IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Local Search Metaheuristics for the Critical Node Problem

This is the author's manuscript

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/1509060 since 2016-11-14T17:36:08Z

Published version:

DOI:10.1002/net.21671

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

13 October 2026

UNIVERSITÀ DEGLI STUDI DI TORINO

This is an author version of the contribution published on:

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

Local Search Metaheuristics for the Critical Node Problem.

Networks, 67(3):209–221, 2016. Advance online publication 19 February 2016.

DOI: 10.1002/net.21671

The definitive version is available at:

http://onlinelibrary.wiley.com/doi/10.1002/net.21671/abstract

Local Search Metaheuristics for the Critical Node Problem *

Roberto Aringhieri, Andrea Grosso, Pierre Hosteins
Dipartimento di Informatica
Università degli Studi di Torino, Italy
{andrea.grosso, roberto.aringhieri, pierre.hosteins}@unito.it

Rosario Scatamacchia
Dipartimento di Automatica e Informatica
Politecnico di Torino, Italy
rosario.scatamacchia@polito.it

December 1, 2015

Abstract

We present two metaheuristics for the Critical Node Problem, i.e., the maximal fragmentation of a graph through the deletion of k nodes. The two metaheuristics are based on the Iterated Local Search and Variable Neighbourhood Search frameworks. Their main characteristic is to exploit two smart and computationally efficient neighbourhoods which we show can be implemented far more efficiently than the the classical neighbourhood based on the exchange of any two nodes in the graph, and which we prove is equivalent to the classical neighbourhood in the sense that it yields the same set of neighbours. Solutions to improve the overall running time without deteriorating the quality of the solution computed are also illustrated. The results of the proposed metaheuristics outperform those currently available in literature.
Keywords: Critical Node Problem, Graph Fragmentation, Metaheuristics

1 Introduction

Given an undirected graph $G(V, E)$ and an integer K, the *Critical Node Problem* (CNP) consists in finding a subset of K nodes $S \subseteq V$, such that the number of node pairs still connected in the induced subgraph $G[V \setminus S]$

$$f(S) = |\{i, j \in V \setminus S : i \text{ and } j \text{ are connected by a path in } G[V \setminus S]\}| \quad (1)$$

is as small as possible.

To the authors' knowledge, the problem can be traced back to the so-called *network interdiction problems* studied by Wollmer [33] and later Wood [34]. Although these seminal papers focused on arc deletion, recently attention has centered more on node deletion. This is also due to the renewed emphasis on security-related research which has called attention to network interdiction problems [1] and to works related to the assessment of the robustness of communication networks [12, 13].

*Work supported by a Google Focused Grant on Mathematical Programming, project "Exact and Heuristic Algorithms for Detecting Critical Nodes in Graphs"

Further applications of CNP arise in different contexts. Borgatti [7] studies the CNP in the context of detecting so-called "key players" in a social network; Arulselvan et al. [5] and Ventresca [28] emphasize contagion control via vaccination of a limited number of individuals, where the nodes of the graph represent potentially infected individuals and the edges represent contacts occurring between them.

The CNP is known to be NP-complete [5], although polynomially solvable on trees [10] and other specially structured graphs [3, 25]. For a broad literature review, including problems with different graph fragmentation metrics, we refer to comprehensive references given in other works, for example [25, 26].

In this paper, we present two metaheuristics for the CNP based on Iterated Local Search (**ILS**) [23] and Variable Neighbourhood Search (**VNS**) [19, 20]. The main contribution of this paper concerns the development of two smart and computational efficient neighbourhoods which are also proved to be equivalent to the classical but computationally complex neighbourhood exchange of two nodes u and v in such a way that $u \in S$ and $v \in V \setminus S$. This allows the development of two metaheuristics for the CNP which significantly improve the best known solutions available in the literature. Further, solutions to improve the overall running time without deteriorating the solution quality are also illustrated.

The paper is organized as follows. Section 2 reports a detailed analysis of the algorithm competitors available in literature. Section 3 describes the two efficient neighbourhood explorations. Section 4 depicts the two metaheuristics proposed for solving CNP and also describes the methods adopted to improve the overall running time while maintaining solution quality. Section 5 discusses the computational results of the proposed algorithm on benchmark instances and compares them with those obtained by the state of the art algorithms for the CNP. Finally, conclusions are provided in Section 6.

2 Existing algorithms

From a practical point of view, the CNP on general graphs has been tackled by Arulselvan et al. [5] by proposing a MILP model and a heuristic approach based on a greedy heuristic coupled with a successive local search phase. Sophisticated metaheuristics – namely population-based incremental learning and simulated annealing – are studied and experimentally compared by Ventresca [28]. The latter work also presents a set of benchmark instances available online that we will use in order to compare our algorithms. All the algorithms presented here have been explicitly developed for the version of the CNP considered here, i.e., the minimization of pair-wise connectivity (number of node pairs still connected).

2.1 Population Based Incremental Learning and Simulated Annealing

In [28], a Population Based Incremental Learning (PBIL) and a Simulated Annealing (SA) algorithms have been proposed together with the first set of benchmark instances publicly available. A probability matrix $\mathbf{M}$ is used to model the probability distribution over the possible values for each element of the solution. At each iteration t, $\mathbf{M}_t$ is used to generate a sample H of h solutions. The best solution in H is then compared with the current best B found and, in the case of improvement, B is updated. Finally, the matrix $\mathbf{M}_t$ is updated using the best current solution B and then a mutation operator introduces some diversification. Algorithm 1 reports the pseudo-code of PBIL.

SA is a standard implementation of the well-known algorithm performing a Monte-Carlo-like exploration of the solution space with a varying temperature parameter updated at each step of

Algorithm 1: PBIL

Data: graph G, K, ω, α, mutation parameters (β,γ);
Result: **B**
Initialize M_0;
for $t = 1$ *to* ω **do**
  $H \longleftarrow$ **generateSamples**$(h, \mathbf{M}_{t-1})$;
  if $f(bestSol(H)) < f(B)$ **then** $B \longleftarrow$ bestSol(H);
  ;
  $\mathbf{M}_t \longleftarrow (1-\alpha)\mathbf{M}_{t-1} + \alpha\mathbf{B}$;
  $\mathbf{M}_t \longleftarrow$ **mutate**$(\beta, \gamma, \mathbf{M}_t)$;

the search.

According to [28], computational results concerning solution quality are clearly in favour of the PBIL algorithm over its SA counterpart. On the contrary, SA found solutions faster than PBIL.

2.2 Algorithms based on Depth First Search

The algorithms reported in Section 2.1 are typically quite slow. Faster algorithms are developed in [14] based on a Depth First Search [27] (DFS) which exploits local properties of nodes in V.

The local properties considered are the degree of a node $u \in V$ and the fact that a node $u \in V$ is an articulation point – a node whose removal results in splitting the graph into two or more connected components – or has edges that act as (local) bridges.

The algorithms developed in [14] compute a ranking of the nodes in V based on a numerical function which combines the nodes' local properties whose coefficients are tuned by statistical experiments on a large set of graphs.

Among the methods tested, the best performing algorithm is the $DFSH-Post$ whose main characteristic is to have a post-processing procedure after the ranking determined by the DFS. The post-processing procedure consists in deselecting those nodes $v \in S$ (the set of deleted nodes) such that the ratio between the number of neighbours belonging to S and the degree of the node is greater than a given threshold. The rationale here is to identify such nodes having the majority of their neighbours already deleted whose replacement by other nodes could be more beneficial to the graph disconnection.

