Memetic search for identifying critical nodes in sparse graphs

Yangming Zhou, Jin-Kao Hao*, Fred Glover

***Abstract*—Critical node problems involve finding a set of critical nodes from a 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). Extensive evaluations on 42 synthetic and real-world benchmark instances show that the proposed algorithm discovers 24 new upper bounds and matches 15 previous best-known 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 node set V and edge set E, *critical node problems* aim to identify a limited number of nodes $|S| \subseteq V$ from G such that their deletion optimizes a predefined connectivity metric over the residual graph $G[V \setminus S]$ (i.e., the sub-graph of G induced by $V \setminus S$). The deleted nodes in S are commonly called *critical nodes*.

CNPs have natural applications in a number of fields, such as network security issues [30], epidemic control [36], biological interaction networks [33], and social network analysis [8], [37]. For instance, in a social network, each node corresponds to a person, edges represent 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) [8].

In this work, we are interested in the classic critical node problem (defined in Section III and denoted as CNP hereinafter) [5]. Informally, CNP involves finding a subset S of at most K nodes, whose removal minimizes the total number of node pairs that are connected by a path in the residual graph. CNP is known to be $\mathcal{NP}$-hard on general graphs [5], even if there are polynomially solvable special cases [1], [31].

Y. Zhou is with the Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai, 200237, China (e-mail: zhou.yangming@yahoo.com). J.K. Hao (Corresponding author) is with the Department of Computer Science, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers and also with the Institut Universitaire de France, 1 rue Descartes, 75231 Paris, France (e-mail: 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).

The computational challenge and wide range of applications of CNP have motivated a variety of solution approaches in the literature, which are reviewed in Section II-A.

To enrich the set of solution methods for solving this computationally challenging problem, we propose in this work the first population-based memetic algorithm for CNP. The algorithm combines an original component-based neighborhood search procedure using an effective large component-based node exchange strategy and a node weighting scheme (to find high-quality local optima, Section IV-D), a double backbone-based crossover operator (to generate promising new solutions, Section IV-E) as well as a rank-based pool updating technique (to maintain a healthy population, Section IV-F).

We assess the effectiveness of the proposed algorithm on 16 synthetic and 26 real-world benchmark instances commonly used in the literature and especially report 24 improved best-known results (new upper bounds). We show the generality of the proposed approach by adapting the method to the related cardinality-constrained critical node problem and report additional 25 new best-known results. We also investigate each key ingredient of the proposed approach to shed light on their impacts on the performance of the method.

II. Literature Review

A. Algorithms for CNP

Due to their theoretical and practical significance, critical node problems have attracted considerable research effort in the last decade and several solution methods (both exact and heuristic algorithms) have been proposed.

In general, exact algorithms are theoretically able to guarantee the optimality of the solutions they find. These methods are particularly useful for special cases of graphs. For instance, optimal polynomial-time dynamic programming algorithms are proposed to solve two CNP variants for tree structures and series-parallel graphs in [31]. A polynomial-time dynamic programming algorithm for graphs with a bounded treewidth is studied in [1]. For general graphs, a branch-and-cut algorithm is proposed in [32] for CNP, using an integer linear programming model with a non-polynomial number of constraints. A more compact linear 0-1 formulation of the problem that requires $\Theta(n^2)$ constraints is recently developed and optimal solutions for small networks are ascertained in [38], [39].

Given the $\mathcal{NP}$-hard nature of critical node problems, it is expected that any exact algorithm for finding the optimal solution in the general case will require an exponential computation time. Consequently, as for many $\mathcal{NP}$-hard problems, heuristics constitute a valuable alternative that are used to find

high-quality approximate solutions within a reasonable time when exact methods are not usefully applicable. For instance, an early heuristic starts with an independent set and uses a greedy criterion to remove vertices [5]. Two other greedy algorithms are presented in [27], [37], using advanced greedy criteria for node selections. Moreover, several neighborhood-based local search algorithms have been studied, like simulated annealing [35], iterated local search and variable neighborhood search [3]. These algorithms typically use the unconstrained node exchange operation that swaps a node in S against a node in $V \setminus S$, leading to a large and expensive neighborhood of quadratic size. Population-based algorithms have also been investigated, including path relinking with greedy randomized adaptive search procedure [28] and genetic algorithm [4]. These algorithms maintain a pool of candidate solutions and generate new solutions from existing ones by local changes, crossover and random mutations.

B. Memetic algorithms

It is noteworthy that the popular memetic algorithms (MAs) [23] have not been investigated for CNP. Indeed, as discussed in several comprehensive surveys of memetic computation (e.g., [10], [24], [25], [26]), MAs represent a highly effective approach for solving difficult problems including numerous $\mathcal{NP}$-hard problems (e.g., graph coloring [21], graph partition and biclique [7], [41], maximum diversity [40], quadratic knapsack [11]) and routing problems [14].

MAs provide a general computational framework that typically combines population-based approaches with neighborhood-based local search. This framework offers interesting opportunities for designing search algorithms able to ensure a desirable balance between exploration and exploitation of large search spaces [10], [19], [25], [26]. To further improve the MA framework, interesting extensions incorporating learning schemes have been proposed in the literature. For example, [14] presents an evolutionary memetic computing paradigm which is capable of learning and evolving a knowledge meme that traverses different but related problem domains. [15] proposes a memetic computational paradigm based on evolutionary optimization + transfer learning for search, which mimics human problem-solving by transferring structured knowledge learned from solving related problems in the form of memes. In [44], opposition-based learning is introduced into memetic search for solving the maximum diversity problem.

This work presents the first memetic algorithm dedicated to the classic critical node problem. As demonstrated with our computational results, the proposed algorithm competes very favorably with the existing CNP approaches.

III. Problem Description and Notation

Critical node problems in a graph $G = (V, E)$ aim to remove a "limited" subset of nodes $S \subseteq V$ in order to optimize 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 disjoint connected components $\mathcal{H} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_T\}$, where a connected component $\mathcal{C}_i$ is a set of nodes such that all nodes in this set are mutually connected (reachable by a path), and there is no edge between any two connected components.

According to the particular interests, different connectivity measures have been studied, which can be divided into three categories.

(i) To optimize the pair-wise connectivity, i.e., the number of pairs of nodes connected by a path in the residual graph [1], [2], [3], [4], [5], [13], [27], [28], [37], [39].
(ii) To optimize the size of the largest connected component in the residual graph [4], [27], [31], [39].
(iii) To optimize the number of connected components in the residual graph [31], [39], [4].

One notices that most studies in the literature have focused on the classic Critical Node Problem (denoted as CNP), which involves minimizing the pair-wise connectivity measure [5] and belongs to the first category mentioned above. Formally, given a graph $G = (V, E)$ and an integer K, CNP is to identify a subset $S \subseteq V$ such that $|S| \leqslant K$, whose removal minimizes the following pair-wise connectivity objective $f(S)$:

$$f(S) = \sum_{i=1}^{T} \binom{|\mathcal{C}_i|}{2} \tag{1}$$

where T is the total number of connected components $\mathcal{C}_i$ in the residual graph $G[V \setminus S]$.

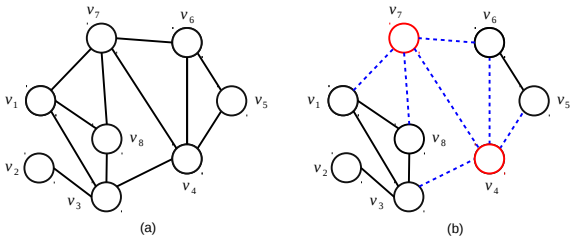

Fig. 1. A CNP example. (a) An undirected graph G with 8 nodes and $K = 2$. (b) A candidate solution for the instance (i.e., a subset $S = \{v_4, v_7\}$) and its cost $f(S)$ can be computed as $\binom{2}{2} + \binom{4}{2} = 7$.

Fig. 1 shows a CNP instance (i.e., the graph of Fig. 1(a) with eight nodes and $K = 2$) and a candidate solution $S = \{v_4, v_7\}$ (Fig. 1(b)), whose removal from the graph leads to two connected components induced by $\{v_1, v_2, v_3, v_8\}$ and $\{v_5, v_6\}$ with an objective value $f(S) = 7$.

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 [37].

In this paper, we focus on solving this classic critical node problem. In Section VI, 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.

IV. The Proposed Memetic Approach for CNP

Given a CNP instance defined by $G = (V, E)$ and $K \in \mathbb{Z}^+$, let S ($|S| < K$) be a candidate solution of G. It is easy to

verify that any solution $S' = S \cup \{u\}$ with one more node $u \in V \setminus S$ is a feasible solution no worse than S (i.e., $f(S') \leqslant f(S)$). Indeed, if u is an isolated node in $G[V \setminus S]$, adding u to S has no impact on the objective value. Otherwise, let u belong to a connected component, say $\mathcal{C}_i$, moving u from $\mathcal{C}_i$ to S reduces the size of $\mathcal{C}_i$ by one or disconnects $\mathcal{C}_i$ into two or more new components. In both cases, we have $f(S') \leqslant f(S)$. Since K is the largest possible value for a solution to be feasible, wa can safely consider only candidate solutions S with exactly K selected elements.

