

Variable Population Memetic Search: A Case Study on the Critical Node Problem

Yangming Zhou, Jin-Kao Hao*, Zhang-Hua Fu*, Zhe Wang, and Xiangjing Lai

Abstract—Population-based memetic algorithms have been successfully applied to solve many difficult combinatorial problems. Often, a population of fixed size is used in such algorithms to record some best solutions sampled during the search. However, given the particular features of the problem instance under consideration, a population of variable size would be more suitable to ensure the best search performance possible. In this work, we propose variable population memetic search (VPMS), where a strategic population sizing mechanism is used to dynamically adjust the population size during the search process. Our VPMS approach starts its search from a small population of only two solutions to focus on exploitation, and then adapts the population size according to the search status to continuously influence the balancing between exploitation and exploration. We illustrate an application of the VPMS approach to solve the challenging critical node problem (CNP). We show that the VPMS algorithm integrating a variable population, an effective local optimization procedure and a backbone-based crossover operator performs very well compared to state-of-the-art CNP algorithms. The algorithm is able to discover new upper bounds for 12 instances out of the 42 popular benchmark instances, while matching 23 previous best-known upper bounds.

Index Terms—Memetic search, Population sizing, local search, critical node problem.

I. INTRODUCTION

Canonical memetic algorithms (denoted by MAs hereafter) are a hybrid metaheuristic that combines local search and population-based search [28]. MAs aim to benefit from the synergy between the exploitation power offered by local search and the exploration capacity provided by population-based search. Since their introduction, MAs have been applied with success to numerous combinatorial search problems including popular NP-hard problems (e.g., graph coloring [27], [34],

maximum diversity [43], graph partitioning [5]) and real applications (e.g., identification of critical nodes in graphs [45], influence maximization in multiplex networks [41], vehicle routing [13], [38]). Comprehensive surveys of representative research on canonical memetic algorithms and additional application examples can be found in, e.g., [10], [30].

From a perspective of computational models, canonical memetic algorithms can be considered as a particular member of the more recent memetic computation (MC) paradigm that emphasizes simultaneous problem learning and optimization via knowledge memes [16]. This work basically concerns canonical MAs given that they follow simple design principles and are known to be a quite useful framework for combinatorial optimization.

Typically, the design of an effective MA for a given problem requires specifying a number of algorithmic components including the local optimization procedure, the crossover operator, and the pool updating strategy [17], [19], [25], [31]. Additionally, since MAs rely on a population of individuals, the population size needs to be identified as well. We observe that existing studies on MA applications focus mainly on designing algorithmic components such as local optimization and crossover, while the issue of population size is typically neglected.

Generally, it is known that the population size of an evolutionary algorithm impacts the solution quality and running time. Indeed, there is a consensus that a small population implies a low population diversity and may lead to premature convergence of the algorithm, whereas a large population promotes diversity, nevertheless consumes more computational resources. However, the optimal population size of a population-based algorithm is generally problem dependent [12] and can even vary at different evolution stages of the search process [42]. As we observe from the literature review of Section II, a number of studies on population sizing have been dedicated to various evolutionary methods such as genetic algorithm, differential evolution, artificial bee colony algorithm and particle swarm optimization, while most of them have been studied for solving continuous optimization problems such as non-linear and multi-modal function optimization. On the other hand, very little effort has been made to investigate population sizing schemes for memetic algorithms in discrete optimization.

This work aims to fill the gap by focusing on memetic algorithms for combinatorial optimization and presenting the variable population memetic search (VPMS) method where a strategic population sizing mechanism is introduced in the MA framework. We summarize our contributions as follows.

First, from an algorithmic perspective, the proposed variable population memetic search enhances the popular MA framework with a strategic population sizing scheme to dynamically

Work partially supported by the National Natural Science Foundation of China (Grant 61903144, the Shanghai Sailing Program (Grant 19YF1412400), the Key Project of Science and Technology Innovation 2030 Supported by the Ministry of Science and Technology of China (Grant 2018AAA0101302), the Fundamental Research Funds for the Central Universities of China (Grant 222201817006), and the Shenzhen Science and Technology Innovation Commission (Grant JCYJ20180508162601910), the funding from Shenzhen Institute of Artificial Intelligence and Robotics for Society (Grant 2019-INT003). (Corresponding authors: Jin-Kao Hao and Zhang-Hua Fu)

Y. Zhou and Z. Wang are with the Key Laboratory of Advanced Control and Optimization for Chemical Processes, Ministry of Education and the School of Information Science and Engineering, East China University of Science and Technology, 130 Meilong Road, 200237 Shanghai, China (e-mails: ymzhou@ecust.edu.cn, wangzhe@ecust.edu.cn).

J.-K. Hao is with the Department of Computer Science, LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France and the Institut Universitaire de France, 1 rue Descartes, 75231 Paris, France (e-mail: jin-hao.hao@univ-angers.fr).

Z.-H. Fu is with the Shenzhen Institute of Artificial Intelligence and Robotics for Society, and The Chinese University of Hong Kong, Shenzhen, 518172 Shenzhen, China (e-mail: fuzhanghua@cuhk.edu.cn).

X. Lai is with the Institute of Advanced Technology, Nanjing University of Posts and Telecommunications, 210023 Nanjing, China (e-mail: laixiangjing@gmail.com).

influence the balancing between exploration and exploitation. A VPMS algorithm starts its search with a small population of two individuals (solutions) to favor exploitation. Upon reaching local optima solutions, the population is augmented with new high-quality solutions to strengthen population diversity and enhance exploration of the search space. When the population reaches a maximum allowable size while the search is still stagnating, it is shrunk to two individuals while maintaining the best solution found so far to start a new round of exploitation and exploration. It is expected that this strategic population sizing mechanism helps the MA algorithm to make its search more focused and more effective.

Second, from a computational perspective, we apply the proposed method to the challenging (NP-hard) critical node problem (CNP). In particular, we integrate a dedicated local improvement procedure and a structured crossover within the variable population memetic search framework. We demonstrate the competitiveness of the resulting VPMS algorithm on two sets of 42 synthetic and real-world benchmark instances in the literature and present new record results (improved upper bounds) for 12 instances. The VPMS approach is also the first heuristic algorithm able to steadily reach the optimal solutions for all 9 instances with known optima in only one minute.

Third, even if the work focuses on MAs for combinatorial optimization, the strategic population sizing mechanism of the VPMS method is of generic nature. As such, it can be used within other population-based algorithms to improve their search performances – we showcase such an example in this work for an existing MA (i.e., the MACNP algorithm of [45]). Finally, it is expected that the VPMS method will contribute to better solve various optimization problems.

The rest of this paper is organized as follows. Section II presents a brief literature review of studies on population sizing in evolutionary algorithms. Section III introduces the proposed VPMS approach. Section IV shows the case study of applying the general VPMS approach to the critical node problem, including detailed computational results and comparisons with state-of-the-art CNP algorithms. Finally, Section V summarizes the work and presents research perspectives.

II. RELATED WORK ON POPULATION CONTROL IN EVOLUTIONARY ALGORITHMS

Evolutionary algorithms (EAs) are population-based computation methods. One important issue concerning EAs is population control. Indeed, this issue has been investigated to improve genetic algorithms, differential evolution, artificial bee colony algorithms and particle swarm optimization [15], [33]. Existing studies on population control can be roughly divided into three categories as follows.

Deterministic methods change the population size during the evolution process according to some deterministic rules. For example, Brest and Maučec [7] presented such a method for differential evolution, which gradually decreases the population size by half each time a given condition is satisfied during the evolution. Brest et al. [8] introduced another method, which, starting from a small population, increases with a specific size determined by a constant value and then

reduces by half the population during the evolution. Instead of changing a specific number of individuals after a specific number of generations during evolution, a few methods have been proposed to automatically adjust the population size based on predefined functions. For example, Koumouis et al. [24] used a saw-tooth shape function to adjust the population size of a genetic algorithm.

Adaptive methods utilize feedback information from the search to determine the direction and magnitude of change of population size. For example, Arabas et al. [1] presented a genetic algorithm with variable population, which eliminates the population size as an explicit parameter by using features such as “age” and “maximal lifetime” of individuals. Eiben et al. [12] introduced a population sizing technique based on improvements of the best fitness in the population of a genetic algorithm: increase the population on fitness improvement or long period search stagnation, and decrease the population on short term stagnation. Besides the fitness of individuals, information on fitness diversity of the population was also used to control the population size. For example, Tirronen and Neri [39] proposed a population control method for differential evolution based on fitness diversity measured by the distances between pairs of individuals along with their fitness values.

Parameter-less methods try to get around the need of setting a fixed population size. For instance, Harik and Lobo [18] introduced the first parameter-less genetic algorithm where an unbounded number of populations with different sizes are dynamically created. Goldman and Funch [14] presented a parameter-less population pyramid (P3) method based on a pyramid-like structure of multiple-populations for performing evolutionary optimization without requiring any user-specified parameters. For both the linkage tree genetic algorithm (LTGA) and the dependency structure matrix genetic algorithm II (DSMGA-II) where the population size is the only parameter, Bosman et al. [6] proposed the population-sizing LTGA algorithm (psLTGA), while Komarnicki and Przewozniczek [23] developed the population-sizing DSMGA-II algorithm (psDSMGA-II).

We observe that although a variety of methods have been reported for population control in various EAs, studies on population control in memetic algorithms for discrete optimization are almost nonexistent. To our knowledge, the only study on population sizing in MAs was presented in [22]. In their work, Karapetyan and Gutin introduced a memetic algorithm for the multidimensional assignment problem, where the population size is adjustable according to a function of the running time of the whole algorithm and the average running time of the local search for the given instance. Specifically, the average running time of the local search procedure is obtained during the algorithm’s run. Their technique supposes a limited running time for the MA and varying the population size aims to best utilize the given time budget.

In this work, we aim to investigate the issue of population control to reinforce the MA framework for combinatorial optimization. Specifically, inspired by existing studies on population management for EAs such as [7], [8], [12] and successful memetic algorithms with small populations such as [27], [34], [38], [45], we introduce the variable population

memetic search (VPMS) which integrates a strategic population sizing mechanism into memetic search. As we show in Section IV, the VPMS approach indeed enables us to design highly effective algorithms for hard optimization problems.

III. VARIABLE POPULATION MEMETIC SEARCH

In this section, we present the variable population memetic search (VPMS) framework, which introduces a strategic population sizing mechanism into memetic algorithms.

