Chen ZH, Cai SW, Gao J *et al*. Heuristic search with cut point based strategy for critical node problem. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 39(6): 1328-1340 Nov. 2024. DOI: 10.1007/s11390-024-2850-0

Heuristic Search with Cut Point Based Strategy for Critical Node Problem

Zhi-Han Chen[1, 2] (陈志翰), Shao-Wei Cai[1, 2, *] (蔡少伟), *Senior Member, CCF*, Jian Gao[3] (高　健)
Shi-Ke Ge[4] (葛士可), Chan-Juan Liu[4] (刘婵娟), *Member, IEEE*, and Jin-Kun Lin[5] (林锦坤)

[1] *School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 101408, China*

[2] *Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China*

[3] *College of Information Science and Technology, Northeast Normal University, Changchun 130024, China*

[4] *School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China*

[5] *SeedMath Technology Limited, Beijing 100086, China*

E-mail: chenzh@ios.ac.cn; caisw@ios.ac.cn; gaojian@nenu.edu.cn; shike.ge@mail.dlut.edu.cn; chanjuanliu@dlut.edu.cn linjk@seedmaas.com

Received September 21, 2022; accepted July 3, 2024.

Abstract The critical node problem (CNP) aims to deal with critical node identification in a graph, which has extensive applications in many fields. Solving CNP is a challenging task due to its computational complexity, and it attracts much attention from both academia and industry. In this paper, we propose a population-based heuristic search algorithm called CPHS (Cut Point Based Heuristic Search) to solve CNP, which integrates two main ideas. The first one is a cut point based greedy strategy in the local search, and the second one involves the functions used to update the solution pool of the algorithm. Besides, a mutation strategy is applied to solutions with probability based on the overall average similarity to maintain the diversity of the solution pool. Experiments are performed on a synthetic benchmark, a real-world benchmark, and a large-scale network benchmark to evaluate our algorithm. Compared with state-of-the-art algorithms, our algorithm has better performance in terms of both solution quality and run time on all the three benchmarks.

Keywords local search, cut point, heuristic search, critical node problem

1 Introduction

Identifying critical nodes is an essential issue in complex network analysis, and the critical node problem (CNP) plays an important role in many application fields, such as network security[1], biological interaction networks[2, 3], smart grid[4], pandemic prevention[5], and social network analysis[6, 7]. The task of CNP is to find a subset of nodes such that a predefined connectivity measure of the remaining graph is minimized.

1.1 Previous Work

CNP has proven to be NP-hard in the general case, though some special cases can be solved in polynomial time such as the tree-structured graphs[8, 9]. The research direction of developing algorithms for solving CNP has drawn much attention from the AI community due to its significant importance in practice[10].

There are two major classes of practical approaches for solving CNP: exact algorithms and heuristic algorithms. Exact algorithms are mostly based on the integer programming model, and usually solve a CNP instance through adopting the branch-and-cut framework. However, transforming CNP instances into integer programming models suffers from introducing exponential number of constraints, and exact algo-

Regular Paper

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant Nos. XDA0320000 and XDA0320300, and the National Natural Science Foundation of China under Grant No. 61972063.

*Corresponding Author

rithms become ineffective when solving large CNP instances. For example, Di Summa *et al.*[11] proposed an integer linear programming model to formulate CNP: for the formulated model, the number of constraints is not polynomial with regard to the size of the instance. Also, the computational experiments conducted in [11] were performed on only small and random instances. Recently, Pavlikov[12] has provided some improvements to mixed-integer linear programming formulations discussed in the literature[13], which can reduce the number of constraints. Nevertheless, the number of constraints is still large, which results in the moderate performance in practice. Walteros *et al.*[14] also formulated CNP with a mixed-integer linear programming model, and proposed novel valid inequalities and preprocessing techniques to speed up the search process of the branch-and-cut algorithm. Their algorithm was tested on both real-world and random instances, showing an advantage over previous work. These exact approaches can prove the optimal solution, but they usually fail to solve large graphs.

An alternative way is heuristic search. Some early researches include variable neighborhood search[15], approximation algorithms[16], and global search algorithms[17]. Recently, local search based heuristic CNP algorithms have witnessed remarkable progress, and an obvious tendency for CNP heuristic algorithms is to solve larger instances. In this research direction, Ventresca and Aleman[18] proposed a depth-first search greedy algorithm whose time complexity is linear to the graph size. Six real-world graphs with up to 20 000 nodes were used to test their algorithm. Aringhieri *et al.*[15] developed a variable neighborhood search algorithm for CNP, and further improved the algorithm with efficient neighborhoods, leading to improved best-known results on some randomly generated graphs with up to 5 000 nodes[19]. With the purpose of dealing with sparse real-world graphs, Pullan[20] proposed a multi-start greedy algorithm CNA1 which showed better results than previous heuristic algorithms on solving graphs containing up to about 10 000 nodes and 25 000 edges. Zhou *et al.*[21] proposed a memetic algorithm named MACNP, which combines several search strategies, including a double backbone-based crossover operator, a component-based neighborhood search procedure, and a rank-based pool updating strategy. They tested their algorithm on real-world graphs with more than 20 000 nodes, showing better performance than previous algorithms including those from [19, 20, 22, 23], and reported new upper bounds for some instances. Later, MACNP was enhanced by a sizing mechanism which dynamically adjusts the population size during the search[24], leading to the VPMS algorithm, and improved upper bounds for some instances were discovered. Seen from the literature[24], MACNP and VPMS represent the latest state-of-the-art in solving CNP. For more details on CNP algorithms, we refer to a survey paper[10].

1.2 Contributions

In this paper, we propose an efficient heuristic search algorithm called CPHS (Cut Point Based Heuristic Search) to solve CNP. Our algorithm is a population-based heuristic search algorithm, and has two main ideas.

Cut Point Based Node Selection Strategy. Local search is an important component of the memetic algorithm, and the node selection strategy directly affects the performance of the local search. Previous algorithms[20, 21, 24] for CNP usually select nodes according to the information of the nodes such as age and degree, and ignore the structural information of the graph. We propose a cut point based node selection strategy and prove that it can minimize the size of a certain connected component so as to make the best movement to minimize the objective value.

Dynamic Pool Updating Strategy. Our algorithm maintains a solution pool. It is desirable that the pool contains high-quality and diverse solutions. Previous heuristic algorithms for CNP usually use static scoring functions to decide which nodes should be removed or added to the pool. However, such static functions cannot adapt well according to the algorithmic behavior. For example, when the solutions in the pool have similar structures, then we should increase the diversity of solutions in the pool, to avoid being trapped in a small search area. Based on this consideration, we propose a metric called overall average similarity and propose a dynamic scoring function based on this metric, to make the pool more robust.

We carry out experiments to evaluate CPHS on the benchmarks in the literature as well as a group of large graphs which are popular for testing algorithms for solving large-scale combinatorial optimization problems. We compare our algorithm with state-of-the-art heuristic algorithms CNA1[20], FastCNP[25], MACNP[21], and VPMS[24], and the strong results indicate that CPHS has much better performance on both traditional benchmarks and a large-scale network benchmark. Further analyses confirm the effectiveness of the important ideas in our algorithm.

1.3 Paper Organization

The remainder of this paper is structured as follows. Section 2 introduces preliminary knowledge. Section 3 presents the CPHS algorithm on a top level, while Section 4 and Section 5 introduce the key functions. Section 6 describes the improvements compared with previous algorithms. Experimental studies are presented in Section 7. Finally, we give some concluding remarks in Section 8.