A. Solution representation and evaluation

Given a graph $G = (V, E)$ and an integer K, any subset $S \subset V$ of K nodes is a feasible solution and can be represented as $S = \{v_{S(1)}, v_{S(2)}, \ldots, v_{S(K)}\}$ $(1 \leqslant S(i) \neq S(j) \leqslant |V|$ for all $i \neq j)$ where $S(l)$ $(1 \leqslant l \leqslant K)$ is the index of a selected node in S. Therefore, the solution space Ω contains all possible subsets $S \subseteq V$ such that $|S| = K$.

According to Equation (1), the corresponding objective value $f(S)$ of a feasible solution S is the total number of node pairs still connected by a path in the residual graph $G[V \setminus S]$. $f(S)$ can be computed in $O(|V| + |E|)$ time with a modified depth-first search algorithm by identifying the connected components of a graph [20] as follows. One finds the connected components of a graph by performing the depth-first search on each connected component. Each new node visited is marked. When no more nodes can be reached along the edges from the marked nodes, a connected component is found. Then, an unvisited node is selected, and the process is repeated until the entire graph is explored.

B. General scheme

The proposed MACNP algorithm is composed of four main procedures: a population initialization procedure, a component-based neighborhood search procedure, a double backbone-based crossover procedure and a rank-based pool updating procedure. MACNP starts from a set of distinct elite individuals which are obtained by the population initialization procedure (Section IV-C). At each generation, an offspring solution is generated by the double backbone-based crossover procedure (Section IV-E). This offspring solution is further improved by the component-based neighborhood search procedure (Section IV-D) and then considered for acceptance by the rank-based pool updating procedure (Section IV-F). The process is repeated until a stopping condition (e.g., time limit) is satisfied. We show the flowchart and pseudo-code of the MACNP algorithm in Fig. 2 and Algorithm 1. We present the key procedures in the following sections.

C. Population initialization

Our MACNP algorithm starts its search with an initial population composed of diverse and high-quality solutions. 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 IV-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.

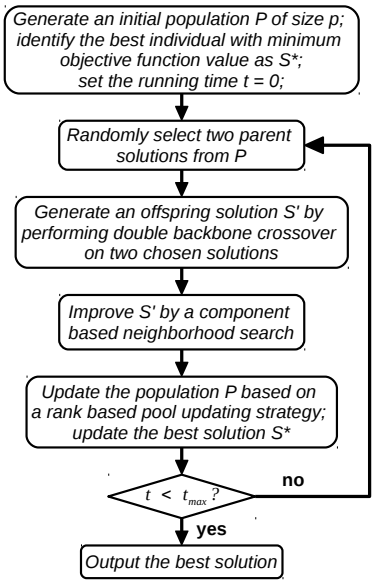

Fig. 2. Flowchart of the proposed MACNP algorithm

Algorithm 1: Pseudo-code of MACNP algorithm.

Input: An undirected graph G = (V,E) and an integer K
Output: The best solution S^* found so far

```
begin
    // build an initial population, Section IV-C
    P = {S^1, S^2, ..., S^p} ← PoolInitialize();
    S* ← arg min{f(S^i) : i = 1, 2, ..., p};
    while a stopping condition is not reached do
        randomly select two parents S^i and S^j from P;
        // generate an offspring by crossover, Section IV-E
        S' ← DoubleBackboneBasedCrossover(S^i, S^j);
        // perform a local search, Section IV-D
        S' ← ComponentBasedNeighborhoodSearch(S');
        if f(S') < f(S*) then
            S* ← S';
        // insert or discard the improved solution, Section IV-F
        P ← RankBasedPoolUpdating (P, S')
return the best solution found S*;
```

D. Component-based neighborhood search

To ensure an effective local optimization, MACNP employs a fast and effective Component-Based Neighborhood Search (CBNS) procedure (see Algorithm 2). CBNS integrates two key techniques, i.e., a large component-based node exchange strategy and a node weighting technique.

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 conven-

Algorithm 2: Component-based neighborhood search

Input: A solution S, number of idle iterations $MaxIdleIters$.
Output: The best solution S^* found so far

$S^* \leftarrow S$;
$iter \leftarrow 0$, $idle\_iter \leftarrow 0$;
while $idle\_iter < MaxIdleIters$ **do**
  select a large component c at random;
  remove a node v from component c with the node weighting scheme;
  $S \leftarrow S \cup \{v\}$;
  $u \leftarrow \arg\min_{w \in S}\{f(S \setminus \{w\}) - f(S)\}$;
  $S \leftarrow S \setminus \{u\}$;
  if $f(S) < f(S^*)$ **then**
    $S^* \leftarrow S$;
    $idle\_iter \leftarrow 0$;
  else
    $idle\_iter \leftarrow idle\_iter + 1$;
  $iter \leftarrow iter + 1$;
return the best solution found;

tional exchange operator which swaps a node $u \in S$ with a node $v \in V \setminus S$ [2], [28], [35]. For a given solution, this neighborhood yields $O(K(|V|-K))$ neighbor solutions. To evaluate a neighbor solution, no incremental technique is known and a full computation from scratch is required by running the modified depth-first search algorithm of complexity $O(|V|+|E|)$ [20]. 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).

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 $v = \arg\max\{f(S)-f((S\setminus\{u\})\cup\{v'\})\}$, $\forall v' \in V \setminus S$ so that swapping u and v disconnects the graph as much as possible. For $v \in V \setminus S$, the second neighborhood tries to identify the node $u \leftarrow \arg\min\{f((S \cup \{v\}) \setminus \{u'\}) - f(S)\}$, $\forall u' \in S$. The computational complexity of examining these neighborhoods is $O(K(|V|+|E|))$ and $O((|V|-K)(|V|+|E|+K \times degree(G)))$ respectively, where $degree(G)$ 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 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 number of connected node pairs in the residual graph $G[V \setminus S]$. This can be achieved by fragmenting the largest connected components in $G[V \setminus S]$ 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 $Z \subset V \setminus S$ such that $Z = \cup_{|\mathcal{C}_i| \geqslant L} \mathcal{C}_i$ as candidate nodes for exchanges. Consequently, the neighborhood size is reduced to $K|Z|$, which is generally far smaller than $K(|V|-K)$ for reasonable L values.

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 $L = (max\_n_c + min\_n_c)/2$, where max_n_c and min_n_c are respectively the number of nodes in the largest and smallest connected components in the residual graph $G[V \setminus S]$.

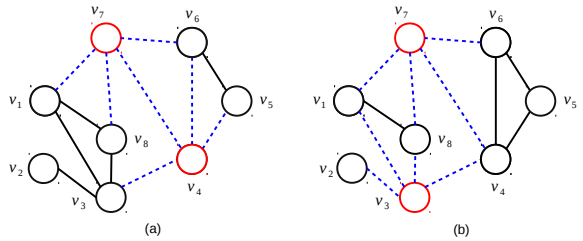

Fig. 3. An illustrative example of component-based neighborhood structure. (a) A candidate solution S. (b) An improved neighbor solution S' of S.

Fig. 3(a) shows a CNP solution $S = \{v_4, v_7\}$ ($K = 2$) with $f(S) = 7$. After deleting v_4 and v_7 from the original graph G, the resulting residual graph is composed of two components $\mathcal{C}_1 = \{v_5, v_6\}$ and $\mathcal{C}_2 = \{v_1, v_2, v_3, v_8\}$. Our component-based neighborhood always swaps a node $u \in S$ with a node selected from a predefined large component (instead of a random component). Based on the definition (1), we have $L = (4+2)/2 = 3$. Therefore we only consider the swapping operations between S and a large component $\mathcal{C}_2$ ($|\mathcal{C}_2| > L = 3$), thus reducing the neighbor solutions to be considered from 12 to 8. Fig. 1(b) shows an improved neighbor solution S' with $f(S') = 0 + \binom{2}{2} + \binom{3}{2} = 4$.

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 neighborhood decomposition [17], [29], [44]. This 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 node to the residual graph. In our case, we first randomly select a *large* component and removes a node v from the selected component with the node weighting scheme (see below). 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 this exchange strategy, the computational effort required to examine the candidate solutions greatly decreases. For instance, for the graph 'BA1000' with 1000 vertices and $K = 75$, using the large component-based two-phase node exchange strategy reduces the number of candidate solutions from $(1000-75) \times 75 = 69375$ to $75 \ll 69375$.