A. General scheme

Like any population-based search algorithm, the performance of a memetic algorithm depends critically on its ability of maintaining a suitable balance of exploration and exploitation of the search space. The proposed variable population memetic search (VPMS) framework aims to encourage such a search balance via a dynamic population sizing mechanism.

From a search perspective, the VPMS approach starts with a small population of two individuals (solutions) to favor exploitation and then strategically adjusts the populations size to influence the population diversity and thus the balance of exploitation and exploration.

From an algorithmic perspective, VPMS mainly consists of five components: population building (Section III-B), offspring solution construction (Section III-C), local improvement (Section III-D), population updating (Section III-E) and population sizing (Section III-F). As shown in Algorithm 1, VPMS starts with an elite population of only two solutions that are obtained by the *PopulationBuilding()* procedure (line 4). From this small elite population, VPMS enters a “while” loop (lines 8-26) to perform its evolutionary search until a given stopping condition is satisfied. At each generation, two or more parents are selected to create an offspring solution based on the *OffspringSolutionConstruction()* procedure (line 10). Afterward, the offspring solution is further improved by the *LocalImprovement()* procedure (line 12). The improved offspring solution is then inserted into the population according to the *PopulationUpdating()* procedure (line 22). In addition to these basic components of a general memetic algorithm, the proposed VPMS approach specifically integrates a new component to dynamically control the population size according to the *PopulationSizing()* procedure (line 24). With the help of this strategic population sizing mechanism, the algorithm adapts (i.e., increases or decreases) its population size according to the current search status.

B. Population building

VPMS uses a **population building strategy** to build an initial population of two solutions. Generally, these initial solutions can be obtained by any means, for instance, randomly or with a fast greedy procedure. In the spirit of MAs, it is preferable to start the search with some high-quality local optima. Thus, the initial solutions are typically improved by a local improvement procedure before being added in the population. Our population building strategy has the particularity of starting with a very small population of only two solutions.

Algorithm 1: Variable population memetic search

Input: Problem instance I with a minimization objective f .

Output: The best found solution S^*

```

1 begin
2   //build an elite population of two solutions; Sect. III-B
3    $ps \leftarrow 2$ ;
4    $P = \{S_1, S_2\} \leftarrow \text{PopulationBuilding}(ps)$ ;
5   //record the best solution
6    $S^* \leftarrow \arg \min_{i \in [1,2]} f(S_i)$ ;
7    $gens \leftarrow 0$ ,  $idle\_gens \leftarrow 0$ ;
8   while a stopping condition is not reached do
9     //construct an offspring solution; Sect. III-C
10     $S' \leftarrow \text{OffspringSolutionConstruction}(P)$ ;
11    //improve it by local optimization; Sect. III-D
12     $S' \leftarrow \text{LocalImprovement}(S', \text{MaxIdleIters})$ ;
13    //update the best solution
14    if  $f(S') < f(S^*)$  then
15       $S^* \leftarrow S'$ ;
16       $idle\_gens \leftarrow 0$ ;
17    end
18    else
19       $idle\_gens \leftarrow idle\_gens + 1$ ;
20    end
21    //update the population; Sect. III-E
22     $P \leftarrow \text{PopulationUpdating}(P, S')$ ;
23    //control population size; Sect. III-F
24     $P \leftarrow \text{PopulationSizing}(P, idle\_gens)$ ;
25     $gens \leftarrow gens + 1$ ;
26  end
27 end
28 return The best found solution  $S^*$ 

```

This is based on three considerations. First, since the search space is not examined yet at the beginning of the search, it is desirable to perform, with a reasonable computation time, an intensified exploitation to locate as fast as possible some first promising regions (represented by high-quality local optima). Second, building an initial population of multiple high-quality solutions in a MA for large problems may be time-consuming due to the application of local optimization. In some settings where the allowable time budget is short, the total time can fully be consumed during the population building phase, leaving no time for further search (see [45] for an example). Third, many successful MAs for difficult optimization problems including those mentioned in Section I employ small populations of ten to a few dozen individuals (some highly powerful MAs, such as [27], even use a fixed population of only two solutions).

C. Offspring solution construction

Offspring solution construction is an important component of a memetic algorithm and forms one leading force for exploration. It aims to create new solutions (offspring) by blending existing solutions. Crossover is a typical recombination operator, which is responsible for exploring new search areas of the solution space. Crossover operator usually consider two or more parents to form one or more new solutions. Within the VPMS method, the choice of the most suitable crossover operators follows the general recommendations for evolutionary and memetic algorithms. As the first approach, various popular crossover operators can be

considered according to the adopted representations [32]. In addition to these (general) operators, it is often advantageous to design dedicated crossovers for the studied problem such that offspring solutions inherit meaningful features (building blocks) from the parent solutions, as shown in many successful MAs (e.g., [21], [27], [34], [45]).

D. Local improvement

Local improvement (also called local optimization) plays a critical role in a memetic algorithm and ensures essentially the role of intensive exploitation of the search space by focusing on a limited region. In principle, the local improvement procedure of a VPMS algorithm can benefit from many general local search methods [20] such as hill climbing, simulated annealing, tabu search, threshold accepting, and variable neighborhood search. Still, these general methods need to be adapted to the given problem in particular by designing suitable search components. In particular, neighborhood is one critical ingredient that structures the way of the space being examined and thus largely determines the performance of the local improvement procedure. Also, specific techniques should be sought to streamline the evaluation of neighbor solutions. Moreover, for constrained problems, decisions should be made concerning whether the search is restricted to feasible solutions or allowed to explore infeasible solutions.

E. Population updating

For each offspring solution obtained by the solution construction component (Section III-C) and further improved by the local improvement component (Section III-D), a decision is made to determine whether and how the offspring is inserted into the population according to a pool updating strategy. For this purpose, existing population replacement strategies for evolutionary algorithms can be used in the VPMS approach. A simple and fitness based updating strategy replaces the worst solution if the offspring has a better quality and is different from any solution in the population. To better manage the population diversity, it is often advantageous to apply a more elaborated updating method (eg. [34], [36]) that simultaneously considers offspring's quality and its distance to the individuals in the population.

F. Population sizing

As its key component, our VPMS approach integrates a **strategic population sizing** (SPS) mechanism (see Algorithm 2) to dynamically adjust the population size during the evolutionary search. This mechanism is composed of a population expanding strategy (to add new individuals) and a population rebuilding strategy (to shrink the population to two individuals). In general terms, we expand the population with new *elite* solutions when a search stagnation is detected. If the population becomes too large but the search still stagnates, we reduce the population to two solutions. A search stagnation occurs when the best recorded solution S^* has not been updated after $MaxIdleGens$ consecutive generations.

Algorithm 2: The pseudo code of the strategic population sizing mechanism.

Input: Population P of size p , maximum allowable population size p_{max} , population size increment p_{inc} and counter of generations without improvement $idle_gens$.

Output: A new population P

```

1 begin
2   if  $idle\_gens > MaxIdleGens$  then
3     //expand the population by adding new solutions;
4     if  $p < p_{max}$  then
5        $p \leftarrow p + p_{inc}$ ;
6        $P \leftarrow PopulationExpanding(P, p)$ ;
7     end
8     //rebuild population based on the best solution;
9     else
10       $p \leftarrow 2$ ;
11       $P \leftarrow PopulationRebuilding(S^*, p)$ ;
12    end
13    //update the best solution;
14     $S^* \leftarrow \arg \min_{i \in [1, \dots, p]} f(S_i)$ ;
15     $idle\_gens \leftarrow 0$ ;
16  end
17 end
18 return A new population  $P$ 

```

1) *Population expanding*: When the search stagnates, we try to break the stagnation by introducing more diversity into the algorithm. It is a common sense that the greater the population size, the greater the population diversity and vice versa. Therefore, we increase the diversity by expanding the population upon search stagnation. Specifically, our **population expanding strategy** adds p_{inc} new high quality solutions into the population, where each new solution is generated according to the population building strategy of Section III-B and added to the population only if the new solution is not the same as any existing solution in the population. p_{inc} is an adjustable parameter, which is able to influence the population diversity. Intuitively, a large p_{inc} value impacts more the diversity of the population than a small value.

2) *Population rebuilding*: When the population size reaches an allowable threshold value p_{max} , but the search is still stagnating, the **population rebuilding strategy** is triggered to rebuild the population. This decision relies on two considerations. First, when the above condition is met, the population is sampling non-promising search regions where no better solution can be expected. Second, large populations usually consume more computation resources. To displace the search to new regions, we shrink the population to a small population of only two solutions. The new population retains always the best recorded solution S^* and includes another *elite* solution generated in the same way as the population building strategy of Section III-B. By mixing the historic best solution and a new high-quality solution, the new population is expected to benefit both from previous search outcome and fresh diversity. One notices that a large p_{max} value leads to a less frequent population rebuilding and vice versa. Varying p_{max} can thus change the behavior and search trajectory of the algorithm.

Finally, we mention that like [1], [12], [22], [39], our SPS scheme follows the general idea of adaptively varying

the population size during evolution. However, the conditions governing population re-sizing and the way of re-sizing the population of our method are different from the existing studies. As showcased in Section IV, MAs equipped with our SPS scheme are capable of reaching very high performances.

G. Discussions

From the perspective of computational models, the VPMS approach enhances the canonical memetic framework with the adaptive population sizing strategy. As such, VPMS can be considered to belong to the broader memetic computation paradigm presented in [16] where optimization and data-driven adaption take place simultaneously and create a desirable symbiosis for a better search performance.

Moreover, our presentation was focused on the integration of the strategic population sizing mechanism within the memetic algorithm framework for combinatorial optimization. It should be clear that the SPS mechanism is general and could be advantageously used by or tailored to other population-based optimization algorithms. With the help of our SPS mechanism, the algorithm would be able to adaptively adjust its population size to a suitable value based on the search situation, increasing thus the search performance while getting rid of the difficult task of tuning this parameter by hand.

IV. VPMS APPLIED TO THE CRITICAL NODE PROBLEM

This section presents a practical application of the VPMS approach to solve the critical node problem (CNP) and demonstrates its competitiveness compared to the state of the art.

A. Critical node problem

Let $G = (V, E)$ be an undirected graph with $|V| = n$ nodes and $|E| = m$ edges, the critical node problem (CNP) involves identifying a subset of nodes $S \subseteq V$ ($|S| \leq k$) such that the removal of the vertices in S leads to a residual graph $G[V \setminus S]$ with the minimum pairwise connectivity. These removed nodes are usually called as *critical nodes*. Once the critical nodes have been removed from G , the residual graph $G[V \setminus S]$ can be represented by a set of disjoint connected subgraphs (i.e., components) $\mathcal{H} = \{C_1, C_2, \dots, C_T\}$, where a connected component C_i is a set of nodes such that there exists a path from a node to any other node in this component, and no edge exists between any two connected components [4].