$DFSH-Post$ has very short running time, less than 0.06 seconds on all benchmark instances introduced in [28].

2.3 Combined greedy approaches

Combined greedy approaches are proposed in [2]. The basic idea is to alternate two simple greedy rules for determining a feasible solution. The first rule removes a node $u \in V$ from S, that is

$$\textbf{Greedy rule 1}: \quad S' = S \setminus \{u\} \quad \text{s.t.} \quad u = \arg\min\{f(S \setminus \{u\}) - f(S)\}. \tag{2}$$

The second rule adds a node $u \in V$ to S, that is

$$\textbf{Greedy rule 2}: \quad S' = S \cup \{u\} \quad \text{s.t.} \quad u = \arg\max\{f(S) - f(S \cup \{u\})\}. \tag{3}$$

Simple greedy algorithms can be derived exploiting the two rules. In [5], an initial solution S is obtained by determining a vertex cover on G and then, if $|S| > K$, nodes are deleted from

S using greedy rule 1. On the contrary, a greedy heuristic can be obtained starting from $S = \emptyset$ and then adding nodes to S using greedy rule 2. Algorithm 2 depicts the pseudo-code of the greedy adopting rule 1.

The procedure **VertexCover** is a modified version of the greedy heuristic which selects the node with highest degree, adds it to the cover, deletes all adjacent edges, and then repeats until the graph is empty (see, e.g., [24]). Our modifications are concerned with an initial shuffling of the nodes in such a way to consider them in random order instead of by decreasing order of node degree, and a pre-processing phase to take out the nodes of degree 1.

Algorithm 2: *Greedy1*

```
   Data: graph G, K
   Result: S*
1  S ⟵ VertexCover(G);
2  while |S| > K do
3  |   u ⟵ arg min{f(S \ {u}) − f(S)};
4  |   S ⟵ S \ {u};
5  S* := S;
```

The idea is to combine these two rules, applying them in sequence so as to introduce an exploration around the feasible solutions with $|S| = K$ and have a chance to get out of local minima. Among many others discussed in [2], two algorithms *Greedy3d* and *Greedy4d* have been selected for the quality of their solutions. The pseudo-code for *Greedy3d* is reported in Algorithm 3: lines 3–6 and 8–12 represent the application of the greedy rule 1 and 2, respectively, breaking ties randomly. A pseudo-code for *Greedy4d* is very similar: it starts from the full graph G, and the greedy rule 2 precedes the greedy rule 1. These two algorithms sequentially delete and add more nodes to S (up to Δ_K) for ℓ times in order to perturb a feasible solution with K nodes. When the best solution is not improved after generating $\mathcal{I}$ feasible solutions, both algorithms restart the search from the vertex cover (*Greedy3d*) or from the full graph (*Greedy4d*).

Algorithm 3: *Greedy3d*

```
   Data: Graph: G, K, Δ_K, ℓ, I;
   Result: S*
1  n ⟵ 0; S ⟵ VertexCover(G); count ⟵ 0;
2  repeat
3  |   while |S| > K − Δ_K do
4  |   |   u ⟵ arg min{f(S \ {u}) − f(S)}; S ⟵ S \ {u};
5  |   |   if |S| = K then
6  |   |   |   if f(S) ≤ f(S*) then S* ⟵ S; count ⟵ 0; ;
7  |   |   |   else count ⟵ count + 1;;
8  |   n ⟵ n + 1;
9  |   while |S| < K + Δ_K do
10 |   |   u ⟵ arg max{f(S) − f(S ∪ {u})}; S ⟵ S ∪ {u};
11 |   |   if |S| = K then
12 |   |   |   if f(S) ≤ f(S*) then S* ⟵ S; count ⟵ 0; ;
13 |   |   |   else count ⟵ count + 1;;
14 |   n ⟵ n + 1;
15 |   if count = I then S ⟵ VertexCover(G); count ⟵ 0;
16 |   ;
17 until n > ℓ;
```

2.4 Exact algorithms

CNP can be modelled as a maximization problem, as reported in [11], with binary variables $\mathbf{x} = (x_i \colon i \in V)$ where $x_i = 1$ iff node $i \in V$ is deleted, and $\mathbf{y} = (y_{ij} \colon i, j \in V, i < j)$ where $y_{ij} = 1$ iff the node pair $\{i, j\}$ is disconnected in the residual graph $G[V \setminus S]$.

$$\begin{aligned} \text{maximize } z = & \sum_{i,j \in V : i<j} y_{ij} \\ \text{subject to } & \sum_{i \in V} x_i \leq K \\ & (\mathbf{x}, \mathbf{y}) \in X \\ & x_i \in \{0,1\} \quad i \in V \\ & y_{ij} \in \{0,1\} \quad i, j \in V,\, i < j. \end{aligned}$$

The polytope X links the values of $\mathbf{x}$ and $\mathbf{y}$ accordingly with the above specification. Note that the objective function can be easily transformed to match function $f(S)$ defined in (1), that is $f(S) = \frac{|V|(|V|-1)}{2} - \sum_{i<j} y_{ij}$.

The branch and cut algorithm presented in [11] is based on a formulation that – although potentially exponential in size – does not overwhelm the solver:

$$X = \left\{ \mathbf{x}, \mathbf{y} \;\middle|\; \textstyle\sum_{r \in V(P)} x_r \geq y_{ij} \quad \text{for each } P \in \mathcal{P}(i,j), i, j \in V,\, i < j \right\} \tag{4}$$

where $\mathcal{P}(i,j)$ is the set of paths linking i to j in G and $V(P)$ is the set of nodes in path P. Constraints (4) state that a pair of nodes i and j can be disconnected only if, for any path P linking i and j, a node r belonging to P is deleted from the graph. Constraints (4) can be separated by solving shortest-path problems. We note that an interesting model requiring only $\mathcal{O}(|V|^2)$ constraints has been recently proposed by Veremyev et al. [30].

3 Neighbourhoods

In order to build efficient metaheuristics based on a local search mechanism, we need to devise efficient local search strategies.

Given a solution S, the value $f(S)$ can be computed through a modified version of the algorithm computing the connected components of a graph (see, e.g., [21]) requiring $\mathcal{O}(|V|+|E|)$. Hereafter, we refer to this algorithm as *Connect* as in [21]. A new solution S' can be obtained from S exchanging a pair of nodes $u \in S$ and $v \in V \setminus S$. The value $f(S')$ of such a new solution can not be computed by updating the value $f(S)$ (as usually done in a Local Search framework) but requires a computation from scratch, applying again the above algorithm. This fact poses a challenge regarding the computational efficiency of any neighbourhood exploration for the CNP.

After discussing the classical neighbourhood exploration N_0, we will present two efficient neighbourhoods N_1 and N_2. We will also show that the three neighbourhoods select the same move leading to the same next incumbent solution, in the case where no ties must be broken.

Neighbourhood N_0. As discussed above, a classical way to get a new solution S' from S is to exchange a node $u \in S$ with another node $v \in V \setminus S$ (*2-node-exchange*). Its main drawback lies in its computational complexity: actually, a complete neighbourhood evaluation is required to select all the nodes $u \in S$, with $|S| = K$, pair them with all the nodes $v \in V \setminus S$ and compute the new objective function using *Connect*. The total complexity of this operation is therefore

$\mathcal{O}(K(|V|-K)(|V|+|E|))$. This can be very time consuming, with a complexity growing in the worst case with the cube of the number of vertices, that is $\mathcal{O}(|V|^3)$.