2 Preliminary

2.1 Notions and Notation

For convenience, we provide a brief introduction of notions and notations about the graph theory used in this paper.

An undirected graph $G=(V,\ E)$ consists of a set of nodes V and a set of edges $E\subseteq V\times V$. The notations $V(G)$ and $E(G)$ denote the node set and the edge set of graph G, respectively. $N(v)=\{u\in V|(u,\ v)\in E\}$ is the set of neighbors of node v, and $d(v)=|N(v)|$ is the degree of node v. Given that a subset of node set $S\subset V$, the induced subgraph $G[S]$ is the graph whose node set $V(G[S])=S$, and edge set $E(G[S])=\{(u,\ v)\in E(G)|u,v\in S\}$.

A pair of nodes are connected if there is an edge path from one to the other. A graph can be divided into several connected components $\{C_1,\ C_2,\ \ldots,\ C_L\}$, in which each pair of nodes in the same connected component are connected and each pair of nodes in the different connected components are not connected. In a connected component C, a node $v\in C$ is a cut point iff removing it (and its incident edges) disconnects the graph. For example, the graph shown in Fig.1 has three cut points 3, 4, and 5.

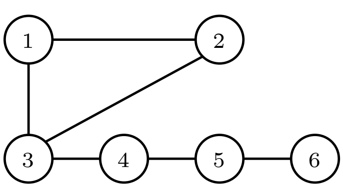

Fig.1. Connected component with three cut points.

Given an unconnected graph, which is divided into connected components (disjoint connect subgraphs) $\{C_1,\ C_2,\ \ldots,\ C_L\}$, without loss of generality, let us assume $|C_1|\leqslant|C_2|\leqslant\ldots\leqslant|C_L|$, and then we call a connected component C_i a large component iff $|C_i|\geqslant(|C_1|+|C_L|)/2$.

2.2 Problem Description

Given a graph $G=(V,\ E)$ and an integer K, the critical node problem (CNP) aims to extract a subset of nodes $S\subset V$, where $|S|\leqslant K$, to minimize the total number of connected pairs in the residual graph $G[V\backslash S]$. The residual graph is divided into several connected components $\{C_1,\ C_2,\ \ldots,\ C_L\}$, and the objective function $f(S)$ of the CNP is defined as

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

where L is the total number of connected components of the residual graph $G[V\backslash S]$.

Considering the easy graph in Fig.2(a) with $K=2$, Fig.2(b) shows a feasible solution $S=\{2,\ 4\}$ and its cost is 3, while Fig.2(c) shows the optimal solution $S=\{3,\ 4\}$ and its cost is 2.

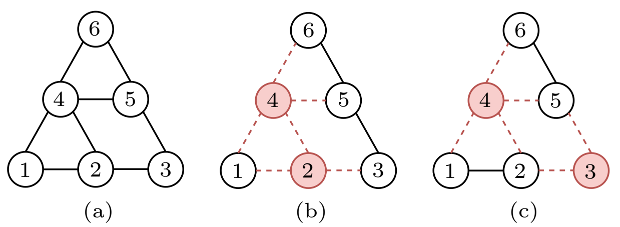

Fig.2. (a) Easy graph with 6 nodes and $K=2$. (b) Feasible solution ($S=\{2,\ 4\}$), whose objective function $f(S)=\binom{3}{2}=3$. (c) Optimal solution ($S=\{3,\ 4\}$), whose objective function $f(S)=\binom{2}{2}+\binom{2}{2}=2$.

3 Framework of Algorithm CPHS

In this section, we introduce the main procedure of our proposed CPHS algorithm (as shown in Algorithm 1), and we will leave the two important sub-algorithms ImproveLS and UpdatePool for Section 4 and Section 5, respectively.

The algorithm maintains a solution pool $\mathcal{P}$. The solutions in $\mathcal{P}$ will be updated during the search. In the beginning, the algorithm generates some initial solutions as the solution pool (line 1), and the best-found solution S^* is initialized as the best initial solution (line 2). After that, the algorithm executes the main loop (lines 3–8) until reaching the preset cutoff time. In each iteration, a new solution is generated by a cross operation, which absorbs the advantages of two random solutions from $\mathcal{P}$ (lines 4 and 5). The resulting solution S is improved by a local search procedure (the *ImproveLS* function), leading to an improved (possibly the same) solution S' (line 6). At the end of each iteration, if S' is better than S^*, then S^* is updated as S' (line 7). Also, a procedure is used to decide whether S' should be added to the solution pool $\mathcal{P}$ (line 8). Finally, when reaching the time limit, CPHS returns the best-found solution S^* (line 9). Below we explain the *Initialization* and *Cross* functions of CPHS.

Algorithm 1. Pseudocode of CPHS

Input: graph $G = (V, E)$, the $cutoff$ time, an integer K
Output: the best solution S^* found

1 $\mathcal{P} \leftarrow Initialization()$;
2 $S^* \leftarrow \text{argmin}_{S^i \in P} f(S^i)$;
3 **while:** $elapsedtime < cutoff$ **do**
4   randomly select two solutions S^i, S^j from $\mathcal{P}$
5     $S \leftarrow Cross(S^i, S^j)$;
6     $S' \leftarrow ImproveLS(S)$;
7   **if** $(f(S') < f(S^*))$ **then** $S^* \leftarrow S'$;
8     $UpdatePool(S')$;
9 **return** S^*;

3.1 Initialization

To construct an initial solution, we first pick K nodes randomly as a solution and then improve it by the *ImproveLS* procedure. This is repeated $Poolsize$ times so as to form a solution pool, where $Poolsize$ is a parameter representing the size of the solution pool.

3.2 Cross

The *Cross* procedure selects common nodes from two solutions in the solution pool, and adds some other nodes to construct a new solution randomly.

To be specific, let S^i and S^j be the two solutions randomly chosen from $\mathcal{P}$, and let us denote the newly generated solution as S. Firstly, we put the common nodes of S^i and S^j to S, i.e., $S = S^i \bigcap S^j$. Then, for each node v appearing in either S^i or S^j but not both (formally, $v \in (S^i \bigcup S^j) \backslash (S^i \bigcap S^j)$), it is added to S with a probability p_0, where p_0 is an algorithmic parameter. For this resulting node set S, we have three different cases.

- $|S| = K$. In this case, we directly pass it to the *ImproveLS* procedure for further improvements.
- $|S| > K$. This means S is not a solution, and we remove $|S| - K$ nodes from S with a greedy manner w.r.t. the objective function f, making $|S| = K$, and thus S becomes a valid solution.
- $|S| < K$. $f(S)$ can be reduced by adding more nodes to S until $|S| = K$. As we will prove in Section 4, adding cut points to the solution is usually a good choice for this aim, although not necessarily optimal. Thus, after picking a random large component, CPHS iteratively adds a random cut point to S (and removes it from the component) until there is no cut point in the component or $|S| = K$. If the size of S is still smaller than K after adding all the cut points in the component, then some more random nodes from the component are added to S until $|S| = K$, which further brings down the value of $f(S)$.

4 Local Search for Improving Solution

A main idea of this work lies in the *ImproveLS* procedure, which is the critical function of the CPHS algorithm. This section presents the details of this sub-algorithm, and provides some theoretical insights of our cut point based strategy.