Since any subset $S \subset V$ of k nodes is a feasible solution for the given graph, the search space Ω is composed of all possible k -node subsets of V , i.e., $\Omega = \{S \subset V : |S| = k\}$. Clearly this search space has a size of $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, which increases extremely fast with n and k . Unsurprisingly, CNP is NP-hard and thus computationally challenging.

Recall that $\sum_{i,j \in V} u_{ij}$ is a measure of the total pairwise connectivity of a graph, where $u_{ij} = 1$ if and only if node i and node j are in the same component, otherwise $u_{ij} = 0$. Therefore, the objective function (to be minimized) can be rewritten as

$$f(S) = \sum_{C_i \in \mathcal{H}} \frac{|C_i|(|C_i| - 1)}{2} \quad (1)$$

where S is a set of critical nodes, $|C_i|$ is the size of the i -th component of the residual graph $G[V \setminus S]$. It is known that $f(S)$ can be easily computed by fast algorithms like breadth or depth first search algorithms in $O(|V| + |E|)$ time using an adjacency list representation of the graph.

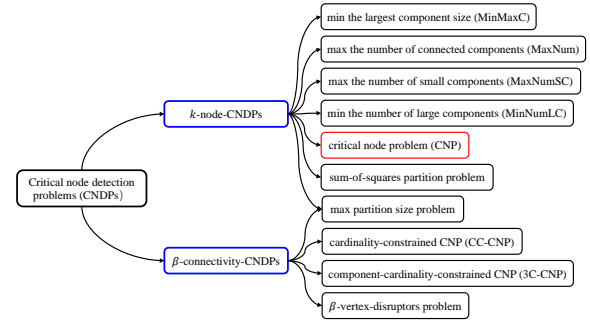


Fig. 1. A taxonomy of critical node detection problems.

CNP is one of the critical node detection problems (CNDPs). There are many interesting variants of CNP, which optimize different objectives, such as minimizing the size of the largest connected component and maximizing the number of connected components. A detailed classification of the main CNDPs is provided in Fig. 1, while more details about these CNDPs can be found in the recent survey [26].

Given its practical and theoretical significance, CNP has been widely studied in the literature. Compared to exact algorithms, heuristic algorithms demonstrated better performances on the CNP benchmark instances. Representative heuristic algorithms include population-based incremental learning approach [40], sophisticated multi-start greedy algorithm (CNA1) [35], fast iterated local search approach (FastCNP) [44] and memetic search approach (MACNP) [45]. It is worth noting that most state-of-the-art results on the CNP benchmark instances were reported by CAN1, FastCNP and MACNP. These algorithms will thus be used as reference algorithms in our comparisons in Section IV-C5.

B. Variable population memetic algorithm for CNP

Our variable population memetic search algorithm for CNP (denoted by VPMS_{CNP}) strictly follows Algorithm 1, while specifying the solution construction component and the local improvement component. Additionally, the initial population of two solutions is obtained in the same way as in MACNP [45], and the rank-based quality-and-distance pool updating strategy is adopted to manage the population.

1) *Double backbone-based crossover*: For solution construction, we adopt the double backbone-based crossover (DBC) [45], which performs structured combinations by inheriting common elements from two parents. Specifically, given two parents S_1 and S_2 randomly taken from the population, DBC generates an offspring solution in three steps: a) create a partial solution by inheriting the common elements shared by the parents (i.e., identified by the set $S_1 \cap S_2$); b) add the elements from the set $(S_1 \cup S_2) \setminus (S_1 \cap S_2)$ into the partial solution in a probabilistic way; c) repair the solution structurally until a feasible solution is achieved by either

adding elements from the set $V \setminus (S_1 \cup S_2)$ or removing elements from the solution. Once a feasible offspring solution is obtained, it is further ameliorated by the diversified late acceptance search procedure below.

2) *Diversified late acceptance search*: Diversified late acceptance search (DLAS) [29] is an iterative local search algorithm that is inspired by the late acceptance hill climbing (LAHC) algorithm [9]. Both DLAS and LAHC start their search from an initial solution and iteratively accept or reject candidate solutions until a given stopping condition is met. The LAHC method uses a fitness array of size HL (i.e., *history length*) to memorize the cost of the previous encountered solutions. Initially, all elements of this array are filled with the cost of the initial solution S . At each subsequent iteration $iters$, a candidate solution S' is generated. Then, an acceptance decision is made according to a comparison between the cost of S' and the previous solution cost stored at position v (the virtual beginning of the fitness array, $v \leftarrow iters \bmod HL$). Specifically, the candidate solution S' is accepted if its cost is not worse than the cost f_v at position v of the fitness array. After the transition from the current solution to S' (i.e., S' becomes the new current solution), the value of position v of the fitness array is updated by $f_v \leftarrow f(S')$. The process repeats until the given stopping condition is met.

DLAS (Algorithm 3) enhances LAHC by increasing the diversity of the accepted solutions and improving the diversity of the values stored in the fitness array. This is achieved by adopting a new acceptance strategy and a new replacement strategy that take into account worsening, improving, and sideways movement scenarios [29] (lines 14-35). Specifically, the new acceptance strategy compares at each iteration the fitness value $f(S')$ of the candidate solution S' with the maximum fitness value f_{max} (instead of f_v) in the fitness array (lines 14-23). For the new replacement strategy, the replacement occurs only in two cases: 1) if $f(S) > f_v$, and 2) if $f(S) < f_v$ and $f(S) < f_{prev}$ (line 27). Our experiments showed that the combination of the new acceptance and replacement strategies in DLAS is indeed quite effective in increasing the search diversity, and helps the algorithm to reach high quality solutions in less time.

To generate a candidate solution, DLAS relies on the component-based two-phase node exchange operator (denoted by $swap$) introduced in [45], which swaps a node $u \in S$ and a node $v \in V \setminus S$ from a *large component*. Let $G[V \setminus S]$ be the residual graph $G[V \setminus S]$ induced by the current solution S and $\mathcal{H} = \{C_1, C_2, \dots, C_T\}$ be the set of connected components of $G[V \setminus S]$. For a swap operation, we consider as candidate nodes a restricted set of nodes $W \subset V \setminus S$ such that $W = \cup_{|C_i| \geq L} C_i$, where L is a predefined threshold value to qualify large components in the residual graph. Thus, a candidate neighbor solution S' is obtained by $swap(u, v)$, where $u \in S$ and $v \in W$. For a given candidate solution, its quality can be evaluated in $O(|V| + |E|)$ time with a modified depth-first search algorithm according to Eq. (1).

C. Computational studies of VPMS for CNP

This section is devoted to an experimental evaluation of the performance of the $VPMS_{CNP}$ algorithm in comparison with

Algorithm 3: The pseudo code of the DLAS procedure.

Input: Initial solution S and maximum allowable number of idle iterations $MaxIdleIters$.

Output: The best found solution S^*

```

1 begin
2   Initialize the history length  $HL$ ,  $S^* \leftarrow S$ ;
3   for  $\forall i \in \{0, \dots, HL - 1\}$  do
4      $f_i \leftarrow f(S)$ ;
5   end
6    $f_{max} \leftarrow f(S)$ ,  $nbr\_max \leftarrow HL$ ;
7   Initialize  $iters \leftarrow 0$ ,  $idle\_iters \leftarrow 0$ ;
8   while  $idle\_iters < MaxIdleIters$  do
9      $f_{prev} \leftarrow f(S)$ ;
10     $S' \leftarrow SwapOperation(S)$ ;
11    Calculate its cost function  $f(S')$ ;
12    //calculate the virtual beginning;
13     $v \leftarrow iters \bmod HL$ ;
14    if  $f(S') = f(S)$  or  $f(S') < f_{max}$  then
15       $S \leftarrow S'$ ,  $f(S) \leftarrow f(S')$ ;
16      if  $f(S) < f(S^*)$  then
17         $S^* \leftarrow S$ ,  $f(S^*) \leftarrow f(S)$ ;
18         $idle\_iters \leftarrow 0$ ;
19      end
20    else
21       $idle\_iters \leftarrow idle\_iters + 1$ ;
22    end
23  end
24  if  $f(S) > f_v$  then
25     $f_v \leftarrow f(S)$ ;
26  end
27  else if  $f(S) < f_v$  and  $f(S) < f_{prev}$  then
28    if  $f_v = f_{max}$  then
29       $nbr\_max \leftarrow nbr\_max - 1$ ;
30    end
31     $f_v \leftarrow f(S)$ ;
32    if  $nbr\_max = 0$  then
33      compute  $f_{max}, nbr\_max$ ;
34    end
35  end
36   $iters \leftarrow iters + 1$ ;
37 end
38 end
39 return The best found solution  $S^*$ 

```

state-of-the-art CNP algorithms.

1) *Benchmark instances*: Our computational experiments were performed on two widely-used benchmark datasets: synthetic dataset¹ and real-world dataset². The **synthetic dataset** presented in [40] is composed of 16 graphs with various structures. The **real-world dataset** introduced in [2] includes 26 instances from several practical applications. The main features of these two datasets are provided in Table I.

2) *Experimental settings*: All our algorithms³ were implemented in the C++ programming language, and compiled using GNU gcc 4.1.2 with '-O3' option on an Intel E5-2670 with 2.5GHz and 2GB RAM under Linux. With a '-O3' flag, running the DIMACS machine benchmark program $dfmax$ ⁴

¹ Available at <http://individual.utoronto.ca/mventresca/cnd.html>

² Available at <http://www.di.unito.it/~aringhie/cnp.html>

³ The best solution certificates and our programs will be made available at <http://www.info.univ-angers.fr/pub/hao/VPMS.html>

⁴ Available at $dfmax$: <ftp://dimacs.rutgers.edu/pub/dsj/cliue>

TABLE I
CHARACTERISTICS OF THE SYNTHETIC AND REAL-WORLD DATASETS
USED IN THE EXPERIMENTS.