Neighbourhood N_1. For a given $u \in S$, we can directly determine the node $v' \in V \setminus S$ that disconnects the graph as much as possible, i.e., such that $v' = \arg\max\{f(S) - f((S \setminus \{u\}) \cup \{v'\})\}$. This can be done by reintroducing u in the graph ($S = S \setminus \{u\}$) and performing a modified *Connect* (with complexity $\mathcal{O}(|V| + |E|)$) that will track all the articulation points and the possible impact of each node if removed from the graph (ties are broken randomly).

In other words, the best exchange with a node $u \in S$ is found avoiding the explicit evaluation of the objective function for all pairs (u, v), with $v \in V \setminus S$. Since all the nodes $u \in S$ have to be considered, the computational complexity of the N_1 exploration is then $\mathcal{O}(K(|V| + |E|)$, which is less than the one of N_0. In addition, the worst case complexity is reduced to $\mathcal{O}(|V|^2)$.

Neighbourhood N_2. We consider an approach complementary to the one discussed for neighbourhood N_1, i.e., for a given node $v \in V \setminus S$ we identify the node $u' = \arg\min\{f((S \cup \{v\}) \setminus \{u'\}) - f(S)\}$. This can be computed in two steps. In the first step, *Connect* computes the new connected components in G without node v in $\mathcal{O}(|V| + |E|)$. Then, we evaluate the ensemble of each node in S with the components in the graph by exploiting the information computed by *Connect*. In particular, we can compute the value $f((S \cup \{v\}) \setminus \{u\})$ for each $u \in S$ in $\mathcal{O}(D(G))$ at most, where $D(G)$ is the maximum degree of a node in G, thus finding u' in $O(K \times D(G))$ at most. Therefore, since all the nodes in $V \setminus S$ have to be evaluated, the total complexity of N_2 is bounded by $\mathcal{O}((|V| - K)(|V| + |E| + K \times D(G)))$.

We remark that the functions determining v' in N_1 and u' in N_2 are respectively the greedy rules 2 and 1 defined by the equations (3) and (2).

Theorem 1. *Starting from the same current incumbent S, each of the neighbourhoods N_0, N_1 and N_2 selects the same move (u^*, v^*) – with $u^* \in S$ and $v^* \in V \setminus S$ – yielding the same next incumbent S^*, in the case where no ties must be broken.*

Proof. In order to avoid the selection of different moves due to breaking ties, let us suppose there exists one and only one best possible move (u^*, v^*) for a given current incumbent S.

By definition N_0 will identify such a best move since it tests all possible pairs of nodes (u, v) with $u \in S$ and $v \in V \setminus S$.

Let us now consider N_1. First we consider a generic move (u, v) in such a way that $u \neq u^*$. By construction, this move leads to an incumbent S' such that $f(S^*) < f(S')$. On the contrary, if $u = u^*$ let us suppose the algorithm extracts a move (u^*, v) where $v \neq v^*$ leading to a solution S' such that $f(S') < f(S^*)$. This is equivalent to saying that the algorithm – which evaluates the impact of all possible nodes $v \in (V \setminus S) \cup \{u\}$ – finds a pair (u^*, v) with higher impact than (u^*, v^*), which is a contradiction. Therefore, N_1 extracts the best possible move (u^*, v^*).

Similar reasoning proves that N_2 will also select (u^*, v^*). □

An illustrative example. Let us consider the graph in Figure 1. Assume that nodes 1 and 2 have been removed. The surviving connections are three, namely the ones between the nodes 3-4, 3-5, 4-5. If we look for an exchange disconnecting the graph as fully as possible, it is easy to see that the insertion of node 1 and the removal of node 3 is uniquely the best move, leaving just nodes 4 and 5 still connected.

N_0 finds the best move after evaluating all the possible exchanges. N_1 evaluates first the insertion in the graph of node 1 that leads to the removal of node 3. Then it considers the

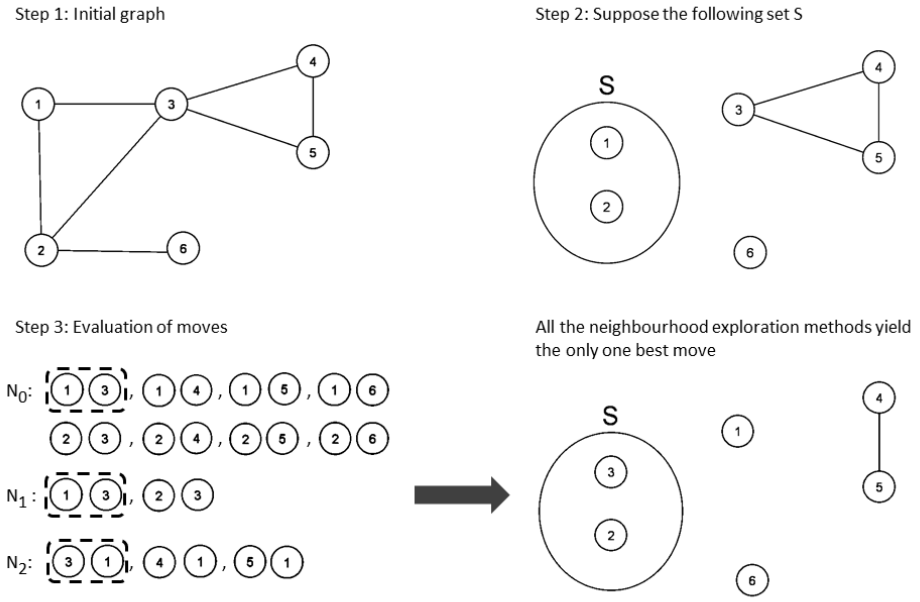

Figure 1: Example of neighbourhoods N_0, N_1 and N_2.

insertion of node 2 and removes node 3 again. Finally, N_1 chooses the swap $(1, 3)$ since $(2, 3)$ results in a graph with two connections (nodes $2-6$, $4-5$).

N_2 evaluates the deletion of nodes 3, 4, 5 and the consequent insertion of nodes 1[1], while the removal of node 6 leads to its reinsertion. N_2 then chooses the exchange producing the highest impact, that is again $(1, 3)$. Therefore all the exploration methods yield the same best move as Theorem 1 states.

4 Local Search Metaheuristics

This Section presents the two metaheuristics proposed for solving CNP. We describe the general framework for the proposed algorithms in order to highlight their main components. First we discuss the Local Search engine (sect. 4) which is used in both algorithms depicted in Sections 4.1 and 4.2. Finally, we briefly introduce all the algorithm variants tested in Section 5.

We discuss the common Local Search engine which is then exploited within the ILS and VNS framework making use of the neighbourhoods proposed in Section 3.

A priori, the neighbourhoods are designed using a best improvement strategy. However, even though N_1 and N_2 are more efficient than N_0 from a complexity point of view, dealing with large instances might require an additional gain in computational efficiency. Therefore, we propose an alternative search strategy in order to improve the total running time of the algorithms, without impairing the quality of the solution provided.

[1]Notice that as far as the removal of nodes 4 and 5 is concerned, their reinsertion in the graph or the insertion of node 1 produces the same impact on the graph (three surviving connections). Without loss of generality, we selected node 1 in the example.

A first common approach is to use a first improvement strategy to explore a given neighbourhood rather than a best improvement one: instead of fully exploring the neighbourhood to search the best move, the first improvement strategy selects the first improving move.

The computational effectiveness of a first improvement strategy depends on how long it takes to find an improving move. To facilitate this process, we introduce a ranking of the nodes so as to identify the nodes to be evaluated first for the goal of finding such a move faster. We tested the use of a centrality measure to get some information about the importance of the nodes.

