretically by the fact that we have no means to foresee the combinations of nodes that will disconnect the graph efficiently when deleted together. It is thus interesting

to combine some of the worst solutions together in the early stages of the evolution.

Algorithm 6: New_Generation

Data: $\mathcal{N}$, $\mathcal{P}$, γ, n, x
Result: $\mathcal{P}'$
$\mathcal{P}' := \{\}$;
for $i = 1...\mathcal{N}$ **do**
  $i_1 := \text{IntRand}(\{0, ..., \mathcal{N}\})$; $i_2 := \text{IntRand}(\{0, ..., \mathcal{N}\} \setminus \{i_1\})$;
  $S'_i := S_{i_1} \cup S_{i_2}$;
  while $|S'_i|$ *can be lowered* **do**
    $u =$Select(GR1(S, n, x)); $S'_i := S'_i \setminus \{u\}$;
  $\mathcal{P}' := \mathcal{P}' \cup \{S'_i\}$;

Note also that the union of several sets S_i, $i \in \{1...p\}$, can be achieved in much the same way as described above with two parents, using the same rules to obtain the best possible child S'. Although this is an interesting possibility, numerical tests have shown no significant improvement when $p \geq 3$.

3.3. Mutation

For each individual of the newborn population, there is a probability of mutation controlled by π. A random integer number is generated from a uniform probability distribution between 1 and 100: if it is smaller than π, the solution is modified. This is usually done by randomly fixing to n_g the number of genes (meaning the number of nodes belonging to S) to be changed. The value of n_g is chosen between 1 and $|S|$ with a probability distribution inversely proportional to n_g: $p(n_g) \propto 1/n_g$. This choice favours the mutation of fewer genes so as not to disturb the solution too much while potentially allowing us to escape the local minimum. After deselecting n_g nodes from S, the solution can be suboptimal: this is the case for the Budget Constrained versions (CNPna), where usually $|S| = K$. It is typically infeasible with Connectivity Constrained (CNPnb) versions, given that a solution is always obtained through the application of a greedy rule that tries to reduce $|S|$ as much as possible while remaining feasible. It is then mandatory to add back nodes to S in order to optimise the connectivity value or to recover feasibility.

In order to do this we can either select nodes at random inside $V \setminus S$ or use the greedy rules designed in Section 2, which is the aim of the function GR2(S, n, x) in the pseudocode of Algorithm 7. This is a crucial step from an evolutionary computing point of view since choosing to use a greedy rule will actually narrow the exploration in the space of solutions and prevent the algorithm from converging to the best solution given an infinite computing time. However, given the combinatorial explosion of the number of solutions with $|S|$, we expect that a random selection has very little chances of significantly improving the solutions within a reasonable time limit. This consideration seems to hold in practice as evidenced by the numerical tests presented in Section 4.3. This is the reason why we choose to orient the mutation phase towards potentially better quality solutions by using GR2(S, n, x) as shown in the pseudocode of Algorithm 7.

Algorithm 7: Mutate

```
Data: 𝒩, 𝒫′, π, n, x
for i = 1...𝒩 do
    if IntRand({1, ..., 100}) ≤ π then
        n_g := IntRand({0, ..., K}, p);
        for j = 1...n_g do
            m := IntRand({1...|S′_i|}); S′_i := S′_i \ {m};
        while |S′_i| can be increased do
            u =Select(GR2(S, n, x)); S′_i := S′_i ∪ {u};
```

3.4. Ordering and Selection

Populations $\mathcal{P}$ and $\mathcal{P}'$ are then merged together and ordered according to the fitness function $F(S, \gamma, S^*, n, x)$ and the best individual is updated when a better solution is found. For the CNP*na* (where the number of deleted nodes is fixed at value K), we try to avoid an excessive convergence of the population towards the best individual by modifying a solution if it is found to be similar to another one in the population. Such a solution is simply modified by exchanging one node between S and $V \setminus S$. The best $\mathcal{N}$ solutions are selected inside the merged population and then they replace the solutions in $\mathcal{P}$.

We finally update the value of γ according to Eq. (11) using the newly computed values for S^* and $\langle \Sigma(S, S^*) \rangle_{\mathcal{P}}$. As concerns π, if a new best solution has been found, it is set to $\pi := \pi_{\min}$, otherwise it is increased at $\pi := \min(\pi + \delta_\pi, \pi_{\max})$. This allows us to diversify the solutions more whenever no improvement of the best solution has been found in the last generations.

3.5. Local Search

At the end of the genetic evolution, a local search is applied to the best found solution. The mechanism of the local search for the CNP1a is proposed in Arulselvan et al. (2009) and improved in Aringhieri et al. (2016b) (of which preliminary results were presented in Aringhieri et al. (2015)). We can basically stay in the feasibility region while optimising the connectivity measure by simply exchanging a node $u \in S$ with a node $v \in V \setminus S$. Two different neighbourhood explorations for this problem are detailed in Aringhieri et al. (2016b). These neighbourhoods explore two–node exchanges around the best solution found and are directly adaptable to the CNP2a and the CNP3a. We use one of these two neighbourhoods to improve the final solution for the CNPna versions (typically the one that guarantees the smallest complexity given the value of K compared to $|V|$, see Aringhieri et al. (2015) and Aringhieri et al. (2016b) for more details).

The local search for Connectivity Constrained CNPs is slightly more challenging: to improve a feasible solution, the cardinality of set S must be lowered. Maintaining the search as local as possible, this means that a node $v \in V \setminus S$ must be exchanged with two nodes $(u_1, u_2) \in S^2 = S \times S$ to lower the objective by one unit, as was done in Arulselvan et al. (2011) with exchanges of random nodes v and (u_1, u_2). We proceed here more systematically by evaluating the deletion of each node $v \in V \setminus S$ from the graph. The connected components of the graph are then computed through a DFS and the connectivity variation for reintroducing each couple $(u_1, u_2) \in S^2$ can be estimated very quickly by enumerating all the components that would be merged together. The first improving move is validated in any case and the local search procedure goes on until no more improving moves are found. Compared to the neighbourhood search for the CNPna versions, the complexity of each move is multiplied by a factor of $K/2$ as we have to evaluate couples $(u_1, u_2) \in S^2$ and not single nodes in S. This makes the local search much slower for the CNPnb, especially for large graphs.

4. Numerical Results

In this section, we will present extensive results over two benchmark sets of instances to demonstrate the overall quality of the solutions found by our genetic algorithm. When previous results exist in the literature (i.e. for the CNP1a), we will compare our results with the best known results, otherwise we will use the greedy algorithms of Section 2 to show that our algorithmic framework provides an added value with respect to the independent use of the greedy procedures.

4.1. Instances and Numerical Setup

The first benchmark set we use is composed of the graphs presented in Ventresca (2012) and Edalatmanesh (2013) and is called Set 1. For the CNP1a, many results are available for these graphs (Ventresca, 2012; Edalatmanesh, 2013; Addis et al., 2016; Aringhieri et al., 2016b) and we will compare them with our results. There are 16 graphs in total belonging to 4 groups with different characteristics. Barabasi-Albert (BA) graphs are scale-free networks and proved to be the easiest to process while the Watts-Strogatz (WS) are designed to mimic a small-world structure with a denser structure and they turn out to be the most challenging to solve. Erdos-Renyi (ER) are random graphs and Forest-Fire (FF) graphs reproduce the behaviour of how a fire spreads through a forest, with a scale-free structure like BA graphs but a densest structure. None of these graphs is expected to reproduce a real network. However, real networks usually display a mix of the characteristics of each category. This makes them an interesting benchmark set to determine the particular characteristics of a generic complex network which are critical for a given algorithm.

In order to characterise the graphs precisely, Table 2 displays the following quantities: the number of nodes $|V|$ and edges $|E|$ with the resulting average degree $\langle d \rangle = 2 * |E|/|V|$; the number of articulation points (AP), as a larger fraction of APs usually results in a graph which is easier to fragment (especially with respect to metric $H(S)$); the value of the clustering coefficient C which signals the tendency of nodes to cluster together; the average shortest path length D which indicates the average distance between two nodes taken at random inside the graph. Finally, we also provide two important quantities for solving the CNP, which are the number of nodes having degree 1, written $|D_1|$ according to the notation of Veremyev et al. (2014a), and the number of nodes which are neighbours of those in D_1, written as $|N(D_1)|$. It

has been noted (Shen and Smith, 2012; Veremyev et al., 2014a,b) that nodes of D_1 are always sub-optimal for the CNP, i.e. they will never be found in an optimal solution since it is more interesting to suppress their neighbours.

The fraction $|D_1|/|V|$, therefore, already gives a hint about the real difficulty of finding a good solution for a given graph. For example, the BA and FF graphs have a high value for $|D_1|$ and the number of APs. These features probably explain the relatively low difficulty of solving these instances exactly through the use of a linear solver, even for graphs with up to 5000 nodes (Aringhieri et al., 2016b). On the contrary, even small dense graphs with no nodes of degree 1 (e.g. WS250 graph) are in practice intractable when using a linear formulation of the problem. As for the value of $|D_1|/|N(D_1)|$, which by definition is greater than 1, it tells us on average how many nodes we will disconnect from the graph by deleting a node of $N(D_1)$. A large value for that parameter should indicate how attractive these nodes will be for the CNP formulation based on maximising $H(S)$.

