PAPER

Cluster Expansion Method for Critical Node Problem Based on Contraction Mechanism in Sparse Graphs

Zheng WANG[†], ***Member and*** **Yi DI**[†a)], ***Nonmember***

SUMMARY The objective of critical nodes problem is to minimize pair-wise connectivity as a result of removing a specific number of nodes in the residual graph. From a mathematical modeling perspective, it comes the truth that the more the number of fragmented components and the evenly distributed of disconnected sub-graphs, the better the quality of the solution. Basing on this conclusion, we proposed a new Cluster Expansion Method for Critical Node Problem (CEMCNP), which on the one hand exploits a contraction mechanism to greedy simplify the complexity of sparse graph model, and on the other hand adopts an incremental cluster expansion approach in order to maintain the size of formed component within reasonable limitation. The proposed algorithm also relies heavily on the idea of multi-start iterative local search algorithm, whereas brings in a diversified late acceptance local search strategy to keep the balance between interleaving diversification and intensification in the process of neighborhood search. Extensive evaluations show that CEMCNP running on 35 of total 42 benchmark instances are superior to the outcome of KBV, while holding 3 previous best results out of the challenging instances. In addition, CEMCNP also demonstrates equivalent performance in comparison with the existing MANCNP and VPMS algorithms over 22 of total 42 graph models with fewer number of node exchange operations.

key words: *critical node problem, contraction mechanism, incremental cluster expansion, iterative local search, diversified late acceptance local search*

1. Introduction

Given an undirected graph $G(V, E)$, associated with a predefined integer K, the main objective of Critical Node Problem (CNP) is trying to remove a subset of K nodes, labeled as S, $S \subset V$, such that the number of pair-wise connectivity in the residual graph $G[V \backslash S]$ is minimum. It is clear that the leaving graph separated by the critical node set S consists of various connected branches, usually removing any deleted node in S to the residual sub-graph is bound tightly to increase the target value of CNP.

Due to its nature properties of physical significance, there exists a number of applications which are closely related with CNPs. Starting from identifying a small number of key nodes in a network in academic area [1], [2], to annotate critical nodes in order to ensure they operate reliably for transporting people and goods throughout the field of transportation engineering [3]. In addition, CNPs likewise applications equally play an important role in military and terrorist networks, for the sake of finding specific vital facilities and key leaders whose deletion would result in the maximum breakdown of communication between individuals in the battlefield and terrorist networks. Whereas the second one stands for the correlation tracking, whose ultimate goal is to discovery a series of crony influences because of certain important nodes infection in a dynamic way. Similar cases exist in the hot spot and sensitive news analysis on social media, as well as COVID-19 pandemic infection chain tracking [4]. Under such conditions, government and scientists should not only pay attention to "critical nodes" (known as super-spreaders), but also keep a watchful eye on the wide spectrum of radiation arising from super-spreaders (acknowledged as asymptomatic carriers or close contacts).

When turning to the theoretical model of CNP, a undirected graph was brought up. However, the CNP has been shown to be NP-hard [5], even though in the tree structure, it is proven to be NP complete when considering non-unit edge costs [6]. Researches crowed by a series of special patterns including interval graphs, regular graphs, planar graphs, bipartite graphs, split graphs and trees have been studied [7], [8], as well as random graphs and complex networks. However, the denser the connections between nodes are, the higher the time complexity of CNP is.

To date, there exists two clades with regard to Critical Node Problem, One is exact approaches and the other is approximation algorithms. From a practical point of view, the former makes every effort to pursue the optimal solutions responding to the proposed impending problems in reasonable time limit, who is usually indicated as an integer linear programming model with a non-polynomial number of constraints. And the latter often returns a solution for a combinatorial optimization challenging that is provably close to the optimal. A detailed classification of the extant methods is provided in Fig. 1.

Exact approaches are more attractive due to the nature that they are theoretically able to guarantee the optimality solutions, note that the dynamic programming, branch-and-cut, and path enumeration method are the typical representatives. In fact, for some applications, finding the optimal solution to the intractable problem may not be a necessity, usually a near-optimal or suboptimal will suffice. In these cases, approximation algorithms demonstrate quite predictable performance without provable approximation bounds, note that computation time for very large networks remains an important issue in these methods as well. The lower branch drawn in Fig. 1 falls into this category.

Manuscript received July 12, 2021.
Manuscript revised January 12, 2022.
Manuscript publicized February 24, 2022.

[†]The authors are with the the School of Information and Communication Engineering, Hubei University of Economics, Wuhan, China.
a) E-mail: diyi8710@hbue.edu.cn (Corresponding author)

DOI: 10.1587/transinf.2021EDP7150

This work explores a new designed paradigm to deal with the Critical Node Problem in sparse graphs, according to the findings presented in [5], which indicates that increasing the number of disconnected components, as well as leaving the average of node sizes among connected branches as much as possible result in a better objective of the CNP, we put forward an incremental cluster expansion algorithm which commits to maintain the size of formed components within reasonable limitation. For this purpose, first we estimate the theoretical optimal number of vertex set in ultimately separated sub-graphs in terms of maximal independent set, and then continuously expand the scale of cluster (defined in Sect. 3, which starts from a random or chosen beginning node) by bringing in new neighbors until meets the calculated upper bound. In principle, our proposed method CEMCNP is still based on the idea of heuristic search. Different from traditional algorithms, CEMCNP holds the following characteristics.

- In order to obtain high quality initial solutions, CEMCNP makes use of an incremental cluster expansion mechanism to iteratively separate the graph into many disconnected components, whose vertex numbers are kept within optimum limits;
- In face with the heavy computational cost, CEMCNP introduces a contraction mechanism to greedy alleviate the effect of vertex scale without loss of accuracy;
- To speed up the computations of the objective function for CNP, CEMCNP takes advantage of a Incremental Estimation Mechanism to evaluate the effectiveness between neighborhood interchange in low complexity. Moreover, CEMCNP also adopts a multi-start search to escape the local optimal solutions, as well as a diversified late acceptance local search strategy to guarantee the diversification and intensification of candidate solution set.

The rest of the paper is organized as follows. Section 2 gives our basic idea of Critical Node Problem, which is divided into two subsections recorded as problem domain definition and provenances of idea. Together in Sect. 3 we introduce the proposed CEMCNP algorithm, followed by the comparative evaluation in Sect. 4. And at last conclude the paper.

2. Basic Idea of Critical Node Problem

2.1 Problem Domain Definition

Critical Nodes problem (CNP)

INPUT: An undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$, together with a predefined integer K, $K < n$.

OUTPUT: A subset of vertices S, $S \subset V$, $|S| \leq K$, which minimizes the object function $f(S)$:

$$f(S) = \sum_{C_i \subset G(V \backslash S)} \frac{|C_i|\,(|C_i| - 1)}{2} \tag{1}$$

Where C_i is the set of all connected components in the residual graph $G = (V \backslash S)$ after removing the whole nodes in S.

2.2 Provenances of Idea

2.2.1 Provenance of Cluster Expansion

When we focus our analyses on the scenario given the optimal solution as described in Fig. 2, it comes the truth that maximizing the number of components in the remaining graph, as well as minimizing the total square difference in sub-graph size leads to optimize the ultimately solutions.

Intuitively, separating the undirected graph into pieces of equal size within optimal range contributes to the decline of objective value in terms of Lemma 1 and Lemma 2 in [5]. At the same time, we notice that nodes in each connected component in the residual graph keeps mutual reachable. Therefore, if we iteratively divide the remaining graph into sub-graph whose scale meets the theoretically reasonable value until the removed nodes size reaches the predefined threshold K, these separated pieces of sub-graphs then ultimately consist of a initial settlement which may approach the optimal solution. That's the essence of our proposed cluster expansion mechanism. In order to detailed describe the algorithm, several necessary definitions are given in the

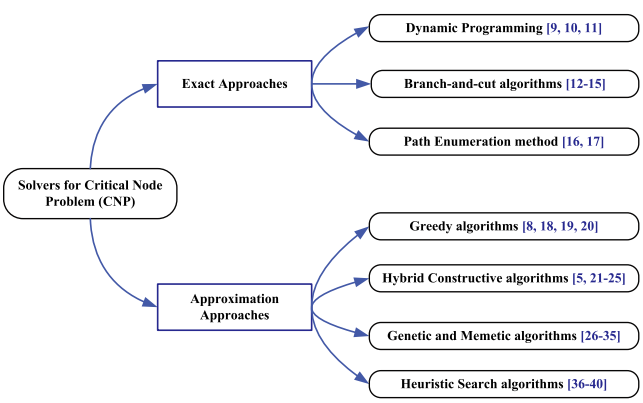


Fig. 1 Existing approaches for solving the SRFLP.

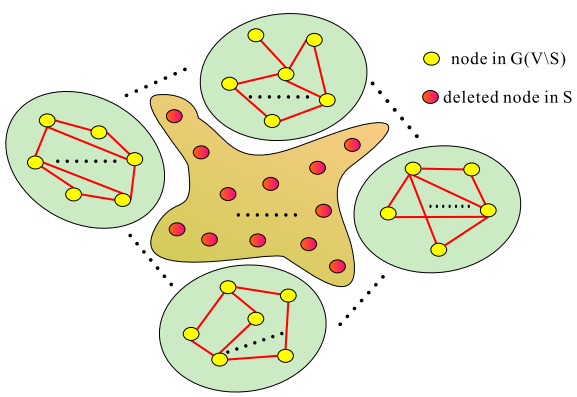


Fig. 2 Scenario of CNP given the optimal solution.