For such a two-phase exchange, selecting a node v to remove from $G[V \setminus S]$ can be performed in $O(T+nbr_v)$, where T is the number of connected components in $G[V \setminus S]$ and nbr_v is the length of the adjacency list of node v. Once a node is added into S, we need to remove a node u from S. We select u that causes 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 its removal will re-connect some existing components to form a large component (as the node being added back must be in the same component as all its neighboring nodes 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 [34], [9] is the second useful technique we adopted in the component-based neighborhood search. Our node weighting scheme works as follows. Each node of a large component is associated with a non-negative integer as its weight, initialized to 0. At each step, we randomly select a component $\mathcal{C}_i$ among the large connected components, and select the node v in $\mathcal{C}_i$ 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 $\mathcal{C}_i$ are increased by one. Additionally, when a node $v \in \mathcal{C}_i$ is exchanged with a node $u \in S$, we set the weight of u to 0.

With 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. This technique helps the search to escape from potential local optima. Our weighting scheme follows the general penalty idea for constraint satisfaction problems, which was first used in this setting in the breakout method [22]. We note that this scheme is also an instance of a tabu search frequency-based memory [16]. 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 [18], 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 [19], 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 [40], [44]. Since CNP is 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^1 and S^2, we divide the set of elements V into three subsets denoted by common elements, exclusive elements and excluding elements, as shown in Definitions 2-4.

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 which are in either S^1 or S^2 and not in X_A, 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 sp_0 ($0 < sp_0 < 1$); (iii) repair the partial solution structurally until a feasible solution is achieved. Specifically, if $|S^0| < K$, we randomly add some excluding elements of X_C to S^0 from a random large connected component in the residual graph. If $|S^0| > K$, 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 and about $sp_0 \times |X_B|$ exclusive elements. Note that sp_0, which could be tuned statically or dynamically (e.g., as a function of $|X_B|$, $|X_A|$ and K), is used to influence the diversity of the offspring solution by including excluding elements.

The proposed crossover shares ideas related to those of the crossovers proposed in [40], [44] by directly inheriting common elements from the parent solutions. However, to complement the partial solution, our crossover favors the transmission of exclusive elements from the parent solutions over the excluding elements that are not in the parent solutions. As such, our crossover is less destructive and has a stronger intensification effect (see also Section VII-C for a comparative study). It is worth noting that the proposed crossover is also different from the crossover for critical node problems introduced in [4] that uses set union as its key operation.

F. Rank-based pool updating

Each offspring solution is submitted for improvement by the component-based neighborhood search procedure presented in Section IV-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, as shown in Algorithm 3). This pool updating strategy resorts to a score function to evaluate each individual, which 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 [11], [21], [44].

First, we temporarily insert S^0 into the population (line 1 of Algorithm 3). Then we evaluate all the solutions of the population according to the score function [44] (lines 2-3 of Algorithm 3) and identify the worst solution S^w (line 4 of Algorithm 3). Finally, if S^0 is different from S^w, we replace S^w by S^0. Otherwise, we discard S^0 (lines 5-6 of Algorithm 3).

Algorithm 3: Rank-based pool updating strategy

Input: A population P of size p and an improved solution S^0
Output: An updated population P

$P' \leftarrow P \cup \{S^0\}$;
for $i = 0, 1, \ldots, p$ **do**
  evaluate solution S^i according to the score function;
identify the worst solution S^w in population P' i.e., $w \leftarrow \max_{j \in \{0,1,\ldots,p\}} Score(S^j, P')$;
if $w \neq 0$ **then**
  replace S^w with S^0, i.e., $P \leftarrow P' \setminus \{S^w\}$;
return the updated population P;

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, each generation of MACNP performs four subroutines: parent selection, double backbone-based crossover, component-based neighborhood search and rank-based pool updating. The parent selection procedure takes time $O(1)$. The double backbone-based crossover operator can be realized in $O(K(|V| + |E|))$. The component-based neighborhood search is bounded in time by $O(K(|V| + |E|)MaxIters$, where $MaxIters$ is the total number of iterations at each neighborhood search. 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 $O(K(|V| + |E|)MaxIters)$.

V. 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 popular benchmark sets composed of 42 graphs[1].

Synthetic benchmark set [35] contains 16 instances (with 235 to 5000 nodes) classified into four categories: Barabasi-Albert (BA) graphs, Erdos-Renyi (ER) graphs, Forest-Fire (FF) graphs and Watts-Strogatz (WS) graphs.

Real-world benchmark set [4] consists of 26 real-world graphs (with 121 to 23133 nodes) from various practical applications in areas like biology, electronics, transportation and complex networks.

B. Experimental settings

The proposed MACNP algorithm[2] was implemented in C++ 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. Running the well-known DIMACS machine benchmark procedure dfmax.c[3] 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.

TABLE I
The parameter settings of the proposed MACNP algorithm.

Parameter	description	value	section
p	population size	20	IV-C
$MaxIdleIters$	maximum number of idle iterations in CBNS	1000	IV-D
sp_0	selection probability	0.85	IV-E

1) Parameter tuning: Like previous CNP algorithms and most heuristic algorithms, MACNP requires some parameters (see Table I). Parameter p is the population size (Section IV-C) and is set to $p = 20$ as suggested in [21] [40] and [44]. Parameter $MaxIdleIters$ is the maximum allowable number of idle iterations, which is used as the stopping condition of the component-based neighborhood search (Section IV-D). sp_0 determines the probability of inheriting an exclusive element during the crossover operation (Section IV-E). Our analyses indicate that $MaxIdleIters$ and sp_0 are sensitive parameters while this is not the case for p. To tune $MaxIdleIters$ and sp_0, we followed the common practice in the heuristic literature by testing a limited number of parameter configurations on a small sample of problem instances as follows.

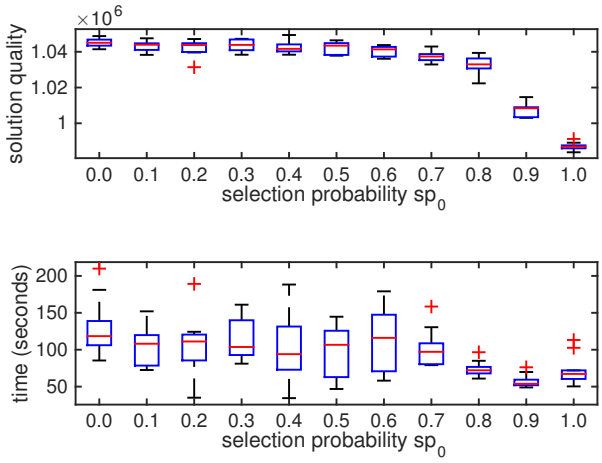

Fig. 4. Box plots corresponding to different values of $sp_0 \in [0, 1.0]$ in terms of solution quality (above sub-figure) and computation time (bottom sub-figure). X-axis indicates each sp_0 value and Y-axis shows the performance.

To avoid over-fitting, we used 9 (out of 42) representative instances (BA5000, ER941, FF500, WS250, TreniR, openflights, H3000a, H4000, OClinks) to cover both synthetic and real-world instances with different sizes and variable levels of difficulty. To identify a suitable value for a given parameter, we evaluated the performance of MACNP by varying the parameter value within a reasonable range, while fixing the other parameters to the default values of Table I. For each parameter configuration, we ran MACNP 30 times to solve each sample instance, each run being limited to 100 generations.

[1] Available at http://www.di.unito.it/~aringhie/cnp.html

[2] Our codes and computational results will be made available at http://www.info.univ-angers.fr/~hao/cnps.html

[3] dfmax: ftp://dimacs.rutgers.edu/pub/dsj/clique

We take the parameter sp_0 as an example. Fig. 4 shows the box plots with eleven values $sp_0 \in [0, 1.0]$. We observe that large values ($sp_0 > 0.8$) yield better results than small ones ($sp_0 \leq 0.8$). We also see that MACNP with $sp_0 = 0.9$ performs the best in terms of computation time. Meanwhile, given that sp_0 influences the diversity of offspring solutions, we adopted $sp_0 = 0.85$ in this work to make a reasonable compromise between solution quality, computation time and offspring diversity. In passing, since our double backbone crossover reduces to the traditional backbone crossover when $sp_0 = 0.0$ (Section IV-E), this experiment also demonstrates the advantage of the proposed crossover over the traditional backbone crossover. To tune the parameter $MaxIdleIters$, we used the same procedure to identify 1000 as its default value. Finally, note that, like many search algorithms, finding the best parameter setting of our algorithm is not a trivial task and may be time consuming. Possible solutions to cope with this problem in general would be to reduce the number of required parameters, define self-tunable parameters and make use of automatic parameter tuning techniques.

For our computational studies, we used consistently the parameter values of Table I to run our MACNP algorithm. Notice that this parameter setting can be considered as the default setting of MACNP and can conveniently be adopted by a user. Meanwhile, users can also tune some parameters in order to better solve their particular problem instances. Given the stochastic nature of the algorithm, MACNP was independently executed 30 times on each benchmark instance like [35]. Following the standard practice for solving CNPs [3], [4], [28], [42], we adopted a time limit ($t_{max} = 3600$ seconds) as the stopping condition of each run.

