Memetic search for identifying critical nodes in sparse graphs

Yangming Zhou, Jin-Kao Hao*, Fred Glover

Abstract—Critical node problems involve identifying a subset of critical nodes from an undirected graph whose removal results in optimizing a pre-defined measure over the residual graph. As useful models for a variety of practical applications, these problems are computational challenging. In this paper, we study the classic critical node problem (CNP) and introduce an effective memetic algorithm for solving CNP. The proposed algorithm combines a double backbone-based crossover operator (to generate promising offspring solutions), a component-based neighborhood search procedure (to find high-quality local optima) and a rank-based pool updating strategy (to guarantee a healthy population). Specially, the component-based neighborhood search integrates two key techniques, i.e., two-phase node exchange strategy and node weighting scheme. The double backbone-based crossover extends the idea of general backbone-based crossovers. Extensive evaluations on 42 synthetic and real-world benchmark instances show that the proposed algorithm discovers 21 new upper bounds and matches 18 previous best-known upper bounds. We also demonstrate the relevance of our algorithm for effectively solving a variant of the classic CNP, called the cardinality-constrained critical node problem (CC-CNP). Finally, we investigate the usefulness of each key algorithmic component.

Index Terms—Heuristics, memetic search, critical node problems, sparse graph, complex networks.

I. INTRODUCTION

Given an undirected graph \( G = (V, E) \) with vertex (node) set \( V \) and edge set \( E \), critical node problems aim to delete a “limited” subset of nodes \( S \subseteq V \) from \( G \) such that a predefined connectivity metric over the residual graph \( G[V \setminus S] \) (i.e., the sub-graph of \( G \) induced by \( V \setminus S \)) is maximized or minimized. These deleted nodes in \( S \) are commonly called critical nodes.

CNPs have natural applications in a number of fields, such as network vulnerability assessment [36], [12], epidemic control [42], biological molecule studies [8], [39], network immunization [5], [27], network communications [14], network attacks [30] and social network analysis [9], [29], [17]. For instance, in a social network, each node corresponds to a person, edges represent some type of interactions between the individuals (e.g., friendship or collaboration), and critical nodes correspond to the “key players” of the network (e.g., leaders of the organization or community) [9].

Y. Zhou and J.K. Hao (Corresponding author) are with the Department of Computer Science, LERIA, Université d’Angers, 2 Boulevard Lavoisier, 49004 Angers, Cedex 01, France, J.K. Hao is also affiliated with the Institut Universitaire de France, 1 rue Descartes, 75231 Paris, Cedex 05, France (e-mail: yangming@info.univ-angers.fr; jin-kao.hao@univ-angers.fr). F. Glover is with the University of Colorado, Leeds School of Business, Boulder, CO, USA (e-mail: glover@opttek.com).

In this work, we are interested in the classic critical node problem (denoted as CNP hereinafter), which involves optimizing a pair-wise connectivity measure in the residual graph, i.e., minimizing the total number of connected node pairs. CNP is known to be NP-hard on general graphs [5], even if there are polynomially solvable special cases [37]. The computational challenge and wide range of applications of CNP have motivated a variety of solution approaches in the literature, including exact algorithms and heuristic algorithms. Exact solution methods [5], [38], [44] guarantee the optimality of the solutions they find, but may fail on hard and large instances. Heuristic algorithms without guaranteed optimality of their solutions have also been studied to find good approximate solutions for large and hard instances within a reasonable computing time. For instance, an early heuristic starts with an independent set and uses a greedy criterion to remove vertices from the set [5]. Another greedy algorithm using a modified depth-first search is proposed in [33]. More recently, a number of metaheuristic algorithms have been reported for CNP, including iterated local search [3], [49], variable neighborhood search [3], multi-start greedy algorithm [83], greedy randomized adaptive search procedure with path relinking [34], and genetic algorithm [4].

Neighborhood search plays a particularly important role in a metaheuristic search algorithm for CNP. In general, a neighborhood for CNP can be conveniently defined by the exchange (or swap) operation which exchanges a vertex in \( S \) against a vertex in \( V \setminus S \). However, this neighborhood has a quadratic size in terms of the number of nodes, making its exploration highly expensive.

To alleviate this difficulty and create an effective neighborhood search procedure, we propose a component-based neighborhood, which relies on a two-phase node exchange strategy and a node weighting technique. First, the two-phase node exchange strategy decomposes the exchange operation into two phases: a removal phase and an add phase, and performs them separately. Moreover, based on the fact that some swaps are irrelevant for optimizing the objective function, we constrain the exchange operations to some specific nodes (i.e., from large connected components in the residual graph). This constrained component-based neighborhood not only considerably reduces the number of candidate solutions to consider at each search iteration, but also makes the search more focused. Moreover, to make the node exchange operation efficient, we devise a node weighting technique to provide useful information for node selection within each exchange operation. Based on this component-based neighborhood, we introduce an effective local optimization procedure and apply...
it together with a double-backbone based crossover operator which can generate new promising offspring solutions from existing parent solutions. The whole algorithm (called MACNP for memetic algorithm for CNP), which also integrates a rank-based pool updating mechanism, proves to be highly effective for solving CNP. In addition, we extend the proposed algorithm to solve a cardinality constrained version of the classic CNP, i.e., the cardinality-constrained critical node problem (CC-CNP). We summarize our main contributions as follows.

- First, the component-based neighborhood search procedure integrates two original ingredients, i.e., a two phase node exchange strategy and a node weighting scheme, which equips local optimization with a more focused and reduced neighborhood (Section III-B). The double backbone-based crossover operator extends the idea of backbone-based crossovers by adopting a double backbone structure (Section III-E).
- The proposed MACNP algorithm yields highly competitive results compared with the state-of-the-art algorithms on both synthetic and real-world benchmarks. In particular, for the set of 16 synthetic instances, MACNP discovers 6 new upper bounds and matches the best known upper bounds for all 10 remaining instances. For the set of 26 real-world instances, MACNP attains 15 new upper bounds and matches 8 previous best-known upper bounds. Compared to the state-of-the-art algorithms, our algorithm also shows a superior performance on CC-CNP and achieves the best objective values on 39 out of 42 benchmark instances, yielding 22 new upper bounds.

The remainder of the paper is organized as follows. In Section II we recapitulate the description of the critical node problem and indicate its relation to some other problems. In Section III we describe the proposed MACNP. In Section IV we present computational results for MACNP in comparison with the results of state-of-the-art algorithms. To show the generality of the proposed approach, we also verify its performance on the cardinality constrained critical node problem in Section V. In Section VI we experimentally analyze several key ingredients of the proposed approach to understand their impacts on the performance of the algorithm. Concluding remarks are provided in the last section.

II. PROBLEM DESCRIPTION AND NOTATION

Critical node problems in a graph $G = (V, E)$ aim to delete a “limited” subset of nodes $S \subseteq V$ in order to maximize or minimize a pre-defined connectivity measure over the residual graph $G[V \setminus S]$. Once the critical nodes have been removed, the residual graph $G[V \setminus S]$ can be represented by a set of connected components $\mathcal{H} = \{C_1, C_2, \ldots, C_T\}$, where a connected component $C_i$ is a set of nodes such that all nodes in this set are mutually connected (reachable by some paths), and no two nodes in different sets are connected.

Critical node problems have been extensively investigated in the last decade, and different connectivity measures have been studied according to the particular interests. These connectivity measures can be divided into three categories:

(i) To optimize the pair-wise connectivity, i.e., the total number of pairs of nodes connected by a path in the residual graph [5], [6], [1], [45], [43], [33], [2], [34], [3], [4].

(ii) To optimize the size of the largest connected component in the residual graph [47], [45], [63], [4].

(iii) To optimize the total number of connected components in the residual graph [47], [45], [4].

However, most studies in the literature have focused on the classic critical node problem (denoted as CNP), which aims to minimize the pair-wise connectivity measure [5] and belongs to the first category mentioned above. Formally, given an integer $K$, CNP is to identify a subset $S \subseteq V$ where $|S| \leq K$, such that the following pair-wise connectivity objective $f(S)$ is minimized:

$$f(S) = \sum_{i=1}^{T} \binom{|C_i|}{2}$$

where $T$ is the total number of connected components $C_i$ in the residual graph $G[V \setminus S]$.

CNP can be also considered as a problem of maximally fragmenting a graph and simultaneously minimizing the variance among the sizes of connected components in the residual graph. In other words, the resulting residual graph should be composed of a relatively large number of connected components while each connected component has a similar size [43].