Instance	V	E	k	Instance	V	E	k
BA500	500	499	50	FF250	250	514	50
BA1000	1000	999	75	FF500	500	828	110
BA2500	2500	2499	100	FF1000	1000	1817	150
BA5000	5000	4999	150	FF2000	2000	3413	200
ER235	235	350	50	WS250	250	1246	70
ER466	466	700	80	WS500	500	1496	125
ER941	941	1400	140	WS1000	1000	4996	200
ER2344	2344	3500	200	WS1500	1500	4498	265
Bovine	121	190	3	Ham3000c	3000	5996	300
Circuit	252	399	25	Ham3000d	3000	5993	300
E.coli	328	456	15	Ham3000e	3000	5996	300
USAir97	332	2126	33	Ham4000	4000	7997	400
humanDisea	516	1188	52	Ham5000	5000	9999	500
Treni_Roma	255	272	26	powergrid	4941	6594	494
EU_flights	1191	31610	119	Oclinks	1899	13838	190
openflights	1858	13900	186	facebook	4039	88234	404
yeast1	2018	2705	202	grqc	5242	14484	524
Ham1000	1000	1998	100	hepth	9877	25973	988
Ham2000	2000	3996	200	hepph	12008	118489	1201
Ham3000a	3000	5999	300	astroph	18772	198050	1877
Ham3000b	3000	5997	300	condmat	23133	93439	2313

on our machine requires 0.19, 1.17 and 4.54 seconds to solve graphs r300.5, r400.5 and r500.5 respectively.

TABLE II
PARAMETER SETTINGS OF THE PROPOSED VPMS_{CNP} ALGORITHM.

Parameter	Description	Value
p_{max}	maximal population size	20
p_{inc}	increment for population expansion	2
$MaxIdleGens$	maximum number of idle generations in VPMS	100
$MaxIdleIters$	maximum number of idle iterations in DLAS	1000

In the following experiments, we use the well-known two-tailed sign test to check the statistical significance of our comparisons between two algorithms on each comparison indicator. This statistical test is based on the number of instances on which an algorithm is the overall winner, and it is highly recommended in [11]. There are $N = 42$ benchmark instances in our experiments. At a significant level of 0.05, the critical value is $CV_{0.05}^{42} = N/2 + 1.96\sqrt{N}/2 \approx 27$. This means that algorithm A significantly outperforms algorithm B if A wins at least 27 out of 42 instances.

3) *Effectiveness of the strategic population sizing mechanism*: Compared to the conventional memetic search framework, the VPMS_{CNP} algorithm integrates the strategic population sizing mechanism to dynamically adjust the population size during the evolutionary search. To verify the effectiveness of our population sizing mechanism, we compare VPMS_{CNP} with an alternative algorithm named FPMS_{CNP} whose population size is fixed to the maximal population size of VPMS_{CNP} while keeping the other components as the same as VPMS_{CNP}. As such, FPMS_{CNP} is a classical memetic algorithm which is quite similar to the powerful state-of-the-art memetic algorithm MACNP of [45] where a different local improvement procedure is used. We show

additional comparisons between VPMS_{CNP} and FPMS_{CNP} with other population sizes in Table IX of the Appendix.

To make a fair comparison between VPMS_{CNP} and FPMS_{CNP}, we ran them on the same computing platform with the setting shown in Table II. We independently solved each instance 30 times with different random seeds, and the time limit of each run was limited to $t_{max} = 3600$ seconds. Detailed comparative results for both synthetic and real-world datasets are summarized in Table III.

In Table III, columns 1 and 2 present for each instance its name (Instance) and the best-known value (f_{bku}) reported in the literature [3], [35], [45]. Columns 3-7 report the results of the FPMS_{CNP} algorithm, namely the best objective value (f_{best}) found during 30 runs, the average objective value (f_{avg}), the average running time per run to attain a best objective value (t_{avg}), the average number of generations per run required to find the best objective value ($\#gens$), and the number of times to successfully find the best objective value ($\#succ$). Similarly, columns 8-12 give the results of VPMS_{CNP}. The best values of the compared results in terms of f_{best} and f_{avg} are indicated in bold. For the $\#succ$ indicator, we compare them only when the same f_{best} values are obtained by the two algorithms.

From Table III, we observe that in terms of f_{best} , the VPMS_{CNP} algorithm (with a variable population) achieves 14 better, 20 equal and 8 worse results compared to the fixed population algorithm FPMS_{CNP}. However, there is no significant difference between these two algorithms (i.e., $24 < CV_{0.05}^{42}$). For the f_{avg} indicator, VPMS_{CNP} attains 25 better, 10 equal and 7 worse results. At a significant level of 0.05, VPMS_{CNP} is significantly better than FPMS_{CNP} ($30 > CV_{0.05}^{42}$). Although VPMS_{CNP} and FPMS_{CNP} achieve the same f_{best} values for 20 out of 42 synthetic instances, VPMS_{CNP} attains these results with a higher, an equal and a worse success rate on 8, 10 and 2 instance, respectively. It is worth noting that VPMS_{CNP} is the first heuristic to steadily (100%) reach the optimal solutions for all 9 instances with known optima (marked by “*” in Table III) in only one minute. For the last three large instances, VPMS_{CNP} is able to attain better results than FPMS_{CNP} even if the results are still worse than the f_{bku} values. Finally, compared to the f_{bku} values of all 42 benchmark instances, these two algorithms together improve on the best-known results (new upper bounds) on 8 instances (marked by “★”) and match the best-known upper bounds on 22 instances. These results provide thus the first positive indications of our strategic population sizing mechanism.

To further study the behavior of the VPMS_{CNP} algorithm, we report in Table IV the comparative results between VPMS_{CNP} and FPMS_{CNP} with a longer time limit $t_{max} = 7200$ seconds. We observe that both VPMS_{CNP} and FPMS_{CNP} improve their results. Importantly, the performance difference between VPMS_{CNP} and FPMS_{CNP} is more obvious than the results shown in Table III. Specifically, VPMS_{CNP} significantly outperforms FPMS_{CNP} in terms of both f_{best} (i.e., $27 \geq CV_{0.05}^{42}$) and f_{avg} (i.e., $31.5 > CV_{0.05}^{42}$). Moreover, these algorithms are able to find new upper bounds on 12 instances (marked by “★”) and match the best-known

TABLE III
COMPARISON OF VPMS_{CNP} (WITH A VARIABLE POPULATION) AGAINST FPMS_{CNP} (WITH A FIXED POPULATION) UNDER $t_{max} = 3600$ SECONDS.

Instance	FPMS _{CNP}						VPMS _{CNP}				
	f_{bkv}	f_{best}	f_{avg}	t_{avg}	#gens	#succ	f_{best}	f_{avg}	t_{avg}	#gens	#succ
BA500	195*	195	195.0	0.0	0	30	195	195.0	0.0	0	30
BA1000	558*	558	558.1	0.0	0	29	558	558.0	2.4	27	30
BA2500	3704*	3704	3704.6	2.8	6	29	3704	3704.0	7.2	117	30
BA5000	10196*	10196	10196.0	21.3	6	30	10196	10196.0	10.4	50	30
ER235	295*	295	295.0	13.6	3539	30	295	295.0	2.0	435	30
ER466	1524	1524	1524.0	45.0	5181	30	1524	1524.0	30.3	3111	30
ER941	5012	5012	5034.0	442.5	25209	5	5012	5026.5	459.2	22890	3
ER2344	902498	912875	931976.9	2456.7	18838	1	904113	933943.7	3012.8	15202	1
FF250	194*	194	194.0	8.9	23610	30	194	194.0	0.0	0	30
FF500	257*	257	257.3	5.0	4299	28	257	257.0	0.5	50	30
FF1000	1260*	1260	1262.3	354.1	17751	16	1260	1260.0	11.7	554	30
FF2000	4545*	4545	4547.8	20.5	402	13	4545	4545.0	43.9	1851	30
WS250	3083	3083	3093.3	1397.5	63236	23	3083	3083.1	1081.5	52449	29
WS500	2072	2078	2089.5	249.3	21014	1	2072	2083.1	366.7	25120	4
WS1000	109807	109677*	126764.6	2629.1	17445	1	119444	134475.5	1506.9	6696	1
WS1500	13098	13146	13329.1	1873.6	85821	1	13098	13161.5	2114.9	31819	9
Bovine	268	268	268.0	0.0	0	30	268	268.0	0.0	0	30
Circuit	2099	2099	2099.0	1.3	313	30	2099	2099.0	1.0	229	30
Ecoli	806	806	806.0	0.0	0	30	806	806.0	0.0	8	30
USAir97	4336	4336	4897.2	1126.8	60012	12	4336	5075.6	1159.5	37122	7
humanDisea	1115	1115	1115.3	3.1	292	29	1115	1115.0	1.6	180	30
Treni_Roma	918	918	918.0	29.7	10216	30	918	918.0	1.8	765	30
EU_flights	348268	348268	351323.0	74.3	77	2	348268	349265.6	1145.4	2319	18
openflights	26842	26842	28845.3	1812.7	7313	1	26785*	27327.0	2391.7	9806	2
yeast1	1412	1412	1412.0	18.1	104	30	1412	1412.0	35.9	437	30
Ham1000	306349	308731	311422.8	2431.2	22374	1	307117	311169.4	2027.2	13862	1
Ham2000	1243859	1244335	1257388.5	2545.1	9134	1	1247652	1256573.8	3109.8	7229	1
Ham3000a	2844393	2841106*	2861888.3	2553.6	4859	1	2840941*	2859284.4	3084.4	4660	1
Ham3000b	2841270	2839733*	2860997.6	2542.4	4964	1	2839893*	2860810.9	3179.3	4538	1
Ham3000c	2838429	2836076*	2848545.9	2313.0	4411	1	2832073*	2844324.3	2819.7	4080	1
Ham3000d	2831311	2830098*	2854757.2	2903.8	5093	1	2830291*	2857201.4	3090.1	4608	1
Ham3000e	2847909	2846371*	2866095.2	2106.1	3943	1	2846731*	2867000.6	3231.6	4816	1
Ham4000	5044357	5060754	5143157.3	2813.4	3132	1	5082521	5141804.3	3404.7	3705	1
Ham5000	7972525	7986458	8098821.1	3034.5	1943	1	8011565	8151850.1	3214.4	2970	1
powergrid	15862	15899	15954.5	1343.6	10222	1	15873	15909.2	2964.3	15205	1
Oclinks	611326	614467	615030.0	601.6	1276	2	611254*	614296.3	1658.4	4229	1
facebook	420334	703330	798567.9	2708.3	5219	1	691232	780429.1	3397.0	3753	1
grqc	13596	13612	13647.2	802.3	2957	1	13603	13615.5	2499.9	6367	2
hepht	106397	107440	109304.9	2700.6	2459	1	107939	110158.4	3206.6	2198	1
hepph	6156536	9327422	10712034.3	3491.3	7	1	7883063	8689170.1	3423.8	565	1
astroph	53963375	61928888	63311361.7	1684.9	0	1	58322396	59563941.1	2721.5	225	1
condmat	2298596	10352129	10823216.8	1682.5	0	1	6843993	7813436.7	3388.5	414	1

* Optimal solutions obtained by a branch-and-cut algorithm [37] within 5 days.
* Improved upper bounds.

upper bounds on 23 instances. These findings indicate that the strategic population size mechanism enables the VPMS_{CNP} algorithm to use its given computational budget more efficiently and more effectively to find high-quality solutions.