Centrality concepts have been introduced in late seventies in [15, 16]. There exist different centrality measures like degree, closeness, betweenness and Katz centrality among the most common ones [31]. Degree and betweenness centrality have been used in [9] to evaluate the tolerance of complex networks to errors and attacks.

We considered the betweenness centrality, which measures how many times a node belongs to the shortest path between two other nodes. This measure seems closely related to CNP objective function, see for example [29] that studies the correlation of different centrality measures with several graph connectivity measures. Given a node u and a pair of nodes $s \neq u$ and $t \neq u$, let $\sigma_u^{s,t}$ be equal to the ratio of number of shortest paths between s and t containing the node u and the total number of shortest paths between s and t. The betweenness centrality of a node u is equal to the sum of $\sigma_u^{s,t}$ over all different pairs of node $s, t \in V$, that is

$$b_u = \sum_{s,t \in V, s \neq t \neq u} \sigma_u^{s,t}.$$

We implemented Brandes algorithm [8], running in $\mathcal{O}(|V| \times |E|)$, to compute the b_u values for all $u \in V$.

With respect to Algorithm 5 and 7, we compute b_u on the initial graph G and then we use them to sort the nodes to be explored in N_1 and N_2. More specifically, we sort the nodes in a decreasing or increasing value of b_u for N_1 and N_2, respectively.

Algorithm 4: Generic Local Search for CNP

Data: graph G, neighbourhood N, solution S, flag $CentrRanking$, $FirstImpr$;
Result: local optima S^*;

```
1 foundMove ⟵ true;
2 while foundMove do
3 |   (u*, v*) ⟵ selectBestMove (N, S, V, FirstImpr, CentrRanking);
4 |   if f(S ∪ {v*} \ {u*}) < f(S) then S ⟵ S ∪ {v*} \ {u*} ;
5 |   else foundMove ⟵ false ;
6 S* ⟵ S;
```

The pseudo-code reported in Algorithm 4 describes a generic Local Search for CNP returning a local optima S^*. The inputs of this procedure are the graph G, the neighbourhood N and the starting solution S. Furthermore it takes two boolean flags $FirstImpr$ and $CentrRanking$: $FirstImpr$ is equal to **true** when a first improvement strategy is applied, **false** otherwise; $CentrRanking$ is equal to **true** when the nodes are ranked with respect to their betweenness centrality values b_u. Note that the ranking is performed outside the Local Search engine, thus the values b_u are not a parameter of the procedure. The loop simply encodes the exploration of the neighbourhood selected, namely each node $u \in S$ ($v \in V \setminus S$) is considered if N_1 (N_2) is applied and accordingly the best exchange is found (function **selectBestMove**). The loop is repeated until a local optima is reached, that is when $foundMove$ is equal to **false**. The exploration can be full or partial according to the selected search strategy.

4.1 Iterated Local Search for CNP

In this section we present a general ILS solution framework to deal with CNP. In the following we refer to the basic ILS scheme discussed in [23] (cf., page 326). The main ingredients of our ILS algorithm are: the procedure to compute the initial solution, the improvement procedure and the perturbation method. The pseudo-code of our algorithm is depicted in Algorithm 5.

Algorithm 5: A ILS solution framework for CNP

```
Data: graph G, K, t_max, neighbourhood N, flag CentrRanking, flag FirstImpr;
Result: S*
S_0 ⟵ Greedy1(G, K); S* ⟵ S_0;
S' ⟵ LocalSearch(G, N, S_0, FirstImpr, CentrRanking);
if f(S') < f(S*) then S* ⟵ S';
done ⟵ false;
repeat
    S'' ⟵ Perturbation(S');
    if S' ≠ S'' then
        S' ⟵ LocalSearch(G, N, S'', FirstImpr, CentrRanking);
        if f(S') < f(S*) then S* ⟵ S';
    else done ⟵ true;
until t ≥ t_max OR done;
```

In our current implementation, the initial solution S_0 is computed by applying the greedy depicted in Algorithm 2. The improvement procedure is the Local Search procedure depicted in Algorithm 4.

The perturbation method consists in fragmenting the largest connected components in the induced graph in order to reach more homogeneous components so as to attempt to reduce the number of node pairs still connected. The idea behind this is that, from a theoretical point of view, the minimization of the pairwise connectivity in the CNP results also in the maximization of the number of components while at the same time minimizing the variance in component cardinalities. Therefore, the perturbation method aims at providing a suitable diversification of the incumbent solution keeping into account these principles. The pseudo-code of the method is provided in Algorithm 6.

Algorithm 6: Perturbation method for ILS

```
Data: graph G, K, S_0
Result: S
S ⟵ S_0; n_c ⟵ 0;
repeat
    n_c ⟵ n_c + 1;
    LC ⟵ TakeLargestComponents(V \ S_0, n_c);
    V' ⟵ VertexCover(LC);
    S ⟵ S ∪ V';
    while |S| > K do
        u ⟵ arg min{f(S \ {i}) − f(S)};
        S ⟵ S \ {u};
until S ≠ S_0 and n_c < Number of Components in V \ S_0;
```

Given an incumbent solution S_0, the procedure first identifies the largest connected component (n_c=1) in the induced graph ($V \setminus S_0$), then we make the connected component disconnected by removing the set of nodes belonging to the vertex cover of the connected components. The nodes of the vertex cover are added to S generating an infeasible solution. Then a new feasible

solution is built by removing iteratively from S, $|S| - K$ times, the node which yields the minimum increase in the objective function. If the resulting solution S is different from the initial one S_0, the procedure stops and the new solution S is returned. Otherwise, the initial solution is reconsidered and the second largest component (n_c=2) is analysed. If the fragmentation of the second component does not lead to a different solution, the third largest component is evaluated and so on for the other components. The method finally stops when all components are assessed (n_c = Number of Components in $V \setminus S_0$). Finally, we remark that the connected components are computed using the *Connect* procedure while the Vertex Cover is heuristically determined by the procedure **VertexCover** reported in Section 2.3.

4.2 Variable Neighbourhood Search for CNP

In this section we present a general VNS solution framework to deal with CNP extending the work presented in [4]. In the following we refer to the basic VNS scheme discussed in [20] (cf., algorithm 7). From a notational point of view, we use h instead of k to denote the kth neighbourhood, and S instead of x to denote a solution. The main ingredients of our VNS algorithm are: the procedure to compute an initial solution, the improvement and the shake procedures. The pseudocode of our algorithm is depicted in Algorithm 7.

In our current implementation, the initial solution S_0 is computed by applying the greedy depicted in Algorithm 2. The improvement procedure is the Local Search procedure depicted in Algorithm 4.

Algorithm 7: A VNS solution framework for CNP

Data: graph G, K, $t_{\max}$, neighbourhood N, flag $CentrRanking$, flag $FirstImpr$, $OrderH$;
Result: S
1 $S \longleftarrow Greedy1(G, K)$;
2 $h \longleftarrow first(OrderH)$;
3 **repeat**
4 $\quad S' \longleftarrow$ **Shake**(S, h);
5 $\quad S'' \longleftarrow$ **LocalSearch**(G, N, S', $FirstImpr$, $CentrRanking$);
6 $\quad$ **if** $f(S'') < f(S)$ **then** $S \longleftarrow S''$; $h \longleftarrow first(OrderH)$;
7 $\quad$ **else** $h \longleftarrow next(h, OrderH)$;
8 **until** $t \geq t_{\max}$ ***OR*** $h = last(OrderH)$;

The flag $OrderH$ is concerned with the possibility of increasing h ($h = 2, \ldots, h_{\max}$) or decreasing h ($h = h_{\max}, \ldots, 2$): the functions $first$, $next$ and $last$ return respectively the first, the next and the last value of h in the increasing or decreasing sequence.