In this paper, we focus on solving this classic critical node problem. In Section V we also show the applicability of our memetic algorithm to solve an important variant of CNP, i.e., the cardinality-constrained critical node problem (CC-CNP) [6], which falls into the second category mentioned above.

CNP is closely related to a variety of other NP-hard optimization problems. For example, the $k$-cut problem [24], which is a popular graph partitioning problem. Given an undirected weighted graph, the $k$-cut problem is to find a minimum cost set of edges that separates the graph into $k$ connected components. Another similar problem is the minimum contamination problem, which minimizes the expected size of contamination by removing a set of edges of at most a given cardinality [28].

III. THE PROPOSED MEMETIC APPROACH FOR CNP

In this section, we present MACNP, an effective memetic algorithm for solving the classic critical node problem. The memetic framework is a powerful general method which has been successfully applied to solve many NP-hard problems, such as graph coloring [31], graph partition [7], [47], maximum diversity [40], [50] and quadratic knapsack [13]. The memetic framework combines population-based search and single-trajectory local search to achieve a suitable balance between search intensification and diversification.

A. Solution representation and evaluation

Given a graph $G = (V, E)$ and an integer $K$ (the maximum allowed number of nodes that can be removed), a feasible solution of CNP can be represented by $S =$
To construct such a population, we first generate randomly a feasible solution (i.e., any set of at most $K$ nodes), and then we improve it by the component-based neighborhood search procedure described in Section III-D. We insert the improved solution into the population if it is different from the existing individuals of the population. Otherwise, we modify the improved solution with the exchange operation until it becomes different from all existing individuals before inserting it into the population. We repeat the procedure $p$ times to fill the population with $p$ distinct solutions.

D. Component-based neighborhood search

To ensure an effective local optimization, MACNP employs a fast and effective component-based neighborhood search (denoted by CBNS) procedure (see Algorithm 2). CBNS integrates two key techniques, i.e., a two-phase node exchange strategy and a node weighting technique.

**Algorithm 2 Component-based neighborhood search**

1: **Input:** a starting solution $S$
2: **Output:** the best solution $S^*$ found so far
3: $S^* \leftarrow S$
4: $iter \leftarrow 0$
5: while $iter < MaxIter$ do
6:   select a large component $c$ at random
7:   remove a node $u$ from component $c$ with the node weighting scheme
8:   $S \leftarrow S \cup \{u\}$
9:   $v \leftarrow \arg \min_{w \in S} \{ f(S' \setminus \{w\}) - f(S) \}$
10: $S \leftarrow S \setminus \{v\}$
11: if $f(S) < f(S^*)$ then
12:   $S^* \leftarrow S$
13:   $iter \leftarrow 0$
14: else
15:   $iter \leftarrow iter + 1$
16: end if
17: end while

1) Component-based neighborhood structure: The performance of a local search procedure greatly depends on its neighborhood structure for candidate solution exploration. A traditional neighborhood for CNP is defined by the conventional exchange operator which swaps a node $u$ from a node $v \in V \setminus S$. For a given solution, this neighborhood yields $O(K(|V| - K))$ neighboring solutions. To evaluate a neighboring solution, no incremental technique is known and a full computation from scratch is required by running the modified depth-first search algorithm of complexity $O(|V| + |E|)$ [26]. Therefore, examining the whole neighborhood requires a time of $O(K(|V| - K)(|V| + |E|))$, which becomes too expensive when many local search iterations are performed (which is usually the case).

Very recently, two other refined neighborhoods have been proposed in [3]. For a given $u \in S$, the first neighborhood aims to directly determine the node $v = \arg \max \{ f(S) - f((S \setminus \{u\}) \cup \{v\}) \}$, for all $v \in V \setminus S$ so that the swap operation between $u$ and $v$ disconnects the graph as much as possible. For $v \in V \setminus S$, the second neighborhood tries to identify the node $u = \arg \min \{ f((S \cup \{v\}) \setminus \{u\}) - f(S) \}$. The computational complexity of examining these neighborhoods...
is \( O(K(|V| + |E|)) \) and \( O((|V| - K)(|V| + |E| + K \times \text{degree}(G))) \) respectively, where \( \text{degree}(G) \) is the maximum node degree in \( G \). Even if these neighborhoods are more computationally efficient compared to the traditional swap neighborhood, they are still expensive to explore within a local search algorithm.

To overcome the limitation, we design an alternative and more efficient component-based neighborhood which is both smaller in size and more focused with respect to the optimization objective. Recall that CNP involves fragmenting the graph in order to minimize the total number of connected node pairs in the residual graph \( G[V \setminus S] \). This can be achieved by fragmenting the largest connected components in the residual graph in order to obtain more homogeneous components, which helps to minimize the number of node pairs still connected. As a result, when exchanging a node \( u \in S \) with a node \( v \in V \setminus S \), it is preferable to consider \( v \in V \setminus S \) from a large component (see Definition 1 below) instead of a small component. Let \( L \) be a predefined threshold to qualify large components. We consider only a subset of nodes \( C \) such that \( C = \bigcup_{i=1}^{K} C_i \) as candidate nodes for exchanges. Consequently, the neighborhood size is reduced to \( K|C| \), which is generally far smaller than \( K(|V| - K) \) for reasonable \( L \) values we used in this paper.

**Definition 1** (large component): A connected component in the residual graph \( G[V \setminus S] \) qualifies as a large component if the number of its nodes is greater than the predefined threshold \( L = (\max_n + \min_n)/2 \), where \( \max_n \) and \( \min_n \) are respectively the number of nodes in the largest and smallest connected components in the residual graph \( G[V \setminus S] \).

2) **Two-phase node exchange strategy:** To further reduce the size of the above component-based neighborhood, we employ a two-phase node exchange strategy which relies on an extension of a candidate list strategy also called a neighborhood decomposition strategy or a successive filtration strategy [23, 35, 50]. The two-phase node exchange strategy breaks an exchange operation on a node pair into two distinct phases: a "removal phase" removes a node from the residual graph and an "add phase" adds a removed node back to the residual graph. This type of two-phase strategy is often used in conjunction with a candidate list approach in tabu search (see, e.g., [19]). In our case, the two-phase node exchange strategy first selects a component at random among the qualified large components and removes a node \( v \) from the selected component with the node weighting scheme. For the node \( u \in S \) to be moved to the residual graph, we select the node which minimally deteriorates the objective function. With the help of the two-phase node exchange strategy, the computational effort required to examine the candidate solutions greatly decreases. Consider the CNP instance ‘BA1000’ with 1000 vertices and \( K = 75 \) as an example. Using the conventional exchange neighborhood requires consideration of \((1000 - 75) \times 75 = 69375\) candidate solutions. Instead, by adopting our two-phase node exchange strategy and only considering the qualified large connected components, only \( 75 \times 69375 = 5203625 \) candidate solutions need to be evaluated by our local search procedure because the process of removing a node from a connected component is performed regardless of its influence on the objective function. As we show in Section IV, the component-based neighborhood with the two-phase node exchange strategy makes the search much more efficient.

The two-phase exchange strategy yields an efficient neighborhood search, in which the process of selecting a node \( v \) to remove from \( G[V \setminus S] \) is performed regardless of its influence on the objective function. The process can be finished in \( O(T + nbr_u) \), where \( T \) is the number of connected components in \( G[V \setminus S] \) and \( nbr_u \) is the length of the adjacency list of node \( u \). Once a node is added into \( S \), \( S \) is an infeasible solution (\(|S| = K + 1\)) and we need to remove a node \( u \) from \( S \), where we select \( u \) to cause the minimum increase in the objective function. The evaluation of the increase in the objective function for each node in \( S \) is performed by scanning the adjacency list of the node to determine if removing the node will re-construct some existing components to form a large component in the residual graph \( G[V \setminus S] \). This operation requires time \( O(K \times nbr_u) \) where \( nbr_u \) is the length of the adjacency list of node \( u \).

3) **Node weighting scheme:** The node weighting technique is the second useful technique we adopted in the component-based neighborhood search. Weighting is a popular technique, which has been used in a number of heuristic algorithms, such as clause weighting for satisfiability problems [40], edge weighting for the minimum vertex cover problem [11], and row weighting for the set cover problem [18].