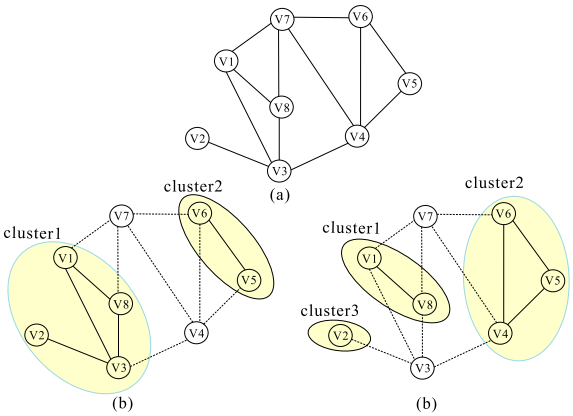


Fig. 3 An illustrative example of cluster expansion.

following.

Definition 1. ***Cluster****, stands for a set of vertices connected in the remaining graph, together with the adjacent edges between each other.*

Definition 2. ***Dominated node set****, is generally accompanied with a predefined cluster, whose removing makes the cluster isolate from the residual graph. For simplicity, the dominated node set of a cluster usually consists of all the neighbors coming from the nodes standed in the cluster.*

According to Fig. 3 (b) (first appeared in the work of [35]), we notice that after deleting the node set $S = \{V_4, V_7\}$, the remaining graph is divided into two subsets of vertices, listed by $cluster1 : \{V_1, V_2, V_3, V_8\}$ and $cluster2 : \{V_5, V_6\}$ respectively. Taking deeply studies for Fig. 3, we found the truth that on the one hand, a vertex collection of $\{V_4, V_7\}$ makes up the dominated node set for cluster1, and on the other hand, it also belongs to the dominated node set for cluster2. In this way, we easily deduce the following corollary.

Corollary 2.1. *A dominated node set (labeled as MNS) for a particular cluster separates the graph into many other clusters, partial or total collections of vertices including in MNS make up of dominated node sets responding to these newly formed clusters respectively in the residual graph.*

Proof. Firstly, it is obvious that the dominated node set of MNS isolates the particular cluster from the current graph, as a result, the remaining sub-graph after eliminating this particular cluster together with its dominated node set is also separated by the same MNS. Moreover, in the worst case, the residual graph still keeps connected. In the general case, there exists a scenario in which the rest vertices constitute several pieces of connected components (clusters) in term of the MNS, usually part of the MNS can separate these clusters from each other. □

Figure 3 (c) gives an example for Corollary 2.1, there exists a dominated node set MNS of $\{V_3, V_7\}$ for cluster1, a partial set of MNS which only containing node $\{V_3\}$ stands for a dominated node set for cluster3, and the whole collection of MNS which including both nodes $\{V_3, V_7\}$ makes up another dominated node set for cluster2 in the remaining graph.

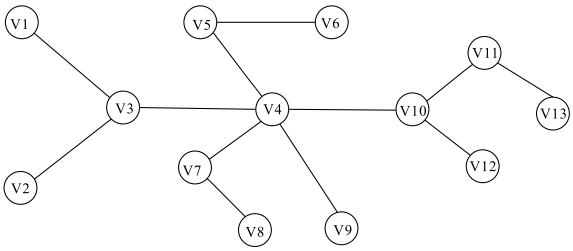


Fig. 4 An example of node selection for CNP in sparse graph.

To grasp the underlying methodology of our proposed cluster expansion, two key points must be taken into consideration. One is the way how the cluster grows, and the other is the optimal size for this chosen cluster. However, the proposed algorithm employs two kinds of methodologies to extend the cluster incrementally, marked as random growth and greedy expansion respectively. These two strategies are detailed described in the following section.

When turning attention to the second point, a optimal number of connected components based on maximum independent set is given according to Lemma 1 in [3]. In combination with Lemma 1 in [5], with considering the circumstance in which the theoretical optimal solution appeared, where a subset collection of K nodes separated the graph $G = (V, E)$ into L equal-sized clusters, the cardinality of each cluster can be computed by $\left\lfloor \frac{n-k}{L} \right\rfloor$. In view of Lemma 1 in [3], the expression has a lower bound $\left\lfloor \frac{n-k}{\alpha(G)} \right\rfloor$. Note that finding the maximum independent set in sparse graph is proved to be NP-hard [41], a maximal independent set is adopted in the process of implementation instead. This is the main idea of cluster expansion, which tries to keep the cardinality of new formed cluster at a certain threshold fluctuating in a small range out of the lower bound.

2.2.2 Provenance of Contraction Mechanism

In face of sparse graphs, connections between nodes are scarce. It is ineffective to choose the leaf node (the degree of node is one) as candidate for CNP on account of less reduction of pair-wise connectivity in the remaining graph. In most cases, it is important to identify the cut vertices and bridges of the graph since the number of disconnected components or isolated nodes (nodes of degree 0) increasing rapidly after removing these cut vertices from the graph. This can be illustrated in Fig. 4.

Figure 4 shows a CNP example with thirteen nodes and $K = 1$, it is clear that $S = \{V_4\}$ is the best solution whose removal from the graph leads to five connected components induced by $\{V_1, V_2, V_3\}$, $\{V_5, V_6\}$, $\{V_7, V_8\}$, $\{V_9\}$ and $\{V_{10}, V_{11}, V_{12}, V_{13}\}$ with a pair-wise connectivity $\binom{3}{2}+$

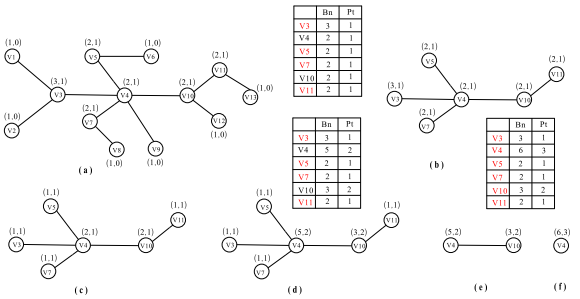


Fig. 5 An implementation of Contraction Mechanism on Fig. 4.

$\begin{pmatrix} 2 \\ 2 \end{pmatrix} + \begin{pmatrix} 2 \\ 2 \end{pmatrix} + \begin{pmatrix} 4 \\ 2 \end{pmatrix} = 11$. While compared with node V_3 and V_{10}, node V_4 gives more powerful capability in splitting the undirected graph. For this purpose, two metrics named branch number and path depth are defined.

Definition 3. ***Branch number****, the branch number of a node v_i represents the number of newly introduced connected components due to the removal of node v_i in the remaining graph.*

Definition 4. ***Leaves contraction****, leaves contraction is an operation which firstly identifies the set of leaf nodes Lf in the graph $G = (V, E)$, and then removes these leaf nodes, together with its adjoin edges including in Lf set from graph G, so as to form a new graph $G(V \backslash Lf)$.*

Definition 5. ***Path depth****, Supposing that each node in the graph $G = (V, E)$ is attached with a scalar which is initialized to zero, we iteratively run the routine of leaves contraction, and during each round the scalars lying upon the neighbors of new founded leaf nodes are incremented by one based on the current values belong to the leaves. This routine of leaves contraction keeps running until no new leaf nodes exist in the residual graph. As a result, the final scalar attached with node V_i is known as the path depth of V_i.*

In combination with the significance of branch number and path length, our contraction mechanism appears in a formal specification in Algorithm 1.

In brief, the contraction mechanism algorithm starts with the leaf nodes which are initialized with specific values, and then refreshes the two-tuples (Bn_k, Pd_k) for each node v_k next to these leaf nodes. As soon as the two metrics updates, the founded leaf nodes are departed from the previous graph, so as to form a new graph. The routine iterates continuously until no new-founded leaf nodes appear in the current graph. Figure 5 gives the detailed implementation on Fig. 4 for Algorithm 1.

From a practical point of view, the proposed contraction mechanism gives the reasons why we choose as the best solution in Fig. 4. Logically, the metric of branch number is important to characterize the splitting capacity which divides the graph into pieces. Meanwhile, another metric of path depth maintains the centrality away from leaves. Theoretically speaking, the bigger the branch number and the path length, the more capable to be a better candidate for CNP. It can be shown in Fig. 5, where node V_4 achieves the ultimate binary arrays (Bn_4, Pd_4) as $(6, 3)$. As a result, we choose node V_4 as the current best candidate for CNP. While look deep into the scenario in Fig. 5, this contraction mechanism has the following property.

