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)
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.
DOI: 10.1002/net.21671

The definitive version is available at:

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 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 $G[V \setminus S]$

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

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 \( M \) is used to model the probability distribution over the possible values for each element of the solution. At each iteration \( t \), \( M_t \) 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 \( M_t \) 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
1 Initialize M₀;
2 for t = 1 to ω do
3     H ← generateSamples(h, Mₜ₋₁);
4     if f(bestSol(H)) < f(B) then B ← bestSol(H);
5     Mₜ ← (1 − α)Mₜ₋₁ + αB;
6     Mₜ ← mutate(β,γ,Mₜ);

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 ∈ V and the fact that a node u ∈ 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 ∈ 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 ∈ V from S, that is

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

The second rule adds a node u ∈ V to S, that is

Greedy rule 2: \[ S' = S \cup \{u\} \quad \text{s.t.} \quad u = \arg \max \{f(S) - f(S \cup \{u\})\}. \] (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 = ∅ 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 ∆K) for ℓ times in order to perturb a feasible solution with K nodes. When the best solution is not improved after generating 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, ∆K, ℓ, I;

Result: S*

1. n ← 0; S ← VertexCover(G); count ← 0;
2. repeat
3.   while |S| > K − ∆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 + ∆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
\( x = (x_i : i \in V) \) where \( x_i = 1 \) iff node \( i \in V \) is deleted, and \( y = (y_{ij} : i, j \in V, i < j) \) where
\( y_{ij} = 1 \) iff the node pair \( \{i, j\} \) is disconnected in the residual graph \( G[V \setminus S] \).

\[
\text{maximize } z = \sum_{i,j \in V: i<j} y_{ij} \\
\text{subject to } \sum_{i \in V} x_i \leq K \\
(x, y) \in X \\
x_i \in \{0, 1\} \quad i \in V \\
y_{ij} \in \{0, 1\} \quad i, j \in V, i < j.
\]

The polytope \( X \) links the values of \( x \) and \( y \) 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:

\[
X = \left\{ x, y \mid \sum_{r \in V(P)} x_r \geq y_{ij} \text{ for each } P \in \mathcal{P}(i, j), i, j \in V, i < j \right\} \tag{4}
\]

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
\( O(|V|^2) \) 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 \( O(|V|+|E|) \).
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
This can be very time consuming, with a complexity growing in the worst case with the cube of the number of vertices, that is \( O(|V|^3) \).

**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 \( v' = \arg \max \{ f(S) - f((S \cup \{u\}) \setminus \{v'\}) \} \). This can be done by reintroducing \( u \) in the graph (\( S = S \setminus \{u\} \)) and performing a modified Connect (with complexity \( O(|V| + |E|) \)) 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 \( O(K(|V| + |E|)) \), which is less than the one of \( N_0 \). In addition, the worst case complexity is reduced to \( O(|V|^2) \).

**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 \( O(|V| + |E|) \). 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 \( O(D(G)) \) 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 \( O(|V| - K)(|V| + |E| + K \times D(G)) \).

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 \( v 
eq v^* \) 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
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 $\sigma_{s,t}^{u}$ 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 $\sigma_{s,t}^{u}$ over all different pairs of node $s, t \in V$, that is