The *ImproveLS* procedure (depicted in Algorithm 2) aims to improve an input solution S by iteratively adding a new node to S and removing a node from S, it terminates when continuous $MaxIter$ rounds have no improvement (line 2), and the best-improved solution S' during this procedure is returned by the *ImproveLS* procedure.

Algorithm 2. ImproveLS

Input: a solution S, step limit $MaxIter$
Output: improved solution S'

1 $S' \leftarrow S$;
2 **while:** $no\_improve\_steps < MaxIter$ **do**
3     $C_R \leftarrow$ a random large connected component in $G[V \setminus S]$;
4   **if** $no\_improve\_steps > Limit$ **then**
5         $v \leftarrow$ the oldest node from C_R;
6   **else**
7   **if** $CutP \leftarrow FindCutnode(C_R) \neq \emptyset$ **then**
8   **if** with half probability **then**
9                 $v \leftarrow$ a random node v from $CutP$;
10   **else**
11                 $v \leftarrow \text{argmin}_{x \in CutP} f(S \bigcup \{x\})$;
12   **else** $v \leftarrow$ the oldest node from C_R;
13     $S \leftarrow S \bigcup \{v\}$;
14     $u \leftarrow argmin_{x \in S} f(S \setminus \{x\})$;
15     $S \leftarrow S \setminus \{u\}$;
16   **if** $(f(S) < f(S'))$ **then** $S' \leftarrow S$;
17 **return** S';

In each iteration, a large connected component C_R is selected randomly (line 3). After that, a node

$v \in C_R$ is selected to be added to the solution. Two node selection strategies are employed alternatively, one of which exploits cut points while the other is a simple diversification strategy. Let $no\_improve\_steps$ be the counter of consecutive no improvement iterations and $Limit$ a parameter which is a positive integer. The $ImproveLS$ procedure switches between two modes according to the behavior.

When improvement is made within last $Limit$ rounds, i.e., $no\_improve\_steps < Limit$, the strategy based on cut point detection is employed to choose the node. Specifically, if there exists a cut point or there exist cut points in C_R, with a probability of 0.5, we greedily select the cut point that can minimize the objective function value the most (lines 10 and 11), and otherwise we randomly select a cut point (line 8 and 9). Otherwise, if the solution is not improved in last $Limit$ rounds or if there is no cut point in C_R, the oldest node in C_R is picked (lines 4, 5, and 12). (We define the *age* of a node as the number of rounds since the last time it changed the state, where a node v has two states w.r.t. the solution S, i.e., $v \in S$ and $v \notin S$. Then the oldest node is the one with the maximum age.) The selected node v is then added into S (line 13). After that, the node u whose removal results in the smallest value of $f(S)$ is removed from S (lines 14 and 15). The algorithm replaces S' with S if it is better than the old one (line 16).

4.1 Greedy Strategy Based on Cut Point

This subsection presents a greedy node selection strategy based on cut point. Recall that in each iteration of the $ImproveLS$ procedure, we randomly choose a large connected component C_R. If there is at least one cut point in C_R, the cut point based strategy selects a cut point $v \in C_R$ to be added into S (equivalently, removing it from the component it belongs to). Proposition 1 states a desirable property of this strategy.

Proposition 1. *In a connected component C_R with at least one cut point, for any node $v \in C_R$, we define $T(C_R,\ v)$ be the number of connected pairs after removing v from C_R. Let v^* be the minimizer of $T(C_R,\ v)$, and then v^* must be a cut point.*

Proof. For a connected component C_R, let $n = |C_R|$, and let $v_1 \in C_R$ be a cut point while $v_2 \in C_R$ is not a cut point. Obviously, $T(C_R,\ v_2) = \binom{n-1}{2}$. In the following of the proof, we mainly calculate $T(C_R,\ v_1)$.

Suppose removing v_1 from C_R divides C_R into k parts, then $T(C_R,\ v_1) = \sum_{i=1}^{k}\binom{n_i}{2}$, where n_i is the size of the i-th part and $\sum_{i=1}^{k} n_i = n-1$.

First we prove the case where $k = 2$,

$$T(C_R,\ v_1) = \sum_{i=1}^{2}\binom{n_i}{2} = \frac{1}{2}(n_1^2 + n_2^2 - n_1 - n_2),$$

while

$$\begin{aligned} T(C_R,\ v_2) &= \binom{n-1}{2} = \binom{n_1+n_2}{2} \\ &= \frac{1}{2}(n_1+n_2-1)(n_1+n_2) \\ &= \frac{1}{2}(n_1^2+n_2^2+2n_1n_2-n_1-n_2). \end{aligned}$$

Comparing $T(C_R,\ v_1)$ and $T(C_R,\ v_2)$,

$$T(C_R,\ v_2) - T(C_R, v_1) = n_1 n_2 \geqslant 0\ (n_1, n_2 \geqslant 1),$$

where$T(C_R,\ v_2)$ is larger than $T(C_R, v_1)$, which means that v^* must be a cut point when $k = 2$.

Now, we prove the case where $k > 2$. According to the arguments above,

$$\binom{n_1}{2} + \binom{n_2}{2} \leqslant \binom{n_1+n_2}{2}.$$

Similarly, we have

$$\begin{aligned} T(C_R,\ v_1) &= \sum_{i=1}^{k}\binom{n_i}{2} \\ &\leqslant \binom{n_1+n_2+\ldots+n_k}{2} \\ &\leqslant \binom{n-1}{2} = T(C_R,\ v_2). \end{aligned}$$

This proves that v^* must be a cut point when $k > 2$. □

Inspired by Tarjan's algorithm[26], we design an algorithm that can not only find all cut points in C_R, but also calculate the number of connected pairs remaining after removing v from C_R (denoted as $Cost(v)$) for each node $v \in C_R$. In detail, during the search process of Tarjan's algorithm, we maintain the children and the subtree size of each node, so that once the algorithm finds a cut point v, $Cost(v)$ can be easily calculated using the subtree size of v's children. Suppose the children of v are $\{x_0,\ x_1,\ \ldots,\ x_k\}$ and the subtree size of x is $size_x$, then $Cost(v)$ is

$$\binom{size_{x_0}}{2} + \binom{size_{x_1}}{2} + \ldots + \binom{size_{x_k}}{2} + \binom{|C_R| - size_{x_0} - size_{x_1} - \ldots - size_{x_k} - 1}{2}.$$

The complexity of this improved algorithm is $O(|V(C_R)| + |E(C_R)|)$, which is the same as Tarjan's algorithm.

If there is no cut point in C_R, we select the oldest node to be added to S. If there exists a cut point or there exist cut points, with a probability of 0.5, we select the node with the smallest *Cost*, and otherwise we randomly select a cut point.

4.2 Enhancement for Cut Point Based Strategy

The cut point based strategy is quite aggressive. Looking at just one iteration, a cut point is the best choice we can make. However, the local search procedure performs many iterations, and the cut point based strategy may lead to a local optimum.

For example, considering the case in Fig.3, we can remove two nodes. Supposing we are using the cut point based strategy, in the first round, the graph only has one connected component, which has exactly one cut point v_9. Therefore, v_9 is chosen to be removed. In the second round, the graph has two connected components, both of which have no cut point, indeed we are leaving with a cycle, and no matter which node we remove, the remaining number of the connected pairs of this graph is $\binom{7}{2} = 21$. However, the optimal solution is removing v_2 and v_5, leading to $\binom{4}{2} + \binom{4}{2} = 12$ connected pairs.