Graph	$\|V\|$	$\|E\|$	$\langle d \rangle$	nb AP	C	D	$\|D_1\|$	$\|N(D_1)\|$
BA500	500	499	1.996	164	0.000	5.663	336	149
BA1000	1,000	999	1.998	324	0.000	6.045	676	290
BA2500	2,500	2,499	1.999	825	0.000	6.901	1,675	729
BA5000	5,000	4,999	2.000	1,672	0.000	8.380	3,328	1475
ER235	235	350	2.979	48	0.006	5.339	39	37
ER466	466	700	3.004	84	0.002	5.974	69	64
ER941	941	1,400	2.976	177	0.005	6.559	147	139
ER2344	2,344	3,500	2.986	419	0.001	7.516	396	354
FF250	250	514	4.112	83	0.276	4.816	57	50
FF500	500	828	3.312	195	0.247	6.026	160	136
FF1000	1,000	1,817	3.634	362	0.216	6.173	280	236
FF2000	2,000	3,413	3.413	725	0.245	7.587	552	477
WS250	250	1,246	9.968	0	0.473	3.327	0	0
WS500	500	1,496	5.984	0	0.420	5.304	0	0
WS1000	1,000	4,996	9.992	0	0.483	4.444	0	0
WS1500	1,500	4,498	5.997	0	0.480	7.554	0	0

Table 2: Benchmark instances of Set 1 (from Ventresca (2012)): main characteristics.

Moreover, the efficiency of our algorithm is evaluated on a second set of

instances deriving from real applications. The graphs *Bovine*, *Circuit* and *E.Coli* are used in Ventresca and Aleman (2014a) and represent respectively protein interactions for a bovine species (Reimand et al., 2008), an electronic circuit (Milo et al., 2004) and interactions within bacteria E. Coli (Yang et al., 2008). *USAir97* and *HumanDis* are used in Edalatmanesh (2013). The first graph represents the flight connections between major US airports in 1997 (USAir, 1997). The second graph represents the relation between genetic disorders. The nodes are the disorders and two nodes are connected if at least one gene is involved in both of them (Goh et al., 2007). *TrainsRome* is presented in Cacchiani et al. (2010) and represents a train network around the city of Rome with train stations as nodes. *EU_flights* is a network of flight connections between airports in the European Union. There is a link between two airports if a direct flight was recorded between them in February or August 2014. The graph *openflights* is a network of flight connections in the USA (Opsahl, 2011), while *yeast* is the graph of the interactions inside the yeast organism S.Cervisiae presented in Yu, H. et al (2008). It was downloaded from the webpage Yeast (2008). The graphs *Ham1000* to *Ham5000* are proposed as graphs with hamiltonian cycles in TSPLIB (Reinelt, 1991). They are not real graphs but help us diversify the set. The *powergrid* network is an electricity distribution network in the USA used in Watts and Strogatz (1998). *OClinks* represents the interactions inside a social network as presented in Opsahl and Panzarasa (2009). The remaining graphs come from the website Leskovec and Krevl (2014): *facebook* is a social network constructed from relations on Facebook while *grqc*, *hepph*, *hepth*, *astroph* and *condmat* represent the interactions between physicists of a same domain. The interactions are measured on the basis of the publications on the website `www.arxiv.org`. The characteristics of these graphs are displayed in Table 3.

The Hamilton graphs are somewhat different from the real graphs in the set since they have a much smaller clustering coefficient and none of them displays any articulation point . On the contrary, graphs representing flight connections or social interactions have a large clustering coefficient and usually a high average degree. For the graph of Set 2, the number of components (written as $|\mathcal{H}|$ in our notations) is also reported. The diameter D is computed for the largest component only. All the graphs of Set 2 are available in the same format at the following address: `http://di.unito.it/cnp`.

The algorithm was programmed in standard C++ and compiled with

Graphs	$\|V\|$	$\|E\|$	$\langle d \rangle$	nb AP	$\|\mathcal{H}\|$	C	D	$\|D_1\|$	$\|N(D_1)\|$
Bovine	121	190	3.140	10	1	0.044	2.861	58	9
Circuit	252	399	3.167	25	1	0.052	5.806	17	17
E.Coli	328	456	2.780	57	1	0.024	4.834	169	52
USAir97	332	2126	12.807	27	1	0.396	2.738	55	26
HumanDis	516	1188	4.605	112	1	0.430	6.509	90	66
TrainsRome	255	272	2.133	79	1	0.018	43.496	4	4
EU_flights	1,191	31,610	53.081	109	2	0.402	2.622	178	108
openflights	1,858	13,900	14.962	125	371	0.331	3.151	339	122
yeast	2,018	2,705	2.681	527	185	0.024	5.612	825	442
Ham1000	1,000	1,998	3.996	0	1	0.002	5.424	0	0
Ham2000	2,000	3,996	3.996	0	1	0.000	6.030	0	0
Ham3000a	3,000	5,999	3.999	0	1	0.000	6.365	0	0
Ham3000b	3,000	5,997	3.998	0	1	0.001	6.366	0	0
Ham3000c	3,000	5,996	3.997	0	1	0.001	6.361	0	0
Ham3000d	3,000	5,993	3.995	0	1	0.000	6.365	0	0
Ham3000e	3,000	5,996	3.997	0	1	0.001	6.366	0	0
Ham4000	4,000	7,997	3.999	0	1	0.001	6.621	0	0
Ham5000	5,000	9,999	4.000	0	1	0.000	6.807	0	0
powergrid	4,941	6,594	2.669	1,229	1	0.103	18.989	1,226	923
OClinks	1,899	13,838	14.574	220	4	0.057	3.055	388	218
facebook	4,039	88,234	43.691	11	1	0.519	3.693	75	10
grqc	5,242	14,484	5.526	813	355	0.630	6.049	843	586
hepth	9,877	25,973	5.259	1,584	429	0.284	5.945	1,581	1,189
hepph	12,008	118,489	19.735	1,168	278	0.659	4.673	1,173	872
astroph	18,772	198,050	21.101	1,107	290	0.318	4.194	1,002	802
condmat	23,133	93,439	8.078	2,096	567	0.264	5.352	1,799	1,395

Table 3: Benchmark instances of Set 2: main characteristics.

`gcc 4.1.2`. All tests were performed on an HP ProLiant DL585 G6 server with two 2.1 GHz AMD Opteron 8425HE processors and 16 GB of RAM. The number of individuals in the population $\mathcal{N}$ is set to 300 for all the graphs but the largest graphs of Set 2. For these graphs, a large value of the initial population $\mathcal{N}$ would drastically limit the work of the evolutionary mechanism. Thus, for the graphs *facebook*, *hepth*, *hepph*, *astroph* and *condmat* we set $\mathcal{N}$ equal to 200, 200, 150, 100 and 100 respectively.

For the CNP1a, to make a useful comparison with previous results from Ventresca (2012) and Aringhieri et al. (2016b), the total running times (in seconds) for the instances of Set 1 are the same as in Aringhieri et al. (2016b). The total running times are also different for the instances of Set 2 since the

size of the graphs varies greatly. The same time limits are also used for all other versions of the CNP. These time limits are reported in the column *time* of tables 4 and 5.

The fraction of the total running time devoted to compute the initial solutions (through algorithm G_1^{nx}) is fixed to $\mathcal{I} = 0.1$. This is usually enough for the graphs of Set 1 and the majority of the graphs in Set 2. Finally, 5% of the total running time is reserved to the local search phase at the end of the evolution process.

As far as the other numerical parameters are concerned, we chose the following values after preliminary computational tests: $\alpha = 0.2$, $\pi_{\min} = 5$, $\pi_{\max} = 50$ and $\delta_\pi = 5$. An indicative study of the stability of the results over Set 1 with different parameters is provided in subsection 4.3. This tuning test validates our final choice of the parameters for the instances considered.

4.2. Solutions for benchmark sets S_1 and S_2

For the CNP1a, the performances of our algorithm can be compared with several competing algorithms[1]. The results on Set 1 are very satisfactory. The best known results in column BK of Table 4 come from the metaheuristics of Aringhieri et al. (2016b) (some numerical results, however, have been updated after an improved implementation of the algorithms. The results are taken from `http://di.unito.it/cnp`). Not only does our Genetic Algorithm find the optimal value for half of the graphs, it also yields six new best known results, with remarkable results for the WS graphs which are very hard to solve. Our algorithm has an average gap to the best known values of 0.5%. This makes the Genetic Algorithm much more robust than any of the 30 VNS and ILS algorithms from Aringhieri et al. (2016b). The best one of these algorithms has an average gap to the best known values of 7.5%.

This robustness of the quality of the solutions is also validated on Set 2. The GA finds the best solution in 17 out of 26 instances and the second best result in 5 other cases (Table 5). On these instances, the competitors are a VNS and an ILS algorithm with a proven efficiency from Aringhieri et al. (2016b), and the two constructive, greedy-like algorithms from Addis et al.