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

We implemented Brandes algorithm [8], running in $O(|V| \times |E|)$, 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 $\text{foundMove} \leftarrow \text{true}$; |
| 2 while $\text{foundMove}$ do |
| 3 \hspace{1em} $(u^*, v^*) \leftarrow \text{selectBestMove} (N, S, V, \text{FirstImpr}, \text{CentrRanking});$ |
| 4 \hspace{1em} if $f(S \cup \{v^*\} \setminus \{u^*\}) < f(S)$ then $S \leftarrow S \cup \{v^*\} \setminus \{u^*\}$; |
| 5 \hspace{1em} else $\text{foundMove} \leftarrow \text{false}$; |
| 6 $S^* \leftarrow S$; |
```

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 $\text{FirstImpr}$ and $\text{CentrRanking}$: $\text{FirstImpr}$ is equal to $\text{true}$ when a first improvement strategy is applied, $\text{false}$ otherwise; $\text{CentrRanking}$ is equal to $\text{true}$ 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 $\text{selectBestMove}$). The loop is repeated until a local optima is reached, that is when $\text{foundMove}$ is equal to $\text{false}$. 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 \( G \), \( K \), \( t_{\text{max}} \), neighbourhood \( N \), flag \( \text{CentrRanking} \), flag \( \text{FirstImpr} \);
Result: \( S^* \)

1. \( S_0 \leftarrow \text{Greedy1}(G, K) \);
2. \( S^* \leftarrow S_0 \);
3. if \( f(S') < f(S^*) \) then \( S^* \leftarrow S' \);
4. done \leftarrow false;
5. repeat
6. \( S'' \leftarrow \text{Perturbation}(S') \);
7. if \( S' \neq S'' \) then
8. \( S' \leftarrow \text{LocalSearch}(G, N, S'', \text{FirstImpr}, \text{CentrRanking}) \);
9. if \( f(S') < f(S^*) \) then \( S^* \leftarrow S' \);
10. else done \leftarrow true;
11. until \( t \geq t_{\text{max}} \) 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: \( S \)

1. \( S \leftarrow S_0; n_c \leftarrow 0; \)
2. repeat
3. \( n_c \leftarrow n_c + 1; \)
4. \( LC \leftarrow \text{TakeLargestComponents}(V \setminus S_0, n_c); \)
5. \( V' \leftarrow \text{VertexCover}(LC); \)
6. \( S \leftarrow S \cup V'; \)
7. while \( |S| > K \) do
8. \( u \leftarrow \arg \min \{ f(S \setminus \{i\}) - f(S) \}; \)
9. \( S \leftarrow S \setminus \{u\}; \)
10. until \( S \neq S_0 \) and \( n_c < \text{Number of Components in } V \setminus S_0; \)

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 ($n_c = \text{Number of Components in } V \setminus S_0$). 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 $k$th 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 $G$, $K$, $t_{\text{max}}$, neighbourhood $N$, flag CentrRanking, flag FirstImpr, OrderH;

**Result:** $S$

1. $S \leftarrow \text{Greedy1}(G, K)$;
2. $h \leftarrow \text{first}(\text{OrderH})$;
3. repeat
4. $S' \leftarrow \text{Shake}(S, h)$;
5. $S'' \leftarrow \text{LocalSearch}(G, N, S', \text{FirstImpr}, \text{CentrRanking})$;
6. if $f(S'') < f(S)$ then $S \leftarrow S''$; $h \leftarrow \text{first}(\text{OrderH})$
7. else $h \leftarrow \text{next}(h, \text{OrderH})$;
8. until $t \geq t_{\text{max}}$ OR $h = \text{last}(\text{OrderH})$;

The flag OrderH is concerned with the possibility of increasing $h$ ($h = 2, \ldots, h_{\text{max}}$) or decreasing $h$ ($h = h_{\text{max}}, \ldots, 2$): 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 $\phi_u$ be the occurrence or frequency in which a node $u$ belongs to a solution $S$. The value $\phi_u$ is updated ($\phi_u \leftarrow \phi_u + 1$) 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 $t_{\text{max}}$ 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 $\phi_u$, 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.