2) Statistical test: To analyze the experimental results, we resort to the well-known two-tailed sign test [12] 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 corresponding Critical Values (CV) of the two-tailed sign test are respectively $CV_{0.05}^{16} = 12$ and $CV_{0.05}^{26} = 18$ 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 $CV_{0.05}^{X}$ 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 under the time limit $t_{max} = 3600$ seconds. Columns 1-3 respectively describe for each instance its name (Instance), the number of critical nodes (K) and the known best value (KBV) reported in [32]. 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 in seconds to attain the objective value (t_{avg}) and the average number of node exchanges to achieve the objective value ($\#exch$).

TABLE II
PERFORMANCE OF MACNP ON SYNTHETIC AND REAL-WORLD BENCHMARKS.

Instance	K	KBV [32]	Δf_{best}	Δf_{avg}	t_{avg}	$\#exch$
BA500	50	195*	0	0.0	0.0	5.8×10^2
BA1000	75	558*	0	0.0	0.3	6.4×10^3
BA2500	100	3704*	0	0.0	0.7	8.6×10^3
BA5000	150	10196*	0	0.0	6.5	3.5×10^4
ER235	50	295*	0	0.0	7.1	2.0×10^5
ER466	80	1524	0	0.0	28.5	9.5×10^5
ER941	140	5012	0	2.1	458.5	1.2×10^7
ER2344	200	959500	-57002	-37160.5	2284.8	1.5×10^7
FF250	50	194*	0	0.0	0.0	2.3×10^3
FF500	110	257*	0	0.0	0.4	8.7×10^3
FF1000	150	1260*	0	0.0	84.9	2.6×10^5
FF2000	200	4545*	0	0.7	107.6	3.3×10^5
WS250	70	3101	-18	-11.6	1140.5	6.1×10^7
WS500	125	2078	-6	4.6	179.3	2.4×10^6
WS1000	200	113638	-3831	10044.6	2675.0	1.8×10^7
WS1500	265	13167	-69	88.1	1012.2	7.3×10^6
Bovine	3	268	0	0.0	0.0	6.8×10^1
Circuit	25	2099	0	0.0	0.2	1.7×10^4
Ecoli	15	806	0	0.0	0.0	6.2×10^2
USAir97	33	4336	0	0.0	756.5	2.5×10^6
humanDi	52	1115	0	0.0	0.6	1.7×10^4
TreniR	26	920	-2	-2.0	0.3	2.2×10^4
EU_fli	119	349927	-1659	1730.0	232.6	1.1×10^5
openfli	186	28671	-1829	33.3	2093.7	2.9×10^6
yeast1	202	1414	-2	-2.0	21.7	1.0×10^5
H1000	100	328817	-22468	-18190.5	2137.5	2.5×10^7
H2000	200	1309063	-65204	-45567.4	2861.9	1.2×10^7
H3000a	300	3005183	-160790	-120401.3	3280.7	1.1×10^7
H3000b	300	2993393	-152123	-108306.0	3252.9	1.2×10^7
H3000c	300	2975213	-136784	-105864.5	3307.5	1.1×10^7
H3000d	300	2988605	-157294	-96042.3	3250.9	1.1×10^7
H3000e	300	3001078	-153169	-113552.3	3437.4	1.2×10^7
H4000	400	5403572	-250595	-136196.5	2907.0	6.6×10^6
H5000	500	8411789	-439264	-316976.4.7	3226.6	6.3×10^6
powergr	494	16099	-237	-197.5	1286.4	1.8×10^6
OClinks	190	614504	-2201	40.0	584.6	4.5×10^5
faceboo	404	420334	222828	319102.6	2978.5	2.2×10^6
grqc	524	13736	-140	-106.8	871.8	9.2×10^5
hepth	988	114382	-7985	-4726.4	3442.0	3.1×10^6
hepph	1201	7336826	1291861	2033389.3	3376.3	1.4×10^6
astroph	1877	54517114	7551852	8030784.1	1911.4	3.9×10^5
condmat	2313	2298596	7155765	7763211.8	1779.5	4.9×10^5

* Optimal results obtained by exact algorithm [32] within 5 days.

From Table II, we observe that MACNP attains the best objective values for all 16 synthetic instances (first part of the table) and finds in particular 5 new upper bounds (see $\Delta f_{best} < 0$). For instances ER2344 and WS250, our average objective values are also better than the previously known best values (see $\Delta f_{avg} < 0$). To the best of our knowledge, our MACNP algorithm is the first heuristic which reaches the optimal solution 4545 of FF2000 within the short computation time of 107.6 seconds, which is far less than 5 days by the branch-and-cut algorithm [32] (as reported in [3]). For the real-world benchmark (second part of Table II), MACNP also shows a highly competitive performance. It matches 5 known best results and finds improved upper bounds for 17 out of the 26 instances. 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 II). However, MACNP failed to attain the known best values for four instances (OClinks, hepph, astroph,and condmat) within the time limit of $t_{max} = 3600$ 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-ran our MACNP algorithm under this condition, MACNP managed to find better solutions, including two new upper bounds for hepph and astroph (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 7 state-of-the-art reference algorithms: 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) [27] and fast heuristic (FastCNP) [42].

Since the source codes of CNA1 [27] and FastCNP [42] are available to us, we first present a detailed comparison between MACNP and these two reference algorithms. It is worth noting that the FastCNP algorithm used in our experiment is an improved version of the initial FastCNP algorithm proposed in [42], by additionally integrating the node weighting scheme (Section IV-D3) in the neighborhood search procedure. To make a fair comparison, the three algorithms were run on our platform with the same time limit $t_{max} = 3600$ seconds, and we ran each algorithm 30 times to solve each instance. In addition to t_{max}, we used the default parameter values of CNA1 and FastCNP: the number of nodes to remove in the partial restart of CNA1 [27] and the maximum number of idle iterations in local search and the perturbation strength for FastCNP [42]. As for MACNP, the default parameter values for CNA1 and FastCNP have been obtained based on sample instances of the same CNP benchmark sets.

The comparative results are shown in Table III where 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}) found during 30 runs, the average objective value (f_{avg}), the average time in seconds (t_{avg}) and the average number of steps ($\#exch$) needed to achieve the best objective value at each run. Correspondingly, columns 6-9 and columns 10-13 respectively present the results of FastCNP and MACNP. The best values of the compared results are in bold, and when the same best objective values are achieved, fewer node exchanges are underlined (which indicates a better performance in terms of computation efficiency). In addition, we give the number of instances (wins) for which our MACNP algorithm obtained a better performance (i.e., f_{best} and f_{avg}) compared to each competing algorithm. The win values for indicators t_{avg} and $\#exch$ are meaningless and are marked by '*'.

Table III indicates 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 instances. For the synthetic benchmark, MACNP is significantly better than CNA1 in terms of the best objective value, winning 12.5 instances (i.e., $12.5 > CV^{16}_{0.05} = 12$). Compared to FastCNP, MACNP is also very competitive and wins 10.5 instances, which is slightly smaller than the critical value $CV^{16}_{0.05} = 12$. As to the average objective value, MACNP significantly outperforms both CNA1 and FastCNP by winning 13 instances. For the real-world benchmark, MACNP again 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 node exchanges to reach its results (see values underlined).

We also compared our MACNP algorithm with five additional algorithms reported in the literature: dynamic restarting greedy algorithms (Greedy3d and Greedy4d) [2], iterated local search (ILS) [3], variable neighborhood search (VNS) [3], and genetic algorithm (GA) [4]. Since the source code of these reference algorithms is not available, we used their best results reported in the corresponding papers. These five algorithms have been recently evaluated on the same platform (i.e., an HP ProLiant DL585 G6 server with two 2.1 GHz AMD Opteron 8425HE 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 different time limits: $t_{max} \in (7200, 10000]$ for most of the synthetic instances and $t_{max} \in [3000, 16000]$ for most of the real-world instances. 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 Greedy3d, Greedy4d, Best VNS, Best ILS, GA, and CNA1, respectively winning 15.0, 15.0, 12.5, 15.5, 12.0, 12.5 instances. Compared to FastCNP, MACNP wins 10.5 instances, which is just slightly smaller than the critical value $CV^{16}_{0.05} = 12$. 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. Specifically, MACNP achieves the best objective values for 23 out of the 26 real-world instances, including 18 new upper bounds and 5 known best objective values.

VI. Application to the Cardinality-Constrained Critical Node Problem

In this section, we show that our approach can also be used to solve other critical node problems, by adapting MACNP to

TABLE III
COMPARATIVE PERFORMANCE OF THE PROPOSED MACNP WITH CNA1 AND FASTCNP ON SYNTHETIC AND REAL-WORLD BENCHMARKS.