Let ϕ_u be the occurrence or frequency in which a node u belongs to a solution S. The value ϕ_u is updated ($\phi_u \longleftarrow \phi_u + 1$) in two cases: if u is added to the solution S during the neighbourhood exploration (line 5), and if u belongs to a solution improving the current best solution (line 6). The shaking procedure replaces the h most frequent nodes in S with the h least frequent nodes in $V \setminus S$. Since we are not using randomness, our shaking procedure is totally deterministic. By consequence, the whole algorithm will stop once all possible attempts have been tried without improvement, possibly before the maximum available time consumption t_{max} is reached (line 7).

4.3 Variants of the algorithms

In the previous sections, we proposed two general frameworks for solving the CNP, that is ILS and VNS. Within these frameworks, we can derive several variants of the same algorithm.

The following alternatives work both for ILS and VNS: using the N_1 or N_2 neighbourhoods, using the best or the first improvement strategies and applying the first improvement strategy combined with the ranking the nodes according to the centrality measure. Regarding VNS, two further alternatives can be considered: the former is concerned with the increasing or decreasing values of h while the latter is concerned with the update of frequency values ϕ_u, that is updated both during neighbourhood exploration and when a new best solution is found or only when a new best solution is found.

Summing up, we obtained 6 ILS and 24 VNS variants which we study in the computational experiments in the next section.

5 Computational analysis

To conduct our computational analysis of the proposed ILS and VNS algorithms and their variants we first identify the computational environment (sect. 5.1), and report a computational test comparing the running time efficiency of the neighbourhoods N_1 and N_2 with that of N_0 (sect. 5.2). In Section 5.3, the quality of the solutions provided by the proposed algorithm variants is discussed in depth. The new best known results for the benchmark instances are summarized in Section 5.4 and an analysis for determining the best variant of the algorithms from a statistical point of view is discussed in Section 5.5. Finally, we discuss the impact of the first improvement strategy and the use of betweenness centrality in terms of running time in Section 5.6.

5.1 Setting up the computational experiments

All the algorithm variants were programmed in standard C++ and compiled with `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. As stated previously, we use the graphs presented in [28] as benchmark instances and compare our results with the best known results coming from the literature. Characteristics of the benchmark instances in terms of cardinality of V and E, and the value of K, are reported in Table 1.

	Barabasi-Albert (BA)	Erdos-Renyi (ER)	Forest-Fire (FF)	Watts-Strogatz (WS)
$\|V\|,\|E\|,K$	500,499,50	235,350,50	250,514,50	250,1246,70
	1000,999,75	466,700,80	500,828,110	500,1496,125
	2500,2499,100	941,1400,140	1000,1817,150	1000,4996,200
	5000,4999,150	2344,3500,200	2000,3413,200	1500,4498,265

Table 1: Benchmark instances from [28]: main characteristics.

In Section 2 we discussed a list of competing algorithms. Given the profound differences between all the heuristics considered here, it is tricky to set up computational conditions that allow a fair comparison of their results. Since the metaheuristics described in Section 2.1 were run 30 times retaining the best results after 30 trials (best results reported in [28]), we ran the greedy-based algorithms reported in Section 2.3 until 30 feasible solutions were encountered (results are presented in [2] and summed up below). The parameter Δ_K was set equal to $\Delta_K = K/2$ from preliminary computational experiments. Finally, the algorithm DFSH-Post reviewed in 2.2 was specifically designed to provide very fast solutions and works in a completely deterministic way. Numerical results for DFSH-Post are extracted from [14].

graph	K	BK	PBIL	DFSH-Post	*Greedy3d*	*Greedy4d*	Exact result
BA500	50	**195**	892	203	195	195	195
BA1000	75	**559**	3057	580	559	559	558
BA2500	100	**3722**	28044	4292	3722	3722	3704
BA5000	150	**10196**	146753	12273	10196	10196	10196
ER235	50	**313**	6700	1141	315	313	295
ER466	80	**1938**	44255	19952	1938	1993	NA
ER941	140	**8106**	229576	114166	8106	8419	NA
ER2344	200	**1112685**	2009132	1606656	1118785	1112685	NA
FF250	50	**197**	1386	302	199	197	194
FF500	110	**262**	1904	344	262	264	257
FF1000	150	**1271**	59594	1880	1288	1271	1260
FF2000	200	**4592**	256905	7432	4647	4592	4545
WS250	70	**11401**	13786	16110	11694	11401	NA
WS500	125	**4818**	53779	55163	4818	11981	NA
WS1000	200	**308596**	308596	319600	316416	318003	NA
WS1500	265	**157621**	703241	653015	157621	243190	NA

Table 2: Best known results for the 16 benchmark instances computed by algorithm competitors.

Overall results are summarized in Table 2. Column BK reports the best knowns computed among all these algorithms. The last column reports the results of the exact algorithm of [11] when it was able to converge to an optimal solution within 5 days. Table 2 showed that *Greedy3d* and *Greedy4d* are capable to compute better solutions than the other ones. Furthermore, they are able to compute solutions close to the exact ones in the case of BA and FF graphs, which are the sparsest graphs.

In order to establish a basis for comparison with these competitors, we defined a time budget for our algorithms. We have chosen the value $t_{max} = \min\{10000, t(\text{PBIL})\}$ seconds where $t(\text{PBIL})$ is the running time of PBIL reported in Table 5 of [28]. We remark that our ILS and VNS frameworks have a tendency to terminate the search before t_{max} – at least, for the smaller benchmark instances – never exceeding the total running time of the PBIL metaheuristic.

5.2 Neighbourhood running time comparison

In Section 3 we showed that N_1 and N_2 are more efficient than N_0 in terms of complexity. To provide further insights, we report the results of a test aiming at comparing the running time of the three neighbourhoods.

The test consists in generating 20 different starting solutions computed by a randomized version of the greedy algorithm described in 2, and then performing 1 full exploration of the neighbourhood considered. This test has been done for each instance and for each neighbourhood. The results of this test are reported in Table 3.

graph	$\|V\|$	K	N_0/N_1	N_0/N_2	N_2/N_1	$(\|V\|-K)/K$
ER466	466	80	57.6	12.9	4.5	4.8
ER941	941	140	106.6	18.8	5.7	5.7
FF500	500	110	54.2	11.7	4.6	3.5
FF1000	1000	150	112.3	18.7	6.0	5.7
WS500	500	125	57.2	18.7	3.1	3
WS1000	1000	200	154.6	37.8	4.1	4

Table 3: Ratio of running times between the different neighbourhood searches.

The results demonstrate the running time efficiency N_1 and N_2 with respect to N_0. Furthermore, they also confirm that N_1 runs faster than N_2 for small values of $K/|V|$. Finally, the last two columns show that the complexity ratio between N_2 and N_1 is close to the ratio $(|V|-K)/K$, since the contribution of the term $K \times D(G)$ is negligible in the graphs considered.