Our node weighting scheme works as follows. Each node of a large component is associated with a positive integer as its weight, initialized to 0. At each step, we randomly select a component \( C_i \) among the large connected components, and select the node \( v \in C_i \) with the largest weight (breaking ties in favor of the node with the largest degree) to move to \( S \). Simultaneously, the weights of the remaining nodes in \( C_i \) are increased by one. Additionally, when a node \( v \in C_i \) is exchanged with a node \( u \in S \), we set the weight of \( u \) to 0.

With the help of the node weighting scheme, the “hard to remove” nodes will have larger weights, and thus have a higher chance to be considered for removal from the component in the following iterations. The node weighting technique helps the search to escape from potential local optima. Our node weighting scheme follows the general penalty idea for constraint satisfaction problems, which was first used in this setting in Morris’s breakout method [32]. We note that this scheme is also an instance of a tabu search frequency-based memory (see, e.g., the six frequency-based memory classes proposed earlier in [19] and their refinements in [22]). To the best of our knowledge, it is the first time that a node weight learning technique is applied to a heuristic procedure for CNP.

**E. Double backbone-based crossover**

Crossover is another important ingredient of the MACNP algorithm. It should be noted that the meaning of crossover has changed from the genetic conception adopted in the early formulation of memetic algorithms. The modern conception embraces the principle of structured combinations introduced in [20], where solutions are combined by domain specific heuristics that map them into new solutions faithful to the
structure of the problems considered. (A similar evolution in the notion of crossover has been occurring within genetic algorithms to incorporate the notion of structured combinations, although often incompletely.) As observed in \([23]\), a successful crossover should be able to generate promising offspring solutions by inheriting good properties of the parents and introducing useful new features, while respecting the domain specific structure of the problem context. The concept of backbone has been used to design some successful crossover operators for subset selection problems \([46], [50]\). The critical node problem being a typical subset selection problem, we adopt the backbone idea and design a double backbone-based crossover operator to create structured combinations as follows.

Let \(S^1\) and \(S^2\) be two solutions of CNP. According to \(S^u\) and \(S^v\), we divide the set of elements \(V\) into three subsets of common elements, unique elements and unrelated elements, as shown in Definition 2, 3 and 4 respectively.

**Definition 2 (common elements):** The set of common elements \(X_A\) is the set of elements of \(V\) shared by \(S^1\) and \(S^2\), i.e., \(X_A = S^1 \cap S^2\).

**Definition 3 (exclusive elements):** The set of exclusive elements \(X_B\) is the set of elements of \(V\) shared by either \(S^1\) or \(S^2\), i.e., \(X_B = (S^1 \cup S^2) \setminus (S^1 \cap S^2)\) (the symmetric difference of \(S^1\) and \(S^2\)).

**Definition 4 (excluding elements):** The set of excluding elements \(X_C\) is the set of elements of \(V\) which are not included in \(S^1\) and \(S^2\), i.e., \(X_C = V \setminus (S^1 \cup S^2)\).

From two parent solutions \(S^1\) and \(S^2\) randomly selected from the population \(P\), an offspring solution \(S^0\) is constructed in three phases: (i) create a partial solution by inheriting all common elements (i.e., the first backbone), i.e., \(S^0 \leftarrow X_A\); (ii) add exclusive elements (i.e., the second backbone) into the partial solution in a probabilistic way. That is, for each exclusive element, we add it into \(S^0\) with probability \(p_0\) \((0 < p_0 < 1)\), otherwise we discard it; (iii) repair the partial solution structurally until a feasible solution is achieved. Specifically, if \(|S^0| < K\), we randomly add some elements to \(S^0\) from a random large connected component in the residual graph; Otherwise we greedily remove some elements from \(S^0\) until \(|S^0| = K\). The elements added in the first two phases form the whole backbone of the parent solutions. Therefore, the double backbones are composed of \(|X_A|\) common elements (i.e., the first backbone) and about \(p_0 \times |X_B|\) exclusive elements (i.e., the second backbone).

This double backbone-based crossover operator shares similar ideas with the crossovers proposed in \([46], [50]\), i.e., directly inheriting all common elements from its parent solutions (see Definition 2). However, our double backbone based crossover operator distinguishes itself from these crossovers by adopting the double backbone structure. That is, it also directly inherits some exclusive elements (see Definition 3) with a selection probability \(p_0\). This strategy of combining solutions by introducing elements beyond their union is shared with the approach of exterior path relinking \([21]\), which likewise has recently been found effective in discrete optimization.

**F. Rank-based pool updating**

Each offspring solution is submitted for improvement by the component-based neighborhood search procedure presented in Section III-D. Then we use a rank-based pool updating strategy to decide whether the improved offspring solution \(S^0\) should be accepted in the population. This pool updating strategy resorts to a score function to evaluate each individual. The score function not only considers the quality of the offspring but also its average distance to other individuals in the population. This strategy is inspired by the population management strategies presented in \([31], [13], [50]\).

The rank-based pool updating strategy applied in our algorithm is described in Algorithm 3. At first, we temporarily insert \(S^0\) to the population \(P\) (line 3 of Alg.3), then we evaluate all individuals of the population according to the score function \([50]\) (lines 4-8 of Alg.3) and identify the worst solution \(S^w\) with the largest score value (line 9 of Alg.3). Finally, if \(S^0\) is different from \(S^w\), we replace \(S^w\) by \(S^0\). Otherwise, we discard \(S^0\) (lines 10-12 of Alg.3).

**Algorithm 3 Rank-based pool updating strategy**

1: **Input:** a population \(P\) and an improved solution \(S^0\).
2: **Output:** a new population \(P^0\).
3: \(P' \leftarrow P \cup \{S^0\}\).
4: \(i \leftarrow 0\)
5: **while** \(i \leq p\) **do**
6: \(P' \leftarrow P' \cup \{S^w\}\)
7: \(i \leftarrow i + 1\)
8: **end while**
9: **if** \(w \neq 0\) **then**
10: **Replace** \(S^w\) **with** \(S^0\), i.e., \(P \leftarrow P' \setminus \{S^w\}\).
11: **end if**

**G. Computational complexity of MACNP**

To analyze the computational complexity of the proposed MACNP algorithm, we consider the main steps in one generation in the main loop of Algorithm 1.

As displayed in Algorithm 1 at each generation, our MACNP algorithm consists of four subroutines: parent selection, double backbone-based crossover, component-based neighborhood search and rank-based pool updating. The parent selection is very simple and takes time \(O(1)\). The double backbone-based crossover operator can be realized in time \(O(|V|K^2)\). The computational complexity of the component-based neighborhood search is \(O(K(|V| + |E|))\maxIter\), where \(\maxIter\) is the maximum allowable number of iterations without improvement. The rank-based pool updating can be achieved in time \(O(p(K^2 + p))\), where \(p\) is the population size. Hence, for each generation, the total complexity of MACNP is \(O(|V|K^2 + K(|V| + |E|))\maxIter\).

**IV. Computational studies**

This section presents computational studies to evaluate the performance of our MACNP algorithm and compare it with state-of-the-art algorithms.
A. Benchmark instances

Our computational studies were based on two benchmarks: Synthetic benchmark was originally presented in [41] and contains 16 instances classified into four categories: Barabasi-Albert (BA) graphs, Erdos-Renyi (ER) graphs, Forest-Fire (FF) graphs and Watts-Strogatz (WS) graphs.

Real-world benchmark was first presented in [4] and consists of 26 real-world graphs from various practical applications, including protein interaction, the electronic circuit, flight network, train network, electricity distribution network, social network and etc.

It is worth noting that both the benchmark instances are all sparse graphs. We use an indicator $\beta = 2|E|/(|V|(|V| + 1))$ ($0 < \beta \leq 1$) to measure the sparse degree of an instance, and we observe that $\beta \leq 0.045$ holds for all instances.

B. Experimental settings

The proposed algorithm is implemented in the C++ programming language and complied with gcc 4.1.2 and flag ‘-O3’. All the experiments were carried out on a computer equipped with an Intel E5-2670 processor with 2.5 GHz and 2 GB RAM operating under the Linux system. Without using any compiler flag, running the well-known DIMACS machine benchmark procedure dfmax on our machine requires 0.19, 1.17 and 4.54 seconds to solve the benchmark graphs r300.5, r400.5 and r500.5 respectively. Our computational results were obtained by running the MACNP algorithm with the parameter settings provided in Table I. Given that the benchmark instances have different structures, it is difficult to obtain a set of parameter values which yield uniformly the best result on all instances. To determine these values, we evaluate the performance of the algorithm for each parameter by varying the chosen parameter within a reasonable range, while fixing the other parameters as the default value of Table I.