4) *Using the strategic population sizing mechanism to enhance a memetic algorithm:* MACNP [45] is a recent state-of-the-art memetic algorithm for both CNP and CC-CNP. We verify now whether the strategic population sizing mechanism can enhance the performance of this memetic algorithm. For this purpose, we replace the fixed population of MACNP by the SPS mechanism and use MACNP^{VP} to denote the resulting MACNP variant. We compare the original MACNP algorithm (with a fixed population) and MACNP^{VP} (with a variable population), based on the 26 real-world benchmark instances. We run both algorithms 30 times on each instance with $t_{max} = 3600$ seconds. The comparative results in terms of the f_{best} and f_{avg} indicators are shown in Fig. 2. The x -axis indicates the instances (named by integer numbers), and the y -axis presents the gap of f (f_{best} or f_{avg}) values to the best-known values f_{bkv} , i.e., $(f - f_{bkv})/f_{bkv}$. Therefore, a

negative gap value indicates an improved best upper bound.

From Fig. 2, we observe that the variable population algorithm MACNP^{VP} significantly outperforms the fixed population algorithm MACNP in terms of f_{best} and f_{avg} . Specifically, Fig. 2(a) indicates that MACNP^{VP} achieves better f_{best} values than MACNP except for the 21th instance (*facebook*). A close look of these results (see Fig. 2(b)) shows that MACNP^{VP} achieves eight new upper bounds. Additionally, In terms of the f_{avg} indicator, MACNP^{VP} achieves 15 better, 9 equal and 2 worse results compared to MACNP (see Fig. 2(c) and 2(d)). This study confirms that the state-of-the-art MACNP algorithm can definitively benefit from the strategic population sizing mechanism proposed in this work.

5) *Comparisons with state-of-the-art algorithms:* We report now a comparative study with respect to three recent state-of-the-art CNP algorithms: CAN1 [35], FastCNP [44] and MACNP [45]. To our knowledge, the best-known results available in the literature were achieved by these three algorithms except for instances *facebook* and *condmat*. To ensure the fairness of the experiment, we ran all the algorithms (with

TABLE IV
COMPARISON OF VPMS_{CNP} AGAINST FPMS_{CNP} UNDER $t_{max} = 7200$ SECONDS.

Instance	FPMS _{CNP}						VPMS _{CNP}				
	f_{bkv}	f_{best}	f_{avg}	t_{avg}	#gens	#succ	f_{best}	f_{avg}	t_{avg}	#gens	#succ
BA500	195	195	195.0	0.0	0	30	195	195.0	0.0	0	30
BA1000	558	558	558.1	0.2	39	29	558	558.0	0.3	4	30
BA2500	3704	3704	3704.0	3.3	11	30	3704	3704.0	6.7	47	30
BA5000	10196	10196	10196.0	31.4	7	30	10196	10196.0	11.8	58	30
ER235	295	295	295.0	97.1	20535	30	295	295.0	2.2	536	30
ER466	1524	1524	1524.0	42.1	5178	30	1524	1524.0	28.1	3180	30
ER941	5012	5012	5029.2	254.4	16860	5	5012	5017.0	1754.7	91144	4
ER2344	902498	902875	927689.7	4815.0	35997	1	906904	927865.4	5221.2	27003	1
FF250	194	194	194.0	0.0	0	30	194	194.0	0.0	0	30
FF500	257	257	257.3	243.5	29799	26	257	257.0	0.6	40	30
FF1000	1260	1260	1260.2	500.8	30782	24	1260	1260	12.2	545	30
FF2000	4545	4545	4546.5	862.9	51211	8	4545	4545.0	66.3	1549	30
WS250	3083	3083	3083.1	1483.0	70537	28	3083	3085.1	1199.2	42897	29
WS500	2072	2072	2088.3	338.7	56525	2	2072	2083.1	403.8	20944	4
WS1000	109807	109712*	126642.7	4859.6	36025	1	119795	131959.1	3701.7	17107	1
WS1500	13098	13103	13287.9	2916.2	150386	1	13098	13153.2	3915.5	48240	11
Bovine	268	268	268.0	0.0	0	30	268	268.0	0.0	0	30
Circuit	2099	2099	2099.0	7.0	1927	30	2099	2099.0	1.1	289	30
Ecoli.txt	806	806	806.0	0.0	0	30	806	806.0	0.0	4	30
USAir97	4336	4336	4665.0	2886.5	105951	20	4336	5060.3	3626.5	109348	6
humanDisea	1115	1115	1115.0	2.9	298	30	1115	1115.0	0.5	50	30
Treni_Roma	918	918	918.0	7.8	21591	30	918	918.0	0.6	222	30
EU_flights	348268	348269	351657.1	295.7	771	1	348268	348434.3	2307.4	5682	28
openflights	26842	26842	28688.7	3284.2	12580	1	26783*	26919.0	4186.2	17090	1
yeast1	1412	1412	1412.4	16.9	60	26	1412	1412.0	26.5	324	30
Ham1000	306349	308198	310580.2	4275.0	39910	1	306349	309912.0	4055.1	30476	3
Ham2000	1243859	1243289*	1256645.8	4692.5	20794	1	1242792*	1251189.7	5223.8	14935	1
Ham3000a	2844393	2842100*	2855766.8	4163.2	9260	1	2840690*	2847291.7	4776.0	7607	1
Ham3000b	2841270	2838531*	2845347.5	4466.6	9347	1	2837584*	2843768.2	4122.4	6310	1
Ham3000c	2838429	2836053*	2846084.9	3523.5	8815	1	2835860*	2839192.3	4203.1	6978	1
Ham3000d	2831311	2827366*	2847582.4	4413.3	10764	1	2829102*	2841551.0	5631.8	8778	1
Ham3000e	2847909	2844721*	2856464.3	4286.6	10228	1	2843000*	2847442.4	4263.2	6881	1
Ham4000	5044357	5051404	5120450.3	4405.7	6449	1	5038611*	5091745.6	6416.5	6690	1
Ham5000	7972525	7968669*	8078656.1	4840.9	4582	1	7969845*	8042058.9	6276.3	5383	1
powergrid	15862	15908	15957.8	2148.8	16566	1	15868	15886.1	5594.1	28573	1
Oclinks	611326	613430	615029.9	896.0	2097	1	611260*	614220.9	2992.5	7709	1
facebook	420334	676712	793272.9	5097.0	13767	1	669910	738856.5	6537.4	8250	1
grqc	13596	13607	13642.5	1221.9	6061	1	13592*	13602.4	4743.4	14404	1
hepht	106397	106814	109092.4	3665.1	4509	1	106792	108673.4	6378.2	4633	1
hepph	6156536	6709598	7541345.1	6999.4	182	1	7211646	7960148.5	6710.5	1709	1
condmat	53963375	7810704	9508083.3	6027.8	7	1	56229708	57421239.1	6364.6	592	1
astroph	2298596	62281904	63073287.1	4383.1	0	1	6057949	6593803.2	6702.4	1199	1

* Improved upper bounds.

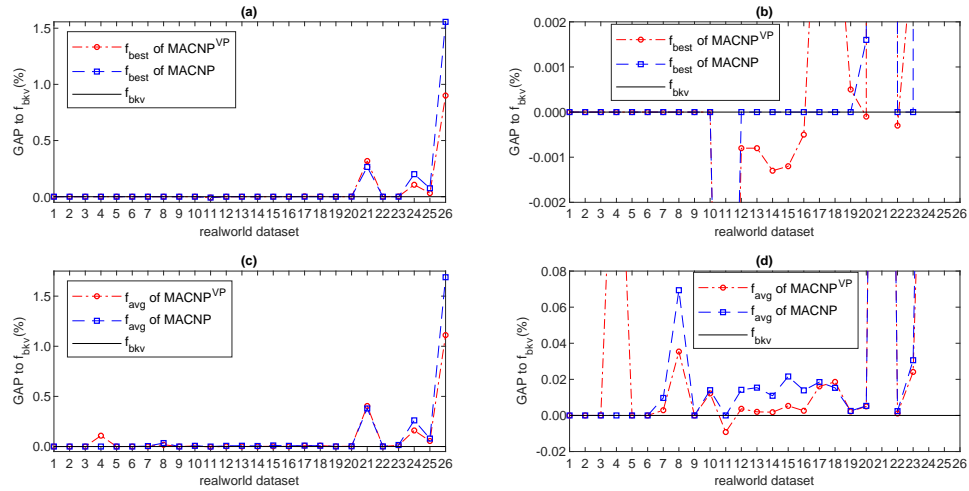


Fig. 2. Comparison between MACNP and MACNP^{VP} under the time limit $t_{max} = 3600$ seconds. Sub-figures (a) and (b) present the *best* results under different ranges of y -axis. Sub-figures (c) and (d) present the *average* results under different ranges of y -axis.

their source codes) on the same computer under the same cutoff time limits. Detailed comparative results between our

algorithms (i.e., $VPMS_{CNP}$ and $MACNP^{VP}$) and the state-of-the-art algorithms with the time limit $t_{max} = 3600$ seconds and a long time limit $t_{max} = 7200$ are provided in Table V and Table VI, respectively. It is worth mentioning that some best known results reported in the literature were achieved with a much larger time budget of 16000 seconds.

Table V shows that under the time limit of 3600 seconds, both $VPMS_{CNP}$ and $MACNP^{VP}$ compete very favorably with the reference algorithms, by attaining 9 new upper bounds and matching 22 best-known bounds. At a significant level of 0.05, $VPMS_{CNP}$ is significantly better than CAN1 (i.e., $35 > CV_{0.05}^{42}$) and FastCNP (i.e., $29.5 > CV_{0.05}^{42}$) in terms of f_{best} . For the f_{avg} indicator, both $VPMS_{CNP}$ and $MACNP^{VP}$ once again significantly outperform CAN1 and FastCNP. Compared to MACNP, $VPMS_{CNP}$ reports 11 better f_{best} values and 23 equal f_{best} values, but the performance difference is statistically marginal (i.e., $22.5 < CV_{0.05}^{42}$). For the f_{avg} indicator, $MACNP^{VP}$ is significantly better than MACNP (i.e., $28.5 > CV_{0.05}^{42}$) (and also better than $VPMS_{CNP}$ ($27 \geq CV_{0.05}^{42}$)).