5.3 Solution quality of ILS and VNS

Table 4 reports the results of the six versions of the ILS algorithms proposed in Section 4.1, that is using N_1 or N_2 neighbourhoods, best (B) or first (F) improvement strategy and first improvement with use of the betweenness centrality (F-C). The results are compared with the values belonging to BK column of Table 2. Values in boldface are those equalling the best known values while those in boldface and underlined improve the best known values.

graph	K	BK	ILS-N_1-F	ILS-N_1-B	ILS-N_2-F	ILS-N_2-B	ILS-N_1-F-C	ILS-N_2-F-C
BA500	50	195	**195**	**195**	**195**	**195**	**195**	**195**
BA1000	75	559	**559**	**559**	**559**	**559**	**559**	**559**
BA2500	100	3722	**3722**	**3722**	**3722**	**3722**	**3722**	**3722**
BA5000	150	10196	10222	10222	10222	10222	10222	10242
ER235	50	313	**313**	**313**	343	**313**	366	324
ER466	80	1938	2272	**1924**	2490	**1924**	**1933**	**1874**
ER941	140	8106	**6878**	**7749**	**5544**	**7749**	**6502**	**5724**
ER2344	200	1112685	**1073490**	**1071968**	**1100639**	1129150	**1062536**	**1066164**
FF250	50	197	206	212	212	**195**	212	212
FF500	110	262	**261**	**261**	**261**	**261**	**261**	**261**
FF1000	150	1271	1300	1298	1288	1276	1290	1278
FF2000	200	4592	**4583**	**4583**	**4583**	**4583**	**4583**	**4583**
WS250	70	11401	**3960**	**3241**	**3857**	**3751**	**3290**	**3266**
WS500	125	4818	**2353**	**2434**	**2383**	**2384**	**2335**	**2282**
WS1000	200	308596	**148856**	**156909**	**151900**	**159201**	**158404**	**158404**
WS1500	265	157621	**14681**	**16641**	**14926**	**15287**	**15608**	**16357**

Table 4: Results of the 6 ILS algorithms: N_1 or N_2 neighbourhoods, best or first improvement strategy and first improvement with use of the betweenness centrality.

The proposed ILS algorithms outperform the best known values. In particular, they significantly improve those corresponding to the most difficult instances, that is ER and WS which are the densest graphs. Only in two instances out of 16, our ILS variants are not able to replicate or to improve the best known values, although the corresponding relative gaps are negligible. Finally, there is not a clear dominance among the six versions of the ILS algorithm.

The following tables 5–7 report the results of the 24 VNS variants derived from the general framework introduced in Section 4.2: best (B), first (F) or first with the use of the betweenness centrality (F-C), and then considering the choice of neighbourhood exploration N_1 or N_2. For each version, we also consider a different strategy for updating the frequencies ϕ_u. The idea is to adopt a less intrusive strategy, that is to update ϕ_u only when u belongs to a solution improving the current best solution (line 6 of Algorithm 7). Thus, while we previously updated the frequency of a node each time it was found in a solution (full), we now only update it when it is found in a local optimum (LO). The results are compared with the values belonging to BK column of Table 2. Values in bold are those equalling the best known values while those in bold and underlined improve the best known values.

Table 5 illustrates the results for the VNS variants that provide the best improvements. The solutions computed outperform the best known values: actually, the 8 VNSs are always able to replicate or to improve the best known values. Similarly to the ILS algorithms, they obtain large improvements on the densest graphs (ER and WS). Although VNS-I-N_2-B with local optima frequency update strategy seems the more robust version since it always replicates or improves the best known values, there is not a clear dominance among the 8 proposed versions.

Table 6 and Table 7 illustrate the results for the VNS variants with first improvements and with first improvement with use of the betweenness centrality. The results of both Tables

graph	K	BK	VNS-D-N_1-B		VNS-D-N_2-B		VNS-I-N_1-B		VNS-I-N_2-B	
			full	LO	full	LO	full	LO	full	LO
BA500	50	195	**195**	**195**	**195**	**195**	**195**	**195**	**195**	**195**
BA1000	75	559	**559**	**559**	**559**	**559**	**559**	**559**	**559**	**559**
BA2500	100	3722	**3722**	**3722**	**3722**	**3722**	**3722**	**3722**	**3722**	**3722**
BA5000	150	10196	**10196**	**10196**	**10196**	**10196**	**10196**	**10196**	10222	**10196**
ER235	50	313	**306**	**303**	**297**	**301**	**297**	**301**	**298**	**306**
ER466	80	1938	**1572**	**1599**	**1589**	**1645**	**1560**	**1585**	**1599**	**1674**
ER941	140	8106	**5709**	**5821**	**6177**	**6039**	**5473**	**5992**	**5316**	**5438**
ER2344	200	1112685	1124027	**1101350**	1127739	**1109129**	**1094844**	**1055841**	**1066958**	**1034575**
FF250	50	197	**194**	**194**	**194**	**194**	**194**	**194**	198	**194**
FF500	110	262	**258**	**258**	**259**	**258**	**258**	**260**	**258**	**257**
FF1000	150	1271	**1260**	**1261**	**1269**	**1260**	**1262**	**1262**	**1261**	**1263**
FF2000	200	4592	**4576**	**4584**	**4582**	**4584**	**4570**	**4562**	**4570**	**4571**
WS250	70	11401	**9027**	**10206**	**11196**	**8209**	**11385**	**10248**	**9009**	**6903**
WS500	125	4818	**2266**	**2336**	**2207**	**2215**	**2179**	**2157**	**2236**	**2266**
WS1000	200	308596	**209640**	**208963**	**154634**	**154634**	**259697**	**255061**	**188866**	**157065**
WS1500	265	157621	201181	201181	**20098**	**26097**	**18083**	**15250**	**19524**	**19472**

Table 5: Results of the 4 VNS algorithms: N_1 or N_2 neighbourhoods, best improvement strategy, different update of parameter ϕ_u.

confirm the validity of the 16 VNS versions as previously discussed. With respect to improving the best known solutions, the use of the best improvement strategy seems to guarantee a general robustness of the algorithm. On the other hand, the first improvement approach seems able to get better solution quality as we report in the next section.

5.4 New best known results

Table 8 summarizes the best values computed by each of the ILS and VNS variants in order to determine the new best results found for each benchmark instance. Column 3 reports the old best known values, columns 4, 5, 6 and 7 report the best results of the algorithm variants reported in Table 4, 5, 6 and 7, respectively. The last two columns report the new best known values and their relative gap with the older ones. Solution values yielding the new best knowns are in boldface.

Our algorithms are able to find 13 new best known values out of 16. The remaining 3, that is those for BA instances, cannot be improved since they correspond to the optimal values reported in the last column of Table 2. The gaps reported showed that the largest improvements are obtained for the WS and ER instances. Considering also the unavailability of optimal solutions

graph	K	BK	VNS-D-N_1-F		VNS-D-N_2-F		VNS-I-N_1-F		VNS-I-N_2-F	
			full	LO	full	LO	full	LO	full	LO
BA500	50	195	**195**	**195**	**195**	**195**	**195**	**195**	**195**	**195**
BA1000	75	559	**559**	**559**	**559**	**559**	**559**	**559**	**559**	**559**
BA2500	100	3722	**3722**	**3722**	**3704**	**3704**	**3722**	**3722**	**3704**	**3704**
BA5000	150	10196	**10196**	**10196**	10218	10218	**10196**	**10196**	**10196**	10218
ER235	50	313	**306**	**303**	**306**	335	**301**	**301**	**298**	**302**
ER466	80	1938	**1562**	**1542**	**1611**	**1727**	**1561**	**1567**	**1725**	**1751**
ER941	140	8106	**5470**	**5503**	**6106**	**6289**	**5722**	**5658**	**5198**	**5628**
ER2344	200	1112685	1112994	**1067397**	**1091185**	**1097573**	**1078895**	**1052406**	**1094239**	**1034333**
FF250	50	197	**194**	**194**	198	199	198	**194**	198	199
FF500	110	262	**257**	**257**	**258**	**258**	**257**	**257**	**257**	**258**
FF1000	150	1271	**1270**	**1270**	1274	1274	**1263**	**1270**	**1265**	1273
FF2000	200	4592	**4578**	**4576**	**4584**	**4584**	**4583**	**4577**	**4549**	**4550**
WS250	70	11401	**7175**	**8833**	**11196**	**6610**	**10237**	**10413**	12457	**7186**
WS500	125	4818	**2148**	**2170**	**2209**	**2199**	**2230**	**2152**	**2209**	**2213**
WS1000	200	308596	**198494**	**200225**	**139653**	**145718**	**268500**	**256239**	**179531**	**154813**
WS1500	265	157621	**16210**	**17198**	**16549**	**26225**	**14623**	**14719**	**14619**	**15692**