[1]While our manuscript was under review, we have been made aware of other competing algorithms from the works of Pullan (2015) and Purevsuren et al. (2016). While it is difficult to compare them with our results since the running times are different, our results are competitive with theirs and provide a higher robustness on WattsStrogatz graphs.

graph	K	time	BK	GA	graph	K	time	BK	GA
BA500	50	3,780	195	*195*	FF250	50	2,640	194	*194*
BA1000	75	7,920	559	***558***	FF500	110	6,690	257	*257*
BA2500	100	10,000	3,704	*3,704*	FF1000	150	10,000	1,260	*1,260*
BA5000	150	10,000	10,196	*10,196*	FF2000	200	10,000	4,549	**4,546**
ER235	50	2,250	295	*295*	WS250	70	4,050	3,241	**3,240**
ER466	80	5,490	1,542	1,560	WS500	125	7,890	2,130	2,199
ER941	140	10,000	5,198	**5,120**	WS1000	200	10,000	115,914	**113,638**
ER2344	200	10,000	997,839	1,039,254	WS1500	265	10,000	13,792	**13,662**

Table 4: Results for the genetic algorithm on the graphs of Set 1 for the **CNP1a**, with existing best known heuristic results in the literature (Aringhieri et al., 2016b) (updated at `http://di.unito.it/cnp`) for comparison. The total running time (column denoted by "time") is set according to the one chosen in Aringhieri et al. (2016b). It is expressed in seconds. K represents the number of deleted nodes. Optimal results are in italic and new best known results from the GA are in bold font.

(2016). These results seem to suggest that the GA is less competitive for very dense graphs with a large average degree $\langle d \rangle$. Nevertheless it still performs well for some dense graphs like *hepph* and the results on large graphs such as *condmat* are also striking. It is interesting that even when we discard the Hamilton graphs which are not real graphs, the GA still finds more best solutions than any other competitor. As concerns the average gap to the best solutions, once again the GA outperforms the competitors with an average gap of 2.2%. The closest competitor is the ILS with a value of 6.8%.

For other types of the CNP, we can only compare with the simple greedy algorithms described in Section 2. They are used in a multi-start manner until the total time used for the GA is exhausted. The results for the CNP1b, the CNP2a, the CNP2b, the CNP3a and the CNP3b are displayed for both Set 1 and Set 2 in Tables A.9 – A.13. We report these tables in Appendix A for readability purposes. We usually run first the CNPna versions with the values of K reported in Tables 4 and 5. The values of the connectivity parameter for the CNPnb versions are similar to the results obtained for the CNPna formulation. This allows us to control the validity of the solutions found.

It is interesting to note that in a few instances the b version of the algorithm reaches better solutions than its a counterpart. For example for the CNP2b (Table A.11), a solution with $|S| = 162$ and a connectivity value (maximum cardinality of the connected components) equal to 488 is found

graph	K	time	VNS-I-N_1-FC	ILS-N_1-FC	*Greedy3d*	*Greedy4d*	GA
Bovine	3	100	**268**	**268**	**268**	**268**	**268**
Circuit	25	150	2,101	2,117	**2,099**	2,100	**2,099**
E.Coli	15	200	**806**	**806**	**806**	834	**806**
USAir97	33	300	5,444	4,442	4,442	4,726	**4,336**
HumanDis	52	300	**1,115**	**1,115**	**1,115**	**1,115**	**1,115**
TrainsRome	26	300	**920**	934	921	936	928
EU_flights	119	2500	356,631	357,486	**349,927**	350,757	351,610
openflights	186	4000	31,620	**28,671**	29,624	29,552	28,834
yeast	202	3000	1,421	1,434	1,416	1,415	**1,414**
Ham1000	100	1000	332,286	344,509	338,574	336,866	**328,817**
Ham2000	200	2000	**1,309,063**	1,417,341	1,372,109	1,367,779	1,315,198
Ham3000a	300	3000	3,058,656	3,235,069	3,087,215	3,100,938	**3,005,183**
Ham3000b	300	3000	3,121,639	3,260,886	3,096,420	3,100,748	**2,993,393**
Ham3000c	300	3000	3,079,570	3,237,528	3,094,459	3,097,451	**2,975,213**
Ham3000d	300	3000	3,027,839	3,242,622	3,090,753	3,100,216	**2,988,605**
Ham3000e	300	3000	3,031,975	3,280,762	3,095,793	3,113,514	**3,001,078**
Ham4000	400	3000	5,498,097	5,877,896	5,534,254	5,530,402	**5,403,572**
Ham5000	500	3000	8,889,904	9,212,984	8,657,681	8,653,358	**8,411,789**
powergrid	494	3000	**16,099**	16,533	16,373	16,406	16,254
OClinks	190	3000	623,366	625,671	**614,504**	614,546	620,020
facebook	404	3000	865,115	**420,334**	608,487	856,642	561,111
grqc	524	3500	13,751	13,817	13,787	13,825	**13,736**
hepth	988	8000	114,933	123,138	232,021	326,281	**114,382**
hepph	1201	10000	10,989,642	11,759,201	10,305,849	10,162,995	**7,336,826**
astroph	1877	13000	65,937,108	65,822,942	54,713,053	**54,517,114**	58,045,178
condmat	2313	16000	6,121,430	**2,298,596**	11,771,033	11,758,662	2,612,548

Table 5: Results for the genetic algorithm on the graphs of Set 2 for the **CNP1a**, with results from competing algorithms from Addis et al. (2016); Aringhieri et al. (2016b) for comparison. K represents the number of deleted nodes. The total running time (column denoted by "time") for each instance (and algorithm) is expressed in seconds. The same time limits are used again in all other numerical experiments concerning Set 2.

for the graph WS1000. The CNP2a formulation of the algorithm only found a solution with $|S| = 200$ and connectivity value equal to 507, which is dominated by the first solution. The advantage of our flexible framework is the possibility of running the "dual" version of the algorithm once a first result is obtained, adapting the constraint parameter to the first result, and checking whether a better solution can be found. In the previous example, if we take the solution of the CNP2b version and apply the greedy rule $GR2(2, b)$ until $|S| = 200$, we find a new solution for the CNP2a variant with $C(S) = 450$. Thus, for the graph WS1000 we can get a reduction of 11.2% with respect to the previous solution computed.

Overall, the Genetic Algorithm compares very favourably to the greedy algorithms. Our algorithm outperforms the greedy procedures in all the instances but one. The improvements of the GA are more relevant for difficult graphs without articulation points like the *WattsStrogatz* and *Hamilton* graphs, as well as for some large or dense graphs of the *condmat* or *facebook* instances.

instance	Min	Max	Average	Dispersion	Dispersion/Average
WS250 (CNP1a)	3186	3678	3314.15	91.10	2.75%
WS1000 (CNP1a)	106902	162573	125947.20	12261.88	9.74%
USAir97 (CNP1a)	4336	4336	4336	0	0.00%
WS1000 (CNP2a)	340	526	432.51	55.60	12.86%
WS250 (CNP3a)	9	16	12.56	1.6812	13.39%
Circuit (CNP1b)	25	26	25.33	0.4702	1.86%
Trains (CNP2b)	27	29	28.09	0.5118	1.82%
ER466 (CNP3b)	79	80	79.37	0.4828	0.61%

Table 6: Results of the GA over 100 runs for different instances and CNP formulations. Minimum, maximum and average results as well as the total and relative dispersion are reported.

Given the stochastic nature of our algorithm, it is important to assess the stability of the results obtained. Since it is very time consuming to perform this task for all instances presented, we computed the dispersion of the results obtained over a hundred runs for only a few select instances. Table 6 indicates the minimum, maximum and average results, as well as the total and relative dispersion, over 8 instances. The total time for each run is limited to $(|V| + |E|)/3$ seconds since this value is often sufficient to obtain good quality solutions. These partial results reveal a reasonable stability of the solutions, even though the relative dispersion can be up to 13% in the most difficult instances. The quality of the results for the CNP1a seems to be quite robust, especially if we consider that the average results for very hard to solve graphs (like WS250 and WS1000) are not far from the best known results.

4.3. Sensitivity to the parameters of the algorithm

In order to provide further insights on the robustness of our algorithm and on our choice of the numerical values of the parameters, we will illustrate the variation of the results (on average) for the CNP1a when these parameters

are varied. First, the graphs in Figure 3 illustrate the variation of the parameters $\alpha, \mathcal{N}$, the number of parents and the parameters $\pi_{\min}$ and δ_π. More precisely, we tested different values taking $\pi_{\min} = \delta_\pi$ and $\pi_{\max} = 10\,\pi_{\min}$. When a parameter is varied, the other parameters are kept fixed to the values indicated in subsection 4.1. To characterize the quality of each set of parameters, we computed the average gap to the values in Table 4 over all instances of Set 1.