| $|V|,|E|,K$ | Barabasi-Albert (BA) | Erdos-Renyi (ER) | Forest-Fire (FF) | Watts-Strogatz (WS) |
|-----------|---------------------|------------------|------------------|---------------------|
| 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 $\Delta_K$ 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].
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 $t_{\text{max}} = \min\{10000, t(\text{PBIL})\}$ 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 $t_{\text{max}}$ – 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.

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.

<table>
<thead>
<tr>
<th>graph</th>
<th>$K$</th>
<th>BK</th>
<th>ILS-$N_1$-F</th>
<th>ILS-$N_1$-B</th>
<th>ILS-$N_2$-F</th>
<th>ILS-$N_2$-B</th>
<th>ILS-$N_1$-F-C</th>
<th>ILS-$N_2$-F-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA500</td>
<td>50</td>
<td>195</td>
<td>195</td>
<td>195</td>
<td>195</td>
<td>195</td>
<td>195</td>
<td>195</td>
</tr>
<tr>
<td>BA1000</td>
<td>75</td>
<td>559</td>
<td>559</td>
<td>559</td>
<td>559</td>
<td>559</td>
<td>559</td>
<td>559</td>
</tr>
<tr>
<td>BA2500</td>
<td>100</td>
<td>3722</td>
<td>3722</td>
<td>3722</td>
<td>3722</td>
<td>3722</td>
<td>3722</td>
<td>3722</td>
</tr>
<tr>
<td>BA5000</td>
<td>150</td>
<td>10196</td>
<td>10222</td>
<td>10222</td>
<td>10222</td>
<td>10222</td>
<td>10222</td>
<td>10222</td>
</tr>
<tr>
<td>ER235</td>
<td>50</td>
<td>313</td>
<td>313</td>
<td>313</td>
<td>343</td>
<td>313</td>
<td>366</td>
<td>324</td>
</tr>
<tr>
<td>ER466</td>
<td>80</td>
<td>1938</td>
<td>2272</td>
<td>1924</td>
<td>2490</td>
<td>1924</td>
<td>1933</td>
<td>1874</td>
</tr>
<tr>
<td>ER941</td>
<td>140</td>
<td>8106</td>
<td>6878</td>
<td>7749</td>
<td>5544</td>
<td>7749</td>
<td>6502</td>
<td>5724</td>
</tr>
<tr>
<td>ER2344</td>
<td>200</td>
<td>112685</td>
<td>1073490</td>
<td>1071968</td>
<td>1100639</td>
<td>1129150</td>
<td>1062536</td>
<td>1066164</td>
</tr>
<tr>
<td>FF250</td>
<td>50</td>
<td>197</td>
<td>206</td>
<td>212</td>
<td>212</td>
<td>195</td>
<td>212</td>
<td>212</td>
</tr>
<tr>
<td>FF500</td>
<td>110</td>
<td>262</td>
<td>261</td>
<td>261</td>
<td>261</td>
<td>261</td>
<td>261</td>
<td>261</td>
</tr>
<tr>
<td>FF1000</td>
<td>150</td>
<td>1271</td>
<td>1300</td>
<td>1288</td>
<td>1276</td>
<td>1290</td>
<td>1276</td>
<td>1276</td>
</tr>
<tr>
<td>FF2000</td>
<td>200</td>
<td>4592</td>
<td>4583</td>
<td>4583</td>
<td>4583</td>
<td>4583</td>
<td>4583</td>
<td>4583</td>
</tr>
<tr>
<td>WS250</td>
<td>70</td>
<td>11401</td>
<td>3960</td>
<td>3241</td>
<td>3857</td>
<td>3751</td>
<td>3290</td>
<td>3266</td>
</tr>
<tr>
<td>WS500</td>
<td>125</td>
<td>4818</td>
<td>2353</td>
<td>2434</td>
<td>2383</td>
<td>2384</td>
<td>2335</td>
<td>2282</td>
</tr>
<tr>
<td>WS1000</td>
<td>200</td>
<td>308596</td>
<td>148856</td>
<td>156909</td>
<td>151900</td>
<td>159201</td>
<td>158404</td>
<td>158404</td>
</tr>
<tr>
<td>WS1500</td>
<td>265</td>
<td>157621</td>
<td>14681</td>
<td>16641</td>
<td>14926</td>
<td>15227</td>
<td>15680</td>
<td>16357</td>
</tr>
</tbody>
</table>

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 $\phi_u$. The idea is to adopt a less intrusive strategy, that is to update $\phi_u$ 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...
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

<table>
<thead>
<tr>
<th>graph</th>
<th>K</th>
<th>BK</th>
<th>VNS-D-N1-B</th>
<th>VNS-D-N2-B</th>
<th>VNS-I-N1-B</th>