For our experiments, we adopted a cutoff time as the stopping condition, which is a standard practice for solving CNPs [3, 44, 49]. Given the stochastic nature of the proposed algorithm, the algorithm was independently executed 30 times on each test instance like [41].

To analyze the experimental results, we resort to the well-known two-tailed sign test [15] to check the significant difference on each comparison indicator between the compared algorithms. When two algorithms are compared, the corresponding null-hypothesis is that the algorithms are equivalent. The null-hypothesis is accepted if and only if each algorithm wins on approximately $X/2$ out of $X$ instances. Since tied matches support the null-hypothesis, we split them evenly between the two compared algorithms, i.e., each one receives the value 0.5. At a significance level of 0.05, the Critical Values (CV) of the two-tailed sign test are respectively $CV_{X/2; 0.05} = 12$ and $CV_{X/2; 0.05} = 18$ when the number of instances in each benchmark is $X = 16$ and $X = 26$. Consequently, Algorithm A is significantly better than algorithm B if A wins at least $CV_{X/2; 0.05}$ instances for a benchmark of $X$ instances.

C. Performance of the MACNP algorithm

Table II shows the computational results for MACNP on the synthetic and real-world benchmarks.

### Table I

| Parameter | Description | Value | Section |
|-----------|-------------|-------|---------|
| $t_{max}$ | time limit (seconds) | 3600 | III-B |
| $p$ | population size | 20 | III-C |
| $MaxIter$ | max no improvement iteration in CBNS | 1000 | III-D |
| $p_0$ | selection probability | 0.85 | III-D |
| $\beta$ | weighting coefficient | 0.6 | III-D |

For our experiments, we adopted a cutoff time as the stopping condition, which is a standard practice for solving CNPs [3, 44, 49]. Given the stochastic nature of the proposed algorithm, the algorithm was independently executed 30 times on each test instance like [41].

To analyze the experimental results, we resort to the well-known two-tailed sign test [15] to check the significant difference on each comparison indicator between the compared algorithms. When two algorithms are compared, the corresponding null-hypothesis is that the algorithms are equivalent. The null-hypothesis is accepted if and only if each algorithm wins on approximately $X/2$ out of $X$ instances. Since tied matches support the null-hypothesis, we split them evenly between the two compared algorithms, i.e., each one receives the value 0.5. At a significance level of 0.05, the Critical Values (CV) of the two-tailed sign test are respectively $CV_{X/2; 0.05} = 12$ and $CV_{X/2; 0.05} = 18$ when the number of instances in each benchmark is $X = 16$ and $X = 26$. Consequently, Algorithm A is significantly better than algorithm B if A wins at least $CV_{X/2; 0.05}$ instances for a benchmark of $X$ instances.

### Table II

| Instance | $K$ | $K'BV$ | $\Delta f_{best}$ | $\Delta f_{avg}$ | $t_{avg}$ | #steps |
|----------|-----|--------|-------------------|------------------|-----------|---------|
| BA500    | 50  | 195    | 0                 | 0                | 0         | 5.8 × 10^2 |
| BA1000   | 75  | 558    | 0                 | 0                | 0         | 6.4 × 10^2 |
| BA2500   | 100 | 3704   | 0                 | 0.7              | 0.9       | 8.6 × 10^4 |
| BA5000   | 150 | 10196  | 0                 | 6.5              | 0.9       | 3.5 × 10^5 |
| ER235    | 50  | 295    | 0                 | 7.1              | 0.9       | 2.0 × 10^5 |
| ER466    | 80  | 1524   | 0                 | 28.5             | 0.9       | 9.5 × 10^5 |
| ER941    | 140 | 5012   | 2.1               | 458.5            | 0.9       | 1.2 × 10^7 |
| ER2344   | 200 | 95990  | -57002            | -37160           | 0.9       | 1.5 × 10^5 |
| FF250    | 50  | 194    | 0                 | 0.0              | 0.0       | 2.0 × 10^6 |
| FF500    | 110 | 257    | 0                 | 0.0              | 0.0       | 8.8 × 10^7 |
| FF1000   | 150 | 1260   | 0                 | 0.0              | 0.0       | 2.6 × 10^8 |
| FF2000   | 200 | 4545   | 0                 | 0.7              | 0.7       | 3.3 × 10^8 |
| WS250    | 70  | 3101   | -11.6             | 1140.5           | 0.7       | 6.1 × 10^8 |
| WS500    | 125 | 2078   | -6                | 4.6              | 0.7       | 1.3 × 10^8 |
| WS1000   | 200 | 113638 | -3831             | 10044.6          | 0.7       | 1.8 × 10^9 |
| WS1500   | 265 | 13167  | -69               | 88.1             | 0.7       | 7.3 × 10^9 |

- **Table I** Performance of MACNP on synthetic and real-world benchmarks.

1. Both synthetic and real-world benchmark instances are publicly available at [http://www.di.unito.it/~aringhe/cnp.html](http://www.di.unito.it/~aringhe/cnp.html).
2. The code of our MACNP algorithm will be made available at [http://www.info.univ-angers.fr/~hao/cnp.html](http://www.info.univ-angers.fr/~hao/cnp.html).
3. dfmax: [ftp://dimacs.rutgers.edu/pub/dsj/clique](ftp://dimacs.rutgers.edu/pub/dsj/clique)
4. Optimal results obtained by exact algorithm [23] within 5 days.
literature. Columns 4-8 report the detailed results of MACNP, including the difference between the best objective value $f_{best}$ and its known best value $KBV$ (i.e., $\Delta f_{best} = f_{best} - KBV$), the difference between the average objective value $f_{avg}$ and $KBV$ (i.e., $\Delta f_{avg} = f_{avg} - KBV$), the average time to attain the objective value ($t_{avg}$) and the average number of steps (i.e., exchanges) to achieve the objective value (#step).

From Table I we observe that MACNP is able to attain the best objective values for all 16 benchmark instances while yielding in particular 5 new upper bounds (see $\Delta f_{best} < 0$ in Table I). For instances ER2344 and WS250, our average objective values are also better than the previously best known upper bound (see $\Delta f_{avg} < 0$ in Table VII). To the best of our knowledge, our MACNP algorithm is the first heuristic which reaches the optimal solution 4545 of FF2000. The average time to find the optimal value 4545 is 107.6, which is far less than 5 days by the exact algorithm (as reported in [3]).

For the real-world benchmark, MACNP also shows a highly competitive performance and achieves the best objective value on 22 out of 26 instances, yielding 17 new upper bounds (see $\Delta f_{best} < 0$ in Table I) and matches 5 previous upper bounds (see $\Delta f_{best} = 0$ in Table I). Also, the average objective value achieved by our MACNP algorithm is better than the previous upper bound for 14 instances (see $\Delta f_{avg} < 0$ in Table I). However, MACNP failed to attain the known best value for three large instances (hepph, astroph, and condmat) within the time limit $t_{max} = 3600$ seconds. Indeed, this time limit is too short for the population-based MACNP algorithm to converge.

Note that in [4], a large time limit of $t_{max} = 16000$ seconds was used. When we re-run our MACNP algorithm under this condition, MACNP managed to find better solutions, including two new upper bounds (see results displayed in italic format in Table IV). This experiment demonstrates the effectiveness of our MACNP algorithm for solving the CNP on both the synthetic and real-world benchmarks.

D. Comparison with the state-of-the-art algorithms

To further assess the performance of our MACNP algorithm, we carried out detailed comparisons between MACNP and state-of-the-art heuristic algorithms. We consider 7 reference algorithms, including the dynamic restarting greedy algorithms (Greedy3d and Greedy4d) [2], iterated local search (ILS) [3], variable neighborhood search (VNS) [3], genetic algorithm (GA) [4], multi-start greedy algorithm (CNA1) [33] and a fast heuristic (FastCNP) [49].

Since the source codes of CNA1 and FastCNP are available to us, we first make a detailed comparison between MACNP and these two reference algorithms. To make a fair comparison, all the three algorithms were run on our platform with the same time limit $t_{max} = 3600$ seconds, and each algorithm was executed 30 trials to solve each instance. The comparative results are shown in Table IV. In this table, the first column provides the name of each instance (Instance), Columns 2-5 report the results of the CNA1 algorithm, including the best objective value ($f_{best}$), the average objective value ($f_{avg}$), the average time to attain the objective value ($t_{avg}$) and the number of steps to achieve the objective value (#step).