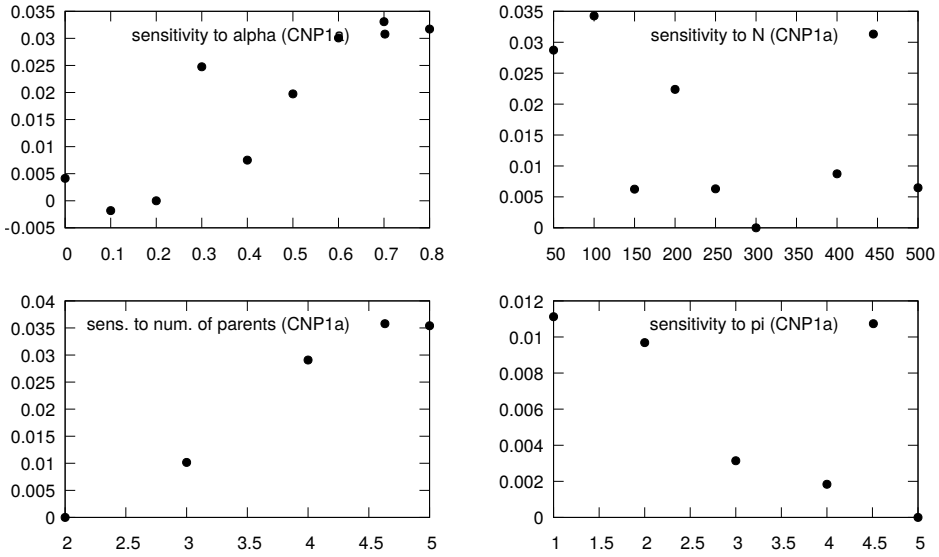

Figure 3: Sensitivity to the parameters α, $\mathcal{N}$, the number of parents, with $\pi_{\min} = \delta_\pi = \pi_{\max}/10$ (from left to right and from top to bottom). The plotted quantity is the average gap to the values in Table 4 over all instances of Set 1 and for the CNP1a.

The plots seem to validate our choice of the parameters. However, the algorithm seems relatively insensitive to their numerical values. The average gap does not exceed 4%. The value of the probability of mutation has the smallest influence. All in all, we can observe that large values of α, which favour solutions that are very different from the best one, disturb too much the selection process. Therefore, it is better to adopt for the parameter α a small, albeit non-zero value. In the same way, merging more than two solutions to create a new solution does not provide any improvement. The sensitivity to the size of the population is more chaotic. We have seen that

it should be adapted to the size of the graph. Nevertheless, the value $\mathcal{N} = 300$ turns out to be a good option also for the majority of the instances in Set 2 and for other versions of the CNP. There are other choices that can impact our numerical results. Choosing random solutions to populate the first generation, for example, would increase the average gap by 2.7% for the instances of Set 1 and deteriorate some of our results on Set 2. Repairing the mutated solutions through random moves instead of using the greedy rules from Section 2 would raise the gap by 1.6%. Adopting a different probability distribution for choosing the number of nodes to mutate has a very little impact on the final results ($< 0.5\%$). Finally, implementing a probability distribution for selecting the potential parents, that is inversely proportional to the fitness function of the parents, actually would increase the average gap by 4.6%. This confirms that a democratic strategy for the selection of parents allows the algorithm to explore the solution space more efficiently.

4.4. Pareto analysis of sample graphs

As outlined above, a solution for the CNP is driven by two quantities: the cardinality of the solution set S and the value of the connectivity measure. This leads to two complementary formulations that we called types a and b. A full understanding of the vulnerability of a graph requires a study of the disruption of the connectivity for all possible values of $|S|$. This analysis is similar to a Pareto study for a bi-objective optimisation problem, using an ε-constrained technique. Considering a CNPna formulation, it amounts to raising parameter $K = |S|$ by one unit and reoptimising the problem. Conversely, in a CNPnb version the connectivity value is fixed to its initial value when the graph is untouched. After that, the connectivity value is slowly lowered or increased according to the type of connectivity n.

This is a very time consuming process, thus we will present such a study for only a few small graphs from Set 1 and Set 2. We report both the Pareto surfaces found for the a and b versions of the problem. Only the non-dominated solutions are considered, namely, with respect to these solutions, no other solution has a lower number of nodes and a better objective function. The solutions are plotted in the $|S|$-connectivity plane in Figure 4 for the CNP1, Figure 5 for the CNP2 and Figure 6 for the CNP3. The first two sets of figures already suggest a certain correlation between solutions of the CNP1a and the CNP2a. The connectivity first diminishes with the increase of $|S|$ at a speed that depends on the density of the articulations points in the graph. For the graph WS250, which has not articulation points, the connectivity

decreases more slowly in the first part of the curve. When enough nodes are deleted, the giant component can be fragmented more efficiently. This seems to occur when $|S|$ is between 40 and 50. In this zone the graph is more challenging to solve since the choice of the nodes will greatly influence the variation of the connectivity. Such a trend in the connectivity reductions can also be spotted for the graph Circuit. The graphs that have a large fraction of articulation points and nodes with degree one, like FF250, exhibit a very fast reduction of the connectivity value with small values of $|S|$ before stalling. These curves generally indicate at which value of $|S|$ we start having a diminishing return on the reduction of connectivity. Obtaining the Pareto front for the b versions is usually more time consuming, particularly for the CNP1. The results are very similar between a and b versions for the graphs FF250 and USAir97. The Pareto front for the CNPnb versions is slightly better for the graph WS250.

The curves for the CNP3a are necessarily different in shape since the problem is a maximisation problem. The curves are more regular and almost linear in several cases. The difficulty of fragmenting the graph WS250 is again visible, since in some cases we have to add up to five nodes to S to create a new independent connected component in the graph. When $|S|$ is large enough and the graph is sufficiently sparse, the increase in the number of connected components we can obtain by raising $|S|$ by one starts to grow. This feature is somewhat opposed to what happens for the pair-wise connectivity minimisation. These observations are a first hint that the CNP3 has peculiarities that make it different from the CNP1 and the CNP2.

Furthermore, we provide an analysis of the number of nodes present in a solution with K deleted nodes, that are not present in the solution with $K-1$ deleted nodes (for the CNPna formulations). This is a way of understanding whether a good solution with $|S| = K + 1$ can be obtained simply from a solution with $|S| = K$ (using for example the greedy rules detailed in Section 2). Table 7 displays the average results over K for the previous 3 instances plus the graphs BA500, ER235 and Circuit. The values of K range from 1 to the value for which the graph is completely fragmented. A straightforward deduction of a CNP solution using smaller solutions of the same problem formulation can rarely work in practice, except for graphs that have a tree structure like the graph BA500. Surprisingly, the second best graph with less differences among the solutions would seem to be the graph USAir97, although it shares many characteristics with the graph WS250, such as a high average degree $\langle d \rangle$, a large clustering coefficient C and a

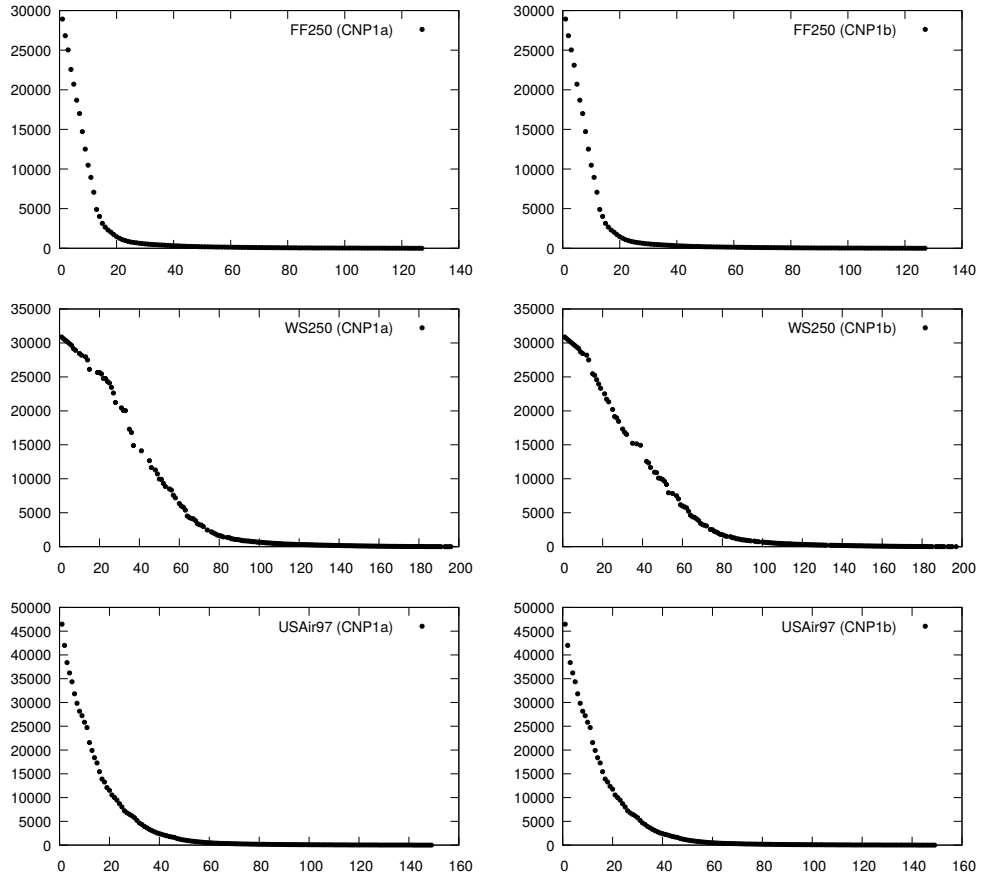

Figure 4: Pareto solutions for the CNP1, for graphs FF250, WS250, USAir97 (from top to bottom) and for versions a and b (from left to right). The x axis represents the number of nodes in the solution ($|S|$) while the y axis is the pair-wise connectivity value ($f(S)$).

small parameter D. We take this as a clue that the information given by the average degree, clustering coefficient and average shortest paths length is only partial for characterising a CNP instance. We can observe that USAir97 presents a much larger dispersion in the degree of the nodes. This feature could lead to a different graph structure and CNP solutions.