	CNA1				FastCNP*				MACNP			
Instance	f_{best}	f_{avg}	t_{avg}	$\#exch$	f_{best}	f_{avg}	t_{avg}	$\#exch$	f_{best}	f_{avg}	t_{avg}	$\#exch$
BA500	**195**	**195.0**	2.5	1.4×10^5	**195**	**195.0**	< 0.1	4.1×10^3	**195**	**195.0**	< 0.1	5.8×10^2
BA1000	**558**	558.7	5.4	1.1×10^5	**558**	**558.0**	29.8	1.6×10^6	**558**	**558.0**	0.3	6.4×10^3
BA2500	**3704**	**3704.0**	1.7	1.4×10^4	**3704**	3710.6	649.9	1.7×10^7	**3704**	**3704.0**	0.7	8.6×10^3
BA5000	**10196**	**10196.0**	103.7	3.8×10^5	**10196**	10201.4	104.9	1.5×10^6	**10196**	**10196.0**	6.5	3.5×10^4
ER235	**295**	**295.0**	6.8	1.0×10^6	**295**	**295.0**	11.7	4.2×10^6	**295**	**295.0**	7.1	2.1×10^5
ER466	**1524**	**1524.0**	825.4	7.2×10^7	**1524**	**1524.0**	364.9	7.0×10^7	**1524**	**1524.0**	28.5	9.5×10^5
ER941	5114	5177.4	1606.1	6.7×10^7	**5012**	**5013.3**	1516.7	1.4×10^8	**5012**	5014.1	458.5	1.2×10^7
ER2344	996411	1008876.4	1379.2	8.8×10^6	953437	979729.2	1793.8	3.4×10^7	**902498**	**922339.5**	2284.8	1.5×10^7
FF250	**194**	**194.0**	158.0	2.2×10^7	**194**	**194.0**	1.9	4.6×10^5	**194**	**194.0**	< 0.1	2.3×10^3
FF500	263	265.0	197.5	9.4×10^6	**257**	258.4	55.1	5.3×10^6	**257**	**257.0**	0.4	8.7×10^3
FF1000	1262	1264.2	1743.0	3.8×10^7	**1260**	1260.8	23.6	1.0×10^6	**1260**	**1260.0**	84.9	2.6×10^5
FF2000	4548	4549.4	1571.0	2.4×10^7	4546	4558.3	1160.8	2.8×10^7	**4545**	**4545.7**	107.6	3.3×10^5
WS250	3415	3702.8	1424.4	7.6×10^7	3085	3196.4	1983.5	4.2×10^8	**3083**	**3089.4**	1140.5	6.1×10^7
WS500	2085	2098.7	1581.4	1.5×10^8	**2072**	2083.3	1452.6	2.7×10^8	**2072**	**2082.6**	179.3	2.4×10^6
WS1000	141759	161488.0	116.5	1.2×10^6	123602	127493.4	2120.2	6.3×10^7	**109807**	**123682.6**	2675.0	1.8×10^7
WS1500	13498	13902.5	1787.2	5.7×10^7	13158	13255.7	1554.9	8.8×10^7	**13098**	**13255.1**	1012.2	7.3×10^6
wins	12.5	13.0	*	*	10.5	13.0	*	*	*	*	*	*
Bovine	**268**	**268.0**	< 0.1	3.0×10^2	**268**	**268.0**	< 0.1	2.4×10^3	**268**	**268.0**	< 0.1	6.8×10^1
Circuit	**2099**	**2099.0**	0.3	6.8×10^4	**2099**	**2099.0**	1.2	5.9×10^5	**2099**	**2099.0**	0.2	1.7×10^4
E.coli	**806**	**806.0**	< 0.1	1.3×10^3	**806**	**806.0**	< 0.1	7.8×10^3	**806**	**806.0**	< 0.1	6.3×10^2
USAir97	**4336**	**4336.0**	254.9	1.1×10^7	**4336**	**4336.0**	90.8	8.6×10^6	**4336**	**4336.0**	756.5	2.5×10^6
HumanDi	**1115**	**1115.0**	5.8	4.8×10^5	**1115**	**1115.0**	2.5	4.3×10^5	**1115**	**1115.0**	0.6	1.7×10^4
TreniR	**918**	**918.0**	1.3	4.3×10^5	**918**	**918.0**	2.4	1.4×10^6	**918**	**918.0**	0.3	2.2×10^4
EU_fli	**348268**	**348347.0**	914.8	1.6×10^6	**348268**	348697.7	1495.0	6.0×10^6	**348268**	351657.0	232.6	1.1×10^5
openfli	29300	29815.3	1835.0	7.4×10^6	28834	29014.4	499.3	3.5×10^6	**26842**	**28704.3**	2093.7	2.9×10^6
yeast	1413	1416.3	1461.9	1.1×10^7	**1412**	**1412.0**	252.0	2.8×10^6	**1412**	**1412.0**	21.7	1.0×10^5
H1000	314152	317805.7	1412.4	2.5×10^7	314964	316814.8	1821.6	6.7×10^7	**306349**	**310626.5**	2137.5	2.5×10^7
H2000	1275968	1292400.4	1200.0	8.6×10^6	1275204	1285629.1	1620.1	2.5×10^7	**1243859**	**1263495.6**	2861.9	1.2×10^7
H3000a	2911369	2927312.0	1598.5	7.4×10^6	2885588	2906965.5	2041.5	1.8×10^7	**2844393**	**2884781.7**	3280.7	1.1×10^7
H3000b	2907643	2927330.5	963.3	4.3×10^6	2876585	2902893.9	1596.2	1.3×10^7	**2841270**	**2885087.0**	3252.9	1.2×10^7
H3000c	2885836	2917685.8	1142.6	4.7×10^6	2876026	2898879.3	1927.9	1.7×10^7	**2838429**	**2869348.5**	3307.5	1.1×10^7
H3000d	2906121	2929569.2	1463.9	5.7×10^6	2894492	2907485.4	2005.4	1.7×10^7	**2831311**	**2892562.7**	3250.9	1.1×10^7
H3000e	2903845	2931806.8	1489.4	6.1×10^6	2890861	2911409.3	1993.0	1.6×10^7	**2847909**	**2887525.7**	3437.4	1.2×10^7
H4000	5194592	5233954.5	1749.1	5.3×10^6	5167043	5190883.7	1954.2	1.2×10^7	**5044357**	**5137528.3**	2907.0	6.6×10^6
H5000	8142430	8212165.9	1342.5	3.0×10^6	8080473	8132896.2	2009.3	8.8×10^6	**7972525**	**8094812.6**	3226.6	6.3×10^6
powergr	16158	16222.1	1532.2	5.6×10^6	15982	16033.5	1610.3	9.8×10^6	**15862**	**15901.5**	1286.4	1.8×10^6
Oclinks	**611326**	614858.5	990.9	2.5×10^6	611344	616783.0	713.1	3.3×10^6	612303	**614544.0**	584.6	4.5×10^5
faceboo	701073	742688.0	2234.4	4.8×10^6	692799	765609.8	3132.9	1.5×10^7	**643162**	**739436.6**	2978.5	2.2×10^6
grqc	15522	15715.7	2201.1	5.2×10^6	13616	13634.8	2002.0	7.8×10^6	**13596**	**13629.2**	871.8	9.2×10^5
hepth	130256	188753.7	2135.6	2.1×10^6	108217	109889.5	2765.3	5.3×10^6	**106397**	**109655.6**	3442.0	3.1×10^6
hepph	9771610	10377853.2	2286.8	7.2×10^5	**6392653**	**7055773.8**	3120.6	2.9×10^6	8628687	9370215.3	3376.3	1.4×10^6
astroph	59029312	60313225.8	3441.4	5.5×10^5	**55424575**	**57231348.7**	3576.4	1.1×10^6	62068966	62547898.1	1911.4	3.9×10^5
condmat	13420836	14823254.9	1481.3	2.9×10^5	**4086629**	**5806623.8**	3511.9	1.6×10^6	9454361	10061807.8	1779.5	4.9×10^5
wins	21.5	22.5	*	*	19.0	19.5	*	*	*	*	*	*

* Our FastCNP is an improved version of original FastCNP proposed in [42], which additionally integrating the node weighting scheme.
Note that, it is meaningless to calculate the wins in terms of average time and average node exchanges when different best objective values are achieved, and we represent them by *.

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 [6]. 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 minimize the subset S, we solve a series of CC-CNP with decreasing K values. For a fixed K, we try to find a set $S \subseteq V$ of K nodes, whose removal minimizes the number of nodes in each connected component which exceeds the cardinality threshold W [27]. For this purpose, we define an auxiliary (minimization) function f':

$$f'(S) = \sum_{i=1}^{T} \max(|\mathcal{C}_i| - W, 0) \tag{2}$$

which calculates the total number of nodes in excess of W in all T connected components of the residual graph. It is clear that if $f'(S) = 0$, then S is a feasible solution of CC-CNP.

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 MACC-CNP to find a set S of K nodes whose removal minimizes the number of exceeded

TABLE IV
COMPARISON BETWEEN MACNP AND THE STATE-OF-THE-ART ALGORITHMS ON SYNTHETIC AND REAL-WORLD BENCHMARKS.