Correspondingly, columns 6-9 and columns 10-13 respectively represent the results of algorithms FastCNP and MACNP. The best values of the compared results are in bold, and when the same best objective values are achieved, fewer steps are underlined (which indicates a better performance in terms of computational efficiency). In addition, we give the number of instances (wins) for which our algorithm obtained a better performance (i.e., $f_{best}$ and $f_{avg}$) compared to the corresponding algorithms. The win values for indicators $t_{avg}$ and #steps are meaningless and are marked by ‘*’.

From Table III we observe that our MACNP algorithm significantly outperforms CNA1 and FastCNP, achieving the best objective values for 38 out of the 42 instances, and the best average objective values for 37 out of 42 instances. For the synthetic benchmark, MACNP is significantly better than CNA1 in terms of the best objective value, winning 12.5 instances (i.e., $12.5 > CV_{0.05} = 12$). Compared to FastCNP, MACNP is also very competitive and wins 10.5 instances, which is slightly smaller than the critical value $CV_{0.05} = 12$. As to the average objective value, MACNP significantly outperforms both CNA1 and FastCNP by winning 13 instances. For the real-world benchmark, MACNP also proves to be significantly better than CNA1 and FastCNP both in terms of the best objective value and the average objective value. Moreover, for the 14 instances where all three algorithms attain the same best objective values, our MACNP algorithm needs the least number of steps to reach its results (see values underlined).

We also compared our MACNP algorithm with five additional algorithms reported in the literature. As the source code of these five reference algorithms is not available, we used their best results reported in the corresponding papers. Fortunately, these five algorithms have been evaluated on the same platform (i.e., an HP ProLiant DL585 G6 server with two 2.1 GHz AMD Opteron 8425SHE processors and 16 GB of RAM) [3], [4], which is slower than our machine with a factor 0.84 according to the Standard Performance Evaluation Corporation (www.spec.org). However, their results were obtained under a longer time limit, i.e., $t_{max} \in \{7200, 10000\}$ for the most of the synthetic instances and $t_{max} \in \{3000, 16000\}$ for most of the real-world instances. Note that, in our comparison, we do not consider the simulated annealing algorithm [41], the population-based incremental learning algorithm [41], and the greedy randomized adaptive search procedure with path relinking [44] because they are completely dominated by FastCNP proposed in [49].

The comparative results of MACNP with the seven state-of-the-art heuristic algorithms on the synthetic and real-world benchmarks are summarized in Table IV. Note that the result of “Best ILS” for each instance is the best result among 6 ILS variants, and ‘Best VNS’ corresponds to the best result among all 24 VNS variants [3].

Table IV shows that our MACNP algorithm attains the best results for all instances. Specifically, MACNP finds 6 new upper bounds and reaches the best objective values for the remaining 10 instances. MACNP is significantly better than the reference algorithms except for FastCNP, respectively winning 15.0, 15.0, 12.5, 15.5, 12.0, 12.5 compared to Greedy3d.
| Instance | f_{best} | f_{avg} | t_{avg} | #steps | f_{best} | f_{avg} | t_{avg} | #steps | f_{best} | f_{avg} | t_{avg} | #steps |
|----------|---------|---------|---------|--------|---------|---------|---------|--------|---------|---------|---------|--------|
| BA500    | 195     | 195.0   | 2.5     | 1.4 × 10^9 | 195     | 195.0   | < 0.1   | 4.1 × 10^9 | 195     | 195.0   | < 0.1   | 5.8 × 10^9 |
| BA1000   | 558     | 558.7   | 5.4     | 1.1 × 10^9 | 558     | 558.7   | 29.8    | 1.6 × 10^9 | 558     | 558.7   | 0.3     | 6.4 × 10^9 |
| BA2500   | 3704    | 3704.0  | 1.7     | 1.4 × 10^9 | 3704    | 3704.0  | 649.9   | 1.7 × 10^9 | 3704    | 3704.0  | 0.7     | 2.6 × 10^9 |
| BA5000   | 10196   | 10196.0 | 103.7   | 3.8 × 10^9 | 10196   | 10201.4 | 104.9   | 1.5 × 10^9 | 10196   | 10196.0 | 6.5     | 5.2 × 10^9 |
| ER235    | 295     | 295.0   | 6.8     | 1.0 × 10^9 | 295     | 295.0   | 11.7    | 4.2 × 10^9 | 295     | 295.0   | 7.1     | 2.1 × 10^9 |
| ER466    | 1524    | 1524.0  | 825.4   | 7.2 × 10^9 | 1524    | 1524.0  | 364.9   | 7.0 × 10^9 | 1524    | 1524.0  | 28.5    | 9.5 × 10^9 |
| ER941    | 5114    | 5177.4  | 1606.1  | 6.7 × 10^9 | 5012    | 5011.3  | 1516.7  | 1.4 × 10^9 | 5012    | 5014.1  | 458.5   | 1.2 × 10^9 |
| ER2344   | 996411  | 1008867.6 | 1379.2  | 8.8 × 10^9 | 953437  | 979729.2 | 1793.8  | 3.4 × 10^9 | 902498  | 922339.5 | 2284.8  | 1.5 × 10^9 |
| FF250    | 194     | 194.0   | 150.0   | 2.2 × 10^9 | 194     | 194.0   | 1.9     | 4.6 × 10^9 | 194     | 194.0   | < 0.1   | 2.3 × 10^9 |
| FF500    | 263     | 265.0   | 197.5   | 9.4 × 10^9 | 257     | 258.4   | 55.1    | 5.3 × 10^9 | 257     | 257.0   | 0.4     | 8.7 × 10^9 |
| FF1000   | 1262    | 1264.2  | 1743.0  | 3.8 × 10^9 | 1260    | 1260.8  | 23.6    | 1.0 × 10^9 | 1260    | 1260.0  | 84.9    | 2.6 × 10^9 |
| FF2000   | 4548    | 4549.4  | 1571.0  | 2.4 × 10^9 | 4546    | 4558.3  | 11608.0 | 2.8 × 10^9 | 4545    | 4545.7  | 107.6   | 3.3 × 10^9 |
| WS250    | 3415    | 3702.8  | 1424.7  | 7.6 × 10^9 | 3085    | 3196.4  | 1938.5  | 4.2 × 10^9 | 3083    | 3089.4  | 1140.5  | 6.1 × 10^9 |
| WS2000   | 2085    | 2098.7  | 1581.4  | 1.5 × 10^9 | 2072    | 2083.3  | 1452.6  | 2.7 × 10^9 | 2072    | 2082.6  | 179.3   | 2.4 × 10^9 |
| WS1000   | 141759  | 161488.0 | 116.5   | 1.2 × 10^9 | 123602  | 127493.4 | 2120.3  | 6.2 × 10^9 | 109807  | 123862.8 | 2657.0  | 1.8 × 10^9 |
| WS1500   | 13498   | 13902.7 | 1787.2  | 5.7 × 10^9 | 13158   | 13255.7 | 15549.9 | 8.8 × 10^9 | 13098   | 13325.1 | 1012.2  | 7.3 × 10^9 |

Note that, it is meaningless to calculate the wins in terms of average time and average steps when different best objective values are achieved, and we represent them by *.

5. Application to the cardinality-constrained critical node problem

In this section, we show that our algorithm can also be used to solve other critical node problems, by testing MACNP on the cardinality-constrained critical node problem (CC-CNP). The experiments were again conducted on the synthetic and real-world benchmarks described in Section V-A.

A. Cardinality-constrained critical node problem

CC-CNP is a cardinality constrained version of the classic CNP. CC-CNP aims to identify a minimum subset \( S \subseteq V \) such that any connected component in the residual graph \( G[V \setminus S] \) contains at most \( W \) nodes where \( W \) is a given threshold value.

To approximate \( K^* \), we solve a series of CC-CNP with decreasing \( K > K^* \) values. For a fixed \( K \), we try to find

Greedy4d, Best VNS, Best ILS, GA, and CNA1. Compared to FastCNP, MACNP wins 10.5 instances, which is just slightly smaller than the critical value \( CV_{10.5}^* = 12 \). These observations indicate that compared to the state-of-the-art algorithms, our MACNP algorithm is highly competitive for solving the synthetic instances.