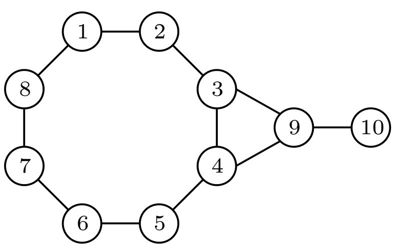

Fig.3. Simple graph with only one cut point.

To avoid being trapped in a sub-optimal solution by choosing cut points constantly, our algorithm introduces more diversification. If the solution was updated within previous *Limit* rounds, we choose a cut point; otherwise, the oldest node is chosen to be added to the solution.

5 Similarity-Aware Solution Pool Updating

CPHS maintains a solution pool during the search, which is updated by the *UpdatePool* procedure. In this section, we introduce the details of the *UpdatePool* procedure. The principle is to strike a balance between the quality of the solutions and the diversity of the solution pool.

The pseudo-code of *UpdatePool* is shown in Algorithm 3. Suppose at the time when *UpdatePool* is called, the solution pool $\mathcal{P} = \{S_1, S_2, \ldots, S_p\}$. Let us denote $\mathcal{P}^+ = \mathcal{P} \cup \{S'\}$ (S' is the newly generated solution) (line 1). We should take into account both the quality of the solutions and the diversity of the solutions in the solution pool $\mathcal{P}$. This is easy to understand, if the solutions in the pool are similar, the algorithm would visit only a small part of the solution space; on the other hand, since the new solutions are generated on the basis of the solution pool, the quality of the solutions in the pool has a direct impact on the quality of the newly generated solutions. We use a scoring mechanism based on the population management strategy in [27] to measure each solution in the pool:

$$Score(A) = rank_f(A) \times p_1 + rank_{Sim}(A) \times (1 - p_1),$$

where $rank_f$ represents the ranking of the solution w.r.t the f value and $rank_{Sim}$ represents the ranking of the solution w.r.t the similarity value.

Algorithm 3. UpdatePool

Input: a solution pool $\mathcal{P} = \{S_1, S_2, \ldots, S_p\}$ and a newly generated solution S'

Output: an updated solution pool $\mathcal{P}$

```
1  P+ ← P ⋃ {S'};
2  S_{p+1} ← S';
3  for i = 1, 2, ..., p+1 do
4      Score_i ← score of S_i calculated by the scoring mechanism;
5  w ← argmax_{x∈{1, 2, ..., p+1}} Score_x;
6  P ← P+ \ {S_w};
7  S_w ← S_{p+1};
8  Sim‾ ← the overall average similarity of P;
9  Update the proportion parameter in the scoring mechanism according to Sim‾
10 for i = 1, 2, ..., p do
11     S_i mutates with a probability based on Sim‾;
12 Return P;
```

The proportion of the quality and diversity in the scoring mechanism of [27] is fixed. Differently, we propose to dynamically adjust this proportion p_1 based on the overall average similarity

$$\overline{Sim} = \sum_{A,\, B \in \mathcal{P},\, A \neq B} \frac{|A \cap B|}{K|\mathcal{P}|(|\mathcal{P}| - 1)},$$

and an obvious intuition is that if the overall average similarity is higher, which means the solutions in the solution pool have similar structures, the proportion of diversity in the scoring function should be larger to prevent the search from getting trapped in a local space. Otherwise the proportion of diversity should be smaller in order to reserve better quality solutions. Therefore we set p_1 like this:

$$p_1 = \begin{cases} 0.7, & \text{if } \overline{Sim} < 0.6, \\ 1 - 0.5\overline{Sim}, & \text{otherwise.} \end{cases}$$

The details will be presented in Section 7.

After calculating the *Score* value of each solution $A \in \mathcal{P}^+$ (lines 2–4), the solution pool $\mathcal{P}$ is updated as $\mathcal{P} \leftarrow \mathcal{P}^+ \setminus \{W\}$ (lines 6 and 7), where W is the solution with the biggest *Score* value (line 5). Then $\overline{Sim}$ is updated (line 8), causing a change of p_1 (line 9).

Moreover, to introduce diversification to the solution pool, our algorithm employs mutation operation (lines 10 and 11). After updating the solution pool, each solution will mutate with a probability also based on the overall average similarity:

$$p_2 \times \overline{Sim}^{p_3},$$

where $0 < p_2 < 1, p_3 \geqslant 1$. Specifically, for each $S_i \in \mathcal{P}$, when it mutates, each node in S_i is replaced with a node which is not in S_i with a probability of 0.5, and then S_i is improved by the *ImproveLS* procedure.

6 Discussion

Compared with the state-of-the-art CNP algorithms MACNP and VPMS, CPHS also adopts the framework of the memetic algorithm. However, it has made improvements in each component, including the following points.

- *Cross*. If the size of the result set S generated from the initial operation is less than K, CPHS will prioritize selecting cut points from large connected components to be added to S, while MACNP and VPMS select nodes randomly.
- *Local Search*. In each round of node selection, CPHS prioritizes selecting cut points to accelerate the search convergence process, while MACNP and VPMS prioritize selecting the oldest node.
- *Pool Updating*. The proportion of diversity and quality in the pool management scoring mechanism of CPHS is dynamically adjusted, while it is fixed in MACNP and VPMS.
- *Mutation*. In CPHS, a mutation strategy is applied to the solutions with a probability based on the overall average similarity to enhance solution diversity.

7 Experiments

We evaluate the performance of CPHS and compare it with state-of-the-art algorithms. In addition, we perform experimental analyses on the strategies in CPHS. The source code and detailed experiment results are available online①.

7.1 Benchmarks

Our computational studies are carried out with three benchmarks.

- The synthetic benchmark is used in the literature[16, 20, 21, 24]. The number of critical nodes (K) is given along with each graph.
- The real-world benchmark consists of 26 real-world graphs from various practical applications in areas like biology, electronics, transportation, and complex networks[21]. The number of critical nodes (K) is given along with each graph.
- The network benchmark comes from Network Data Repository[28]②, which collects massive graphs from the real world. We only report the results on the graphs with no more than 1 000 000 edges, resulting in 23 graphs. Those larger graphs are too difficult to solve even within two hours③. For each instance, the K values are set as $K{=}|V|/5, |V|/10, |V|/20$, respectively, as the K value is always in the range $[|V|/20, |V|/5]$ in the synthetic and real-world benchmarks. This finally leads to $23 \times 3 = 69$ instances in total. This benchmark has been widely used for graph-theoretic combinatorial optimization problems including maximum clique[29], coloring[30], and dominating set problems[31].

7.2 Implementations

CPHS was implemented in C++, and was compiled using GNU gcc 9.2.0 with "-O2" option. For parameter tuning, we randomly select five instances

①https://github.com/iHaN-o/CPHS, Nov. 2024.

②http://www.graphrepository.com/networks.php, Nov. 2024.

③For those graphs with more than one million edges, CPHS can find a solution for 10 graphs within two hours while the other algorithms fail to solve any of them within two hours.

from each benchmark with various sizes and various levels of difficulty. The algorithmic parameters of CPHS are divided into the following two categories.

• *Static Parameters*. $PoolSize = 20$, $MaxIter=1\,000$, $Limit = 100$, $p_0 = 0.9$, $p_2 = 0.001$, and $p_3 = 8$. We tune each of the parameters while fixing other parameter values unchanged. For each configuration, CPHS is executed for 10 runs on each instance within 3 600 seconds.