Instance	K	KBV	Greedy3d	Greedy4d	Best VNS	Best ILS	GA	CNA1$^{\diamond}$	FastCNP$^{\diamond}$	MACNP
BA500	50	195*	**195**	**195**	**195**	**195**	**195**	**195**	**195**	**195**
BA1000	75	558*	559	559	559	559	**558**	**558**	**558**	**558**
BA2500	100	3704*	3722	3722	**3704**	3722	**3704**	**3704**	**3704**	**3704**
BA5000	150	10196*	**10196**	**10196**	**10196**	10222	**10196**	**10196**	**10196**	**10196**
ER235	50	295*	315	313	**295**	313	**295**	**295**	**295**	**295**
ER466	80	1524	1938	1993	1542	1874	1560	**1524**	**1524**	**1524**
ER941	140	5012	8106	8419	5198	5544	5120	5114	**5012**	**5012**
ER2344	200	959500	1118785	1112685	997839	1038048	1039254	996411	953437	**902498***
FF250	50	194*	199	197	**194**	195	**194**	**194**	**194**	**194**
FF500	110	257*	262	264	**257**	261	**257**	263	**257**	**257**
FF1000	150	1260*	1288	1271	**1260**	1276	**1260**	1262	**1260**	**1260**
FF2000	200	4545*	4647	4592	4549	4583	4546	4548	4546	**4545***
WS250	70	3101	11694	11401	6610	3241	3240	3415	3085	**3083***
WS500	125	2078	4818	11981	2130	2282	2199	2085	**2072**	**2072***
WS1000	200	113638	316416	318003	139653	115914	113638	141759	123602	**109807***
WS1500	265	13167	157621	243190	13792	14681	13662	13498	13158	**13098***
wins		10.5	15.0	15.0	12.5	15.5	12.0	12.5	10.5	*
Bovine	3	268	**268**	**268**	**268**	**268**	**268**	**268**	**268**	**268**
Circuit	25	2099	**2099**	2100	2101	2117	**2099**	**2099**	**2099**	**2099**
E.coli	15	806	**806**	834	**806**	**806**	**806**	**806**	**806**	**806**
USAir97	33	4336	4442	4726	5444	4442	**4336**	**4336**	**4336**	**4336**
HumanDi	52	1115	**1115**	**1115**	**1115**	**1115**	**1115**	**1115**	**1115**	**1115**
TreniR	26	926	926	936	920	934	928	**918**	**918**	**918***
EU_fli	119	349927	349927	350757	356631	355798	351610	**348268**	**348268**	**348268***
openfli	186	28834	29624	29552	31620	29416	28834	29300	28834	**26842***
yeast1	202	1414	1416	1415	1421	1434	1414	1413	**1412**	**1412***
H1000	100	328817	338574	336866	332286	344509	328817	314152	314964	**306349***
H2000	200	1309063	1372109	1367779	1309063	1417341	1315198	1275968	1275204	**1243859***
H3000a	300	3005183	3087215	3100938	3058656	3278172	3005183	2911369	2885588	**2844393***
H3000b	300	2993393	3096420	3100748	3121639	3250497	2993393	2907643	2876585	**2841270***
H3000c	300	2975213	3094459	3097451	3079570	3202002	2975213	2885836	2876026	**2838429***
H3000d	300	2988605	3090753	3100216	3027839	3237495	2988605	2906121	2894492	**2831311***
H3000e	300	3001078	3095793	3113514	3031975	3255390	3001078	2903845	2890861	**2847909***
H4000	400	5403572	5534254	5530402	5498097	5877896	5403572	5194592	5167043	**5044357***
H5000	500	8411789	8657681	8653358	8889904	9212984	8411789	8142430	8080473	**7972525***
powergr	494	16099	16373	16406	16099	16533	16254	16158	15982	**15862***
Oclinks	190	614504	614504	614546	623366	625671	620020	**611326**	611344	612303
faceboo	404	420334	608487	856642	865115	**420334**	561111	701073	692799	643162
grqc	524	13736	13787	13825	13751	13817	13736	15522	13616	**13596***
hepth	988	114382	232021	326281	114933	123138	114382	130256	108217	**106397***
hepph	1201	7336826	10305849	10162995	10989642	11759201	7336826	9771610	6392653	***6156536****
astroph	1877	54517114	54713053	54517114	65937108	65822942	58045178	59029312	55424575	***53963375****
condmat	2313	2298596	11771033	11758662	6121430	**2298596**	2612548	13420836	4086629	*4871607*
wins		22.5	24.0	25.0	24.5	22.5	22.5	21.5	21.0	*

* Optimal results obtained by branch-and-cut algorithm [32] within 5 days.
* Improved best upper bounds.
$^{\diamond}$ The best results of CNA1 and FastCNP are obtained by running their source codes in our platform, which are slightly different their reported results in [27] and in [42], respectively.
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.

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 [21], [43], [45].

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 five state-of-the-art algorithms: greedy algorithms (G1 and G2) [4], genetic algorithm (GA) [4], multi-start greedy algorithm (CNA2) [27], fast heuristic (FastCNP) [42]. Among these reference algorithms, FastCNP was originally proposed for the classic critical node problem, and we adapted it to CC-CNP in the same way as for MACNP. The source code of CNA2 was provided by its author [27]. For algorithms G1, G2 and GA, whose source codes are not available, we used the results reported in [4]. These results have been obtained with different time limits $t_{max} \in [100, 16000]$ seconds, which are, for most instances, larger than our time limit of $t_{max} = 3600$ seconds. For our comparative study, we ran CNA2 and FastCNP with their default parameters under the time limit $t_{max} = 3600$ 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 between our MACC-CNP algorithm and 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) for which

MACC-CNP has a better result according to the two-tailed sign test [12], as shown in the last row for each benchmark.

TABLE V
COMPARISON BETWEEN MACC-CNP AND THE STATE-OF-THE-ART ALGORITHMS ON SYNTHETIC AND REAL-WORLD BENCHMARKS.

Instance	L	KBV	G1	G2	GA	CNA2	FA$^{\circ}$	MA$^{\diamond}$
BA500	4	47	**47**	**47**	**47**	**47**	**47**	**47**
BA1000	5	61	**61**	**61**	**61**	**61**	**61**	**61**
BA2500	10	100	101	**100**	**100**	**100**	**100**	**100**
BA5000	13	149	154	151	**149**	**149**	**149**	**149**
ER235	7	47	49	50	**47**	**47**	**47**	**47**
ER466	14	81	86	85	81	**79**	**79**	**79***
ER941	25	139	149	152	139	141	**135**	**135***
ER2344	1400	204	252	270	204	194	189	**185***
FF250	5	48	**48**	49	**48**	**48**	**48**	**48**
FF500	4	100	102	102	**100**	**100**	**100**	**100**
FF1000	7	142	145	145	**142**	**142**	**142**	**142**
FF2000	12	182	191	187	**182**	**182**	**182**	**182**
WS250	40	73	79	80	72	71	**70**	**70***
WS500	15	126	145	144	126	124	**123**	**123***
WS1000	500	162	195	418	162	180	166	**157***
WS1500	30	278	339	332	278	273	256	**254***
wins		11.5	14.5	14.5	11.5	11.0	9.5	*
Bovine	15	4	**4**	**4**	**4**	**4**	**4**	**4**
Circuit	30	24	25	26	**24**	**24**	**24**	**24**
E.coli	20	15	16	**15**	**15**	**15**	**15**	**15**
USAir97	70	33	34	40	**33**	**33**	**33**	**33**
HumanDi	10	49	51	50	**49**	**49**	**49**	**49**
TreniR	10	28	30	31	28	**27**	**27**	**27***
EU_fli	850	113	127	118	113	113	**112**	**112***
openfli	140	184	194	206	184	183	**180**	**180***
yeast1	6	195	202	199	195	**193**	**193**	**193***
H1000	800	103	172	151	103	97	95	**92***
H2000	1600	221	362	313	221	207	195	**188***
H3000a	2500	279	448	402	279	276	252	**242***
H3000b	2500	279	456	401	279	270	251	**244***
H3000c	2500	276	446	404	276	274	250	**244***
H3000d	2500	276	452	402	276	272	250	**244***
H3000e	2500	280	455	403	280	270	252	**244***
H4000	3300	398	651	571	398	388	354	**347***
H5000	4200	458	745	662	458	459	413	**410***
powergr	20	428	449	440	428	430	**397**	**397***
Oclinks	1100	197	209	200	197	193	193	**192***
faceboo	450	324	472	821	**324**	523	375	378
grqc	20	480	497	501	480	486	462	**461***
hepth	70	981	1040	1042	981	1029	955	**944***
hepph	3600	1228	1416	1572	1228	1103	**994**	1120
astroph	12000	1322	3284	1769	1322	1364	**1249**	1329
condmat	500	2506	2506	2651	2506	2357	2357	**2320***
wins		21.5	25.5	25.0	21.5	22.5	19.0	*

* Improved best upper bounds.
$^{\circ}$ Adaptation of the FastCNP algorithm [42] to CC-CNP (denoted as FA).
$^{\diamond}$ The proposed MACC-CNP (MA) algorithm.