Similar observations are found on the real-world benchmark in Table IV. MACNP significantly outperforms the reference algorithms, respectively winning 24.0, 24.0, 24.5, 22.5, 23.5, 21.0, 21.0 instances with respect to the reference algorithms Greedy3d, Greedy4d, Best VNS, Best ILS, GA, CNA1, and FastCNP. Specifically, MACNP achieves the best objective values for 21 out of the 26 real-world instances, including 13 new upper bounds and 8 known best objective values.
**TABLE IV** Comparison between MACNP and the state-of-the-art algorithms on synthetic and real-world benchmarks.

| Instance | $K$ | $K_{BV}$ | Greedy3d | Greedy4d | Best VNS | Best ILS | GA | CNA1 | FastCNP | MACNP |
|----------|-----|----------|----------|----------|------------|------------|----|------|----------|-------|
| BA500    | 50  | 195*     | 195      | 195      | 195        | 195        | 195| 195  | 195      | 195   |
| BA1000   | 75  | 558*     | 559      | 559      | 1544       | 1544       | 558| 558  | 558      | 558   |
| BA2500   | 100 | 3704*    | 3722     | 3722     | 5690       | 5690       | 3704| 3704 | 3704     | 3704  |
| BA5000   | 150 | 10196*   | 10196    | 10196    | 10196      | 10196      | 10196| 10196 | 10196    | 10196 |
| ER235    | 50  | 295*     | 315      | 313      | 295        | 295        | 295| 295  | 295      | 295   |
| ER466    | 80  | 1524     | 1938     | 1542     | 1542       | 1542       | 1524| 1524 | 1524     | 1524  |
| ER941    | 140 | 5012     | 8106     | 5198     | 5198       | 5198       | 5120| 5120 | 5120     | 5120  |
| ER2344   | 200 | 959500   | 1118785  | 997839  | 1038048    | 996411     | 953437| 953437 | 902498*  |       |
| F250     | 50  | 194*     | 199      | 197      | 194        | 194        | 194| 194  | 194      | 194   |
| F650     | 110 | 257*     | 262      | 267      | 261        | 261        | 257| 257  | 257      | 257   |
| FF1000   | 150 | 1260*    | 1288     | 1271     | 1260       | 1260       | 1260| 1260 | 1260     | 1260  |
| FF2000   | 200 | 4545*    | 4647     | 4592     | 4549       | 4583       | 4546| 4546 | 4546     | 4546  |
| WS250    | 70  | 3101     | 1169     | 1140     | 6610       | 3241       | 3240| 3415 | 3415     | 3085  |
| WS500    | 125 | 2078     | 4818     | 11981    | 2130       | 2199       | 2085| 2072 | 2072     |       |
| WS1000   | 250 | 113638   | 316416   | 113638   | 13662      | 13498      | 13158| 13158 | 13098*   |       |
| WS1500   | 265 | 13167    | 157621   | 243190   | 2130       | 2199       | 2085| 2072 | 2072     |       |

*Optimal results obtained by exact algorithm [38] within 5 days.

Note that, for the results of our MACNP algorithm on hepph, astroph and condmat (in italic format), they are reached with $t_{max} = 16, 000$ seconds.

B. Solving CC-CNP with MACNP

To solve CC-CNP, we adapt MACNP slightly and denote the new algorithm by MACC-CNP. Basically, we replace the objective function $f$ used in MACNP with the minimization function $f'$ defined by Equation (2). To solve CC-CNP, we start with an initial $K$ value (obtained with a construction method, see below), and apply MACNP-CNP to find a set $S$ of $K$ nodes whose removal minimizes the number of exceeded nodes (i.e., minimizing $f'(S)$). If $f'(S) = 0$, then $S$ is a feasible solution of CC-CNP. At this moment, we decrease $K$ by one and solve the problem again. We repeat this process until no feasible solution can be found and report the last $S$ found with $f'(S) = 0$. This general solution procedure is inspired by a popular approach for solving the classic graph coloring problem [31], [48].

The initial $K$ value is obtained with an initial feasible solution $S^0$. We first set $S^0$ to be empty. Then we iteratively pick a node $v$ from a large connected component whose cardinality exceeds $W$ and move $v$ to $S^0$. We repeat this process until a feasible solution $S^0$ is obtained (i.e., $f'(S^0) = 0$, meaning that all components contain at most $W$ nodes). We set the initial $K$ to equal $|S^0|$.

C. Comparison with the state-of-the-art algorithms

Based on the synthetic and real-world benchmarks, we compared our MACC-CNP algorithm with the five state-of-the-art algorithms: greedy algorithms (G1 and G2) [4], genetic algorithm (GA) [3], multi-start greedy algorithm (CNA2)
reported in [4]. These results have been obtained with different GA, whose source codes are not available, we used the results in the same way as for MACNP. The source code of CNA2 critical node problem, and we adapted it for solving CC-CNP.

For our comparative study, we ran CNA2 and FastCNP with time limits of 60 and 3600 seconds, and each instance was solved 30 times. For our MACC-CNP algorithm, we also solved each instance 30 times independently under the same time limit.

The comparative results of running our MACC-CNP algorithm against the reference algorithms on the synthetic and real-world benchmarks are displayed in Table V. To analyze these results, we calculated the number of instances (wins) in which MACC-CNP proved superior according to the two-tailed sign test [15], as shown in the last row for each benchmark.

As indicated in Table V, MACC-CNP achieves the best objective values for all synthetic instances, and yielding in particular 2 new upper bounds. At a significance level of 0.05, MACC-CNP is significantly better than G1 and G2. For algorithms GA, CNA2 and FastCNP, MACCNP is better but the differences are not significant, winning 11.5, 11.0 and 9.5 instances. We also observe that MACC-CNP is very effective on the real-world benchmark. For this benchmark, at a significance level 0.05, our MACC-CNP algorithm significantly outperforms all reference algorithms. Specifically, MACC-CNP discovers new upper bounds for 15 instances and reaches the known best upper bounds on 8 out of 11 remaining instances. These observations show that MACC-CNP is also highly competitive compared to state-of-the-art algorithms for solving CC-CNP.

VI. DISCUSSION

We now analyze the key ingredients of the proposed MACNP algorithm: the two-phase node exchange strategy and node weighting scheme used in the component-based neighborhood search (Sections VI-A and VI-B) and the backbone-based crossover (Section VI-C). Based on the classic critical node problem, the experiments were carried out on 4 representative synthetic instances from different families (BA5000, ER941, FF500 and WS250) as well as 4 representative real-world instances (TreniR, H3000a, H4000 and hepth). These instances cover different classes with different sizes and have different levels of difficulties. For each algorithm variant, we ran it on each instance 15 times with a time limit $t_{max} = 3600$ seconds.

A. Benefit of the two-phase node exchange strategy

Our component-based neighborhood search decomposes the exchanging procedure into two phases, i.e., the “add-phase” and “removal-phase”, and performs them separately. To investigate the benefit of the two-phase node exchange strategy, we compare MACNP with an alternative algorithm MACNP$^0$ which uses the conventional two node exchange strategy to swap a node $u \in S$ (being the current solution) with a node $v \in V \setminus S$. MACNP$^0$ and MACNP share thus the same components except the neighborhood.