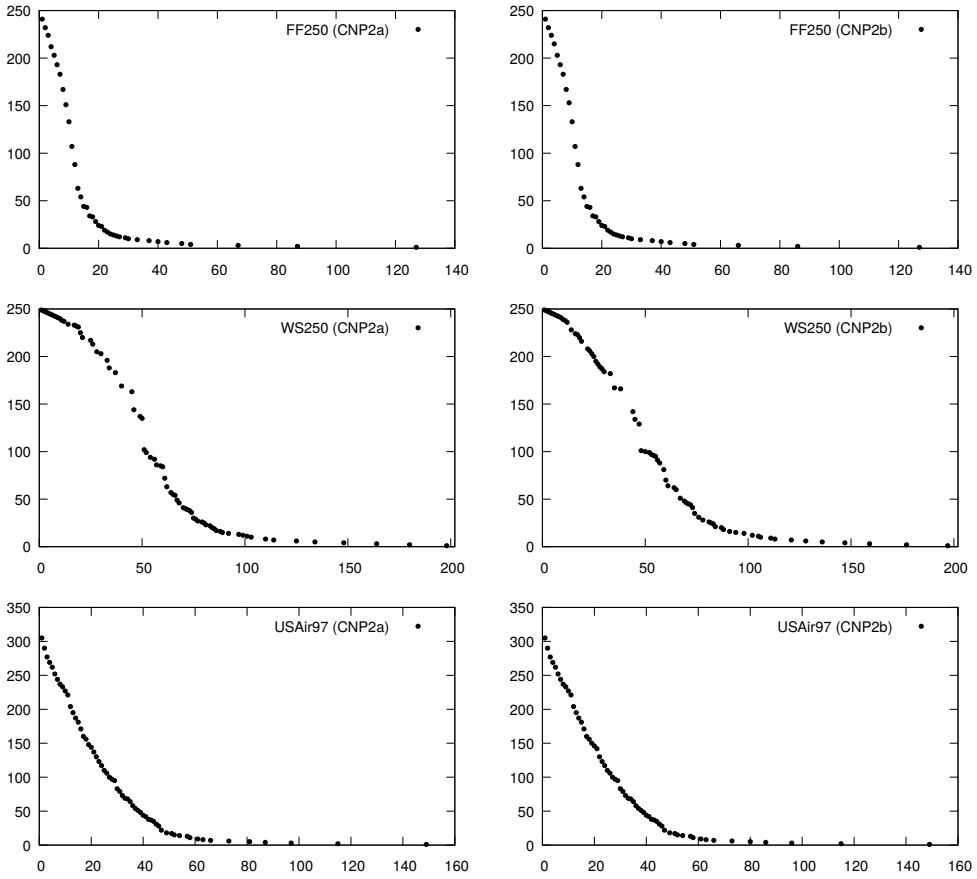

Figure 5: Pareto solutions for the CNP2, for graphs FF250, WS250, USAir97 (from top to bottom) and for versions a and b (from left to right). The x axis represents the number of nodes in the solution ($|S|$) while the y axis is the maximum cardinality of the connected components ($C(S)$).

4.5. Compatibility of the solutions of different CNP versions

The optimal solutions for the different versions of the CNP will not be the same in general. At the same time, they have a certain level of correlation. Ventresca and Aleman (2015b) highlighted this aspect for the CNP1a and the CNP2a. The study of some of the smaller graphs in the last section also suggests the presence of such a correlation. Therefore, it is interesting to investigate how much the solutions of different CNP versions usually differ.

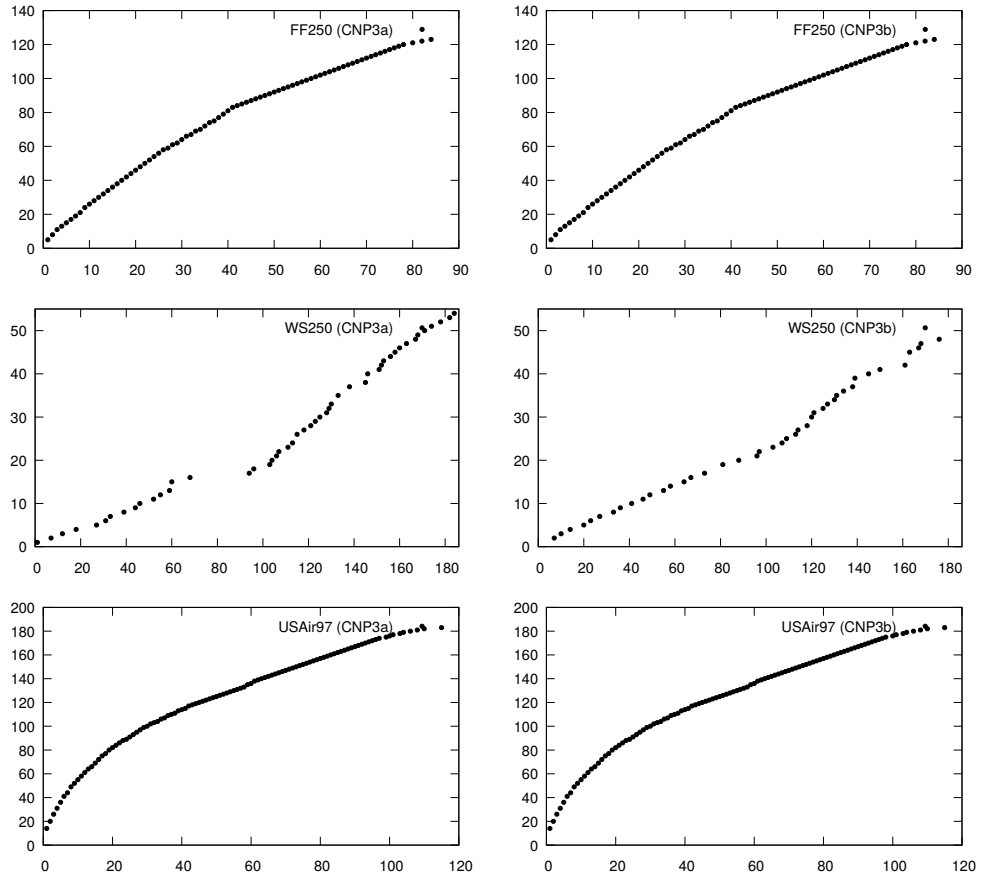

Figure 6: Pareto solutions for the CNP3, for graphs FF250, WS250 and USAir97 (from top to bottom) for versions a and b (from left to right). The x axis represents the number of nodes in the solution ($|S|$) while the y axis is the number of connected components ($H(S)$).

It is also appealing to try to characterise the nodes that are favoured in a CNP variant and disfavoured in other versions of the problem.

This could allow us to take a solution for a specific connectivity measure and quickly transform it into a good quality solution for another CNP version. Since these solutions were computed heuristically, such a study cannot pretend to characterise the differences between different CNP formulations in absolute terms. However, it can give us some hints about how the dif-

	BA500	ER235	FF250	WS250	Circuit	USAir97
CNP1a	2.122	5.632	7.373	21.959	7.008	5.953
CNP2a	1.230	9.588	4.230	31.675	10.640	6.047
CNP3a	1.000	6.527	4.506	28.565	10.911	3.573

Table 7: Average number of nodes present in a solution with $|S| = K$ that are not present in the solution with $|S| = K - 1$, for six small instances and all three versions of the CNPna.

ferent versions of our algorithm relate to each other. In addition, given the good results obtained against the competitors, we can reasonably hope that our solutions provide an indication about the general properties of the CNP problems treated here.

We provide an analysis of the solutions of the Budget Constrained CNPna versions, since they are the easiest to compare as they share the same value of $|S|$. For each solution of a CNPna problem, we can easily compute and compare the value of all three connectivity functions $f(S)$, $C(S)$ and $H(S)$. Table 8 displays the results on a subset of the graphs from Set 1 and Set 2. A preliminary conclusion that can be drawn is that in some cases our approach for the CNP2a is outperformed by the solutions obtained for the CNP1a (graphs *EU_flights* and *Ham5000*). This result can be linked to our choice of the greedy rules. However, it is evident from these results that the solutions of the CNP1a and the CNP2a have similar characteristics. The values of $f(S)$ and $C(S)$ are relatively close. Instead, those connectivity values in the CNP3a tend to show an increase by 20% or even more.