Algorithm 1: Pseudo-code of Contraction Mechanism

Input: an undirected graph $G = (V, E)$;
a two-tuples(Bn_i, Pd_i): the Branch number and Path depth of node v_i;
$(Bn_i, Pd_i) = (1, 0)$, for each node v_i, $v_i \in V$ and v_i is a leave node;
$(Bn_i, Pd_i) = (0, 0)$, for each node v_i, $v_i \in V$ and v_i is not a leave node.
Output: the final convergent set of two-tuples (Bn_i, Pd_i) for each node v_i

```
while there exists at least one leaf node in graph G do
    find the set of leaf nodes set Lf;
    if there are only two leaf nodes V_p and V_q in Lf then
        choose the node with the bigger value of Bn_x for V_x;
        (Bn_x, Pd_x) ← (Bn_x, Pd_x + 1);
        G ← (V_x);
    else
        foreach V_j in Lf do
            (Bn_j_Tmp, Pd_j) ← (1, Pd_j) ;
            foreach V_k, (V_k, V_j) ∈ E do
                Pd_k ← Pd_j + 1 ;
                remove the node V_j from the current graph,
                  then G ← (V\{V_j});
                if the neighbor number of V_k is greater than
                  1 in the graph G(V\{V_j}) then
                    Bn_k ← Bn_k + Bn_j_Tmp;
                end
            end
        end
    end
end
Return the final convergent set of two-tuples (Bn_i, Pd_i) for
each node V_i
```

Corollary 2.2. *Each graph without loops will converge to a single node with maximal path depth based on the contraction mechanism.*

Proof. Supposing that there exists a graph $G = (V, E)$ with n nodes and m edges left after deleting L nodes during the K_{th} iteration of contraction operation, and meanwhile no leaf nodes can be found in the next $(K + 1)_{th}$ iteration. It is clear that each node v has a degree of at least two since no leaves exist. As a result, we have the conclusion that $m \geq n$. If we hold the hypothesis that there is no circuit in the remaining graph $G = (V, E)$, thus the graph $G = (V, E)$ must be a tree with n nodes. According to the nature of the tree, it meets the requirement that $m < n$, which is in contradiction with the condition $m \geq n$. In this way the assumption does not hold. There must be at least one loop in the graph $G = (V, E)$. Therefore, graphs without loops will eventually reduce to one node after iteratively eliminating all the other nodes with their adjacent edges, and the path depth lying upon the last node is the longest iteration depth of this

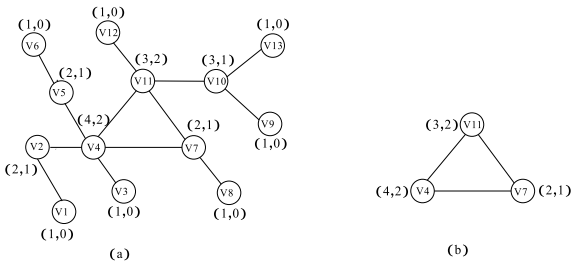


Fig. 6 An implementation of contraction mechanism in cyclic graph.

contraction mechanism. □

In principle, it is practical to take advantage of the proposed contraction mechanism while the graph is loop-free. Furthermore in most case, the sparse graphs usually contain a certain number of loops, our proposed contraction mechanism also plays an important role in reducing the scale of node numbers without loss of accuracy in CNP applications, this can be illustrated in Fig. 6.

Contrary to the loop-free graphs, models with circuit fail to converge to a single node based on the leaves contraction mechanism. However, it still makes a great effect on decreasing the node scale, while at the same time keeps the exactly calculated paired metrics (Bn_k, Pd_k) upon each node V_k. These new formed tuples help tremendously in choosing the candidate solution for CNP, for example, a node with bigger value of two-tuples (Bn_k, Pd_k) is more advisable. Figure 6, node V_4 presents the best solution because the elimination of which splits the graph into more pieces compared with any other nodes holding the same or less path depth metric.

3. Method

In this section, we present the framework of Cluster Expansion Method for Critical Node Problem (CEMCNP) based on the contraction mechanism, which is brought up in combination with a multi-start and diversified late acceptance local search strategy.

3.1 General Scheme

Up to now, iterated local search still plays a great role in widely-used meta-heuristic approaches, which iteratively applies local search to modifications of the newly introduced search point. To keep the diversity of solutions, a multi-start framework is introduced in order to enlarge the searching scope while trapping in the local optimum. Algorithm 2 shows the pseudo code of proposed CEMCNP. It is well known that better initial solution usually guarantees fast convergence and more satisfying results in heuristic search algorithms. As a result, our CEMCNP method starts with a novel cluster expansion based construction phase at the beginning of each round (see "CEBaseConstruction" in line 4), which increasingly separates out specified dimension clusters from residual graph in terms of the cluster

Algorithm 2: Pseudo-code of Cluster Expansion Method for CNP

Input: an undirected graph $G = (V, E)$, node number $n = |V|$, node set $N = \{v | v \in V\}$; time limit t_{max}, historical length HL; maximum number of iterations $MaxIters$; weighting coefficient β, parameter best cluster size L; fluctuate value D.
Output: the best solution S^* found so far

```
1  foreach round r = 1 : n do
2      randomly select a mode v from set N;
3      N ← N\{v};
4      S0 ← CEBasedConstruction(v, β, L, D),
5      S* ← S0;
6      S' ← DLALS(S0, HL, MaxIters);
7      if f(S) < f(S*) then
8          S* ← S';
9      end
10     terminateFlag ← false;
11     repeat
12         S'' ← NMBasedPerturbation(S');
13         if S' ≠ S'' then
14             S' ← DLALS (S'', HL, MaxIters);
15         end
16         if f(S') < f(S*) then
17             S* ← S';
18         else
19             terminateFlag ← true;
20         end
21     until t >= t_max OR terminateFlag;
22 end
23 Return the best found solution S*
```

expansion process, as well as a contraction mechanism (see Sect. 3.2). Afterwards, CEMCNP then employs a diversified late acceptance local search procedure (DLALS in line 5 for short) to acquire a local optima around the initial solution (see Sect. 3.3). With resorting to an iteratively paired node migration based perturbation (see the notation "NMBasedPerturbation" in line 11), together with the local search operation, CEMCNP can rapidly jump to a new search region, and attain a local optimal solution as soon as possible. At each round, CEMCNP approach repeats the above-mentioned procedure until no better result found or time limit is met.

3.2 Cluster Expansion Based Construction

In general, a good initial solution usually speeds up the search process moving towards the direction where a good solution arises. And meanwhile, the time consumed in constructing it should also be considered. In this algorithm, a better initial solution is build by iteratively separating out right-sized clusters from the residual graph, this process terminates when the total number of dominated node set belonging to such clusters exceeds the upper bound K. Once this iterative process stops, a greedy approach which continuously deletes a node from current solution with minimal increment of objective function is brought in. Notice that when the cardinality of current solution reaches K, a

Algorithm 3: Pseudo-code of ClusterExpansion-BasedConstruction

Input: an undirected graph $G = (V, E)$, node number $n = |V|$, node set $N = \{v|v \in V\}$; solution upper bound K, weighting coefficient β; parameter best cluster size L, fluctuate value D; the selected node v.
Output: the initial solution $S_0, |S_0| = K$

```
1  S_0 ← {};
2  while |S_0| < K do
3      find the largest cluster of G as G';
4      contractionMechanism(G');
5      if graph G' can converge to a single node u then
6          S_0 ← S_0 ∪ {u};
7          G ← G(V\{u});
8          continue;
9      else
10         newCluster C ← {},
11         newClusterDominatedNodes Dns_C ← {};
12         if node v is in the sub-graph G' then
13             C ← C ∪ {v};
14         else
15             randomly select a node v' from the sub-graph G';
16             C ← C ∪ {v'};
17             Dns_C ← ClusterExpansion(C, G', b, L, D)
18             S_0 ← S_0 ∪ Dns_C;
19             G ← G(V\(S_0 ∪ C));
20         end
21     end
22 end
23 while |S_0| > K do
24     u* ← arg_{u∈S_0} min{f(S_0\{u}) − f(S_0)};
25     S_0 ← S_0\{u*};
26 end
27 Return the initial solution S_0
```

good initial solution comes up. The detailed peusedo-code of cluster expansion based construction is provided in Algorithm 3. With considering the conclusion declared in [5], we hold the hypothesis that better solution of CEMCNP should maximally split the graph and simultaneously minimize the variance among the size of divided components. In this way, the proposed construction method always begins to fragment the graph from a large connected component instead of a small one, this can be shown in line 3.

In order to cut down the heavy time consuming cost, a contraction mechanism runs at the beginning of each round (line 4). With the help of leaves contraction operation, each node v covers a paired tuple (Bn_v, Pd_v). the number of searches will drop by several orders of magnitude by means of these metrics. At the same time, the iteratively calculated metrics branch number and path depth still provide the comprehensive reference information for deciding the best candidate. In particular, when the graph turns to be an acyclic undirected graph, greedily picking up the node with the biggest paired metrics (Bn, Pd) from the largest component well consists of our initial solution (line 5-9). Taken from the results of multiply runs, we found the truth that once a graph converges to a single node by means of the leaves contraction operation, any of its sub-graph exhibits the same characteristic. However, most of the time there still exists denumerable loops in the current graph, we then take advantage of a cluster expansion process (line 10-18) which iteratively cuts the graph into pieces with reasonable numbers and size to decrease the pair-wise connectivity in the residual graph. In the end, a greedy approach to keep the size of initial solution under the upper bound of K is needed (line 23-25). Obviously, the cluster expansion operation consists of the core component of this cluster expansion based construction algorithm. In order to put into practice, three critical points should be considered: the starting point of a cluster, the way to broaden the cluster and terminal condition of expansion operation.