From Table VI which reports the results under the longer time limit of 7200 seconds, we first observe that all the algorithms improve their results with the extended time limit and this is especially true for our $VPMS_{CNP}$ and $MACNP^{VP}$ algorithms. Indeed, for the 42 instances, $VPMS_{CNP}$ and $MACNP^{VP}$ find 12 new upper bounds and reach 23 best-known results. At a significant level of 0.05, both $VPMS_{CNP}$ and $MACNP^{VP}$ perform significantly better than CNA1 (i.e., $34 > CV_{0.05}^{42}$ and $35 > CV_{0.05}^{42}$) and FastCNP (i.e., $28.5 > CV_{0.05}^{42}$ and $29.5 > CV_{0.05}^{42}$) in terms of the f_{best} indicator. Similar observations hold for the f_{avg} indicator. We also find that $MACNP^{VP}$ performs significantly better than MACNP (i.e., $31.5 > CV_{0.05}^{42}$) in terms of the average results. For the f_{best} indicator, $MACNP^{VP}$ performs marginally better than MACNP (i.e., $24 < CV_{0.05}^{42}$) with 13 better, 22 equal and 7 worse results, respectively. Remarkably, $VPMS_{CNP}$ significantly outperforms MACNP both in terms of f_{best} (i.e., $27.5 > CV_{0.05}^{42}$) and f_{avg} (i.e., $29.5 > CV_{0.05}^{42}$).

Finally, even if we do not show timing information of the compared algorithms, we mention that the reference algorithms typically attain their reported best solutions long before the limit of 7200 seconds and as a result, it is unlikely that additional time budget will benefit them. Therefore, they are not tested with still longer time limits.

V. CONCLUSION AND FUTURE WORK

The proposed variable population memetic search (VPMS) framework uses a strategic population sizing mechanism to dynamically adjust the population size of a memetic algorithm during the evolutionary search. By strategically varying the population size, the memetic algorithm is able to adapt the population diversity during the search and thus favors a continuing balancing between exploitation and exploration. We showcased the effectiveness of the VPMS approach by applying it to solve the challenging critical node problem. Our experiments indicated that the memetic algorithms with the strategic population sizing mechanism compete very favorably

with the state-of-the-art algorithms, and remarkably discover new upper bounds for 12 instances out of the 42 benchmark instances in the literature.

There are several perspectives for future research. First, to further improve the proposed VPMS approach, alternative population sizing schemes can be explored. For this purpose, population control techniques developed for various evolutionary algorithms in the literature including those reviewed in Section II could serve as a natural basis and provide useful ideas. Second, this work focuses on enhancing the canonical memetic search framework for combinatorial optimization. It is worth investigating the proposed strategic population sizing mechanism and alternative schemes within other population-based algorithms, including the broader memetic computation paradigm [16]. Moreover, the key idea of simultaneous problem learning and optimization via knowledge memes promoted by the memetic computation paradigm could be useful to design more powerful memetic algorithms with self-learned components and parameters. Third, the proposed VPMS approach is a general framework. Consequently, it would be interesting to check its effectiveness for solving additional large combinatorial problems.

REFERENCES

- [1] J. Arabas, Z. Michalewicz, and J. Mulawka, "GAVaPS-a genetic algorithm with varying population size," in *Proceedings of the First IEEE Conference on Evolutionary Computation*, IEEE, 1994, pp. 73–78.
- [2] 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*, vol. 55, pp. 128–145, 2016.
- [3] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia, "Local search metaheuristics for the critical node problem," *Networks*, vol. 67, no. 3, pp. 209–221, 2016.
- [4] A. Arulselvan, C. W. Commander, L. Eleftheriadou, and P. M. Pardalos, "Detecting critical nodes in sparse graphs," *Computers & Operations Research*, vol. 36, no. 7, pp. 2193–2200, 2009.
- [5] U. Benlic and J. K. Hao, "A multilevel memetic approach for improving graph k-partitions," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5, pp. 624–472, 2011.
- [6] P. A. N. Bosman, N. H. Luong and D. Thierens, "Expanding from discrete Cartesian to permutation gene-pool optimal mixing evolutionary algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2016, pp. 637–644.
- [7] J. Brest, M. S. Maućec, "Population size reduction for the differential evolution algorithm," *Applied Intelligence*, vol. 29, no. 3, pp. 228–247, 2008.
- [8] J. Brest, A. Zamuda, I. Fister, M. S. Maućec *et al.*, "Self-adaptive differential evolution algorithm with a small and varying population size," in *2012 IEEE Congress on Evolutionary Computation*. IEEE, 2012, pp. 1–8.
- [9] E. K. Burke and Y. Bykov, "The late acceptance hill-climbing heuristic," *European Journal of Operational Research*, vol. 258, no. 1, pp. 70–78, 2017.
- [10] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5, pp. 591–607, 2011.
- [11] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.
- [12] A. E. Eiben, E. Marchiori, and V. Valko, "Evolutionary algorithms with on-the-fly population size adjustment," in *International Conference on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, vol. 3242. Springer, 2004, pp. 41–50.
- [13] L. Feng, Y.-S. Ong, M.-H. Lim, and I. W. Tsang, "Memetic search with interdomain learning: A realization between CVRP and CARP," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 644–658, 2014.

TABLE V
COMPARISONS OF OUR PROPOSED ALGORITHMS WITH STATE-OF-THE-ART ALGORITHMS UNDER $t_{max} = 3600$ SECONDS.

Instance	f_{bkv}	CAN1 [35]		FastCNP [44]		MACNP [45]		MACNP ^{VP}		VPMS _{CNP}	
		f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
BA500	195	195	195.0	195	195.0	195	195.0	195	195.0	195	195.0
BA1000	558	558	558.7	558	558.0	558	558.0	558	558.0	558	558.0
BA2500	3704	3704	3704.0	3704	3710.6	3704	3704.0	3704	3704.0	3704	3704.0
BA5000	10196	10196	10196.0	10196	10201.4	10196	10196.0	10196	10196.0	10196	10196.0
ER235	295	295	295.0	295	295.0	295	295.0	295	295.0	295	295.0
ER466	1524	1524	1524.0	1524	1524.0	1524	1524.0	1524	1524.0	1524	1524.0
ER941	5012	5114	5177.4	5012	5013.3	5012	5014.1	5012	5015.9	5012	5026.5
ER2344	902498	996411	1008876.4	953437	979729.2	902498	922339.5	912205	929024.1	904113	933943.7
FF250	194	194	194.0	194	194.0	194	194.0	194	194.0	194	194.0
FF500	257	263	265.0	257	258.4	257	257.0	257	257.0	257	257.0
FF1000	1260	1262	1264.2	1260	1260.8	1260	1260.0	1260	1260.0	1260	1260.0
FF2000	4545	4548	4549.4	4546	4558.3	4545	4545.7	4545	4545.0	4545	4545.0
WS250	3083	3415	3702.8	3085	3196.4	3083	3089.4	3083	3087.5	3083	3083.1
WS500	2072	2085	2098.7	2072	2083.3	2072	2082.6	2072	2082.1	2072	2083.1
WS1000	109807	141759	161488.0	123602	127493.4	109807	123682.6	123253	135187.8	119444	134475.5
WS1500	13098	13498	13902.5	13158	13255.7	13098	13255.1	13098	13175.7	13098	13161.5
Bovine	268	268	268.0	268	268.0	268	268.0	268	268.0	268	268.0
Circuit	2099	2099	2099.0	2099	2099.0	2099	2099.0	2099	2099.0	2099	2099.0
E.coli	806	806	806.0	806	806.0	806	806.0	806	806.0	806	806.0
USAir97	4336	4336	4336.0	4336	4336.0	4336	4336.0	4336	5275.0	4336	5075.6
HumanDisea	1115	1115	1115.0	1115	1115.0	1115	1115.0	1115	1115.0	1115	1115.0
Treni_Roma	918	918	918.0	918	918.0	918	918.0	918	918.0	918	918.0
EU_flights	348268	348268	348347.0	348268	348697.7	348268	351657.0	348268	349265.6	348268	349265.6
openflights	26842	29300	29815.3	28834	29014.4	26842	28704.3	26842	27792.3	26785*	27320.0
yeast	1412	1413	1416.3	1412	1412.0	1412	1412.0	1412	1412.0	1412	1412.0
H1000	306349	314152	317805.7	314964	316814.8	306349	310626.5	306353	310081.3	307117	311169.4
H2000	1243859	1275968	1292400.4	1275204	1285629.1	1243859	1263495.6	1242999*	1251826.9	1247652	1256573.8
H3000a	2844393	2911369	2927312.0	2885588	2906965.5	2844393	2884781.7	2842072*	2855005.3	2840941*	2859284.4
H3000b	2841270	2907643	2927330.5	2876585	2902893.9	2841270	2885087.0	2839018*	2847010.7	2839893*	2860810.9
H3000c	2838429	2885836	2917685.8	2876026	2898879.3	2838429	2869348.5	2834802*	2843661.7	2832073*	2844324.3
H3000d	2831311	2906121	2929569.2	2894492	2907485.4	2831311	2892562.7	2827859*	2846261.0	2830291*	2857201.4
H3000e	2847909	2903845	2931806.8	2890861	2911409.3	2847909	2887525.7	2846412*	2855333.6	2846731*	2867000.6
H4000	5044357	5194592	5233954.5	5167043	5190883.7	5044357	5137528.3	5077298	5125589.3	5082521	5141804.3
H5000	7972525	8142430	8212165.9	8080473	8132896.2	7972525	8094812.6	8012229	8120955.9	8011565	8151850.1
powergr	15862	16158	16222.1	15982	16033.5	15862	15901.5	15870	15897.1	15873	15909.2
Oclinks	611326	611326	614858.5	611344	616783.0	612303	614544.0	611280*	614364.0	611254*	614296.3
faceboo	420334	701073	742688.0	692799	765609.8	643162	739436.6	687604	760335.1	691232	780429.1
grqc	13596	15522	15715.7	13616	13634.8	13596	13629.2	13592*	13611.4	13603	13615.5
hepht	106397	130256	188753.7	108217	109889.5	106397	109655.6	106778	108961.1	107939	110158.4
hepht	6156536	9771610	10377853.2	6392653	7055773.8	8628687	9370215.3	7465746	8128758.7	7883063	8689170.1
astroph	53963375	59029312	60313225.8	55424575	57231348.7	62068966	62547898.1	57411990	59897908.4	58322396	59563941.1
condmat	2298596	13420836	14823254.9	4086629	5806623.8	9454361	10061807.8	6438018	7407961.4	6843993	7813436.7

* Improved upper bounds.