We broadened the analysis to the whole Set 1 and Set 2 (a total of 42 instances from which 17 are real instances). In the last row of Table 8, we provide the number of instances for which the solutions of each CNPna problem reached the best overall value of $f(S)$, $C(S)$ and $H(S)$. Our preliminary analysis is confirmed, in the sense that algorithms for the CNP1a and the CNP3a do a very good job concerning their own connectivity measure. For the CNP2a, our genetic algorithm does not find the best value in 25% of the instances. This is interesting since, in our framework, the algorithm for the CNP1a guarantees on average to find a good solution with respect to the CNP2a connectivity measure. This could be due to the fact that the size of all components are taken into account in the CNP1a. Lifting the degeneracies by exploiting this information could provide a better guide through the

graph	CNP1a $f(S)$	CNP1a $C(S)$	CNP1a $H(S)$	CNP2a $f(S)$	CNP2a $C(S)$	CNP2a $H(S)$	CNP3a $f(S)$	CNP3a $C(S)$	CNP3a $H(S)$
BA5000	**10,196**	14	1,931	10,263	**13**	1,902	13,119	37	**1,999**
ER2344	**1,039,254**	1434	126	114,2031	**1,412**	99	1,412,231	1,681	**336**
FF2000	**4,546**	12	452	4,865	**11**	425	9,445	55	**498**
WS1500	**13,662**	33	56	14,533	**30**	52	54,545	301	**73**
TrainsRome	**928**	15	28	990	**11**	25	1,821	42	**31**
EU_flights	**351,610**	**839**	199	358,312	847	207	364,240	854	**211**
yeast	**1,414**	7	1,033	1,475	**6**	1,025	1,542	15	**1,050**
Ham5000	**8,411,789**	**4,099**	20	8,561,041	4,138	60	9,178,493	4,285	**196**
powergrid	**16,254**	20	964	17,713	**17**	921	1,505,833	1,721	**1,228**
facebook	**561,111**	722	141	648,971	**442**	109	1,125,806	1,062	**361**
hepth	**114,382**	91	1,649	125,794	**66**	1,599	8,864,918	4,210	**2,566**
total wins	41	16	3	2	31	1	1	2	42

Table 8: Value of the three connectivity metrics of the solutions for all the CNPna on a subset of instances of Set 1 and Set 2. Values of parameter $K = |S|$ are the same as in the previous tables. Best results between all three algorithms are in bold font. The last row displays the number of instances for which the solutions of each CNPna found the best $f(S)$, $C(S)$ and $H(S)$, over all instances of Set 1 and Set 2.

solution space in the CNP2a variant.

Let us try to characterise the solutions of the different CNPna. We start by counting the proportion of nodes that are common among them. For the sake of simplicity, we only provide the average values over all 42 instances. The proportion of nodes common to the CNP1a and the CNP2a solutions is 62.8% on average. The proportion goes down to 50.8% and 49.9% when the CNP3a solutions are compared with the solutions of the CNP1a and the CNP2a respectively. The total proportion of the nodes shared by the three types of solutions is only 41.7%, suggesting a high variability in their overall structure. We computed the average proportion of articulation points selected by the individual solutions in each instance (we limited the analysis to the instances that contain APs). The solutions of the CNP1a and the CNP2a only select 32.3% and 31.8% of the APs, while the solutions of the CNP3a incorporate 44.2% of all APs. The difference is even larger when the proportion of nodes from $N(D_1)$ is considered. This value goes up to 47.9% for the CNP3a while it is equal to 32.5% and 32.3% for the CNP1a and CNP2a.

The previous results are confirmed by the analysis of the nodes that are exclusive to each type of solution. The nodes that are only present in the CNP3a solutions are composed of 42.0% of nodes from $N(D_1)$. For the

solutions of the CNP1a and the CNP2a, the percentage is equal to 17.2% and 17.0% respectively. This confirms our first intuition that the articulation points, in particular those belonging to $N(D_1)$, are extremely attractive for the solutions of the CNP3a. At the same time, the articulation points seem to be less crucial to the design of the solutions of the CNP1a and the CNP2a. There is a simple possible explanation of this general behaviour. In the CNP3a, only the number of connected components matters while their cardinality is not relevant. On the contrary, in the CNP1a and CNP2a the largest component influences greatly the objective function value. However, since the minimal number of components in a graph $G[V \setminus S]$ is bounded by $\lceil |V|/C(S) \rceil$, reducing efficiently $C(S)$ requires to fragment the graph into a large number of components. Therefore, the CNP1a and CNP2a need to take into account both $C(S)$ and $H(S)$.

It is very difficult to detect the characteristics of the nodes that are more suitable for one type of connectivity measure rather than for the others. Ventresca and Aleman (2015b) and Aringhieri et al. (2016b) outlined that the use of the betweenness centrality values does help in searching for a good quality solution for the CNP1a. We computed the betweenness centrality values of the nodes in each graph and we looked for the proportion of the nodes in each solution that are part of the K nodes with the highest centrality value. We found that 51.4% and 50.4% of the nodes in the solutions of the CNP1a and the CNP2a belong to the set of nodes with the highest centrality values. This percentage goes down to 45.1% for the CNP3a. Therefore, on average, only one half of the K nodes with the highest betweenness centrality are part of a complete solution. This underlines a complexity of the CNP that is hardly reducible to characteristics of the single nodes like the centrality values. As a final remark, we stress that an intrinsic difficulty of the CNP is to identify sets of nodes capable of fragmenting the graph only when they are deleted together. Such "articulation sets", also called "node cuts" or "node separators", are extremely difficult to spot right away and cannot be determined considering the node-dependent quantities alone.

5. Conclusions

We presented a general Evolutionary Framework to solve a general problem known as the Critical Node Problem. Our framework is based on a simple genetic algorithm structure that makes use of appropriate greedy rules to repair and correct the solutions during the reproduction and mutation phases.

The proposed hybrid heuristic is quickly adaptable to several formulations of the problem since only the criteria for the greedy rules have to be redesigned and implemented for a new formulation. We outlined new greedy rules (when needed) and presented numerical results for six formulations of the problem, including "dual" formulations which involve the same connectivity measure.

Our results compare favourably to the best known results. This suggests the good quality of the solutions found and above all their robustness. Benchmark results for all versions of the problem are provided. They may constitute an interesting basis for future comparison especially for the variants of the CNP where efficient metaheuristics were not available up to now. We also tested our approach on a new set of benchmark instances with real graphs. By comparing solutions of different types of the CNP, we could confirm intrinsic differences of their structure for connectivity measures that use global features of the fragmented graphs (number of connected components and their cardinalities). It would be very interesting to investigate the application of this algorithmic framework to real problems linked to the CNP, such as vaccination problems or the analysis of biological networks, by exploiting the flexibility of our Evolutionary Framework. Such problems often require the ability to deal with oriented or weighted graphs. These features would slow down all existing efficient heuristic algorithms for the CNP. It would be appealing to devise effective procedures for tackling these aspects computationally in a reasonable time.

Acknowledgments. The authors would like to thank Valentina Cacchiani, Francesca Cordero, Guglielmo Guastalla and Mario Ventresca for providing or indicating useful real graphs of interest for this work. We would also like to thank two anonymous referees for their invaluable help in improving the overall clarity and consistency of the paper. This work was supported by a Google Focused Grant on Mathematical Programming, project "Exact and Heuristic Algorithms for Detecting Critical Nodes in Graphs".

References

Addis, B., Aringhieri, R., Grosso, A., Hosteins, P., 2016. Hybrid Constructive Heuristics for the Critical Node Problem. Annals of Operations Research 238 (1), 637–649.

Addis, B., Di Summa, M., Grosso, A., 2013. Removing critical nodes from

a graph: complexity results and polynomial algorithms for the case of bounded treewidth. Discrete Applied Mathematics 161, 2349–2360.

Albert, R., Jeong, H., Barabási, A. L., 2000. Error and attack tolerance of complex networks. Nature 406, 378–382.

Alevras, D., Grötschel, M., Wessäly, R., 1997. Capacity and survivability models for telecommunication networks. Tech. rep., in Proceedings of EURO/INFORMS Meeting.

Aringhieri, R., Grosso, A., Hosteins, P., 2016a. A genetic algorithm for a class of Critical Node Problems. Electronic Notes in Discrete Mathematics 52, 359–366, proceedings of the INOC2015 conference.

Aringhieri, R., Grosso, A., Hosteins, P., Scatamacchia, R., 2015. VNS solutions for the critical node problem. Electronic Notes in Discrete Mathematics 47, 37–44, Proceedings of the VNS'14 conference.

Aringhieri, R., Grosso, A., Hosteins, P., Scatamacchia, R., 2016b. Local search metaheuristics for the critical node problem. Networks 67, 209–221.

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

Arulselvan, A., Commander, C. W., Shylo, O., Pardalos, P. M., 2011. Cardinality-constrained critical node detection problem. In: Gülpnar, N., Harrison, P., Rüstem, B. (Eds.), Performance Models and Risk Management in Communications Systems. Vol. 46 of Springer Optimization and Its Applications. Springer New York, pp. 79–91.
URL `http://dx.doi.org/10.1007/978-1-4419-0534-5\_4`

Balas, E., de Souza, C., 2005. The vertex separator problem: a polyhedral investigation. Mathematical Programming 103, 583–608.