3.2.1 The Starting Point of a Cluster

The definition of a starting point covers two meanings: one is the beginning node when gets down to constructing a new solution in face of the original graph. As described in Algorithm 2, CEMCNP takes turns to choose a node as the beginning entry of the cluster expansion during each round of solution construction phase. In this way, every node standed in the original graph can take the role of sponsor to obtain a new solution. And the other one stands for the beginner when starts to separate a cluster from a large connected component. Normally a random approach is desirable (see line 15 in Algorithm 3).

3.2.2 The Way to Broaden the Cluster

Once a starting point, together with a complementary graph is determined, cluster expansion routine tries best to partition the graph into appropriate clusters in terms of the continuously growing dominated node set. With considering the scenario in Fig. 2, we assume that the deleted nodes in S consist of the best solution for CNP in this graph.

It is not hard to find that the nodes in the identical cluster are mutual reachable, and the nodes standing in different clusters are inaccessible. The reason behind this is that the nodes assembling in S break down the connections between these nodes. From this perspective, the solution S carries another form of expression: the union of dominated node set associated with scattered clusters in the remaining graph. Based on this idea, cluster expansion routine continuously constructs new cluster from neighborhoods, and simultaneously leaves the dominated node as part of solution for CNP. Once the total size of accumulated dominated nodes exceeds the upper bound K, this iterative process then terminated.

To our knowledge, many factors govern the way to broaden the cluster. As described in Fig. 7, coupled nodes V_6 and V_7 consist of the incipient cluster $C_0 = \{V_6, V_7\}$ (region marked by solid line eclipse), as long as we further enlarge the cluster, three nodes labeled as V_5, V_9 and V_{10} should take into consideration, note that the regions covered by candidate are drawn by dotted line eclipse. In order to distinguish the best one among those candidates, cluster expansion routine adopts three measurable scalars.

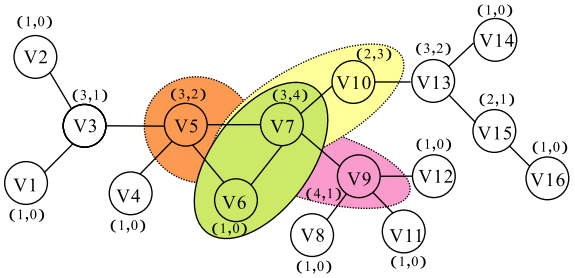


Fig. 7 An illustration for cluster expansion operation.

A. The incremental cardinality of dominated node set: stands for the increasing or decreasing number of dominated node set while putting a new candidate node into the original cluster. For example, consider the simple scenario of expanding node V_9 into the cluster C_0 (thus form a new cluster $C_1 = \{V_6, V_7, V_9\}$), then the dominated node set for C_0 (labeled as Dns_0) is $Dns_0 = \{V_5, V_9, V_{10}\}$, as well as the dominated node set $Dns_1 = \{V_5, V_8, V_{10}, V_{11}, V_{12}\}$ for C_1. The incremental cardinality of dominated node set is 2, which means it needs two more nodes to isolate C_1 from the residual graph in comparison with C_0. In the normal course of things we are inclined to choose the candidate with less or even negative growth of cardinality of dominated node set, due to the reason that the less nodes stayed in dominated node set, the more nodes left to cut the residual graph.

B. The branch number of candidate: a metric to measure the number of dispersed clusters after removing the candidate from the residual graph.

C. The path depth of candidate: represents the length starting from the furthest leaf to this candidate, which is an emblem of centrality. Taken altogether, the main principle behind the three metrics can be briefly stated.

Rule 1: the candidate with less incremental cardinality of dominated node set for the newly formed cluster is preferred to be the right node added into the increasing cluster;

Rule 2: the candidate with larger branch number and path depth is inclined to be the dominated node, so as to partition the residual graph into more pieces with high equalization among the divided clusters.

If we take the notations C, Dns_C, $N(u)$ as a newly formed cluster, the associated dominated node set for cluster C and the neighborhoods for node u in graph $G = (V, E)$ respectively. For a candidate node v which is about to join the cluster C, expressions such as Bn_v and Pd_v represent the abovementioned branch number of candidate and path depth of candidate in order. By definition, it is easy to deduce the following formula:

$$Dns_C = \bigcup_{\mu \in C} N(\mu) - C \tag{2}$$

Especially, as long as we put forward a candidate v into the newly cluster C, the set of $Dns_{C\cup\{v\}}$ can be quickly calculated by:

$$Dns_{C\cup\{v\}} = (N(v) - C) \cup Dns_C - \{v\} \tag{3}$$

Based on the established set of rules, three corresponding joint statistics are taken into consideration when we put a candidate node v into the current cluster C.

A. The incremental cardinality of dominated node set $\Delta Dns_{C\to(C\cup\{v\})}$: is denoted by $\Delta Dns_{C\to(C\cup\{v\})} = \left|Dns_{C\cup\{v\}}\right| - |Dns_C|$ in the evolution from current cluster C to a newly expanded cluster $C \cup \{v\}$, which can be easily calculated by:

$$\left|Dns_{C\cup\{v\}}\right| - |Dns_C| = |N(v) - C| - 1 \tag{4}$$

B. The sum of branch number in dominated node set Dns_C is defined by $Sum\_Bd_{Dns_C}$:

$$Sum\_Bn_{Dns_C} = \sum_{v\in Dns_C} Bn_v - |S_{cross}| + 1,$$
$$S_{cross} = \{\mu | \mu \in Dns_C \wedge Bn_u > 1\} \tag{5}$$

This statistic carries a distinct physical meaning, which reveals the number of connected components left after removing the dominated node set Dns_C from the residual graph. For example, if we remove the dominated node set $Dns_{C_0} = \{V_5, V_9, V_{10}\}$ for cluster C_0 in Fig. 7, the dispersed connected components are made up of seven collections, listed by $\{V_1, V_2, V_3\}$, $\{V_4\}$, $\{V_5, V_6, V_7\}$, $\{V_8\}$, $\{V_{11}\}$, $\{V_{12}\}$, $\{V_{13}, V_{14}, V_{15}, V_{16}\}$ respectively. However, the total number of scattered clusters can be easily computed according to Eq. (5):

$$Sum\_Bn_{Dns_{C_0}} = Bn_{V_5} + Bn_{V_9} + Bn_{V_{10}} -$$
$$|\{V_5, V_9, V_{10}\}| + 1 = 3 + 4 + 2 - 3 + 1 = 7$$

Thus we hold the principle that the larger the value of $Sum\_Bn_{Dns_C}$, the more connected components can be found.

C. The sum of mean difference of path depth in dominated node set Dns_C is described as $SMD\_Pd_{Dns_C}$:

$$SMD\_Pd_{Dns_C} = \sum_{v\in Dns_C} \left|Pd_v - \overline{Pd_{Dns_C}}\right|,$$
$$\overline{Pd_{Dns_C}} = \frac{\sum\limits_{v\in Dns_C} Pd_v}{|Dns_C|} \tag{6}$$

Actually, the direct motivation behind the statistic of $SMD\_Pd_{Dns_C}$ is that it is recommendable to keep the total square difference in connected component size as small as possible. As a result, we hope to isolate the graph from the center of topology throughout each round of partition. Therefore we take the principle that the smaller the $SMD\_Pd_{Dns_C}$, the more consistent in terms of cluster sizes.

With the help of the above-mentioned joint statistics, together with the proposed rules, we then give a detailed presentation how it works. To begin with, considering the suitcase in which a cluster $C_0 = \{V_6, V_7\}$ is about to expand to a new cluster $C_1 = \{V_6, V_7\} \cup \{v\}$, $v \in Dns_{C_0}$ in Fig. 7. Note that the corresponding reference quantities $\Delta Dns_{C_0\to(C_0\cup\{v\})}$, $Sum\_Bn_{Dns_{C_0\cup\{v\}}}$, $SMD\_Pd_{Dns_{C_0\cup\{v\}}}$ for each

Table 1 An illustration of parameter calculation

cluster C_0	Dns_{C_0}	Candidate v	$Dns_{C_0\cup\{v\}}$	f	$\Delta Dns_{C_0\to(C_0\cup\{v\})}$	$Sum\_Bn_{Dns_{C_0\cup\{v\}}}$	$SMD\_Pd_{Dns_{C_0\cup\{v\}}}$
$\{V_6, V_7\}$	$\{V_5, V_9, V_{10}\}$	V_5	$\{V_3, V_4, V_9, V_{10}\}$	7	1	8	7/2
		V_9	$\{V_5, V_8, V_{10}, V_{11}, V_{12}\}$	10	2	7	6
		V_{10}	$\{V_5, V_9, V_{13}\}$	8	0	8	4/3

candidate are given in Table 1.