- [14] B. W. Goldman and W. F. Funch, "Fast and efficient black box optimization using the parameter-less population pyramid," *Evolutionary Computation*, vol. 23, no. 3, pp. 451–479, 2015.
- [15] Y. Guan, L. Yang, and W. Sheng, "Population control in evolutionary algorithms: review and comparison," *Bio-inspired Computing: Theories and Applications*, pp. 161–174, 2017.
- [16] A. Gupta and Y. S. Ong, *Memetic computation: the mainspring of knowledge transfer in a data-driven optimization era*. Springer International Publishing, 2019.
- [17] J. K. Hao, "Memetic algorithms in discrete optimization," In F. Neri, C. Cotta, P. Moscato (Eds.) *Handbook of Memetic Algorithms. Studies in Computational Intelligence*, vol. 379, Chapter 6, pp 73–94, Springer, 2012.
- [18] G. R. Harik and F. G. Lobo, "A parameter-less genetic algorithm," In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999, pp. 258–265.
- [19] W. E. Hart, N. Krasnogor, and J. E. Smith, "Recent advances in memetic algorithms," *Studies in Fuzziness and Soft Computing*, vol. 166, Springer, 2005.
- [20] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. Elsevier, 2004.
- [21] Y. Jin, and J. K. Hao, "Solving the latin square completion problem by memetic graph coloring," *IEEE Transactions on Evolutionary Computation*, vol. 33, no. 6, pp. 1015–1028, 2019.
- [22] D. Karapetyan and G. Gutin, "A new approach to population sizing for memetic algorithms: a case study for the multidimensional assignment problem," *Evolutionary Computation*, vol. 19, no. 3, pp. 345–371, 2011.
- [23] M. M. Komarnicki and M. W. Przewozniczek, "Parameter-less, population-sizing DSMGA-II" In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019, pp. 289–290.
- [24] V. K. Koumoussis and C. P. Katsaras, "A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 1, pp. 19–28, 2006.
- [25] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: model, taxonomy, and design issues," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 5, pp. 474–488, 2005.
- [26] M. Lalou, M. A. Tahraoui, and H. Kheddouci, "The critical node detection problem in networks: A survey," *Computer Science Review*, vol. 28, pp. 92–117, 2018.
- [27] L. Moalic and A. Gondran, "The sum coloring problem: A memetic algorithm based on two individuals," in *2019 IEEE Congress on Evolutionary Computation*, IEEE, 2019, pp. 1798–1805.
- [28] P. Moscato and C. Cotta, "A gentle introduction to memetic algorithms," In *Handbook of Metaheuristics*, Kluwer, Norwell, Massachusetts, USA, pp. 105–144, 2003.
- [29] M. Namazi, C. Sanderson, M. A. H. Newton, M. M. A. Polash, and A. Sattar, "Diversified late acceptance search," in *Proceedings of the 31st Australasian Joint Conference on Artificial Intelligence*, Wellington, New Zealand, 2018, pp. 299–311.
- [30] F. Neri and C. Cotta, "Memetic algorithms and memetic computing optimization: A literature review," *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, 2012.
- [31] F. Neri, C. Cotta, and P. Moscato (Eds.), "Handbook of Memetic

TABLE VI
COMPARISONS OF OUR PROPOSED ALGORITHMS WITH STATE-OF-THE-ART ALGORITHMS UNDER $t_{max} = 7200$ SECONDS.

Instance	f_{bkv}	CNA1 [◊]		FastCNP [◊]		MACNP [◊]		MACNP ^{VP}		VPMS _{CNP}	
		f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
BA500	195	195	195.0	195	195.0	195	195.0	195	195.0	195	195.0
BA1000	558	558	558.0	558	558.0	558	558.1	558	558.0	558	558.0
BA2500	3704	3704	3704.0	3704	3704.0	3704	3704.0	3704	3704.0	3704	3704.0
BA5000	10196	10196	10196.0	10196	10202.9	10196	10196.0	10196	10196.0	10196	10196.0
ER235	295	295	295.0	295	295.0	295	295.0	295	295.0	295	295.0
ER466	1524	1524	1524.0	1524	1524.0	1524	1524.0	1524	1524.0	1524	1524.0
ER941	5012	5066	5132.7	5012	5014.2	5012	5016.1	5012	5015.9	5012	5026.5
ER2344	902498	984677	1002569.7	963785	979021.9	911274	929358.3	912205	929024.1	904113	933943.7
FF250	194	194	194.0	194	194.0	194	194.0	194	194.0	194	194.0
FF500	257	262	265.4	257	258.3	257	257.3	257	257.0	257	257.0
FF1000	1260	1261	1262.2	1260	1261.3	1260	1262.6	1260	1260.0	1260	1260.0
FF2000	4545	4548	4548.0	4545	4557.0	4545	4547.6	4545	4545.0	4545	4545.0
WS250	3083	3361	3678.8	3094	3198.9	3083	3083.2	3083	3087.5	3083	3083.1
WS500	2072	2085	2092.3	2078	2085.8	2072	2086.1	2072	2082.1	2072	2083.1
WS1000	109807	138343	156129.6	113656	121002.5	110342	125548.4	123253	135187.8	119444	134475.5
WS1500	13098	13557	13802.2	13143	13247.5	13150	13339.1	13098	13175.7	13098	13161.5
Bovine	268	268	268.0	268	268.0	268	268.0	268	268.0	268	268.0
Circuit	2099	2099	2099.0	2099	2099.0	2099	2099.0	2099	2099.0	2099	2099.0
Ecoli.txt	806	806	806.0	806	806.0	806	806.0	806	806.0	806	806.0
USAir97	4336	4336	4336.0	4336	4336.0	4336	4343.1	4336	5151.5	4336	5060.3
humanDisea	1115	1115	1115.0	1115	1115.0	1115	1115.0	1115	1115.0	1115	1115.0
Treni_Roma	918	918	918.0	918	918.0	918	918.0	918	918.0	918	918.0
EU_flights	348268	348268	348419.6	348268	350051.2	348268	351573.9	348268	349016.2	348268	348434.3
openflights	26842	29266	29679.2	26896	28820.1	26842	28724.9	26842	27821.1	26783*	26919.0*
yeast1	1412	1414	1415.1	1412	1412.0	1412	1412.6	1412	1412.0	1412	1412.0
Ham1000	306349	315267	317389.6	313967	318063.3	306353	310254.2	306349	310348.5	306349	309912.0
Ham2000	1243859	1280776	1291835.7	1272769	1284846.2	1243810*	1255525.9	1242739*	1249217.5	1242792*	1251189.7
Ham3000a	2844393	2889116	2922475.9	2889026	2907454.3	2841893*	2851070.2	2841487*	2845235.8	2840690*	2847291.7
Ham3000b	2841270	2900645	2920695.8	2888279	2901745.0	2839435*	2845280.4	2839098*	2841822.5	2837584*	2843768.2
Ham3000c	2838429	2885201	2917668.6	2881202	2898648.5	2836103*	2841923.0	2835369*	2837858	2835860*	2839192.3
Ham3000d	2831311	2894637	2924121.0	2879509	2903890.5	2829328*	2839602.4	2828492*	2834729.6	2829102*	2841551.0
Ham3000e	2847909	2905662	2929507.7	2890137	2910922.6	2844979*	2858484.1	2845437*	2850598.1	2843000*	2847442.4
Ham4000	5044357	5169509	5226214.6	5144613	5186840.6	5042395*	5105351.2	5045783	5089596.9	5038611*	5091745.6
Ham5000	7972525	8158935	8212347.4	8080428	8117117.3	7964765*	8060826.0	7969299*	8039418.4	7969845*	8042058.9
powergrid	15862	16103	16166.0	15985	16024.3	15897	15943.7	15865	15882.7	15868	15886.1
Oclinks	611326	611285	615343.2	614469	616631.3	612328	614732.8	611253*	613861.5	611260*	614220.9
facebook	420334	662680	746744.5	676009	766879.0	680936	783374.6	630564	732633.6	669910	738856.5
grqc	13596	15488	15630.5	13614	13634.0	13601	13644.0	13591*	13598.4	13592*	13602.4
hepht	106397	134863	164474.5	106362*	108138.5	106926	108238.5	106276*	108079.9	106792	108673.4
hepht	6156536	9657653	10051432.7	6299554	7108586.7	6155877*	6991782.6	7087968	7724431.6	7211646	7960148.5
astroph	53963375	57054795	58119058.3	56625063	57713404.8	58941340	60665177.4	55800209	56920216.6	56229708	57421239.1
condmat	2298596	12862556	13834807.8	3754050	4225322.2	5205685	6580912.8	5393192	6403204.6	6057949	6593803.2

◊ The results were obtained by re-running CAN1 [35], FastCNP [44] and MACNP [45] with $t_{max} = 7200$ seconds.

* Improved upper bounds.