Boginski, V., Commander, C. W., 2009. Identifying critical nodes in protein-protein interaction networks. In: Butenko, S., Chaovalitwongse, W. A., Pardalos, P. M. (Eds.), Clustering Challenges in Biological Networks. World Scientific Publishing, pp. 153–168.

Borgatti, S. P., 2006. Identifying sets of key players in a social network. Computational and Mathematical Organization 12, 21–34.

Brown, G., Carlyle, M., Salmerón, J., Wood, K., 2006. Defending critical infrastructure. Interfaces 36 (6), 530–544.

Cacchiani, V., Caprara, A., Toth, P., 2010. Scheduling extra freight trains on railway networks. Transportation Research Part B 44, 215–231.

Cavalcante, V. F., de Souza, C. C., 2011. Exact algorithms for the vertex separator problem in graphs. Networks 57, 212–230.

Cohen, R., Ben-Avraham, D., Havlin, S., 2003. Efficient immunization strategies for computer networks and populations. Physical Review Letters 91, 247901–247905.

Cormican, K. J., Morton, D. P., Wood, R. K., 1998. Stochastic network interdiction. Operations Research 46, 184–197.

Di Summa, M., Grosso, A., Locatelli, M., 2011. The critical node problem over trees. Computers and Operations Research 38, 1766–1774.

Di Summa, M., Grosso, A., Locatelli, M., 2012. Branch and cut algorithms for detecting critical nodes in undirected graphs. Computational Optimization and Applications 53, 649–680.

Dinh, T. N., Xuan, Y., Thai, M. T., Park, E. K., Znati, T., 2010. On approximation of new optimization methods for assessing network vulnerability. In: Proceedings of the 29th IEEE Conference on Computer Communications (INFOCOM). pp. 105–118.

Edalatmanesh, M., 2013. Heuristics for the critical node detection problem in large complex networks. Ph.D. thesis, Faculty of Mathematics and Science, Brock University, St. Catharines, Ontario.

Goh, K. I., Cusick, M. E., Valle, D., Childs, B., Vidal, M., Barabasi, A. L., 2007. The human disease network. Proceedings of the National Academy of Sciences of the United States of America 104, 8685–8690.

Grubesic, T. H., Murray, A. T., 2006. Vital nodes, interconnected infrastructures, and the geographies of network survivability. Ann Association American Geographers 96, 64–83.

Jenelius, E., Petersen, T., Mattsson, L.-G., 2006. Importance and exposure in road network vulnerability analysis. Transportation Research Part A: Policy and Practice 40 (7), 537 – 560.

Lalou, M., Tahraoui, M. A., Kheddouci, H., 2015. Component-cardinality-constrained critical node problem in graphs. Discrete Applied Mathematics.
URL http://dx.doi.org/10.1016/j.dam.2015.01.043

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

Lim, C., Smith, J. C., 2007. Algorithms for discrete and continuous multicommodity flow network interdiction problems. IIE Transactions 39, 15–26.

Matisziw, T. C., Murray, A. T., 2009. Modeling s-t path availability to support disaster vulnerability assessment of network infrastructure. Computers & Operations Research 36, 16–26.

Milo, R., Itzkovitz, S., Kashtan, N., Levitt, R., Shen-Orr, S., Ayzenshtat, I., et al, 2004. Superfamilies of evolved and designed networks. Science 303, 1538–42.

Opsahl, T., 2011. Why anchorage is not (that) important: Binary ties and sample selection. http://wp.me/poFcY-Vw.

Opsahl, T., Panzarasa, P., 2009. Clustering in weighted networks. Social Networks 32, 155–163.

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

Purevsuren, D., Cui, G., Win, N., Wang, X., 2016. Heuristic algorithm for identifying critical nodes in graphs. Advances in Computer Science: an International Journal 5.

Reimand, J., Tooming, L., Peterson, H., Adler, P., Vilo, J., 2008. Mining heterogeneous biological networks for gene modules with functional significance. Nucleic Acids Research 36, W452–9.

Reinelt, G., 1991. TSPLIB–a traveling salesman problem library. ORSA Journal on Computing 3, 376–384.

Salmerón, J., Wood, K., Baldick, R., 2004. Analysis of electric grid security under terrorist threat. IEEE Trans Power Syst 19, 905–912.

Sánchez-Oro, J., Mladenović, N., Duarte, A., 2016. General variable neighborhood search for computing graph separators. Optimization Letters. In Press.

Shen, S., Smith, J. C., 2012. Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs. Networks 60 (2), 103–119.

Shen, S., Smith, J. C., Goli, R., 2012. Exact interdiction models and algorithms for disconnecting networks via node deletions. Discrete Optimization 9, 172–88.

USAir, 1997. Computational analysis of social and organizational systems. The USAir97 network. Freely available at `http://www.casos.cs.cmu.edu/computational_tools/datasets/external/USAir97/index11.php`.

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

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

Ventresca, M., Aleman, D., 2015a. Efficiently identifying critical nodes in large complex networks. Computational Social Networks 2 (6).

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

Ventresca, M., Aleman, D. M., 2014b. A region growing algorithm for detecting critical nodes. In: COCOA'14. pp. 593–602.

Veremyev, A., Boginski, V., Pasiliao, E., 2014a. Exact identification of critical nodes in sparse networks via new compact formulations. Optimization Letters 8, 1245–1259.

Veremyev, A., Prokopyev, O. A., Pasiliao, E. L., 2014b. An integer programming framework for critical elements detection in graphs. Journal of Combinatorial Optimization 28, 233–273.

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

Watts, D. J., Strogatz, S. H., 1998. Collective dynamics of small-world networks. Nature 393, 400–442.

Wollmer, R., 1964. Removing arcs from a network. Operations Research 12, 934–940.

Wood, R. K., 1993. Deterministic network interdiction. Mathematical and Computer Modelling 17, 1–18.

Yang, R., Huang, L., Lai, Y., 2008. Selectivity-based spreading dynamics on complex networks. Physical Review E 78.

Yeast, 2008. Yeast interaction network. `http://interactome.dfci.harvard.edu/S_cerevisiae/index.php?page=download`.

Yu, H. et al, 2008. High-quality binary protein interaction map of the yeast interactome network. Science 322.

Zhou, T., Fu, Z. Q., Wang, B. H., 2006. Epidemic dynamics on complex networks. Progress in Natural Sciences 16, 452–457.

Appendix A. Numerical results for the different versions of the CNP

We present here the numerical results for the following versions of the CNP: CNP1b, CNP2a, CNP2b, CNP3a and CNP3b on Sets 1 and 2 in Tables A.9 to A.13.

graph	P	$G_1^{(1b)}$	$G_2^{(1b)}$	GA	graph	P	$G_1^{(1b)}$	$G_2^{(1b)}$	GA
BA500	200	**50**	**50**	**50**	FF250	200	**50**	51	**50**
BA1000	550	**76**	**76**	**76**	FF500	300	106	**104**	**103**
BA2500	3,700	102	**101**	**101**	FF1000	1,250	154	157	**151**
BA5000	10,000	157	**153**	**153**	FF2000	4,500	208	207	**202**
ER235	300	52	52	**51**	WS250	3,000	77	86	**72**
ER466	1,500	84	89	**82**	WS500	2,000	142	156	**129**
ER941	5,000	148	162	**142**	WS1000	115,000	259	423	**194**
ER2344	1,000,000	248	263	**209**	WS1500	14,000	330	375	**266**
Bovine	270	**3**	**3**	**3**	Ham3000c	3,000,000	504	439	**301**
Circuit	2,100	**26**	28	**26**	Ham3000d	3,000,000	498	436	**300**
E.Coli	800	**16**	**16**	**16**	Ham3000e	3,000,000	505	434	**304**
USAir97	4,000	**34**	41	**34**	Ham4000	5,500,000	632	557	**382**
HumanDis	1,100	**53**	**53**	**53**	Ham5000	8,500,000	813	720	**498**
TrainsRome	1,000	26	26	**25**	powergrid	16,000	525	535	**499**
EU_flights	350,000	136	124	**120**	OClinks	615,000	203	197	**193**
openflights	30,000	197	200	**185**	facebook	420,000	634	1,203	**465**
yeast	1,400	207	211	**204**	grqc	13,800	538	576	**524**
Ham1000	328,000	159	145	**100**	hepth	115,000	1046	1,221	**994**
Ham2000	1,300,000	351	303	**208**	hepph	7,500,000	1478	1,594	**1,339**
Ham3000a	3,000,000	503	438	**302**	astroph	55,000,000	3225	2,732	**1,870**
Ham3000b	3,000,000	506	436	**305**	condmat	2,600,000	2487	3,483	**2,460**

Table A.9: Results for the genetic algorithm on the graphs of Set 1 and Set 2 for the **CNP1b**, with results from greedy algorithms $G_1^{(1b)}$ and $G_2^{(1b)}$ for comparison. P represents the maximum pair-wise connectivity of the solution. Best results between all three algorithms are in bold font.