Table 6: Results of the 4 VNS algorithms: N_1 or N_2 neighbourhoods, first improvement strategy, different update of parameter ϕ_u.

graph	K	BK	VNS-D-N_1-F-C		VNS-D-N_2-F-C		VNS-I-N_1-F-C		VNS-I-N_2-F-C	
			full	LO	full	LO	full	LO	full	LO
BA500	50	195	**195**	**195**	**195**	**195**	**195**	**195**	**195**	**195**
BA1000	75	559	**559**	**559**	**559**	**559**	**559**	**559**	**559**	**559**
BA2500	100	3722	**3722**	**3722**	**3704**	**3704**	**3722**	**3722**	**3704**	**3704**
BA5000	150	10196	**10196**	**10196**	10218	10218	**10196**	**10196**	**10196**	**10196**
ER235	50	313	**301**	**297**	**295**	**301**	**301**	**301**	**298**	**295**
ER466	80	1938	**1584**	**1600**	**1566**	**1569**	**1551**	**1560**	**1595**	**1655**
ER941	140	8106	**5412**	**5201**	**5420**	**5564**	**5372**	**5349**	**5556**	**5326**
ER2344	200	1112685	**1032976**	**1053202**	**1102577**	**1069662**	**1035696**	**1012849**	**1108855**	**1059239**
FF250	50	197	198	198	**194**	**194**	198	**194**	**194**	**194**
FF500	110	262	**258**	**258**	**258**	**259**	**258**	**258**	**259**	**259**
FF1000	150	1271	**1261**	**1263**	1281	1290	1273	1274	1279	1279
FF2000	200	4592	**4579**	**4592**	4593	**4569**	**4561**	**4555**	**4565**	**4551**
WS250	70	11401	**10235**	**10235**	**8721**	**7327**	**10741**	**10231**	**9274**	**7977**
WS500	125	4818	**2188**	**2135**	**2139**	**2154**	**2209**	**2196**	**2224**	**2130**
WS1000	200	308596	**208125**	**211303**	**190892**	**184609**	**268747**	**265095**	**204311**	**236279**
WS1500	265	157621	**92855**	**15009**	**16156**	**15242**	**14527**	**14538**	**14665**	**14138**

Table 7: Results of the 4 VNS algorithms: N_1 or N_2 neighbourhoods, first improvement strategy with use of the betweenness centrality, different update of parameter ϕ_u.

by applying the exact algorithm (see Table 2), the WS instances seem to be the hardest to solve. As a matter of fact, they are the densest graphs in our benchmark set: for the same number of nodes, WS has 4 times more arcs than BA and twice more than ER and FF; moreover, they present a "small-world" structure where every node's degree is close to the average degree and no node of degree 1 is present. Regarding the results of the algorithms, the VNS versions with first improvement strategy are those yielding 11 new best known values out of 13. The ILS and VNS-B versions are able to find unique new best known values only for instances WS250 and FF1000, respectively.

5.5 Statistical comparison

In order to determine, if possible, the best algorithm variant among all our proposals, we compute the Friedman average ranking [17] over the 16 benchmark instances. We also compute the average gap of each algorithm to the best found solution, that is, for each instance we extract the best found value between all the algorithms and compute the gap of each algorithm to this best found value; then for each algorithm we compute the average gap over all 16 instances.

graph	K	old BK	ILS	VNS-B	VNS-F	VNS-F-C	new BK	gap %
BA500	50	195	195	195	195	195	195	0.00%
BA1000	75	559	559	559	559	559	559	0.00%
BA2500	100	3722	3722	3722	**3704**	**3704**	3704	-0.48%
BA5000	150	10196	10222	10196	10196	10196	10196	0.00%
ER235	50	313	313	297	298	**295**	295	-5.75%
ER466	80	1938	1874	1560	**1542**	1551	1542	-20.43%
ER941	140	8106	5544	5316	**5198**	5201	5198	-35.87%
ER2344	200	1112685	1062536	1034575	1034333	**1012849**	1012849	-8.97%
FF250	50	197	195	**194**	**194**	**194**	194	-1.52%
FF500	110	262	261	**257**	**257**	258	257	-1.91%
FF1000	150	1271	1276	**1260**	1263	1261	1260	-0.87%
FF2000	200	4592	4583	4562	**4549**	4551	4549	-0.94%
WS250	70	11401	**3241**	6903	6610	7327	3241	-71.57%
WS500	125	4818	2282	2179	2148	**2130**	2130	-55.79%
WS1000	200	308596	148856	154634	**139653**	184609	139653	-54.75%
WS1500	265	157621	14681	18083	14619	**14138**	14138	-91.03%

Table 8: New best known results.

Both results are reported in Table 9 and favour algorithms VNS-I-N_2-FC-LO and ILS-N_2-FC. Unfortunately, the *Wilcoxon's matched-pairs signed-ranks test* [32] – between each algorithm and the best ranking – indicates that the dominance cannot be considered statistically significant.

algorithm	VNS-D-N_1-F-full	VNS-D-N_1-B-full	VNS-D-N_2-F-full	VNS-D-N_2-B-full	VNS-I-N_1-F-full	VNS-I-N_1-B-full
avg. ranking	13.25	16.47	17.44	16.28	15.25	13.91
avg. gap to BK	0.07	0.14	0.08	0.09	0.09	0.10
algorithm	VNS-I-N_2-F-full	VNS-I-N_2-B-full	VNS-D-N_1-F-LO	VNS-D-N_1-B-LO	VNS-D-N_2-F-LO	VNS-D-N_2-B-LO
avg. ranking	11.97	14.81	12.78	17.81	17.84	15.75
avg. gap to BK	0.08	0.09	0.08	0.14	0.10	0.10
algorithm	VNS-I-N_1-F-LO	VNS-I-N_1-B-LO	VNS-I-N_2-F-LO	VNS-I-N_2-B-LO	VNS-D-N_1-FC-full	VNS-D-N_2-FC-full
avg. ranking	12.94	14.19	13.84	13.09	13.59	13.38
avg. gap to BK	0.09	0.09	0.07	0.07	0.08	0.07
algorithm	VNS-I-N_1-FC-full	VNS-I-N_2-FC-full	VNS-D-N_1-FC-LO	VNS-D-N_2-FC-LO	VNS-I-N_1-FC-LO	VNS-I-N_2-FC-LO
avg. ranking	13.31	14.19	13.59	13.38	11.75	**10.84**
avg. gap to BK	0.09	0.08	0.08	0.07	0.08	0.07
algorithm	ILS-N_1-F	ILS-N_2-F	ILS-N_1-B	ILS-N_2-B	ILS-N_1-FC	ILS-N_2-FC
avg. ranking	20.16	20.00	21.12	20.72	20.56	19.81
avg. gap to BK	0.07	0.07	0.07	0.07	0.07	**0.05**

Table 9: Average ranking and average gap to best known of each algorithm over the 16 benchmark instances. The best results are displayed in bold face.