| Instance | $L$ | $K_{BV}$ | G1 | G2 | CNA2 | FA | MA |
|----------|-----|----------|----|----|------|----|----|
| BA5000   | 4   | 47       | 47 | 47 | 47   | 47 | 47 |
| BA1000   | 5   | 61       | 61 | 61 | 61   | 61 | 61 |
| BA2500   | 10  | 101      | 100| 100| 100  | 100| 100|
| BA5000   | 13  | 149      | 154| 149| 149  | 149| 149|
| ER235    | 7   | 47       | 49 | 50 | 47   | 47 | 47 |
| ER466    | 14  | 81       | 86 | 85 | 79   | 79 | 79 |
| ER941    | 25  | 139      | 149| 152| 139  | 141| 135|
| ER2344   | 1400| 204      | 252| 270| 204  | 194| 189|
| FF250    | 5   | 48       | 49 | 48 | 48   | 48 | 48 |
| FF500    | 4   | 100      | 102| 102| 100  | 100| 100|
| FF1000   | 7   | 142      | 145| 145| 142  | 142| 142|
| FF2000   | 12  | 182      | 191| 187| 182  | 182| 182|
| WS250    | 40  | 73       | 79 | 79 | 72   | 71 | 70 |
| WS500    | 15  | 126      | 144| 144| 126  | 124| 123|
| WS1000   | 500 | 162      | 195| 182| 162  | 180| 166|
| WS1500   | 30  | 278      | 339| 332| 278  | 237| 256|

| Instance | $L$ | $K_{BV}$ | G1 | G2 | CNA2 | FA | MA |
|----------|-----|----------|----|----|------|----|----|
| BA5000   | 4   | 47       | 47 | 47 | 47   | 47 | 47 |
| BA1000   | 5   | 61       | 61 | 61 | 61   | 61 | 61 |
| BA2500   | 10  | 101      | 100| 100| 100  | 100| 100|
| BA5000   | 13  | 149      | 154| 149| 149  | 149| 149|
| ER235    | 7   | 47       | 49 | 50 | 47   | 47 | 47 |
| ER466    | 14  | 81       | 86 | 85 | 81   | 79 | 79 |
| ER941    | 25  | 139      | 149| 152| 139  | 141| 135|
| ER2344   | 1400| 204      | 252| 270| 204  | 194| 189|
| FF250    | 5   | 48       | 49 | 48 | 48   | 48 | 48 |
| FF500    | 4   | 100      | 102| 102| 100  | 100| 100|
| FF1000   | 7   | 142      | 145| 145| 142  | 142| 142|
| FF2000   | 12  | 182      | 191| 187| 182  | 182| 182|
| WS250    | 40  | 73       | 79 | 79 | 72   | 71 | 70 |
| WS500    | 15  | 126      | 144| 144| 126  | 124| 123|
| WS1000   | 500 | 162      | 195| 182| 162  | 180| 166|
| WS1500   | 30  | 278      | 339| 332| 278  | 237| 256|

| Instance | $f_{best}$ | $f_{avg}$ | $f_{step}$ | $f_{best}$ | $f_{avg}$ | $f_{step}$ |
|----------|-------------|------------|------------|-------------|------------|------------|
| BA5000   | 3083        | 3257.7     | 37.6       | 3083        | 3101.1     | 39954.5    |
| ER941    | 257         | 257.0      | 134.9      | 257         | 257.0      | 21445.2    |
| FF500    | 5014        | 5171.1     | 5.3        | 5012        | 5033.7     | 29428.5    |
| WS250    | 10198       | 10208.9    | 10.7       | 10198       | 10208.9    | 102275.0   |
| TreniR   | 918         | 918.0      | 1322.5     | 918         | 918.0      | 2849170    |
| H3000a   | 3451908     | 3483709.9  | < 0.1      | 3451908     | 3483709.9  | 2883554.5  |
| H4000    | 6165357     | 6224768.8  | < 0.1      | 6165357     | 6144352.2  | 514352.2   |
| hepth    | 23964174    | 24528950.9 | < 0.1      | 23964174    | 24528950.9 | 108354.0   |

The comparative results for MACNP and MACNP$^0$ are
summarized in Table [V]. In this table, for each instance, we report the best objective value ($f_{\text{best}}$), the average objective value ($f_{\text{avg}}$), and the average number of steps per second ($\text{steps}_s$) over 15 trials achieved by each algorithm. The comparative results show that MACNP performs significantly better than MACNP_0 in terms of all comparison indicators primarily due to its much lower computational complexity per step. In each second, MACNP is able to perform seventy or even tens of thousands times more steps than that of MACNP_0.

Finally, even if we do not show a direct comparison between our component-based neighborhood with the two neighborhoods proposed in [3, Table [V]] (columns 10 and 7) shows that FastCNP (using our component-based neighborhood) clearly dominates the Best ILS algorithm (using the two complementary neighborhoods), which provides an additional indicator of the interest of the proposed component-based neighborhood.

B. Effectiveness of the node weighting scheme

To assess the effectiveness of the node weighting scheme in the component-based neighborhood search (CBNS), we compared our MACNP algorithm with its alternative algorithm MACNP_1 where the node weighting scheme is disabled in CBNS. In MACNP_1, we also decompose the exchanging operation into two phases (“add-phase” and “removal-phase”), and execute them separately. However, for “add-phase”, a node is randomly removed from a large connected component instead of selecting the node by the node weighting scheme.

Table [VII] shows the comparative results of MACNP and MACNP_1 on the tested instances, based on four indicators: best objective value ($f_{\text{best}}$), average objective value ($f_{\text{avg}}$), average time to find the best objective value ($t_{\text{avg}}$), average number of steps required to find the objective value (steps). An obvious observation from this table is that the algorithm with the node weighting scheme (i.e., MACNP) significantly outperforms MACNP_1 (which lacks the node weighting scheme) on almost all instances except WS250. For WS250, both MACNP and MACNP_1 achieve the best objective value 3083, while the average objective value 3093.8 of MACNP_1 is slightly better than 3101.1 of MACNP. More importantly, MACNP needs less time and fewer steps to achieve the best objective values. These observations demonstrate the effectiveness of the node weighting scheme.

C. Rationale behind the double backbone-based crossover

As introduced in Section III-E, we specially designed a double backbone-based crossover operator to generate offspring solutions. To investigate the rationale behind this crossover, we compare MACNP with an alternative version MACNP_2. MACNP_2 is obtained from MACNP by replacing our dedicated backbone-based crossover with a single backbone-based crossover which only treats the common elements as the backbone. Specifically, the single backbone-based crossover operator first constructs a partial solution $S^0$ by directly inheriting all the common elements of two parent solutions and then completes the partial solution $S^0$ by removing a node from a large component of the residual graph until $|S^0| = K$.

Comparative performance of MACNP and its alternative MACNP_2 in terms of the best objective value and average objective value are displayed in the left and right part of Figure [I] respectively. The X-axis indicates the instance, and the Y-axis shows the gap between our results (eight best values or average values) to the known best values in percentage, which is defined as $(f - KBV) \times 100/KBV$ where $f$ is the best objective value or average objective value, and $KBV$ is the known best objective (see 3rd column of Table [V]). A gap smaller than zero means the algorithm obtains a new upper bound for the corresponding instance.

Fig. 1. Comparative results of MACNP and its alternative MACNP_2.

From Figure [I] we observe that compared to MACNP_2, our MACNP algorithm is able to attain a better $f_{\text{best}}$ for all 8 instances including 4 new upper bounds (see values below 0 on the left part of Figure [I]). MACNP also outperforms MACNP_2 on all 8 tested instances in terms of the average objective value ($f_{\text{avg}}$), as shown in the right part of Figure [I]. This experiment confirms the value of our double backbone-based crossover.

VII. CONCLUSIONS AND FUTURE WORK

In this work, we proposed an effective memetic search approach for solving the classic critical node problem (MACNP), which combines a component-based neighborhood search for local optimization, a double backbone-based crossover operator for solution recombination and a rank-based pool updating strategy to guarantee a healthy diversity of the population. To ensure its effectiveness, the component-based neighborhood search relies on a focused and reduced neighborhood owing to its two-phase node exchange strategy and the node weighting scheme. Additionally, the double backbone-based crossover not only conserves solution features from the parent solutions, but also introduces diversity by including exclusive elements from parent solutions in a probabilistic way.

To demonstrate the competitiveness of the proposed algorithm, we evaluated MACNP on a broad range of synthetic and real-world benchmarks. The computational results showed that MACNP significantly outperforms state-of-the-art algorithms on both two benchmarks. We also assessed the performance of MACNP for solving the cardinality-constrained critical node problem (MACC-CNP), which is an important variant of the
classic CNP. Our results showed that the approach is also highly competitive compared with state-of-the-art algorithms. Finally, we performed experiments to investigate the benefit of different search components and techniques.

Future work motivated by our findings will be to investigate opportunities for further improving the performance of our algorithm by incorporating machine learning techniques (e.g. reinforcement learning and opposition-based learning). Another inviting avenue for research is to adapt the proposed algorithm by incorporating machine learning techniques.

Finally, we performed experiments to investigate the benefit of highly competitive compared with state-of-the-art algorithms.

### ACKNOWLEDGMENT

We would like to thank Dr. W. Pullan for kindly sharing the source codes of the CNA1 and CNA2 algorithms.

### REFERENCES

[1] B. Addis, M. Di Summa, and A. Grosso, “Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth,” Discrete Applied Mathematics, 161(16):2349–2360, 2013.

[2] B. Addis, R. Aringhieri, A. Grosso, and P. Hosteins, “Hybrid constructive heuristics for the critical node problem,” Annals of Operations Research, 238(1-2):637–649, 2016.