Table V shows that MACC-CNP achieves the best objective values for all synthetic instances, and yielding in particular 7 new upper bounds. At a significance level of 0.05, MACC-CNP is significantly better than G1 and G2. Compared to GA, CNA2 and FastCNP, MACNP is better but the differences are not significant, winning 11.5, 11.0 and 9.5 instances. MACC-CNP is also very effective on the real-world benchmark. At a significance level 0.05, MACC-CNP significantly outperforms all reference algorithms. MACC-CNP discovers new upper bounds for 18 instances and reaches the best upper bounds for 5 out of 8 remaining instances. These observations show that MACC-CNP is highly competitive for solving CC-CNP.

VII. DISCUSSION AND ANALYSIS

We now study some key ingredients of the proposed algorithm. This study was performed on 4 representative synthetic instances from different families (BA5000, ER941, FF500, WS250) and 4 representative real-world instances (TreniR, H3000a, H4000, hepth). We ran each algorithm variant 15 times on each instance with a time limit $t_{max} = 3600$ seconds.

A. Benefit of the two-phase node exchange strategy

To study the benefit of the large component-based two-phase node exchange strategy (Section IV-D), we compare MACNP with an alternative version MACNP$_0$ where the neighborhood search using the large component-based two-phase node exchange is replaced by using the conventional two node exchange strategy. In other words, at each neighborhood search iteration of MACNP$_0$, we choose at random a node $u \in S$ (S being the current solution) and a node $v \in V \setminus S$ and accept the node exchange if the exchange leads to a solution better than S; Otherwise, we test another random pair (u, v). MACNP$_0$ and MACNP share thus the same ingredients except the neighborhood search procedure.

TABLE VI
COMPARISON BETWEEN LARGE COMPONENT-BASED TWO-PHASE NODE EXCHANGE (MACNP) AND TRADITIONAL NODE EXCHANGE (MACNP$_0$).

	MACNP$_0$			MACNP		
Instance	f_{best}	f_{avg}	$\frac{\#exch}{sec}$	f_{best}	f_{avg}	$\frac{\#exch}{sec}$
BA5000	10198	10200.8	1.5	**10196**	**10196.0**	5708.9
ER941	5014	5171.1	5.8	**5012**	**5013.7**	29428.5
FF500	**257**	**257.0**	134.9	**257**	**257.0**	21445.3
WS250	**3083**	3257.7	37.6	**3083**	**3101.1**	59954.5
TreniR	**918**	**918.0**	1322.5	**918**	**918.0**	102275.0
H3000a	3451908	3483709.9	< 0.1	**2849170**	**2883554.5**	3909.4
H4000	6165357	6224768.6	< 0.1	**5081209**	**5144354.2**	2117.6
hepth	23964174	24528095.0	< 0.1	**106552**	**108354.0**	1100.8

The results for MACNP and MACNP$_0$ are summarized in Table VI. For each instance, we report the best objective value (f_{best}), the average objective value (f_{avg}), and the average number of node exchanges per second ($\frac{\#exch}{sec}$) over 15 trials achieved by each algorithm. The results show that MACNP performs significantly better than MACNP$_0$ in terms of all comparison indicators primarily due to its much lower computational complexity to perform a node exchange. We observe that in each second, MACNP performs seventy or even tens of thousands times more node exchanges than 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 IV (columns 10 and 7) shows that FastCNP (using our component-based neighborhood) dominates the best ILS algorithm of [3] (using two complementary refined neighborhoods), which further demonstrates the interest of the component-based neighborhood.

B. Effectiveness of the node weighting scheme

To assess the interest of the node weighting scheme in the Component-Based Neighborhood Search (CBNS) (Section IV-D3), we compared MACNP 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 comparison of MACNP and MACNP$_1$ on the tested instances, based on four indicators: best objective value (f_{best}), average objective value (f_{avg}), average time to find the best objective value (t_{avg}), and average number of node exchanges required to attain the best objective value ($\#exch$). An obvious observation 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 node exchanges to achieve the best objective values. These observations demonstrate the effectiveness of the node weighting scheme.

C. Benefit of the double backbone-based crossover

To study the benefit of the double backbone-based crossover (Section IV-E), we compare MACNP with an alternative version MACNP$_2$. MACNP$_2$ is obtained from MACNP by replacing the double backbone-based crossover with a single backbone-based crossover which only treats the common elements as the backbone. The single backbone-based crossover operator first constructs a partial solution S^0 by inheriting all the common elements of two parent solutions and then completes the partial solution S^0 by adding nodes from a large component of the residual graph until $|S^0| = K$.

Comparative results of MACNP and MACNP$_2$ in terms of the best objective value and average objective value are displayed in the left and right part of Figure 5 respectively. The X-axis indicates the instance, and the Y-axis shows the gap of 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 or average objective value, and KBV is the known best objective (see 3rd column of Table IV). A negative gap indicates an improved upper bound for the corresponding instance.

Figure 5 shows that compared to MACNP$_2$, MACNP is able to attain a superior best objective value for all 8 instances including 4 new upper bounds (see values below 0 on the left part of Figure 5). MACNP also outperforms MACNP$_2$ on all 8 tested instances in terms of the average objective value, as shown in the right part of Figure 5. This experiment confirms the value of our double backbone-based crossover.

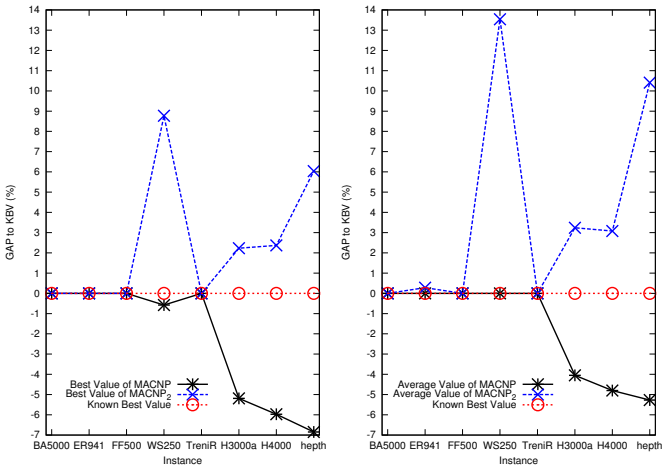

Fig. 5. Comparative results of MACNP (with double backbone-based crossover) and MACNP$_2$ (with traditional backbone-based crossover).

VIII. 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. Given that most ideas of this work are not limited to CNP, another inviting avenue for research is to adapt the proposed method to solve other critical node problems with different measures (e.g., distance-based connectivity and betweenness centrality).

Acknowledgment

We are grateful to our anonymous reviewers for their insightful comments that have helped us to significantly improve this work. We 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 Appl. Math.*, 161(16):2349–2360, 2013.

[2] B. Addis, R. Aringhieri, A. Grosso, and P. Hosteins, "Hybrid constructive heuristics for the critical node problem," *Ann. Oper. Res.*, 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.

TABLE VII
COMPARISON OF MACNP (WITH NODE WEIGHTING) AGAINST MACNP_1 (WITHOUT NODE WEIGHING).

	MACNP_1				MACNP			
Instance	f_{best}	f_{avg}	t_{avg}	$\#exch$	f_{best}	f_{avg}	t_{avg}	$\#exch$
BA5000	**10196**	**10196.0**	44.7	2.7×10^5	**10196**	**10196.0**	5.8	3.3×10^4
ER941	5014	5014.4	310.1	9.3×10^6	**5012**	**5013.7**	612.5	1.8×10^7
FF500	**257**	**257.0**	0.5	1.2×10^4	**257**	**257.0**	0.3	6.4×10^3
WS250	**3083**	**3093.8**	1360.2	8.5×10^7	**3083**	3101.1	1185.6	7.1×10^7
TreniR	**918**	**918.0**	0.3	3.5×10^4	**918**	**918.0**	0.2	2.0×10^4
H3000a	2862217	2913716.6	3202.4	1.2×10^7	**2849170**	**2883554.5**	3270.2	1.3×10^7
H4000	5187236	5313477.9	3202.0	9.0×10^6	**5081209**	**5144354.2**	2985.9	6.3×10^6
hepth	108453	111804.3	3431.6	3.3×10^6	**106552**	**108354.0**	3369.1	3.7×10^6

[4] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia, "A general evolutionary framework for different classes of critical node problems," *Eng. Appl. Artif. Intell.*, 55:128–145, 2016.

[5] A. Arulselvan, C. W. Commander, L. Elefteriadou, and P. M. Pardalos, "Detecting critical nodes in sparse graphs," *Comput. Oper. Res*, 36(7):2193–2200, 2009.

[6] A. Arulselvan, C. W. Commander, O. Shylo, and P. M. Pardalos, "Cardinality-constrained critical node detection problem," *Performance Models and Risk Management in Communications Systems*, vol. 46, pp. 79–91, 2011.

[7] U. Benlic and J.-K. Hao, "A multilevel memetic approach for improving graph k-partitions," *IEEE Trans. Evol. Comput.*, 15(5):624–642, 2011.