• *Dynamic Parameters*. In CPHS, p_1 is an adaptive dynamic parameter based on the overall average similarity $\overline{Sim}$, and its setting is described in Section 5. The constants 0.7 and 0.5 in the formula are tuned according to an observation that CPHS reaches its peak performance at $p_1 \in [0.5, 0.7]$ when we set it statically. When $\overline{Sim}$ is particularly small, the search is insufficient probably, and thus we fix p_1 to 0.7. Otherwise, p_1 decreases to 0.5 as $\overline{Sim}$ increases.

7.3 Competitors

We compare CPHS with state-of-the-art heuristic CNP algorithms, including CNA1[20], FastCNP[25], MACNP[21], and VPMS[24]. The codes of these algorithms are kindly provided by their authors. For the synthetic benchmark and the real-world benchmark, we set their parameters as described in [21] and [24], respectively, which are tuned by their authors for these two benchmarks. For the network benchmark, we tune its parameters in the same way as we tune our Static parameters. Because CNA1 is worse than the other two algorithms on nearly all the instances, except for some easy instances, we do not report its results.

7.4 Experimental Settings

The experiments are conducted on a server with Intel® Xeon® Platinum 8153 256-core processor with 2.00 GHz and 1 024 GB RAM under the Linux system. Each algorithm is executed 10 runs for each instance with different random seeds (1, 2, ..., 10). The time limit for each run is 3 600 seconds, as suggested in the previous CNP heuristic algorithms[20, 21, 25].

For each instance, we report for each algorithm the best objective value among the 10 trials (f^*) as well as the average objective value ($\overline{f}$). The best f^* and $\overline{f}$ found among the algorithms are shown in bold. However, for some instances of the synthetic benchmark and real-world benchmark, all algorithms obtain the same quality solution (i.e, the same minimal and average values), and for such instances, we report the average run time.

7.5 Comparative Performance

Results on Synthetic Benchmark. As seen from

Table 1. Comparison Results on the Synthetic Benchmark

Instance	K	FastCNP		MACNP		VPMS		CPHS	
		f^*	$\overline{f}$	f^*	$\overline{f}$	f^*	$\overline{f}$	f^*	$\overline{f}$
BA500	50	**195.0**	**195.0**	**195.0**	**195.0**	**195.0**	**195.0**	**195.0**	**195.0**
BA1000	75	**558.0**	**558.0**	**558.0**	**558.0**	**558.0**	**558.0**	**558.0**	**558.0**
BA2500	100	**3 704.0**	**3 704.0**	**3 704.0**	**3 704.0**	**3 704.0**	**3 704.0**	**3 704.0**	**3 704.0**
BA5000	150	**10 196.0**	**10 196.0**	**10 196.0**	**10 196.0**	**10 196.0**	**10 196.0**	**10 196.0**	**10 196.0**
ER250	50	**295.0**	**295.0**	**295.0**	**295.0**	**295.0**	**295.0**	**295.0**	**295.0**
ER500	80	**1 524.0**	1 525.5	**1 524.0**	**1 524.0**	**1 524.0**	**1 524.0**	**1 524.0**	**1 524.0**
ER1000	140	5 030.0	5 183.6	**5 012.0**	5 025.3	5 020.0	5 037.2	**5 012.0**	**5 013.4**
ER2500	200	996 023.0	1 025 661.8	904 494.0	926 635.3	918 082.0	936 760.9	**903 273.0**	**915 874.6**
FF250	50	**194.0**	**194.0**	**194.0**	**194.0**	**194.0**	**194.0**	**194.0**	**194.0**
FF500	110	**257.0**	**257.0**	**257.0**	**257.0**	**257.0**	**257.0**	**257.0**	**257.0**
FF1000	150	**1 260.0**	**1 260.0**	**1 260.0**	**1 260.0**	**1 260.0**	**1 260.0**	**1 260.0**	**1 260.0**
FF2000	200	**4 545.0**	**4 545.0**	**4 545.0**	4 545.5	**4 545.0**	**4 545.0**	**4 545.0**	**4 545.0**
WS250	70	3 179.0	3 386.5	**3 083.0**	3 130.5	**3 083.0**	**3 089.4**	**3 083.0**	3 120.1
WS500	125	2 101.0	2 120.5	**2 072.0**	**2 082.0**	2 085.0	2 085.0	**2 072.0**	2 083.3
WS1000	200	135 856.0	139 744.4	126 496.0	154 264.6	121 788.0	135 236.8	111 594.0	119 758.1
WS1500	265	13 923.0	14 212.8	13 099.0	13 224.7	**13 098.0**	**13 189.6**	13 221.0	13 395.8

Note: The underlined CPHS's results indicate that they are significantly superior to other algorithms according to the Wilcoxon signed-rank test.

Table 1, FastCNP is far too weak in comparison with MACNP, VPMS and CPHS. These three algorithms have similar results on the benchmark, with CPHS being the best among them. Specifically, CPHS has better performance than FastCNP on all instances. CPHS finds better solutions than MACNP on five instances and worse on two instances, and these figures are three and two when compared with VPMS, while obtaining the same results for the remaining instances.

Results on Real-World Benchmark. The results on the real-world benchmark are summarized in Table 2 (we round off the average objective value ($\overline{f}$) due to the space limit), which apparently demonstrates that CPHS performs significantly better than FastCNP, MACNP and VPMS on real-world instances. For the five easy instances (from Bovine to Treni_Roma), the four algorithms find solutions of the same quality. For the remaining 21 instances, CPHS dominates the competitors. Specifically, CPHS dominates FastCNP on all instances, while it finds better solutions than MACNP on 20 out of these 21 instances, in terms of both f^* and $\overline{f}$. Similarly, CPHS dominates VPMS on 20 out of these 21 instances. It is worth noting that no algorithm dominates CPHS on any instance.

Results on Network Benchmark. The detailed results are provided as supplementary file[4] due to the space limit. As MACNP and VPMS have much better performance than FastCNP on almost all instances, especially on massive instances, we focus on the comparison results of CPHS against MACNP and VPMS. CPHS finds better f^* than MACNP on 55 out of the 69 instances, while finding the same result on seven instances, and the averaged solution quality ($\overline{f}$) is better on 62 instances and same results on two instances. When compared with VPMS, CPHS finds