From a statistical point of view, the candidate V_{10} carries the largest value of $Sum\_Bn_{Dns_{C_0\cup\{v\}}}$, as well as the least $\Delta Dns_{C\to(C\cup\{v\})}$ and $SMD\_Pd_{Dns_{C_0\cup\{v\}}}$ compared with the other two candidates V_5 and V_9. In other words, the dominated node set for newly formed cluster $C_1 = \{V_6, V_7, V_{10}\}$ segregates the graph into the most fermentations, together with the minimum square difference in connected components size. As a result, the solution of $S = \{V_5, V_9, V_{10}\}$ shares the minimum pair-wise connectivity $\binom{3}{2} + \binom{3}{2} + \binom{2}{2} = 7$ on the one hand, and takes up the least elements itself on the other. Without loss of generality, the performance measurement of candidate v (denoted by Score(v)) is assessed based on the following function.

$$Score(v) = \beta \times \frac{Sum\_Bn_{Dns_{C\cup\{v\}}}}{SMD\_Pd_{Dns_{C\cup\{v\}}}} - (1-\beta) \times \Delta Dns_{C\to(C\cup\{v\})} \quad (7)$$

Where β is the weighting coefficient between the ratio of $Sum\_Bn_{Dns_{C_0\cup\{v\}}}$ to $SMD\_Pd_{Dns_{C_0\cup\{v\}}}$ and the incremental value $\Delta Dns_{C\to(C\cup\{v\})}$, which is empirically set according to the practical application scenarios.

3.2.3 The Terminal Condition of Expansion Operation

According to the inference in Sect. 2, a theoretical optimal solution for CNP happens in the scenario where a subset collection of K nodes separates the graph $G = (V, E)$ into L equal-sized clusters, and the cardinality of each cluster is close or equal to $\left\lceil \frac{n-k}{L} \right\rceil$. In reality, even in the cases where an exact global optimum is known, the size of clusters usually appears not to be the same. As a result, the proposed cluster expansion operation loosens the equal-sized restriction, which allows the scale of newly formed cluster to fluctuating within a small range out of the theoretical optimum. For example, in Algorithm 3, cluster expansion adopts a fluctuate parameter D to keep the cardinality of cluster between $[L - D]$ and $[L + D]$. In other words, the expansion operation terminates once the size of cluster meets this predefined threshold interval $[L - D, L + D]$. In combination with the three coherent subroutines: starting a cluster, broadening the cluster and terminating the expansion, we can summarized the main point of cluster expansion as follows.

As depicted in Algorithm 4, the proposed cluster expansion always choose the candidate with the maximum score value in a greedy way (line 4-7), thus leads to a more efficient and fast search process. However, our cluster expansion operation also adopts a stochastic growth mode, which randomly pick up a candidate into the cluster instead.

Algorithm 4: Pseudo-code of ClusterExpansion

Input: current sub-graph G', initial new cluster C_0, weighting coefficient β; parameter best cluster size L, fluctuate value D.
Output: the newly formed cluster C, associated with its responding dominated node set Dns_C

```
1  C ← C_0;
2  while |C| < L + D do
3      TmpDns_C ← ∪_{μ∈C∪} N(μ) − C;
4      foreach node v in TmpDns_C do
5          calculate the Score(v) based on Eq. (7);
6      end
7      v_max ← arg max_{v∈TmpDns_C} Score(v);
8      C ← C ∪ {v'};
9      Dns_C ← ∪_{μ∈C} N(μ) − C;
10 end
11 Return the ultimately cluster C, Dns_C
```

The performance between them will be released in the comparative evaluation section.

3.3 Diversified Late Acceptance Local Search

3.3.1 Implementation of DLALS

A better initial solution does well in the rapid convergence of search, meanwhile the search strategy has an equally important effect on cutting down the heavily computation cost in the optimization process. Two main factors should be taken into consideration: one is the time cost, and the other is the ability of making good balance between the diversification and intensification in living candidates. With resorting to the idea of late acceptance hill climbing (LAHC) algorithm in [42], we adopt a diversified late acceptance local search (DLALS) [42] approach, which takes into account worsening, improving and sideways candidates with the aim to improve the diversity of proceeding solutions. Similar to LAHC, DLALS imports an array of size HL (also denoted by history length) that is used to stay a number of feasible solutions. Notice that LAHC employs an acceptance criterion in a greedy way which accepts the candidate solution only if its objectivity is better that that of the current solution. As a result, this greedy strategy naturally make the search fall into local optimum quickly. In contrast, DLALS adopts a new acceptance strategy to increase diversity of the values stored in the historical array, as well as a new replacement mechanism to further improve the evolution on the one hand, escape the local optimum in terms of bringing in a bit poor candidates on the other. Algorithm 5 writes out the full implementation of DLALS in our CEMCNP approach.

Using a memorable fitness array with a fixed length HL

Algorithm 5: Pseudo-code of DLALS

Input: an initial solution S ;
fitness array Ψ with historical length HL;
for $0 \le i \le HL, \Psi_i = f(\bullet)$;
maximum number of iterations $MaxIters$.
Output: the best solution S^* found so far.

1 $S^* \leftarrow S, f(S^*) \leftarrow f(S)$;
2 **for** $\forall i \in \{0, 1, 2, \ldots, HL-1\}$ **do**
3 $\quad \Psi_i \leftarrow f(S)$;
4 **end**
5 $f_{\max} \leftarrow f(S), num\_\max \leftarrow HL$;
6 $index\_iter \leftarrow 0, idle\_iters \leftarrow 0$;
7 **while** $idle\_iters < MaxIters$ **do**
8 $\quad \Psi_{pre} \leftarrow f(S)$;
9 $\quad S' \leftarrow ClusterBasedSwap(S)$;
10 $\quad f(S') \leftarrow IterativeDynamicEvaluation(S, S')$;
11 $\quad k \leftarrow (index\_iter) \bmod HL$;
12 $\quad$ **if** $f(S') = f(S)$ *or* $f(S') < \Psi_{max}$ **then**
13 $\quad\quad S \leftarrow S', f(S) \leftarrow f(S')$;
14 $\quad\quad$ **if** $f(S) < f(S^*)$ **then**
15 $\quad\quad\quad S^* \leftarrow S, f(S^*) \leftarrow f(S)$;
16 $\quad\quad\quad idle\_iters \leftarrow 0$;
17 $\quad\quad$ **else**
18 $\quad\quad\quad idle\_iters \leftarrow idle\_iters + 1$;
19 $\quad\quad$ **end**
20 $\quad$ **end**
21 $\quad$ **if** $f(S) > \Psi_k$ **then**
22 $\quad\quad \Psi_k \leftarrow f(S)$;
23 $\quad$ **end**
24 $\quad$ **else if** $f(S) < \Psi_k$ *and* $f(S) < \Psi_{pre}$ **then**
25 $\quad\quad$ **if** $\Psi_k = \Psi_{max}$ **then**
26 $\quad\quad\quad num\_max \leftarrow num\_max - 1$;
27 $\quad\quad$ **end**
28 $\quad\quad \Psi_k \leftarrow f(S)$;
29 $\quad\quad$ **if** $num\_max = 0$ **then**
30 $\quad\quad\quad$ re-compute Ψ_{max}, num_max;
31 $\quad\quad$ **end**
32 $\quad$ **end**
33 **end**
34 Return The best found solution S^*

to store the feasible solutions is the main characteristic of DLALS. In addition, two enhanced operations (denoted by acceptance and replacement strategy respectively) based on the fitness array come into being.

A. Acceptance Strategy: As long as a new candidate solution S' with its fitness value $f(S')$ comes in, it is necessary to compare it with the maximum fitness value in the fitness array, rather than the current value of Ψ_k. The candidate S' would be accepted if $f(S') = f(S)$ or $f(S') < \Psi_{max}$, the first condition allows accepting new candidate with fitness value equal to Ψ_{max} on the one hand, while the second condition holds the candidate with smaller fitness value than Ψ_{max}. In summary, accepting sideways or worsening moves increases the diversity level of the search (see line 12-20).

B. Replacement Strategy: The replace operation happens in two scenarios, one is when the fitness value $f(S)$ of the new candidate S is larger than Ψ_k, which still under the constraint of $f(S') < \Psi_{max}$, then the value in Ψ_k is replaced by $f(S)$ (see line 21-23). Such a replacement increases the probability of accepting more worsening moves, so as to further improve the solutions in future iterations. The other appears as long as $f(S)$ is smaller than Ψ_k, the replacement is done just when $f(S)$ is smaller than the previous value of Ψ_{pre} as well. Such a replacement avoids wiping off other large values in the fitness array prematurely (see line 24-28).

In comparison with LAHC liked approaches, DLALS tries to hold larger fitness values in the historical array when the search encounters non-improving moves on the one hand, and cautiously replace large fitness values with small ones when the search progressing into improving moves on the other. Once all the maximal elements in the fitness array are substituted, the variables of Ψ_{max} and num_{max} need to be recalculated (see line 29-31).

3.3.2 Cluster Based Swap and Iterative Dynamic Evaluation Mechanism

To generate a candidate solution, DLALS resorts to a cluster-based two-phase node exchange operation (denoted by "ClusterBasedSwap" at line 9) in connection with component-based neighborhood structure [34], which exchange a node $u \in S$ with a node $v \in (V \backslash S)$ from a large cluster C_g. Let $G(V \backslash S)$ be the residual graph separated by the current solution S, and $\{C_1, C_2, \ldots, C_T\}$ be the collection of connected clusters in $G(V \backslash S)$. Here a largest cluster C_g stands for the cluster whose size meets the requirement $|Cg| \ge$ L (L is a predefined threshold). In generally, the value of L is defined as the arithmetical mean of the number of nodes in the largest and smallest connected clusters in the residual graph $G(V \backslash S)$. To further reduce the size of the above-mentioned cluster-based neighborhood, our two-phase node exchange strategy is carried out in two steps: Firstly removes a node v standing in the large cluster C_g from the residual graph $G(V \backslash S)$. For the sake of efficiency, a node v with the highest score value (according to Eq. (7)) after running the contraction routine in cluster C_g is preferred. Secondly adds a node u to the residual graph $G(V \backslash (S \cup \{v\}))$. Usually we choose the node u that minimally deteriorates the objective function. This cluster based swap search can be examined in Algorithm 6. With the help of this greedy exchange strategy, the computational cost to traverse the candidate solution drops dramatically from $K(|V| - K)$ to $K(|Z|)$, where $Z = \cup_{|C_i| \ge L} C_i$.