5.6 Running time

In Table 10 we provide a comparison of the running times of a version of the VNS algorithm reflecting the general trend of our computational experience. In particular, the values of the time to best, namely the time where the best value of the objective function is found, and of the total running time (in seconds) of the VNS-I-N_1 algorithm are presented, considering the application of best and first (with and without the use of the betweenness centrality) improvement strategies. We notice that a first improvement strategy lets the VNS perform faster than a best improvement strategy, and the use of the centrality measure appreciably accelerates performances. In addition, we have seen in the previous tables that the solutions achieved by the various strategies are comparable. Therefore the betweenness centrality criterion seems to let the algorithms converge quickly without compromising the quality of the results.

6 Conclusions

We devised two metaheuristics for the CNP based on the Iterated Local Search and Variable Neighbourhood Search methodologies. We exploited two efficient neighbourhoods that make it possible to outperform the results reported in the literature. A large proportion of improving solutions has been found: 13 new best known values out of 16 benchmark instances. Moreover the remaining 3 ties with best known values are optimal values.

We also evaluated the use of different exploration strategies in order to exploit the potential of the proposed algorithms and to understand if some particular approach turns out to be especially promising. Although there is not a clear dominance of one strategy over the others, it is worth noting that a first improvement strategy yields a large number (11) of new best known values. The use of betweenness centrality plays a role in improving running times without deteriorating the quality of the results. This is a useful finding for the goal of enhancing the computational effort of the algorithms for the CNP. In general, VNS algorithms perform better than ILS methods in our setting, but all in all the results are comparable, and interestingly the ILS approach seems to be able to handle hard instances.

graph	K	VNS-I-N_1-B		VNS-I-N_1-F		VNS-I-N_1-F-C	
		Time to best	Total time	Time to best	Total time	Time to best	Total time
BA500	50	0	39	1	32	0	4
BA1000	75	1	301	0	243	0	24
BA2500	100	4	1991	4	1579	1	255
BA5000	150	10121	10121	197	10088	129	1971
ER235	50	8	34	8	22	11	15
ER466	80	114	410	38	160	65	116
ER941	140	783	5356	24	1935	56	726
ER2344	200	7955	10055	6113	10025	3518	7589
FF250	50	33	61	4	25	1	3
FF500	110	270	761	153	491	59	102
FF1000	150	4474	7654	1594	4075	380	619
FF2000	200	6721	10063	21	10041	878	2047
WS250	70	8	33	7	14	8	17
WS500	125	460	2414	143	1587	200	740
WS1000	200	1568	1963	553	1162	502	771
WS1500	265	8880	10057	9299	10003	2565	10102

Table 10: Time to best and total running time (in seconds) of a version of the VNS algorithm with best improvement, first improvement and first improvement with the use of betweenness centrality values (and full update of the frequence parameter ϕ_u).

Even though the benchmark instances are not real networks and some of them are random graphs with no particular features of real complex networks (ER), some of them display scale free structures (BA, FF) or small-world structures as encountered in many social networks, airline networks or fire propagation situations as discussed in [28, 14]. This is a good hint that our algorithms may be expected to be relatively good for real-world critical node problem cases according to [22, 18] which show that the no free lunch hypothesis does not seem to apply to most real-world problems.

Future work will be devoted to devise auxiliary data structures and different neighbourhoods. The trade-off between the quality of the solutions and computational times could be more extensively addressed, in order to evaluate the application of our algorithms to larger instances and real networks as well. The use of centrality measures to deal with the CNP, in relation also with the structures of the graphs, will be further investigated. Finally, another appealing area of investigation would be to develop effective metaheuristics to cope with other versions of the CNP (e.g., the Cardinality Constrained CNP [6, 25, 26]) or with weighted graphs.

References

[1] Network interdiction applications and extensions, 2014. Virtual issue on Networks.

[2] B. Addis, R. Aringhieri, A. Grosso, and P. Hosteins. Hybrid constructive heuristics for the critical node problem. *Submitted for publication*, 2014. `http://www.optimization-online.org/DB_HTML/2015/02/4764.html`.

[3] B. Addis, M. Di Summa, and A. Grosso. Removing critical nodes from a graph: complexity results and polynomial algorithms for the case of bounded treewidth. *Discrete Applied Mathematics*, 16-17:2349–2360, 2013.

[4] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia. VNS solutions for the Critical Node Problem. In *The 3rd International Conference on Variable Neighborhood Search (VNS'14)*, Electronic Notes in Discrete Mathematics, pages 37–44, February 2015.

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

[6] V. Boginski and C. W. Commander. Identifying critical nodes in protein-protein interaction networks. In S. Butenko, W. A. Chaovalitwongse, and P. M. Pardalos, editors, *Clustering Challenges in Biological Networks*, pages 153–168. World Scientific Publishing, 2009.

[7] S. P. Borgatti. Identifying sets of key players in a network. *Computational and Mathematical Organization Theory*, 12:21–34, 2006.

[8] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

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

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

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

[12] T. N. Dinh and M. T. Thai. Precise structural vulnerability assessment via mathematical programming. In *MILCOM 2011 – 2011 IEEE Military Communications Conference*, pages 1351–1356. IEEE, 2011.

[13] T. N. Dinh, Y. Xuan, M. T. Thai, P. M. Pardalos, and Znati T. On new approaches of assessing network vulnerability: Hardness and approximation on approximation of new optimization methods for assessing network vulnerability. *IEEE/ACM Transactions on Networking*, 20:609–619, 2012.

[14] M. Edalatmanesh. *Heuristics for the Critical Node Detection Problem in Large Complex Networks*. PhD thesis, Faculty of Mathematics and Science, Brock University, St. Catharines, Ontario, 2013.

[15] L. C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215 – 239, 1978–1979.

[16] L. C. Freeman, D. Roeder, and R. R. Mulholland. Centrality in social networks: ii. experimental results. *Social Networks*, 2(2):119 – 141, 1979–1980.

[17] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701, 1937.

[18] C. Garcia, F.J. Rodriguez, and M. Lozano. Arbitrary function optimisation with metaheuristics. no free lunch and real-world problems. *Soft Computing*, 16:2115–2133, 2012.

[19] P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable Neighbourhood Search: methods and applications. *4OR*, 6:319–360, 2008.

[20] P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable Neighbourhood Search: methods and applications. *Ann Oper Res*, 175:367–407, 2010.

[21] J. Hopcroft and R. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, June 1973.

[22] C. Igel and M. Toussaint. On classes of functions for which no free lunch results hold. *Information Processing Letters*, 86:317–321, 2003.

[23] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated Local Search: Framework and applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 363–397. Springer US, 2010.

[24] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

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

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

[27] J.R. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

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

[29] M. Ventresca and D. Aleman. Network robustness versus multi-strategy sequential attack. *Journal of Complex Networks*, 3:126–146, 2015.

[30] A. Veremyev, V. Boginski, and E. Pasiliao. Exact identification of critical nodes in sparse networks via new compact formulations. *Optimization Letters*, 8:1245–1259, 2014.

[31] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[32] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.

[33] R. Wollmer. Removing arcs from a network. *Operations Research*, 12:934–940, 1964.

[34] R. K. Wood. Deterministic network interdiction. *Mathematical and Computer Modelling*, 17:1–18, 1993.