Table 2. Comparison Results on the Real-World Benchmark

Instance	K	FastCNP		MACNP		VPMS		CPHS	
		f^*	$\overline{f}$	f^*	$\overline{f}$	f^*	$\overline{f}$	f^*	$\overline{f}$
Bovine	3	**268**	**268**	**268**	**268**	**268**	**268**	**268**	**268**
Circuit	25	**2 099**	**2 099**	**2 099**	**2 099**	**2 099**	**2 099**	**2 099**	**2 099**
Ecoli	15	**806**	**806**	**806**	**806**	**806**	**806**	**806**	**806**
humanD	52	**1 115**	**1 115**	**1 115**	**1 115**	**1 115**	**1 115**	**1 115**	**1 115**
Treni_Roma	26	**918**	**918**	**918**	**918**	**918**	**918**	**918**	**918**
yeast1	202	**1 412**	**1 412**	**1 412**	1 412	**1 412**	**1 412**	**1 412**	**1 412**
astroph	1 877	59 929 379	60 933 263	61 269 470	61 948 413	56 229 708	57 421 239	**53 436 604**	**53 832 208**
condmat	2 313	12 701 426	13 234 928	9 271 118	9 850 815	6 057 949	6 593 803	**3 732 788**	**4 046 156**
EU_fl	119	349 100	350 596	350 762	354 870	**348 268**	349 848	**348 268**	**349 266**
facebook	404	751 425	790 614	722 113	787 886	696 418	**761 198**	**680 258**	762 154
grqc	524	15 871	16 084	13 631	13 644	13 635	13 653	**13 595**	**13 631**
H1000	100	316 727	322 606	309 362	312 737	**306 349**	311 437	**306 349**	**310 359**
H2000	200	1 317 841	1 331 228	1 264 907	1 284 196	1 247 922	1 259 022	**1 246 172**	**1 254 923**
H3000a	300	2 989 389	3 023 400	2 911 248	2 955 816	2 840 529	2 853 246	**2 799 139**	**2 833 486**
H3000b	300	2 978 787	3 018 403	2 886 180	2 960 133	2 839 488	2 857 123	**2 822 633**	**2 834 821**
H3000c	300	2 968 978	3 011 465	2 889 965	2 936 136	2 835 510	2 844 654	**2 782 091**	**2 821 088**
H3000d	300	3 018 962	3 033 349	2 913 031	2 970 674	2 830 238	2 858 304	**2 783 038**	**2 821 378**
H3000e	300	2 435 534	2 469 112	2 898 302	2 963 916	2 846 889	2 863 676	**2 219 321**	**2 256 196**
H4000	400	5 362 144	5 415 118	5 211 185	5 345 563	5 109 197	5 148 288	**4 973 910**	**5 065 126**
H5000	500	8 486 534	8 529 798	8 415 527	8 581 551	8 102 079	8 146 342	**7 911 029**	**8 023 562**
hepph	1 201	10 769 287	11 448 988	10 080 780	10 590 445	10 046 236	10 508 820	**5 805 879**	**6 133 045**
hepth	988	149 294	252 648	106 674	108 178	113 747	116 076	**105 079**	**105 790**
OClinks	190	617 790	619 790	615 574	616 460	612 313	614 261	**612 303**	**613 823**
openfl	186	29 676	30 083	28 700	29 109	26 875	28 676	**26 777**	**28 315**
powerg	494	16 063	16 146	15 904	15 927	15 952	15 998	**15 856**	**15 863**
USAir97	33	4 726	5 181	**4 336**	**4 336**	**4 336**	5 331	**4 336**	**4 336**

Note: The underlined CPHS's results indicate that they are significantly superior to other algorithms according to the Wilcoxon signed-rank test.

[4]https://github.com/iHaN-o/CPHS/blob/main/Newwork_compare_with_sota.pdf, Nov. 2024.

better f^* on 45 instances, and same on seven instances, and $\overline{f}$ is better on 50 instances and same on three instances. These strong results clearly show the superiority of CPHS on these large-scale instances.

Results on Average Run Time. We also compare the average run time of the algorithms for those instances where they find the same quality solutions, and the results are depicted in Fig.4. The average run time of CPHS is significantly less than that of FastCNP and VPMS, and is usually more than 10x faster. CPHS is also faster than MACNP for most instances, with only two exceptions.

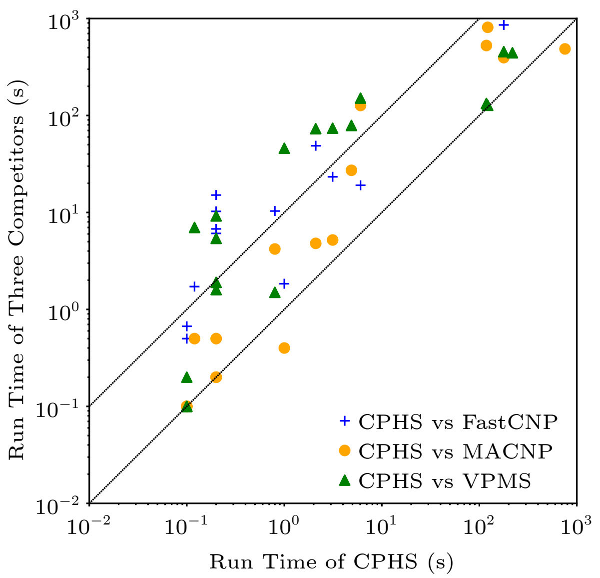

Fig.4. Average run time of CPHS and competitors on all instances where the algorithms find the same quality solutions.

Summarized Results. The results on the three benchmarks are summarized in Table 3. In particular, for any comparison between CPHS and each competitor, we perform the Wilcoxon signed-rank test[32] to examine the statistical significance. For each instance, if all the p-values of Wilcoxon signed-rank tests at 95% confidence level are smaller than 0.05 (indicating statistical significance)[32, 33], the performance improvement of CPHS over all its competitors is considered to be statistically significant, and the results of CPHS are marked using the underline. As seen from Tables 1–3, the performance of CPHS is significantly better than that of all competitors.

7.6 Component Analysis

We also study the effectiveness of the key strategies of our algorithm. We modify CPHS to obtain four alternative versions.

- CPHS_0 removes the age based diversification enhancement.
- CPHS_1 removes the greedy strategy based on cut point (always selects the oldest node to be added to the solution).
- CPHS_2 removes the mutation operation.
- CPHS_3 replaces dynamic parameters in the mutation operation and pool updating with static parameters.

The comparison of CPHS and its alternatives on the synthetic benchmark and the real-world benchmark is shown in Table 4 (15 easy instances are not reported since all variants can find the same optimal solution). For the network benchmark, the results are provided as supplementary file⑤. And we summarize the comparison results: as seen from Table 5, the performance of every alternative is far weaker than CPHS, of which CPHS_1 is the worst. These results demonstrate the effectiveness of the strategies in CPHS especially the cut point based greedy strategy.

8 Conclusions

This paper proposed an effective local search algorithm CPHS to solve the critical node problem (CNP), which integrates two main novel ideas. The first one is a cut point based local search procedure, while the second one is a dynamic pool updating strategy. The comparison results between CPHS and state-of-the-art CNP algorithms showed that CPHS dominates on a wide range of benchmarks. Particularly, CPHS performs much better on real-world graphs

Table 3. Summarized Results of CPHS with MACNP and VPMS on Three Benchmarks

	CPHS vs MACNP						CPHS vs VPMS					
	Synthetic		Real-World		Network		Synthetic		Real-World		Network	
	f^*	$\overline{f}$	f^*	$\overline{f}$	f^*	$\overline{f}$	f^*	$\overline{f}$	f^*	$\overline{f}$	f^*	$\overline{f}$
Better	2	5	19	20	55	62	4	4	17	19	45	50
Same	13	9	7	6	7	2	11	10	9	6	7	3
Worse	1	2	0	0	7	5	1	2	0	1	17	16

⑤https://github.com/iHaN-o/CPHS/blob/main/Network_compare_with_alternatives.pdf, Nov. 2024.