Node-exchange operation is a basic move operator widely adopted by local-search heuristics, which is done in two steps: one is to find the best exchange sequence, and the other is to re-compute the objective function in terms of the selected pair of nodes. As depicted above, DLALS employs a two phases of paired "remove" and "add" routines, which first greedily removes the node with high score from the large clusters, and then adds the node with minimum growth of the objective function into the residual graph. Unfortunately, to re-evaluate the pair-wise connectivity from scratch is time-consuming. In order to further improve the computational efficiency, we take advantage of an iterative dynamic evaluation mechanism to recalculate the objective function in a cumulative manner. For convenience, we first list two key definitions in the following.

Algorithm 6: Pseudo-code of IterativeDynamicEvaluation

Input: a solution S, with cluster configuration $I = (\Delta, map)$; fitness array a solution S' after two-phase node exchange.
Output: objective value of $f(S')$, with new $I' = (\Delta', map')$.

1 obtain the move (u, v) based on the transforming from S to S;
2 **foreach** $m \in N(u)$ *and* $map(m) = map(u)$ **do**
3 $C \leftarrow DFS(m)$;
4 $map(w) \leftarrow C, \forall w \in C$;
5 $\Delta \leftarrow \Delta \cup C$;
6 $\Delta f \leftarrow \Delta f + (|C| \times (|C| - 1))/2$;
7 **end**
8 $\Delta \leftarrow \Delta \backslash map(u)$;
9 $map(u) \leftarrow \Phi, C' \leftarrow \Phi$;
10 **foreach** $n \in N(v)$ *and* $map(n) \subset \Delta$ **do**
11 $C' \leftarrow C' \cup map(n)$;
12 $\Delta \leftarrow \Delta \backslash map(n)$;
13 $\Delta f' \leftarrow \Delta f' + (|map(n)| \times (|map(n)| - 1))/2$;
14 **end**
15 $\Delta \leftarrow \Delta \cup C' \cup \{v\}$;
16 $map(w) \leftarrow C', \forall w \in C'$;
17 $\Delta f' = |C'| \times (|C'| - 1)/2 - \Delta f'$;
18 $I' : (\Delta', map') \leftarrow I : (\Delta, map)$;
19 $f(S') \leftarrow f(S) - \Delta f + \Delta f'$;
20 Return Objective value $f(S')$, with new $I' : (\Delta', map')$

Definition 6. *Given a candidate solution S, a move is a paired node exchange operation, denoted as* (u, v)*, where u is a node to be added to S from the residual graph, and v is the node to be removed from S.*

Definition 7. *The current cluster configuration is defined as a two-tuples* $I = (\Delta, map)$*, where* $\Delta = \cup_{i=1,2,\ldots,m} C_i$ *is a union of all the connected clusters in the residual graph* $G(V \backslash S)$*,* $map(v) : V \rightarrow C$ *is a function which maps a node v to the connected cluster* C_i *containing v.*

In light of the above concepts, our iterative dynamic evaluation mechanism can be explained in Algorithm 6. Considering the suitcase when moves a node u from the residual graph $G(V \backslash S)$ to S, it is necessary to check the partitioned clusters due to the elimination of node u, usually a depth-first traversal search (DFS) rooting at the adjacency list of node u is to succeed (see line 2-7). Once a node v is moved from S to the residual graph, the evaluation of increase in objective function is performed by traversing the neighbor list of node v to testify whether its removal leads to re-connect some existing clusters so as to form a large new cluster (see line 10-14). In this way, the incremental of the objective function from S to $S \cup \{u\} \backslash \{v\}$ is briefly calculated in terms of the number and size of new-formed or disappeared clusters according to the neighborhood structure, instead of the whole graph.

Furthermore, we notice that once a solution S changes to another solution S', the corresponding cluster configuration also shifts from $I : (\Delta, map)$ to $I' : (\Delta', map')$. In most cases, especially for the two node-change scenarios, the difference between Δ and Δ' is subtle, for example most of the clusters both in the two set are nearly the same. Since our evaluation mechanism totally relies on the cluster configuration, it is easy to obtain the change in the value of the objective function before and after (see line 6, 13, 17, 19). Once we begin a new round of search, the new-formed cluster configuration I' immediately become the starting point for the next round of search. This process continually circulates until the solution converges, that is the exact connotation what the "iterative" means.

3.4 Node Migration Based Perturbation

In line with the aforementioned studies in [36], [37], our node migration based perturbation generates a new starting candidate for the next round once the neighbourhood search relapses into a local optima. This node migration based perturbation operates as a destructive-constructive procedure, which first expands S with n_e nodes (here n_e is a indication of the perturbation strength) randomly taken from a large cluster, and then circularly removes a node from S with minimum pair-wise connectivity in the residual graph until the cardinality of S meets the threshold K. Specifically, we make use of a map list to memorize the first expanded n_e nodes, and verify whether the removed n_e nodes are the same. This works like a tabu table, which guarantees the differences between the nodes swapped in and out, so as to the keep the search moving forward. The perturbation method consists in fragmenting the largest connected components in the induced graph in order to reach more homogeneous components so as to reduce the number of node pairs still connected. Therefore, the perturbation method aims at providing a suitable diversification of the incumbent solution keeping into account these principles. The idea behind this is that, from a theoretical point of view, the minimization of pair-wise connectivity in the CNP results also in the maximization of the number of components while at the same time minimizing the variance in component cardinalities.

4. Comparative Evaluation

In this section, we will carry out computational experiment and comparison of the proposed algorithm CEMCNP with the state-of-the-art methods for CNP, as well as an alternative version CEMCNP* where the cluster expansion choosing the candidate with the maximum score value in a greedy way is replaced by randomly picking up a candidate into the cluster instead.

4.1 Benchmark Instances with Characteristics Observed by Contraction Mechanism

Our calculating examples are composed of two widely-used benchmark datasets: synthetic dataset and real-work dataset respectively. Notice that the scale of these examples ranges from 121 to 23133 nodes, the types of coverage models contain sparse graphs and dense graphs, as well as Hamiltonian graphs. In order to characterise the graph precisely, certain known global common features on these two datasets are provided: the average degree $\overline{d} = \frac{2\times|E|}{|V|}$, where $|V|$ and $|E|$

Table 2 The known and deduced characteristics by contraction

Graph	$\overline{d}$	n_C	*Coeff*	MaxPd	Dnodes	$\overline{Pd}$	$\overline{Bn}$	Jns
BA500	1.996	1	0.000	8	499	0.582	1.324	164
BA1000	1.998	1	0.000	10	999	0.585	1.322	324
BA2500	1.999	1	0.000	10	2499	0.580	1.329	825
BA5000	2.000	1	0.000	13	4999	0.61	1.334	1672
ER235	2.979	2	0.006	5	53	0.28	0.430	48
ER466	3.004	4	0.002	5	92	0.24	0.38	84
ER941	2.976	12	0.005	5	203	0.26	0.40	177
ER2344	2.986	14	0.001	4	483	0.22	0.38	419
FF250	4.112	1	0.276	5	79	0.344	0.556	83
FF500	3.312	1	0.247	5	226	0.492	0.796	195
FF1000	3.634	1	0.216	8	384	0.441	0.691	362
FF2000	3.413	1	0.245	5	761	0.427	0.682	725
WS250	9.968	1	0.473	1	0	0	0	0
WS500	5.984	1	0.420	1	0	0	0	0
WS1000	9.992	1	0.483	1	0	0	0	0
WS1500	5.997	1	0.480	1	0	0	0	0
Bovine	3.140	1	0.044	4	63	0.12	0.60	10
Circuit	3.167	1	0.052	3	25	0.13	0.20	25
E.Coli	2.780	1	0.024	4	188	0.21	0.74	57
USAir97	12.80	1	0.396	2	55	0.07	0.24	27
HumanDis	4.605	1	0.430	4	96	0.15	0.32	112
TrainsRome	2.133	1	0.018	55	60	5.87	0.47	79
EU_flights	53.08	2	0.402	3	184	0.10	0.25	109
openflights	14.96	371	0.331	4	350	0.07	0.26	125
Yeast	2.681	185	0.024	6	1090	0.41	0.78	527
Ham1000	3.996	1	0.002	1	0	0	0	0
Ham2000	3.996	1	0.000	1	0	0	0	0
Ham3000a	3.999	1	0.000	1	0	0	0	0
Ham3000b	3.998	1	0.001	1	0	0	0	0
Ham3000c	3.997	1	0.001	1	0	0	0	0
Ham3000d	3.995	1	0.000	1	0	0	0	0
Ham3000e	3.997	1	0.001	1	0	0	0	0
Ham4000	3.999	1	0.001	1	0	0	0	0
Ham5000	4.0	1	0.000	1	0	0	0	0
powergrid	2.669	1	0.103	8	1588	0.33	0.56	1229
Oclinks	14.57	4	0.057	4	398	0.12	0.33	220
facebook	43.69	1	0.519	2	75	0.002	0.021	11
Grqc	5.526	355	0.630	5	1099	0.17	0.32	813
Hepth	5.259	429	0.284	5	1968	0.169	0.324	1584
Hepph	19.73	278	0.659	3	1372	0.091	0.187	1168
astroph	21.10	290	0.318	4	1180	0.053	0.106	1107
condmat	8.078	567	0.264	5	2195	0.081	0.156	2096

indicate the number of nodes and edges respectively; the number of connected branches n_c; the value of the clustering coefficient *Coeff* which signals the tendency of nodes to cluster together. Moreover, a number of additional important quantities based on our contraction mechanism for solving the CNP are also provided in Table 2. Let the notations *MaxPd*, *Dnodes*, $\overline{Pd}$, $\overline{Bn}$ be the maximum path depth, nodes elapsed due to leaves contraction, the average path depth and branch number respectively in terms of the contraction mechanism. We also define the number of joint nodes as *Jns* whose removal separates the residual graph into at least two blocks.