<th>VNS-I-N2-B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>full</td>
<td>LO</td>
<td>full</td>
<td>LO</td>
<td>full</td>
<td>LO</td>
</tr>
<tr>
<td>BA500</td>
<td>50</td>
<td>195</td>
<td>195</td>
<td>195</td>
<td>195</td>
<td>195</td>
</tr>
<tr>
<td>BA1000</td>
<td>75</td>
<td>559</td>
<td>559</td>
<td>559</td>
<td>559</td>
<td>559</td>
</tr>
<tr>
<td>BA2500</td>
<td>150</td>
<td>3722</td>
<td>3722</td>
<td>3722</td>
<td>3722</td>
<td>3722</td>
</tr>
<tr>
<td>BA5000</td>
<td>150</td>
<td>10196</td>
<td>10196</td>
<td>10196</td>
<td>10196</td>
<td>10196</td>
</tr>
<tr>
<td>ER235</td>
<td>50</td>
<td>313</td>
<td>306</td>
<td>303</td>
<td>297</td>
<td>301</td>
</tr>
<tr>
<td>ER466</td>
<td>80</td>
<td>1394</td>
<td>1372</td>
<td>1599</td>
<td>1845</td>
<td>1560</td>
</tr>
<tr>
<td>ER941</td>
<td>140</td>
<td>8106</td>
<td>5709</td>
<td>5821</td>
<td>6177</td>
<td>6039</td>
</tr>
<tr>
<td>ER2344</td>
<td>200</td>
<td>1112685</td>
<td>1101235</td>
<td>1127759</td>
<td>1109125</td>
<td>1094484</td>
</tr>
<tr>
<td>FF250</td>
<td>50</td>
<td>197</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>198</td>
</tr>
<tr>
<td>FF500</td>
<td>110</td>
<td>262</td>
<td>258</td>
<td>258</td>
<td>258</td>
<td>258</td>
</tr>
<tr>
<td>FF1000</td>
<td>150</td>
<td>1271</td>
<td>1269</td>
<td>1269</td>
<td>1269</td>
<td>1269</td>
</tr>
<tr>
<td>FF2000</td>
<td>200</td>
<td>4592</td>
<td>4584</td>
<td>4584</td>
<td>4584</td>
<td>4584</td>
</tr>
<tr>
<td>WS250</td>
<td>70</td>
<td>11401</td>
<td>9027</td>
<td>10206</td>
<td>11196</td>
<td>8209</td>
</tr>
<tr>
<td>WS500</td>
<td>125</td>
<td>4818</td>
<td>2266</td>
<td>2236</td>
<td>2207</td>
<td>2215</td>
</tr>
<tr>
<td>WS1000</td>
<td>200</td>
<td>30856</td>
<td>209640</td>
<td>208963</td>
<td>154634</td>
<td>154634</td>
</tr>
<tr>
<td>WS1500</td>
<td>265</td>
<td>157621</td>
<td>201181</td>
<td>201181</td>
<td>201181</td>
<td>201181</td>
</tr>
<tr>
<td>WS250</td>
<td>70</td>
<td>11401</td>
<td>9027</td>
<td>10206</td>
<td>11196</td>
<td>8209</td>
</tr>
<tr>
<td>WS500</td>
<td>125</td>
<td>4818</td>
<td>2266</td>
<td>2236</td>
<td>2207</td>
<td>2215</td>
</tr>
<tr>
<td>WS1000</td>
<td>200</td>
<td>30856</td>
<td>209640</td>
<td>208963</td>
<td>154634</td>
<td>154634</td>
</tr>
<tr>
<td>WS1500</td>
<td>265</td>
<td>157621</td>
<td>201181</td>
<td>201181</td>
<td>201181</td>
<td>201181</td>
</tr>
</tbody>
</table>

Table 5: Results of the 4 VNS algorithms: N1 or N2 neighbourhoods, first improvement strategy, different update of parameter φu.

Table 6: Results of the 4 VNS algorithms: N1 or N2 neighbourhoods, first improvement strategy, different update of parameter φu.
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.

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 \( \phi_u \).

Table 8: New best known results.
Both results are reported in Table 9 and favour algorithms VNS-I-N\textsuperscript{2}-FC-LO and ILS-N\textsuperscript{2}-FC. Unfortunately, the Wilcoxon’s matched-pairs signed-ranks test\cite{32} – between each algorithm and the best ranking – indicates that the dominance cannot be considered statistically significant.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>VNS-D-N\textsuperscript{1}-F-full</th>
<th>VNS-D-N\textsuperscript{1}-B-full</th>
<th>VNS-D-N\textsuperscript{2}-F-full</th>
<th>VNS-D-N\textsuperscript{2}-B-full</th>
<th>VNS-I-N\textsuperscript{1}-F-full</th>
<th>VNS-I-N\textsuperscript{1}-B-full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. ranking</td>
<td>13.25</td>