Table 4. Comparison Results of CPHS and Its Alternatives on the Synthetic Benchmark and Real-World Benchmark

Instance	$CPHS_0$		$CPHS_1$		$CPHS_2$		$CPHS_3$		CPHS	
	f^*	$\bar{f}$	f^*	$\bar{f}$	f^*	$\bar{f}$	f^*	$\bar{f}$	f^*	$\bar{f}$
ER1000	5 014	5 014.0	**5 012**	**5 013.0**	**5 012**	5 014.1	**5 012**	5 013.2	**5 012**	5 013.4
ER2500	914 658	928 366.7	907 630	924 858.3	909 568	921 895.6	910 112	922 141.2	**903 273**	**915 874.6**
FF2000	**4 545**	4 545.1	**4 545**	**4 545.0**	**4 545**	4 545.2	**4 545**	**4 545.0**	**4 545**	**4 545.0**
WS1000	**100 321**	**111 481.2**	118 019	134 621.5	105 221	117 973.7	102 563	114 768.8	111 594	119 758.1
WS1500	13 212	**13 355.7**	**13 209**	13 450.0	13 277	13 502.4	13 212	13 433.6	13 221	13 395.8
WS500	2 085	2 085.0	2 085	2 086.9	2 085	2 096.8	**2 072**	**2 083.3**	**2 072**	**2 083.3**
WS250	**3 083**	**3 097.2**	**3 083**	3 144.6	**3 083**	3 152.4	**3 083**	3 112.0	**3 083**	3 120.1
astroph	54 845 785	55 335 760.0	61 010 134	61 690 842.9	**53 240 664**	53 780 643.3	53 277 103	**53 743 429.4**	53 436 604	53 832 207.7
condmat	4 897 042	5 418 729.0	5 131 547	5 768 460.5	4 063 158	4 341 527.6	3 969 536	4 393 034.7	**3 732 788**	**4 046 156.0**
EU_flights	350 762	351 681.5	**348 268**	349 764.4	**348 268**	350 263.2	**348 268**	349 515.0	**348 268**	**349 265.6**
facebook	743 835	**757 193.0**	777 553	811 807.0	760 395	796 718.5	714 610	789 844.4	**680 258**	762 153.7
grqc	13 605	13 637.3	13 602	13 639.8	13 701	13 740.1	13 625	13 682.9	**13 595**	**13 631.4**
Ham1000	313 403	318 673.9	309 722	312 194.2	307 279	310 141.5	**306 349**	**309 248.7**	**306 349**	310 358.8
Ham2000	1 280 590	1 298 453.8	1 246 886	1 255 201.6	1 244 002	**1 253 642.4**	**1 239 349**	1 256 853.4	1 246 172	1 254 923.3
Ham3000a	2 888 682	2 938 656.7	2 835 573	2 848 173.8	2 813 353	2 849 744.9	2 827 676	2 856 395.7	**2 799 139**	**2 833 485.7**
Ham3000b	2 863 941	2 921 528.1	2 812 876	2 842 161.8	**2 807 253**	2 839 806.0	2 819 521	2 849 315.0	2 822 633	**2 834 820.9**
Ham3000c	2 859 843	2 907 585.4	2 806 678	2 833 888.5	2 802 685	2 836 148.3	2 802 896	2 843 865.0	**2 782 091**	**2 821 087.8**
Ham3000d	2 901 115	2 952 580.6	2 812 166	2 861 616.0	2 794 781	2 828 627.7	2 808 517	2 846 102.9	**2 783 038**	**2 821 378.2**
Ham3000e	2 295 684	2 339 004.9	2 254 927	2 277 048.2	2 237 303	2 265 403.5	2 226 559	2 267 203.3	**2 219 321**	**2 256 195.7**
Ham4000	5 152 243	5 234 590.2	5 015 960	5 079 097.2	5 055 748	5 099 579.0	5 032 353	5 081 813.6	**4 973 910**	**5 065 125.9**
Ham5000	8 127 622	8 196 694.7	7 978 287	8 034 887.5	**7 896 059**	**7 972 842.2**	7 953 752	8 032 924.6	7 911 029	8 023 561.6
hepph	6 422 337	7 144 325.7	7 581 238	8 008 581.6	5 856 258	6 239 015.8	**5 520 826**	**5 935 976.9**	5 805 879	6 133 044.9
hepth	**104 424**	**105 222.4**	105 182	106 423.9	105 835	106 313.4	105 892	107 002.7	105 079	105 790.2
OClinks	612 303	614 149.2	612 303	614 810.9	**611 250**	614 484.0	612 303	**613 728.1**	612 303	613 822.6
openflights	27 024	28 305.9	**26 777**	**28 211.3**	26 842	28 555.1	**26 777**	28 357.9	**26 777**	28 315.4
powergrid	15 854	**15 858.7**	**15 853**	15 860.2	15 870	15 881.7	15 861	15 902.1	15 856	15 863.2
USAir97	5 418	5 436.2	**4 336**	**4 336.0**	**4 336**	**4 336.0**	**4 336**	**4 336.0**	**4 336**	**4 336.0**

Table 5. Summarized Results of CPHS with Its Alternatives on Three Benchmarks

	CPHS vs $CPHS_0$		CPHS vs $CPHS_1$		CPHS vs $CPHS_2$		CPHS vs $CPHS_3$	
	f^*	$\bar{f}$	f^*	$\bar{f}$	f^*	$\bar{f}$	f^*	$\bar{f}$
Better	39	42	61	64	45	52	40	48
Same	9	5	3	3	8	6	8	6
Worse	21	22	5	2	16	11	21	15

and large-scale graphs. Based on these results, we concluded that our algorithm pushes the state-of-the-art in solving CNP over a broad range of benchmarks.

It would be interesting to study the dynamic updating method to population-based heuristic search algorithms for other problems.

Conflict of Interest The authors declare that they have no conflict of interest.

References

[1] Shen Y, Nguyen N P, Xuan Y, Thai M T. On the discovery of critical links and nodes for assessing network vulnerability. *IEEE/ACM Trans. Networking*, 2013, 21(3): 963–973. DOI: 10.1109/TNET.2012.2215882.

[2] Boginski V, Commander C W. Identifying critical nodes in protein-protein interaction networks. In *Clustering Challenges in Biological Networks*, Butenko S, Chaovalitwongse W A, Pardalos P M (eds.), World Scientific, 2009, pp.153–167. DOI: 10.1142/9789812771667_0007.

[3] Tomaino V, Arulselvan A, Veltri P, Pardalos P M. Studying connectivity properties in human protein–protein interaction network in cancer pathway. In *Data Mining for Biomarker Discovery*, Pardalos P M, Xanthopoulos P, Zervakis M (eds.), Springer-Verlag, 2012, pp.187–197. DOI: 10.1007/978-1-4614-2107-8_10.

[4] Nguyen D T, Shen Y, Thai M T. Detecting critical nodes in interdependent power networks for vulnerability assessment. *IEEE Trans. Smart Grid*, 2013, 4(1): 151–159. DOI: 10.1109/TSG.2012.2229398.

[5] Aspnes J, Chang K, Yampolskiy A. Inoculation strategies for victims of viruses and the sum-of-squares partition problem. *Journal of Computer and System Sciences*, 2006, 72(6): 1077–1093. DOI: 10.1016/j.jcss.2006.02.003.

[6] Kempe D, Kleinberg J, Tardos É. Maximizing the spread

of influence through a social network. In *Proc. the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2003, pp.137–146. DOI: 10.1145/956750.956769.

[7] Borgatti S P. Identifying sets of key players in a social network. *Computational & Mathematical Organization Theory*, 2006, 12: 21–34. DOI: 10.1007/s10588-006-7084-x.