As long as we pay attention to the Barabasi-Albert (BA) graphs, it is clear that all the graphs such as BA500, BA1000, BA2500 and BA5000 can eventually shrink down to a single node by means of our contraction mechanism, which is in accordance with the value of *Dnodes*. Meanwhile, the *Jns* value of these graphs accounts for nearly 33% of the total nodes. Once we greedily choose the joint node as a feasible candidate during each round of iteration, the ultimately formed initial solution is inclined to converge to a near optimum in a fast way. As a result, the number of node exchanges spending on the local search phase will drop down accordingly, this can be confirmed in the characteristic of "#exch" (the average number of node exchanges to achieve the optimum) in the following experiment.

In addition, with the help of contraction mechanism, we can reduce the size of the feasible candidate nodes from 25 to 2195 up to ten iterations (also known as the field of *MaxPd*) on the datasets other than the Barabasi-Albert (BA) graphs, our contraction mechanism plays an equally important role in decreasing the number of neighbourhood exchanges, so as to speed up the search process. However, in face of the datasets of Watts-Strogatz (WS) and Hamilton graphs, we notice that this contraction mechanism lose efficacy due to the strong connection between nodes, as well as the existence of a large number of loops. Fortunately, our follow-up cluster expansion method can well solve the dilemma, which can be proven next. Apart from that, the notation $\overline{Pd}$ gives the average sub-tree depths rooted at each node, whereas the symbol $\overline{Bn}$ expresses the average splitting ability of nodes, both of two joined together well explain the positive effects brought by the contraction mechanism.

Table 3 Parameter settings of the proposed CEMCNP algorithm

Parameter	Value	Description
t_{max}	3600 seconds	time limit for each round
MaxIters	1000	maximum number of idle iterations in DLALS
β	0.5	weighting coefficient
D	8% to 10% of the total node size	fluctuate threshold for the optimum cluster size
HL	5	historical length of fitness array in DLALS

4.2 Experimental Settings

The proposed heuristic was implemented in the *C++* programming language and compiled using Microsoft Visual Studio 2010 (10.0.30319.1 RTMRel). It was tested on a PC equipped with a 3.2 GHz Intel CoreTM i7−8700 Processor and 8.0 gigabytes of RAM operating under the Microsoft Windows 10 environment. In the following experiments, we test our CEMCNP algorithm, together with its variants over the well known benchmark instances discussed above with the parameters setting shown in Table 3.

In principle, The parameters t_{max}, *MaxIters* and HL are initialized as 3600 seconds, 1000 and 5 respectively, which are derived from the literatures [34], [35], [43]. Besides that, the weighting coefficient β is set to 0.5 so as to hold the balance among the path depth, branch number, as well as the dominated node set. As a basis of comparison, we conducted multiple testing for the correction of parameter D, and finally found that the optimum value for D lies within the range of 8% to 10% of the total node size.

4.3 Comparisons with State-of-the-Art Algorithms

As soon as we make a comprehensive survey over the existing solves for Critical Node Problems, a comparative study with respect to two of the recent state of art CNP algorithms, named MANCNP [35] and VPMS [34] are reported. To the best of our knowledge, the above-mentioned two methods basically cover most of the current optimal solutions. Without loss of generality, we also bring in the best-known results (denoted by KBV: Known Best Value) available in the literature which have been achieved by combining the optimums of some other excellent algorithms in [5], [13], [18], [19], [23], [37], [39].

Since the source code of MANCNP and VPMS is not available, we hold their best results appeared in the corresponding papers instead. Detailed comparative performance

Table 4 Comparisons of CEMCNP algorithm with state of the art algorithms

Graph	K	KBV	$MANCNP$	$VPMS$	$CEMCNP$	t_{avg}^{MANCNP}	$\#exch^{\text{MANCNP}}$	t_{avg}^{CEMCNP}	$\#exch^{\text{CEMCNP}}$
BA500	50	195	195	195	195	< 0.1	5.8×10^2	< 0.1	$\mathbf{3.6 \times 10^2}$
BA1000	75	558	558	558	558	0.3	6.4×10^3	0.2	$\mathbf{4.3 \times 10^3}$
BA2500	100	3704	3704	3704	3704	0.7	8.6×10^3	0.2	$\mathbf{5.4 \times 10^3}$
BA5000	150	10196	10196	10196	10196	6.5	3.5×10^4	1.8	$\mathbf{9.7 \times 10^3}$
ER235	50	295	295	295	295	7.1	2.0×10^5	19.8	2.4×10^5
ER466	80	1524	1524	1524	1524	28.5	9.5×10^5	98.4	1.2×10^6
ER941	140	5012	5012	5012	5012	458.5	1.2×10^7	694.6	1.9×10^8
ER2344	200	959500	902498	904113	912346	2284.8	1.5×10^7	3069.0	2.9×10^7
FF250	50	194	194	194	194	< 0.1	2.3×10^3	0.3	4.7×10^3
FF500	110	257	257	257	257	0.4	8.7×10^3	0.2	1.2×10^4
FF1000	150	1260	1260	1260	1260	84.9	2.6×10^5	120.6	3.9×10^5
FF2000	200	4545	4545	4545	4545	107.6	3.3×10^5	486.4	4.2×10^5
WS250	70	3101	3083	3083	3083	1140.5	6.1×10^7	623.1	$\mathbf{4.8 \times 10^5}$
WS500	125	2078	2072	2072	2072	179.3	2.4×10^6	105.8	$\mathbf{6.9 \times 10^5}$
WS1000	200	113638	109807	119444	109935	2675.0	1.8×10^7	1256.2	$\mathbf{5.9 \times 10^6}$
WS1500	265	13167	13098	13098	13098	1012.2	7.3×10^6	658.2	$\mathbf{6.2 \times 10^6}$
Bovine	3	268	268	268	268	< 0.1	6.8×10^1	< 0.1	1.7×10^2
Circuit	25	2099	2099	2099	2099	0.2	1.7×10^4	0.1	$\mathbf{4.6 \times 10^3}$
E.Coli	15	806	806	806	806	< 0.1	6.2×10^2	< 0.1	$\mathbf{3.4 \times 10^2}$
USAir97	33	4336	4336	4336	4336	756.5	2.5×10^6	856.4	3.2×10^6
HumanDis	52	1115	1115	1115	1115	0.6	1.7×10^4	0.5	$\mathbf{8.5 \times 10^3}$
TrainsRome	26	920	918	918	918	0.3	2.2×10^4	0.2	$\mathbf{9.2 \times 10^3}$
EU_flights	119	349927	348268	348268	348325	232.6	1.1×10^5	431.6	2.6×10^5
Openflights	186	28671	26842	26785	26796	2093.7	2.9×10^6	3165.8	5.7×10^6
Yeast	202	1414	1412	1412	1412	21.7	1.0×10^5	9.8	$\mathbf{6.2 \times 10^4}$
Ham1000	100	328817	306349	307117	307113	2137.5	2.5×10^7	1068.4	$\mathbf{7.8 \times 10^6}$
Ham2000	200	1309063	1243859	1247652	1245637	2861.9	1.2×10^7	1527.2	$\mathbf{5.4 \times 10^6}$
Ham3000a	300	3005183	2844393	2840941	2842695	3280.7	1.1×10^7	1204.3	$\mathbf{3.8 \times 10^6}$
Ham3000b	300	2993393	2841270	2839893	2840867	3252.9	1.2×10^7	2040.0	$\mathbf{5.1 \times 10^6}$
Ham3000c	300	2975213	2838429	2832073	**2831643**	3307.5	1.1×10^7	2159.3	$\mathbf{6.5 \times 10^6}$
Ham3000d	300	2988605	2831311	2830291	**2830284**	3250.9	1.1×10^7	3895.2	4.1×10^7
Ham3000e	300	3001078	2847909	2846731	**2846536**	3437.4	1.2×10^7	4035.6	6.7×10^7
Ham4000	400	5403572	5044357	5082521	5096437	2907.0	6.6×10^6	1563.7	$\mathbf{1.8 \times 10^6}$
Ham5000	500	8411789	7972525	8011565	8007638	3226.6	6.3×10^6	2024.1	$\mathbf{3.4 \times 10^6}$
Powergrid	494	16099	15862	15873	15906	1286.4	1.8×10^6	856.7	$\mathbf{9.7 \times 10^5}$
OClinks	190	614504	612303	611254	615467	584.6	4.5×10^5	267.8	$\mathbf{3.2 \times 10^5}$
facebook	404	420334	643162	691232	589763	2978.5	2.2×10^6	3526.9	4.4×10^7
Grqc	524	13736	13596	13603	13743	871.8	9.2×10^5	1025.0	2.5×10^6
Hepth	988	114382	106397	107939	115309	3442.0	3.1×10^6	4521.4	6.3×10^7
Hepph	1201	7336826	8628687	7883063	7556094	3376.3	1.4×10^6	4025.1	2.6×10^7
Astroph	1877	54517114	62068966	58322396	57895042	1911.4	3.9×10^5	3105.2	7.5×10^6
condmat	2313	2298596	9454361	6843993	7658643	1779.5	4.9×10^5	4203.5	3.1×10^7

between our method and the reference algorithms are shown in Table 4.