graph	K	$G_1^{(2a)}$	$G_2^{(2a)}$	GA	graph	K	$G_1^{(2a)}$	$G_2^{(2a)}$	GA
BA500	50	**4**	**4**	**4**	FF250	50	**5**	6	**5**
BA1000	75	**5**	**5**	**5**	FF500	110	**4**	**4**	**4**
BA2500	100	11	11	**10**	FF1000	150	**7**	8	**7**
BA5000	150	14	14	**13**	FF2000	200	**12**	**12**	**12**
ER235	50	**7**	26	**7**	WS250	70	58	170	**41**
ER466	80	18	197	**14**	WS500	125	24	315	**15**
ER941	140	35	453	**23**	WS1000	200	**475**	792	507
ER2344	200	1,870	1,638	**1,412**	WS1500	265	128	1,161	**30**
Bovine	3	**16**	**16**	**16**	Ham3000c	300	2,650	2,643	**2,444**
Circuit	25	30	93	**27**	Ham3000d	300	2,651	2,639	**2,441**
E.Coli	15	21	21	**19**	Ham3000e	300	2,656	2,644	**2,453**
USAir97	33	73	111	**69**	Ham4000	400	3,549	3,528	**3,292**
HumanDis	52	**10**	11	**10**	Ham5000	500	4,437	4,418	**4,138**
TrainsRome	26	12	17	**11**	powergrid	494	18	23	**17**
EU_flights	119	868	**847**	**847**	OClinks	190	1,143	1,125	**1,118**
openflights	186	178	272	**141**	facebook	404	470	1,683	**442**
yeast	202	**6**	**6**	**6**	grqc	524	**18**	504	**18**
Ham1000	100	867	875	**806**	hepth	988	124	3,589	**66**
Ham2000	200	1,758	1,758	**1,613**	hepph	1,201	8,181	6,384	**3,600**
Ham3000a	300	2,649	2,639	**2,457**	astroph	1,877	13,835	12,112	**11,947**
Ham3000b	300	2,657	2,646	**2,444**	condmat	2,313	6,042	10,810	**513**

Table A.10: Results for the genetic algorithm on the graphs of Set 1 and Set 2 for the **CNP2a**, with results from greedy algorithms $G_1^{(2a)}$ and $G_2^{(2a)}$ for comparison. K represents the number of deleted nodes. Best results between all three algorithms are in bold font.

graph	L	$G_1^{(2b)}$	$G_2^{(2b)}$	GA	graph	L	$G_1^{(2b)}$	$G_2^{(2b)}$	GA
BA500	4	**47**	**47**	**47**	FF250	5	**48**	49	**48**
BA1000	5	**61**	**61**	**61**	FF500	4	102	102	**100**
BA2500	10	101	**100**	**100**	FF1000	7	145	145	**142**
BA5000	13	154	151	**149**	FF2000	12	191	187	**182**
ER235	7	**49**	50	**47**	WS250	40	79	80	**73**
ER466	14	86	85	**81**	WS500	15	145	144	**126**
ER941	25	149	152	**139**	WS1000	500	195	418	**162**
ER2344	1,400	252	270	**204**	WS1500	30	339	332	**278**
Bovine	15	**4**	**4**	**4**	Ham3000c	2,500	446	404	**276**
Circuit	30	25	26	**24**	Ham3000d	2,500	452	402	**276**
E.Coli	20	16	**15**	**15**	Ham3000e	2,500	455	403	**280**
USAir97	70	34	40	**33**	Ham4000	3,300	651	571	**398**
HumanDis	10	51	50	**49**	Ham5000	4,200	745	662	**458**
TrainsRome	10	30	31	**28**	powergrid	20	449	440	**428**
EU_flights	850	127	118	**113**	OClinks	1,100	209	200	**197**
openflights	140	194	206	**184**	facebook	450	472	821	**324**
yeast	6	202	199	**195**	grqc	20	497	501	**480**
Ham1000	800	172	151	**103**	hepth	70	1,040	1,042	**981**
Ham2000	1,600	362	313	**221**	hepph	3,600	1,416	1,572	**1,228**
Ham3000a	2,500	448	402	**279**	astroph	12,000	3,284	1,769	**1,322**
Ham3000b	2,500	456	401	**279**	condmat	500	**2,506**	2,651	**2,506**

Table A.11: Results for the genetic algorithm on the graphs of Set 1 and Set 2 for the **CNP2b**, with results from greedy algorithms $G_1^{(2b)}$ and $G_2^{(2b)}$ for comparison. L represents the maximum allowed cardinality of connected components. Best results between all three algorithms are in bold font.

graph	K	$G_1^{(3a)}$	$G_2^{(3a)}$	GA	graph	K	$G_1^{(3a)}$	$G_2^{(3a)}$	GA
BA500	50	**313**	**313**	**313**	FF250	50	**92**	**92**	**92**
BA1000	75	**590**	**590**	**590**	FF500	110	214	214	**215**
BA2500	100	**1,129**	**1,129**	**1,129**	FF1000	150	337	334	**340**
BA5000	150	1,998	**1,999**	**1,999**	FF2000	200	491	497	**498**
ER235	50	67	65	**68**	WS250	70	7	4	**15**
ER466	80	99	105	**110**	WS500	125	25	18	**44**
ER941	140	181	190	**206**	WS1000	200	9	4	**41**
ER2344	200	286	309	**336**	WS1500	265	17	19	**73**
Bovine	3	**77**	**77**	**77**	Ham3000c	300	20	57	**127**
Circuit	25	30	29	**31**	Ham3000d	300	17	56	**132**
E.Coli	15	**169**	168	**169**	Ham3000e	300	16	55	**131**
USAir97	33	103	**104**	**104**	Ham4000	400	16	69	**166**
HumanDis	52	147	**148**	**148**	Ham5000	500	18	81	**196**
TrainsRome	26	30	**31**	**31**	powergrid	494	1,161	1,225	**1,228**
EU_flights	119	203	**211**	**211**	OClinks	190	544	545	**554**
openflights	186	1,101	1,105	**1,109**	facebook	404	229	256	**361**
yeast	202	1,049	1,049	**1,050**	grqc	524	1,499	1,511	**1,539**
Ham1000	100	17	28	**52**	hepth	988	2,452	2,465	**2,566**
Ham2000	200	17	41	**87**	hepph	1,201	2,304	2,272	**2,441**
Ham3000a	300	18	57	**132**	astroph	1,877	2,373	2,728	**2,740**
Ham3000b	300	15	57	**125**	condmat	2,313	4,369	4,358	**4,709**

Table A.12: Results for the genetic algorithm on the graphs of Set 1 and Set 2 for the **CNP3a**, with results from greedy algorithms $G_1^{(3a)}$ and $G_2^{(3a)}$ for comparison. K represents the number of deleted nodes. Best results between all three algorithms are in bold font.

graph	N	$G_1^{(3b)}$	$G_2^{(3b)}$	GA	graph	N	$G_1^{(3b)}$	$G_2^{(3b)}$	GA
BA500	300	**44**	**44**	**44**	FF250	90	**48**	**48**	**48**
BA1000	600	**80**	**80**	**80**	FF500	215	111	111	**110**
BA2500	1,100	**93**	**93**	**93**	FF1000	340	152	154	**150**
BA5000	2,000	**151**	**151**	**151**	FF2000	500	205	202	**201**
ER235	70	**52**	55	**52**	WS250	15	102	67	**62**
ER466	110	91	84	**79**	WS500	45	157	131	**118**
ER941	200	157	149	**134**	WS1000	40	379	233	**174**
ER2344	350	274	235	**209**	WS1500	70	356	236	**216**
Bovine	80	**4**	**4**	**4**	Ham3000c	130	757	300	**267**
Circuit	30	25	26	**24**	Ham3000d	130	744	298	**269**
E.Coli	170	**16**	**16**	**16**	Ham3000e	130	773	299	**273**
USAir97	100	31	**30**	**30**	Ham4000	170	1,051	399	**382**
HumanDis	150	54	**53**	**53**	Ham5000	200	1,302	482	**463**
TrainsRome	30	26	**25**	**25**	powergrid	1,200	516	**481**	**481**
EU_flights	200	114	**108**	**108**	OClinks	550	196	193	**187**
openflights	1,100	186	183	**180**	facebook	350	1344	394	**387**
yeast	1,050	203	203	**202**	grqc	1,500	523	514	**496**
Ham1000	50	221	103	**94**	hepth	2,500	1,021	1013	**943**
Ham2000	90	493	201	**178**	hepph	2,400	1,275	1,306	**1,177**
Ham3000a	130	748	302	**270**	astroph	2,700	2,370	1,840	**1,834**
Ham3000b	130	770	301	**273**	condmat	4,700	2,599	2,594	**2,317**

Table A.13: Results for the genetic algorithm on the graphs of Set 1 and Set 2 for the **CNP3b**, with results from greedy algorithms $G_1^{(3b)}$ and $G_2^{(3b)}$ for comparison. N represents the minimum allowed number of connected components. Best results between all three algorithms are in bold font.