[8] Arulselvan A, Commander C W, Elefteriadou L, Pardalos P M. Detecting critical nodes in sparse graphs. *Computers & Operations Research*, 2009, 36(7): 2193–2200. DOI: 10.1016/j.cor.2008.08.016.

[9] Shen S, Smith J C. Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs. *Networks*, 2012, 60(2): 103–119. DOI: 10.1002/net.20464.

[10] Lalou M, Tahraoui M A, Kheddouci H. The critical node detection problem in networks: A survey. *Computer Science Review*, 2018, 28: 92–117. DOI: 10.1016/j.cosrev.2018.02.002.

[11] Di Summa M, Grosso A, Locatelli M. Branch and cut algorithms for detecting critical nodes in undirected graphs. *Computational Optimization and Applications*, 2012, 53(3): 649–680. DOI: 10.1007/s10589-012-9458-y.

[12] Pavlikov K. Improved formulations for minimum connectivity network interdiction problems. *Computers & Operations Research*, 2018, 97: 48–57. DOI: 10.1016/j.cor.2018.04.012.

[13] Dinh T N, Thai M T. Precise structural vulnerability assessment via mathematical programming. In *Proc. the 2011 Military Communications Conference*, Nov. 2011, pp.1351–1356. DOI: 10.1109/MILCOM.2011.6127492.

[14] Walteros J L, Veremyev A, Pardalos P M, Pasiliao E L. Detecting critical node structures on graphs: A mathematical programming approach. *Networks*, 2019, 73(1): 48–88. DOI: 10.1002/net.21834.

[15] Aringhieri R, Grosso A, Hosteins P, Scatamacchia R. VNS solutions for the critical node problem. *Electronic Notes in Discrete Mathematics*, 2015, 47: 37–44. DOI: 10.1016/j.endm.2014.11.006.

[16] Ventresca M, Aleman D. A derandomized approximation algorithm for the critical node detection problem. *Computers & Operations Research*, 2014, 43: 261–270. DOI: 10.1016/j.cor.2013.09.012.

[17] Ventresca M. Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem. *Computers & Operations Research*, 2012, 39(11): 2763–2775. DOI: 10.1016/j.cor.2012.02.008.

[18] Ventresca M, Aleman D. A fast greedy algorithm for the critical node detection problem. In *Proc. the 8th International Conference on Combinatorial Optimization and Applications*, Dec. 2014, pp.603–612. DOI: 10.1007/978-3-319-12691-3_45.

[19] Aringhieri R, Grosso A, Hosteins P, Scatamacchia R. Local search metaheuristics for the critical node problem. *Networks*, 2016, 67(3): 209–221. DOI: 10.1002/net.21671.

[20] Pullan W. Heuristic identification of critical nodes in sparse real-world graphs. *Journal of Heuristics*, 2015, 21(5): 577–598. DOI: 10.1007/s10732-015-9290-5.

[21] Zhou Y, Hao J K, Glover F. Memetic search for identifying critical nodes in sparse graphs. *IEEE Trans. Cybernetics*, 2019, 49(10): 3699–3712. DOI: 10.1109/TCYB.2018.2848116.

[22] Addis B, Aringhieri R, Grosso A, Hosteins P. Hybrid constructive heuristics for the critical node problem. *Annals of Operations Research*, 2016, 238(1/2): 637–649. DOI: 10.1007/s10479-016-2110-y.

[23] Aringhieri R, Grosso A, Hosteins P, Scatamacchia R. A general evolutionary framework for different classes of critical node problems. *Engineering Applications of Artificial Intelligence*, 2016, 55: 128–145. DOI: 10.1016/j.engappai.2016.06.010.

[24] Zhou Y, Hao J K, Fu Z H, Wang Z, Lai X. Variable population memetic search: A case study on the critical node problem. *IEEE Trans. Evolutionary Computation*, 2021, 25(1): 187–200. DOI: 10.1109/TEVC.2020.3011959.

[25] Zhou Y, Hao J K. A fast heuristic algorithm for the critical node problem. In *Proc. the Genetic and Evolutionary Computation Conference Companion*, Jul. 2017, pp.121–122. DOI: 10.1145/3067695.3075993.

[26] Tarjan R. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1972, 1(2): 146–160. DOI: 10.1137/0201010.

[27] Lü Z, Hao J K. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 2010, 203(1): 241–250. DOI: 10.1016/j.ejor.2009.07.016.

[28] Rossi R A, Ahmed N K. The network data repository with interactive graph analytics and visualization. In *Proc. the 29th AAAI Conference on Artificial Intelligence*, Jan. 2015, pp.4292–4293. DOI: 10.1609/aaai.v29i1.9277.

[29] Rossi R A, Gleich D F, Gebremedhin A H, Patwary M M A. Fast maximum clique algorithms for large graphs. In *Proc. the 23rd Int. Conf. World Wide Web*, Apr. 2014, pp.365–366. DOI: 10.1145/2567948.2577283.

[30] Lin J, Cai S, Luo C, Su K. A reduction based method for coloring very large graphs. In *Proc. the 26th International Joint Conference on Artificial Intelligence*, Aug. 2017, pp.517–523. DOI: 10.24963/ijcai.2017/73.

[31] Cai S, Hou W, Wang Y, Luo C, Lin Q. Two-goal local search and inference rules for minimum dominating set. In *Proc. the 29th International Joint Conference on Artificial Intelligence*, Jul. 2020, pp.1467–1473. DOI: 10.24963/ijcai.2020/204.

[32] Conover W J. Practical Nonparametric Statistics (3rd edition). John Wiley & Sons, 1999.

[33] Luo C, Zhao Q, Cai S, Zhang H, Hu C. SamplingCA: Effective and efficient sampling-based pairwise testing for highly configurable software systems. In *Proc. the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Nov. 2022, pp.1185–1197. DOI: 10.1145/3540250.3549155.

Zhi-Han Chen received his B.S. degree in computer science and technology from Peking University, Beijing, in 2020. He is currently pursuing his Ph.D. degree with the School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing. His current research interests include constraint solving and electronic design automation.

Shao-Wei Cai received his Ph.D. degree in computer science from Peking University, Beijing, in 2012. He is currently a professor in Institute of Software, Chinese Academy of Sciences, Beijing. He has developed efficient SAT/SMT/MaxSAT solvers, which have received many awards in SAT/SMT/MaxSAT competitions (or evaluations). He has won the best paper award of SAT 2021 conference. His current research interests include constraint solving and electronic design automation.

Jian Gao received his Ph.D. degree in computer application technology from Dalian Maritime University, Dalian. He is currently a professor of computer science in Northeast Normal University, Changchun. His research interests include automated reasoning, constraint programming and heuristics.

Shi-Ke Ge received his B.E. degree in computer science and technology from Shenyang University of Technology, Shenyang, in 2021. He is currently a master degree candidate in computer science and technology from Dalian University of Technology, Dalian. His research interest includes constraint solving.

Chan-Juan Liu received her Ph.D. degree in computer software and theory from Peking University, Beijing, in 2016. She is currently an associate professor with School of Computer Science and Technology, Dalian University of Technology, Dalian. Her current research interests include game theory and multi-agent decision making.

Jin-Kun Lin received his Ph.D. degree in computer science and theory from Peking University, Beijing, in 2018. He is currently the chief technology officer with SeedMath Technology Limited, Beijing. His current research interests include software testing and heuristic search.