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

[9] S. Cai, K. Su, and A. Sattar, "Local search with edge weighting and configuration checking heuristics for minimum vertex cover," *Artif. Intell.*, 175(9):1672–1696, 2011.

[10] X. Chen, Y. S. Ong, M. H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evol. Comput.*, 15(5):591–607, 2011.

[11] Y. Chen and J. K. Hao, "Memetic search for the generalized quadratic multiple knapsack problem," *IEEE Trans. Evol. Comput.*, 20(6):908–923, 2016.

[12] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res*, 7:1–30, 2006.

[13] 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 Trans. Netw.*, 20(2):609–619, 2012.

[14] L. Feng, Y. S. Ong, M. H. Lim, and I. W. Tsang, "Memetic search with interdomain learning: A realization between CVPR and CARP," *IEEE Trans. Evol. Comput.*, 19(5):644–658, 2014.

[15] L. Feng, Y. S. Ong, A. H. Tan, and I. W. Tsang, "Memes as building blocks:a case study on evolutionary optimization+transfer learning for routing problems," *Memet. Comput.*, 7(3):159–180, 2015.

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

[17] F. Glover, E. Taillard, and E. Taillard, "A user's guide to tabu search," *Ann. Oper. Res.*, 41(1):1–28, 1993.

[18] F. Glover, "Tabu search for nonlinear and parametric optimization (with links to genetic algorithms)," *Discrete Appl. Math.* 49:231–255, 1994.

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

[20] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Commun. ACM*, 16(6):372–378, 1973.

[21] Z. Lü and J.-K. Hao, "A memetic algorithm for graph coloring," *Eur. J. Oper. Res.*, 203(1):241–250, 2010.

[22] P. Morris, "The breakout method for escaping from local minima," in *Proc. 11th AAAI, USA, July 11-15, 1993*, pp. 40–45, 1993.

[23] P. Moscato, "Memetic algorithms: A short introduction," in: New Ideas in Optimization, McGraw-Hill Ltd., UK, pp. 219234, 1999.

[24] F. Neri, C. Cotta, and P. Moscato, "Handbook of Memetic Algorithms", Springer Berlin Heidelberg, 2012.

[25] Y. S. Ong, M. H. Lim, N. Zhou, and K. W. Wong, "Classification of adaptive memetic algorithms:a comparative study," *IEEE Trans. Systems, Man and Cybernetics, Part B.*, 36(1):141–152, 2006.

[26] Y. S. Ong, M. H. Lim, and X. Chen, "Memetic computation-past, present & future," *IEEE Comp. Int. Mag.*, 5(2):24–31, 2010.

[27] W. Pullan, "Heuristic identification of critical nodes in sparse real-world graphs," *J. Heuristics*, 21(5):577–598, 2015.

[28] D. Purevsuren, G. Cui, N. N. H. Win, and X. Wang, "Heuristic algorithm for identifying critical nodes in graphs," *Adv. Comput. Sci. Int. J.*, 5(3):1–4, 2016.

[29] 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.

[30] 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 Trans. Netw.*, 21(3):963–973, 2013.

[31] 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.

[32] M. D. Summa, A. Grosso, and M. Locatelli, "Branch and cut algorithms for detecting critical nodes in undirected graphs," *Comput. Optim. Appl.*, 53(3):649–680, 2012.

[33] V. Tomaino, A. Arulselvan, P. Veltri, and P. M. Pardalos, "Studying connectivity properties in human protein–protein interaction network in cancer pathway," *Data Mining for Biomarker Discovery*, 65:187–197, 2012.

[34] J. Thornton, "Clause weighting local search for SAT," *J. Autom Reasoning*, 35(1-3):97–142, 2005.

[35] M. Ventresca, "Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem," *Comp. Oper. Res.*, 39(11):2763–2775, 2012.

[36] M. Ventresca and D. M. Aleman, "A derandomized approximation algorithm for the critical node detection problem," *Comp. Oper. Res.*, 43:261–270, 2014.

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

[38] A. Veremyev, V. Boginski, and E. L. Pasiliao, "Exact identification of critical nodes in sparse networks via new compact formulations," *Optim. Lett.*, 8(4):1245–1259, 2014.

[39] A. Veremyev, O. A. Prokopyev, and E. L. Pasiliao, "An integer programming framework for critical elements detection in graphs," *J. Comb. Optim.*, 28(1):233–273, 2014.

[40] Q. Wu and J.-K. Hao, "A hybrid metaheuristic method for the maximum diversity problem," *Eur. J. Oper. Res*, 231(2):452–464, 2013.

[41] B. Yuan, B. Li, H. Chen, and X. Yao, "A new evolutionary algorithm with structure mutation for the maximum balanced biclique problem," *IEEE Trans. Cybern.*, 45(5):1054–1067, 2015.

[42] Y. Zhou and J.-K. Hao, "A fast heuristic algorithm for the critical node problem," in *Proc. Genet. Evol. Comput. Conf. Companion*, Berlin, Germany, July 15-19, 2017, ACM, 121–122.

[43] Y. Zhou, J.-K. Hao, and B. Duval, "Reinforcement learning based local search for grouping problems: A case study on graph coloring," *Expert Syst. Appl.*, 64:412–422, 2016.

[44] Y. Zhou, J.-K. Hao, and B. Duval, "Opposition-based memetic search for the maximum diversity problem," *IEEE Trans. Evol. Comput.*, 21(5):731–745, 2017.

[45] Y. Zhou, B. Duval, and J.-K. Hao, "Improving probability learning based local search for graph coloring," *Appl. Soft Comput.*, 65:542–553, 2018.

APPENDIX

To highlight the contributions of the key algorithmic components of the MACNP algorithm, we compare MACNP

TABLE VIII
COMPARATIVE RESULTS BETWEEN MACNP AND ITS FOUR VARIANTS. THE BEST RESULTS ARE INDICATED IN BOLD.

	MACNP		$MACNP_0$		$MACNP_1$		$MACNP_2$		$MACNP_3$	
Instance	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
BA5000	**10196.0**	**10196.0**	10198.0	10200.8	**10196.0**	**10196.0**	**10196.0**	**10196.0**	**10196.0**	10209.1
ER941	**5012.0**	**5013.7**	5014.0	5171.1	5014.0	5014.4	**5012.0**	5026.1	5019.0	5163.1
FF500	**257.0**	**257.0**	**257.0**	**257.0**	**257.0**	**257.0**	**257.0**	**257.0**	**257.0**	257.5
WS250	**3083.0**	3101.1	**3083.0**	3257.7	**3083.0**	**3093.8**	3373.0	3520.9	3085.0	3838.9
TreniR	**918.0**	**918.0**	**918.0**	**918.0**	**918.0**	**918.0**	**918.0**	**918.0**	**918.0**	**918.0**
H3000a	**2849170.0**	**2883554.5**	3451908.0	3483709.9	2862217.0	2913716.6	3072097.0	3102585.7	2931805.0	2989920.8
H4000	**5081209.0**	**5144354.2**	6165357.0	6224768.6	5187236.0	5313477.9	5531484.0	5569792.9	5322446.0	5377694.8
hepth	**106552.0**	**108354.0**	23964174.0	24528095.0	108453.0	111804.3	121295.0	126286.5	124871.0	137912.0
avg.	**1007049.6**	**1019468.6**	4200113.6	4282047.3	1022171.8	1044809.8	1093079.0	1102322.9	1049824.6	1065739.3

with four variants (named $MACNP_0$, $MACNP_1$, $MACNP_2$, $MACNP_3$) where a particular component of MACNP is disabled or replaced by a traditional alternative operator.

- $MACNP_0$ is obtained from MACNP by replacing the two-phase node exchange strategy of Section IV-D2 with the traditional node exchange strategy (e.g., [3]). See Section VII-A for more details.
- $MACNP_1$ is obtained from MACNP by disabling the node weighting technique of Section IV-D3. See Section VII-B for more details.
- $MACNP_2$ is obtained from MACNP by replacing the double backbone-based crossover of Section IV-E with a traditional backbone-based crossover (e.g., [40], [44]). See Section VII-C for more details.
- $MACNP_3$ is obtained from MACNP by replacing the local optimization component (i.e., the component-based neighborhood search of Section IV-D) with a classic mutation operator, which works as follows. Each time an offspring solution is created by the crossover, n_g swaps are randomly performed on the offspring solution. The value of n_g ($1 \leq n_g \leq |S|$) is given by $n_g = \lceil 1.0/x \rceil$ where $|S|$ is the number of the selected node in the current solution and x is a random value in $(0, 1]$. If $n_g > |S|$, we set $n_g = |S|$.

Table VIII summarizes the comparative results between MACNP and these variants on the set of 8 representative instances used in Section VII. These results are based on 15 runs of each algorithm to solve each instance with a time limit $t_{max} = 3600$ seconds per run. These results show that removing any component of the MACNP algorithm deteriorates its performance and the MACNP algorithm performs the best thanks to the symbiosis of its ingredients.