[3] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia, “Local search metaheuristics for the critical node problem,” Networks, 67(3):209–221, 2016.

[4] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia, “A general evolutionary framework for different classes of critical node problems,” Engineering Applications of Artificial Intelligence, 55:128–145, 2016.

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

[6] A. Arulselvan, C. W. Commander, O. Shylo, and P. M. Pardalos, “Cardinality-constrained critical node detection problem.” Springer Optimization and Its Applications, vol. 46. pp. 79–91, 2011.

[7] U. Benlic and J.-K. Hao, “A multilevel memetic approach for improving graph k-partitions,” IEEE Transactions on Evolutionary Computation, 15(5):624–642, 2011.

[8] V. Boginski and C. W. Commander, “Identifying critical nodes in protein–protein interaction networks,” Clustering Challenges in Biological Networks, pp. 153–167, 2009.

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

[10] S. Cai, K. Su, and A. Sattar, “Local search with edge weighting and configuration checking heuristics for minimum vertex cover,” Artificial Intelligence, 175(9):1672 – 1696, 2011.

[11] S. Cai, J. Lin, and K. Su, “Two weighting local search for minimum vertex cover,” in Proceedings of the AAAI-2015, January 25-30, 2015, Austin, Texas, USA. AAAI Press, 2015, pp. 1107–1113.

[12] X. Chen, “System vulnerability assessment and critical nodes identification.” Expert Systems with Applications, 65:212–220, 2016.

[13] Y. Chen and J. K. Hao, “Memetic search for the generalized quadratic multiple knapsack problem,” IEEE Transactions on Evolutionary Computation, 20(6):908–923, 2016.

[14] C. W. Commander, P. M. Pardalos, V. Ryabchenko, S. Uryasev, and G. Zrazhevsky, “The wireless network jamming problem,” Journal of Combinatorial Optimization, 14(4):481–498, 2007.

[15] J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” Journal of Machine Learning Research, 7:1–30, 2006.

[16] T. N. Dinh, Y. Xuan, M. T. Thai, P. M. Pardalos, and T. Znati, “On new approaches of assessing network vulnerability: hardness and approximation.” IEEE/ACM Transactions on Networking, 20(2):609–619, 2012.

[17] N. Fan and P. M. Pardalos, “Robust optimization of graph partitioning and critical node detection in analyzing networks,” Lecture Notes in Computer Science, 6508:170–183, 2010.

[18] C. Gao, X. Yao, T. Weise, and J. Li, “An efficient local search heuristic with row weighting for the unicost set covering problem,” European Journal of Operational Research, 246(3):750–761, 2015.

[19] F. Glover, “Tabu search: A tutorial,” Interfaces, 20(1):74–90, 1990.

[20] F. Glover, “Tabu search for nonlinear and parametric optimization (with links to genetic algorithms),” Discrete Applied Mathematics, 49:231–255, 1994.

[21] F. Glover, “Exterior path relinking for zero-one optimization,” International Journal of Applied Metaheuristic Computing, 5(3):1–8, 2014.

[22] F. Glover and M. Laguna, “Tabu Search,” in C. Reeves (Ed.) Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scientific Publishing, pp. 71–140, 1993.

[23] F. Glover, E. Taillard, and E. Taillard, “A user’s guide to tabu search,” Annals of Operations Research, 41(1):1–28, 1993.

[24] N. Guttmann-Beck and R. Hassin, “Approximation algorithms for minimum k-cut,” Algorithmica, 27(2):198–207, 2000.

[25] J.-K. Hao, Memetic algorithms in discrete optimization. In F. Neri, C. Cotta, P. Moscato (Eds.) Handbook of Memetic Algorithms. Studies in Computational Intelligence 379, Chapter 6, pp. 73-94, 2012.

[26] J. Hopcroft and R. Tarjan, “Algorithm 447: Efficient algorithms for graph maximum k-cut,” Algorithmica, 14(4):481–498, 1993.

[27] C. J. Kuhlman, V. S. Anil Kumar, M. V. Marathe, S. S. Ravi, and D. J. Rosenkrantz, “Finding critical nodes for inhibiting diffusion of complex contagions in social networks,” Lecture Notes in Computer Science, 6322:111–127, 2010.

[28] V. A. Kumar, R. Rajaraman, Z. Sun, and R. Sundaram, “Existence theorems and approximation algorithms for generalized network security games,” in Proc. of 2010 IEEE 30th International Conference on Distributed Computing Systems, IEEE, 2010, pp. 348–357.

[29] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. Vanbriesen, and T. Glance, “Cost-effective outbreak detection in networks,” in Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 420–429, 2007.

[30] M. Lozano, C. García-Martínez, F. J. Rodríguez, and H. M. Trujillo, “Optimizing network attacks by artificial bee colony,” Information Sciences, 377:30 – 50, 2017.

[31] Z. Lu and J.-K. Hao, “A memetic algorithm for graph coloring,” European Journal of Operational Research, 233(1):241–250, 2010.

[32] P. Morris, “The breakout method for escaping from local minima,” in Proceedings of the 11th AAAI, USA, July 11-15, 1993, pp. 40–45, 1993.

[33] W. Pullan, “Heuristic identification of critical nodes in sparse real-world graphs,” Journal of Heuristics, 21(5):577–598, 2015.
[34] D. Purevsuren, G. Cui, N. N. H. Win, and X. Wang, “Heuristic algorithm for identifying critical nodes in graphs,” *Advances in Computer Science: an International Journal*, 5(3):1–4, 2016.

[35] B. Rangaswamy, A. S. Jain, and F. Glover, “Tabu search candidate list strategies in scheduling,” *Operations Research/Computer Science Interfaces Series*, 9:15–233, 1998.

[36] Y. Shen, N. P. Nguyen, Y. Xuan, and M. T. Thai, “On the discovery of critical links and nodes for assessing network vulnerability,” *IEEE/ACM Transactions on Networking*, 21(3):963–973, 2013.

[37] 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.

[38] M. D. Summa, A. Grosso, and M. Locatelli, “Branch and cut algorithms for detecting critical nodes in undirected graphs,” *Computational Optimization and Applications*, 53(3):649–680, 2012.

[39] V. Tomaino, A. Arulselvan, P. Veltri, and P. M. Pardalos, “Studying connectivity properties in human protein–protein interaction network in cancer pathway,” *Optimization and Its Applications*, 65:187–197, 2012.

[40] J. Thornton, “Clause weighting local search for sat,” *Journal of Automated Reasoning*, 35(1-3):97–142, 2005.

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

[42] M. Ventresca and D. M. Aleman, “A derandomized approximation algorithm for the critical node detection problem,” *Computers & Operations Research*, 43:261–270, 2014.

[43] M. Ventresca and D. Aleman, “Efficiently identifying critical nodes in large complex networks,” *Computational Social Networks*, 2(1):6, 2015.

[44] A. Veremyev, V. Boginski, and E. L. Pasiliao, “Exact identification of critical nodes in sparse networks via new compact formulations,” *Optimization Letters*, 8(4):1245–1259, 2014.

[45] A. Veremyev, O. A. Prokopyev, and E. L. Pasiliao, “An integer programming framework for critical elements detection in graphs,” *Journal of Combinatorial Optimization*, 28(1):233–273, 2014.

[46] Q. Wu and J.-K. Hao, “A hybrid metaheuristic method for the maximum diversity problem,” *European Journal of Operational Research*, 231(2):452–464, 2013.

[47] B. Yuan, B. Li, H. Chen, and X. Yao, “A new evolutionary algorithm with structure mutation for the maximum balanced biclique problem,” *IEEE Transactions on Cybernetics*, 45(5):1054-1067, 2015.

[48] Y. Zhou, J.-K. Hao, and B. Duval, “Reinforcement learning based local search for grouping problems: A case study on graph coloring,” *Expert Systems with Applications*, 64:412–422, 2016.

[49] Y. Zhou and J. K. Hao, “A fast heuristic algorithm for the critical node problem,” *Genetic and Evolutionary Computation Conference Companion (GECCO’17)*, Berlin, Germany, July 15-19, 2017. Companion Material Proceedings, pages 121–122.

[50] Y. Zhou, J. K. Hao, and B. Duval, “Opposition-based memetic search for the maximum diversity problem,” *IEEE Transactions on Evolutionary Computation*, 21(5):731–745, 2017.