- Algorithms,” *Studies in Computational Intelligence*, vol. 379, Springer, 2012.
- [32] G. Pavai and T. V. Geetha, “A survey on crossover operators,” *ACM Computing Surveys*, vol. 49, no. 4, pp. 72:1–72:43, 2016.
- [33] A. P. Piotrowski, “Review of differential evolution population size,” *Swarm and Evolutionary Computation*, vol. 32, pp. 1–24, 2017.
- [34] D. C. Porumbel, J. K. Hao, and P. Kuntz, “An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring,” *Computers & Operations Research*, vol. 37, no. 10, pp. 1822–1832, 2010.
- [35] W. Pullan, “Heuristic identification of critical nodes in sparse real-world graphs,” *Journal of Heuristics*, vol. 21, no. 5, pp. 577–598, 2015.
- [36] K. Sörensen and M. Sevaux, “MA | PM: memetic algorithms with population management,” *Computers & Operations Research*, vol. 33, no. 5, pp. 1214–1225, 2006.
- [37] M. D. Summa, A. Grosso, and M. Locatelli, “Branch and cut algorithms for detecting critical nodes in undirected graphs,” *Computational Optimization and Applications*, vol. 53, no. 3, pp. 649–680, 2012.
- [38] K. Tang, Y. Mei, and X. Yao, “Memetic algorithm with extended neighborhood search for capacitated arc routing problems,” *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1151–1166, 2009.
- [39] V. Tirronen and F. Neri, “Differential evolution with fitness diversity self-adaptation,” in *Nature-Inspired Algorithms for Optimisation*. Springer, 2009, pp. 199–234.
- [40] M. Ventresca, “Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem,” *Computers & Operations Research*, vol. 39, no. 11, pp. 2763–2775, 2012.
- [41] S. Wang, J. Liu, and Y. Jin, “Finding influential nodes in multiplex networks using a memetic algorithm,” *IEEE Transactions on Cybernetics*, DOI: 10.1109/TCYB.2019.2917059, 2019.
- [42] T. Weise, Y. Wu, R. Chiong, K. Tang, and J. Lässig, “Global versus local search: the impact of population sizes on evolutionary algorithm performance,” *Journal of Global Optimization*, vol. 66, no. 3, pp. 511–534, 2016.
- [43] Y. Zhou, J. K. Hao, and B. Duval, “Opposition-based memetic search for the maximum diversity problem,” *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 5, pp. 731–745, 2017.
- [44] Y. Zhou and J. K. Hao, “A fast heuristic algorithm for the critical node problem,” in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ACM, 2017, pp. 121–122.
- [45] Y. Zhou, J. K. Hao, and F. Glover, “Memetic search for identifying critical nodes in sparse graphs,” *IEEE Transactions on Cybernetics*, vol. 49, no. 10, pp. 3699–3712, Oct 2019.

APPENDIX

This appendix reports three additional experiments. To ensure the fairness of the experiments, we ran the compared algorithms on our computer with the same cutoff time limits.

a) Results under an extended time limit of 10800 seconds: We verify whether VPMS_{CNP} has a lasting search capacity with a still larger time limit of $t_{max} = 10800$ seconds. Since this experiment is very time consuming, we run VPMS_{CNP} to solve each of the 42 benchmark instance 15 times (instead of 30 runs as before) and report the results in Table VII. We observe that the algorithm attains new improved upper bounds on 11 instances and matches the best-known bounds on 24 instances. Importantly, it achieves more than 18 best-known bounds with a success rate of 100% and improves its results on the largest instances. Also, the algorithm improves the f_{avg} values on almost all instances. Finally, we observe some variations of the best results compared to those of 7200 seconds. This can be explained by the fact that only 15 runs (instead of 30 runs) were performed. We observe that CNA1, FastCNP and MACNP typically converged to their best possible solutions before reaching 7200 seconds, and did not run them with a longer cutoff time.

b) VPMS_{CNP} against FPMS_{CNP} with different population sizes: In Section IV-C3, we compared VPMS_{CNP} with the fixed population variant FPMS_{CNP} where the population size is fixed to $p_{max} = 20$. We now extend the comparison by testing two other sizes: namely $0.8 * p_{max}$ (smaller than p_{max}) and $1.2 * p_{max}$ (larger than p_{max}). For space reasons, we show the results of this experiment on 10 representative instances in Table VIII where the results of VPMS_{CNP} and FPMS_{CNP} (denoted by FPMS_{CNP} ^{p_{max}}) are also included. The results clearly show a clear dominance of VPMS_{CNP} over all the FPMS variants with different fixed populations in terms of the best and average values, demonstrating the effectiveness of the strategic population sizing mechanism.

c) Other variable population sizing methods: To further show the interest of our strategic population sizing (SPS) mechanism, we compare it with two alternative variable population size methods, namely population size reduction (PSR) and population size expansion (PSE). The PSR method was initially proposed for the differential evolution algorithm [7], which gradually decreases the greatest size (in our case 20) at the beginning of the evolutionary process to the smallest size (in our case 2) at the end of the evolution. On the contrary, PSE starts from a small population of only two individuals and then continuously enlarges the population size during the evolutionary search. For this experiment, we create two VPMS variants by replacing our SPS method with PSE and PSR (i.e., VPMS^{PSE} and VPMS^{PSR}). The comparative results on the 10 selected instances above are summarized in Table IX. From the table, we observe that the VPMS algorithm with our strategic population sizing mechanism performs the best in terms of the f_{best} and f_{avg} indicators, confirming the benefit of the proposed SPS mechanism.

TABLE VII
COMPUTATIONAL RESULTS OF VPMS_{CNP} UNDER A LONG TIME LIMIT
 $t_{max} = 10800$ SECONDS.

Instance	VPMS _{CNP}					
	f_{bkv}	f_{best}	f_{avg}	t_{avg}	#gens	#succ
BA500	195*	195	195.0	0.0	0	15
BA1000	558*	558	558.0	3.5	0	15
BA2500	3704*	3704	3704.0	4.4	113	15
BA5000	10196*	10196	10196.0	8.8	61	15
ER235	295*	295	295.0	2.4	667	15
ER466	1524	1524	1524.0	21.1	3380	15
ER941	5012	5012	5015.3	2791.9	197790	2
ER2344	902498	908061	934925.3	7698.1	33644	1
FF250	194*	194	194.0	0.0	1	15
FF500	257*	257	257.0	0.5	52	15
FF1000	1260*	1260	1260.0	2.4	164	15
FF2000	4545*	4545	4545.0	60.4	1912	15
WS250	3083	3083	3083.0	965.8	60783	15
WS500	2072	2072	2083.0	681.5	55420	2
WS1000	109807	113131	131288.9	4642.6	17811	1
WS1500	13098	13098	13154.7	4700.2	96293	7
Bovine	268	268	268.0	0.0	0	15
Circuit	2099	2099	2099.0	0.6	144	15
Ecoli.txt	806	806	806.0	0.0	0	15
USAir97	4336	4336	4586.5	5106.0	239133	10
humanDisea	1115	1115	1115.0	0.1	0	15
Treni_Roma	918	918	918.0	1.8	768	15
EU_flights	348268	348268	348268.0	3251.1	7564	15
openflights	26842	26785*	26814.2	6468.7	32415	9
yeast1	1412	1412	1412.0	25.4	537	15
Ham1000	306349	306349	309925.4	5054.1	40359	1
Ham2000	1243859	1243316*	1247510.0	6504.7	19250	1
Ham3000a	2844393	2842197*	2845559.9	6792.2	7443	1
Ham3000b	2841270	2839360*	2843720.8	6996.2	7821	1
Ham3000c	2838429	2835969*	2838649.3	5179.0	8345	1
Ham3000d	2831311	2829536*	2833408.6	6719.1	10779	1
Ham3000e	2847909	2843777*	2848707.0	5924.0	8641	1
Ham4000	5044357	5045719	5094052.9	9242.3	6573	1
Ham5000	7972525	7970553*	8029114.2	9405.9	6196	1
powergrid	15862	15862	15878.2	8413.7	45019	1
Oclinks	611326	611264*	613968.2	6749.6	9246	1
facebook	420334	642625	731533.0	10097.1	10476	1
grqc	13596	13591*	13598.9	7080.5	27692	1
hepth	106397	106362*	108017.3	9933.1	6312	1
hepph	6156536	7189023	7444555.0	10675.8	2030	1
astroph	53963375	55280459	55916119.2	10758.9	978	1
condmat	2298596	5521284	5900013.2	10273.1	1949	1

* Optimal solutions obtained by a branch-and-cut algorithm [37] within 5 days.

* Improved best upper bounds.

TABLE VIII
COMPARISON OF VPMS_{CNP} (WITH A VARIABLE POPULATION) AGAINST FPMS_{CNP} (WITH A FIXED POPULATION SIZE OF $0.8 * p_{max}$, $1.0 * p_{max}$, AND $1.2 * p_{max}$) UNDER $t_{max} = 3600$ SECONDS.

Instance	f_{bkv}	VPMS _{CNP}		FPMS _{CNP} ^{0.8*p_{max}}		FPMS _{CNP} ^{p_{max}}		FPMS _{CNP} ^{1.2*p_{max}}	
		f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
BA5000	10196	10196	10196.0	10196	10196.0	10196	10196.0	10196	10196.0
ER235	295	295	295.0	295	295.0	295	295.0	295	295.1
FF1000	1260	1260	1260.0	1260	1263.4	1260	1262.3	1260	1261.4
WS1500	13098	13098	13161.5	13145	13421.6	13146	13329.1	13098	13320.7
EU_flights	348268	348268	349265.6	349100	352467.3	348268	351323.0	348268	350848.6
openflights	26842	26785*	27327.0	26874	28794.9	26842	28845.3	26785*	28577.3
Ham3000a	2844393	2840941*	2859284.4	2843215*	2866172.0	2841106*	2861888.3	2843025*	2859236.9
Ham3000c	2838429	2832073*	2844324.3	2831739*	2853766.5	2836076*	2848545.9	2836163*	2845344.7
powergrid	15862	15873	15909.2	15922	15977.6	15899	15954.5	15895	15952.9
grqc	13596	13603	13615.5	13615	13656.7	13612	13647.2	13608	13636.4

* Improved upper bounds.

TABLE IX
COMPARISON OF VPMS_{CNP} AGAINST VPMS_{CNP} VARIANTS INTEGRATING OTHER VARIABLE POPULATION SIZE METHODS (I.E., PSE AND PSR) UNDER $t_{max} = 3600$ SECONDS.

Instance	f_{bkv}	VPMS _{CNP}		VPMS _{CNP} ^{PSE}		VPMS _{CNP} ^{PSR} [7]	
		f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
BA5000	10196	10196	10196.0	10196	10196.0	10196	10196.0
ER235	295	295	295.0	295	295.0	295	295.0
FF1000	1260	1260	1260.0	1260	1260.1	1260	1261.5
WS1500	13098	13098	13161.5	13201	13436.6	13163	13341.5
EU_flights	348268	348268	349265.6	350762	352522.8	349100	352271.9
openflights	26842	26785*	27327.0	26875	28829.4	26874	28607.5
Ham3000a	2844393	2840941*	2859284.4	2847053	2887510.2	2843404	2876168.9
Ham3000c	2838429	2832073*	2844324.3	2834326	2871369.3	2835775	2862621.5
powergrid	15862	15873	15909.2	15917	15999.6	15934	15974.0
grqc	13596	13603	13615.5	13616	13664.2	13607	13650.1

* Improved upper bounds.