<td>16.47</td>
<td>17.44</td>
<td>16.28</td>
<td>15.25</td>
<td>13.91</td>
</tr>
<tr>
<td>Avg. gap to BK</td>
<td>0.07</td>
<td>0.14</td>
<td>0.08</td>
<td>0.09</td>
<td>0.09</td>
<td>0.10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>VNS-I-N\textsuperscript{2}-F-full</th>
<th>VNS-I-N\textsuperscript{2}-B-full</th>
<th>VNS-I-N\textsuperscript{1}-F-LO</th>
<th>VNS-I-N\textsuperscript{1}-B-LO</th>
<th>VNS-D-N\textsuperscript{1}-F-LO</th>
<th>VNS-D-N\textsuperscript{1}-B-LO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. ranking</td>
<td>11.97</td>
<td>14.81</td>
<td>12.78</td>
<td>17.81</td>
<td>17.84</td>
<td>15.75</td>
</tr>
<tr>
<td>Avg. gap to BK</td>
<td>0.08</td>
<td>0.09</td>
<td>0.08</td>
<td>0.14</td>
<td>0.10</td>
<td>0.10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>VNS-I-N\textsuperscript{2}-F-full</th>
<th>VNS-I-N\textsuperscript{2}-B-full</th>
<th>VNS-I-N\textsuperscript{2}-F-LO</th>
<th>VNS-I-N\textsuperscript{2}-B-LO</th>
<th>VNS-D-N\textsuperscript{2}-F-full</th>
<th>VNS-D-N\textsuperscript{2}-B-full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. gap to BK</td>
<td>0.09</td>
<td>0.09</td>
<td>0.07</td>
<td>0.07</td>
<td>0.08</td>
<td>0.07</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ILS-N\textsuperscript{1}-F-LO</th>
<th>ILS-N\textsuperscript{2}-F-LO</th>
<th>ILS-N\textsuperscript{1}-B-LO</th>
<th>ILS-N\textsuperscript{2}-B-LO</th>
<th>ILS-N\textsuperscript{1}-FC</th>
<th>ILS-N\textsuperscript{2}-FC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. ranking</td>
<td>20.16</td>
<td>20.00</td>
<td>21.12</td>
<td>20.72</td>
<td>20.56</td>
<td>19.81</td>
</tr>
<tr>
<td>Avg. gap to BK</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
</tr>
</tbody>
</table>

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\textsuperscript{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.
<table>
<thead>
<tr>
<th>graph</th>
<th>$K$</th>
<th>VNS-I-$N_1$-B</th>
<th>VNS-I-$N_1$-F</th>
<th>VNS-I-$N_1$-F-C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time to best</td>
<td>Total time</td>
<td>Time to best</td>
</tr>
<tr>
<td>BA500</td>
<td>50</td>
<td>0</td>
<td>39</td>
<td>1</td>
</tr>
<tr>
<td>BA1000</td>
<td>75</td>
<td>1</td>
<td>301</td>
<td>0</td>
</tr>
<tr>
<td>BA2500</td>
<td>100</td>
<td>4</td>
<td>1991</td>
<td>4</td>
</tr>
<tr>
<td>BA5000</td>
<td>150</td>
<td>10121</td>
<td>10121</td>
<td>197</td>
</tr>
<tr>
<td>ER235</td>
<td>50</td>
<td>8</td>
<td>34</td>
<td>8</td>
</tr>
<tr>
<td>ER466</td>
<td>80</td>
<td>114</td>
<td>410</td>
<td>38</td>
</tr>
<tr>
<td>ER941</td>
<td>140</td>
<td>783</td>
<td>5356</td>
<td>24</td>
</tr>
<tr>
<td>ER2344</td>
<td>200</td>
<td>7955</td>
<td>10055</td>
<td>6113</td>
</tr>
<tr>
<td>FF250</td>
<td>50</td>
<td>33</td>
<td>61</td>
<td>4</td>
</tr>
<tr>
<td>FF500</td>
<td>110</td>
<td>270</td>
<td>761</td>
<td>153</td>
</tr>
<tr>
<td>FF1000</td>
<td>150</td>
<td>4474</td>
<td>7654</td>
<td>1594</td>
</tr>
<tr>
<td>FF2000</td>
<td>200</td>
<td>6721</td>
<td>10063</td>
<td>21</td>
</tr>
<tr>
<td>WS250</td>
<td>70</td>
<td>8</td>
<td>33</td>
<td>7</td>
</tr>
<tr>
<td>WS500</td>
<td>125</td>
<td>460</td>
<td>2414</td>
<td>143</td>
</tr>
<tr>
<td>WS1000</td>
<td>200</td>
<td>1568</td>
<td>1963</td>
<td>553</td>
</tr>
<tr>
<td>WS1500</td>
<td>265</td>
<td>8880</td>
<td>10057</td>
<td>9299</td>
</tr>
</tbody>
</table>

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 frequency parameter $\phi_u$).

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