With resorting to the statistical magnitude defined in MANCNP [35], we also take advantage of the "#exch" metric to identify the average number of node exchanges to achieve the corresponding objection, as well as "t_{avg}" indicating the average time in seconds. From the macroscopic point of view, the following are some of our more notable findings:

- Firstly, to our delight, our CEMCNP algorithm running on the three benchmark instances Ham3000c, Ham3000d, and Ham3000e improved the best solutions throughout all the existing reference algorithms in the literatures. Notice that the time-consuming of the corresponding algorithm still remains competitive.
- Secondly, our CEMCNP algorithm holds the best results (equal to the performance of existing MANCNP and VPMS algorithms) over 24 out of total 42 graph models. Moreover, CEMCNP takes fewer number of node exchange operations (as well as less expenditure of time) to pursuit the optimal solutions in half of these 24 examples. This can be seen in both areas of "#exch" and "t_{avg}" in BA500, BA1000, BA2500, BA5000, WS250, WS500, WS1500, Circuit, E.Coli, HumanDis, TrainsRome and Yeast graphs.
- Finally, this CEMCNP algorithm running on 35 out of 42 graphs are superior to the outcomes of KBV. And on the other hand, the proposed CEMCNP method achieves highly competitive performance compared with the famous MANCNP and VPMS algorithms over seven instances (listed by facebook, hepph, astroph, condmat, Ham3000c, Ham3000d and Ham3000e).

4.4 Effectiveness of the Cluster Expansion-Based Construction

In order to study the benefit of the cluster expansion-based construction mechanism, we first conduct a comparison between CEMCNP with an alternative version CEMCNP* where the way to broaden the cluster is replaced by randomly picking up a candidate into the cluster. In other words, without considering the three correlative reference quantities (the incremental cardinality of dominated node set, the sum of branch number in dominated node set and

Table 5 Average time comparison between initial solution and final solution of CEMCNP algorithm

Graph	$T^{init}_{avg}(T_{avg})$	Graph	$T^{init}_{avg}(T_{avg})$	Graph	$T^{init}_{avg}(T_{avg})$	Graph	$T^{init}_{avg}(T_{avg})$	Graph	$T^{init}_{avg}(T_{avg})$
BA500	0.049(<0.1)	FF250	0.049(0.3)	Bovine	0.014(<0.1)	Ham1000	0.918(1068.4)	powergrid	3.564(856.7)
BA1000	0.036(0.2)	FF500	0.027(0.2)	Circuit	0.018(0.1)	Ham2000	3.306(1527.2)	Oclinks	1.513(267.8)
BA2500	0.029(0.2)	FF1000	0.660(120.6)	E.coli	0.012(<0.1)	Ham3000a	7.228(1204.3)	facebook	10.487(3526.9)
BA5000	0.257(1.8)	FF2000	2.237(486.4)	USAir97	0.093(856.4)	Ham3000b	9.111(2040.0)	grqc	8.474(1025.0)
ER235	0.084(19.8)	WS250	0.075(623.1)	humanDisea	0.075(0.5)	Ham3000c	7.277(2159.3)	hepth	21.644(4521.4)
ER466	0.264(98.4)	WS500	0.367(105.8)	Treni Roma	0.028(0.2)	Ham3000d	9.030(3895.2)	hepph	29.877(4025.1)
ER941	1.135(694.6)	WS1000	0.701(1256.2)	EU flights	0.711(431.6)	Ham3000e	9.145(4035.6)	astroph	8.974(3105.2)
ER2344	5.130(3069.0)	WS1500	2.726(658.2)	openflights	1.451(3165.8)	Ham4000	11.992(1563.7)	condmat	7.640(4203.5)
				yeast1	0.428(9.8)	Ham5000	24.648(2024.1)		

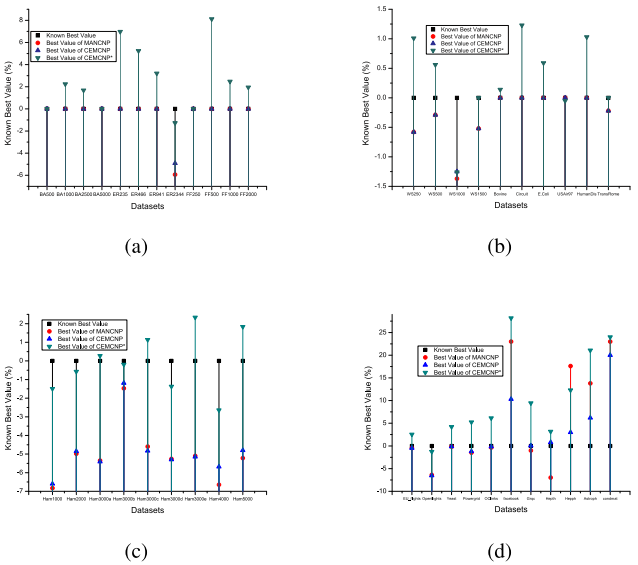


Fig. 8 Comparison between CEMCNP with an alternative version CEMCNP*.

the sum of mean difference of path depth in dominated node set), this CEMCNP* method just chooses the candidate node in a randomly way in terms of fewer cardinality of new-formed dominated node set.

Comparative results of CENCNP and CENCNP*, together with MANCNP in term of the best objective value are described in Fig. 8. Note that the X-axis represents the instance of datasets, for the sake of convenience, we hold the notation of "GAP to KBV (the gap of results to the known best values in percentage)" for the Y-axis, which is defined as $\frac{(f_{best}-KBV)\times 100}{KBV}$†, where f_{best} indicates the best objective obtained by MANCNP, CEMCNP and CENCNP* respectively.

When viewed from the curves between the performance of MANCNP and CENCNP, the two are quite closely to each other, which is also in-keeping with the findings in Table 4. What is noteworthy is that CEMCNP keeps the superior performance in comparison with the best solutions of MANCNP over 11 of total 42 models. For obvious understanding, a negative gap indicates an improved upper bound for the KBV, and a positive gap denotes a lower bound for the corresponding instance. Of all the 42 instances, CEMCNP gives better solution compared with that of CEMCNP*. While focused on the 31 test cases in front, as described from Fig. 8 (a) to 8 (c), there exists an average gap of 4% between the best solutions achieved by CEMCNP and CEMCNP*, the reason behind this is that the strategy to broaden the cluster in CEMCNP can take full advantage of the joint deduced statistics so as to dig deep into the potential characteristics of the residual graph, which has been ignored in the implementation of CEMCNP*.

In addition, another more fascinating benefit in our CEMCNP algorithm belongs to the accuracy of initial solution brought by the cluster expansion based construction. Distinguishing from traditional approaches who are inclined to establish the initial solution in a random way, this proposed CEMCNP walks along the direction towards the optimal solution at the start point. As a result, the original basic feasible solution usually stands much closed to the global optimum, even though the construction process may be a bit of time-consuming. Table 5 presents the average time cost spending on the construction of initial solutions. It is clear that this initial phase only takes up to 17% of the overall algorithm running time. In most cases, especially for the large instances, the ratio has been reduced to less than 0.5%.

As similar with the pattern of manifestation in Fig. 8, we display the 42 instances on the X axis, ranging from Fig. 9 (a) to 9 (d). For each test case, we conduct two experiments: one is to evaluate the performance between KBV and Initial Solution of CEMCNP, and the other concentrates upon the comparison between MANCNP and Initial Solution of CEMCNP. Without loss of generality, we adopt a relative quantification named "GAP to X" (X means KBV and MANCNP) to measure the gap between "X" and Initial Solution of CEMCNP for Y axis. It is observed that for the 42 benchmark instances, CEMCNP finds the feasible initial solutions which maintain a gap under the upper limit of 15% with KBV or Best Value of MANCNP over 33 test cases. Especially for the Hamilton and Barabasi-Albert (BA) graphs, cluster expansion based construction well holds the initial value much closer to the optimal solution. For example, under an upper bound of 5% in Fig. 9 (a) to 9 (c). In this way, high-quality initial solution helps to pursuit the global optimum and accelerates the convergence of local search. These observations further demonstrate the relevance of the cluster expansion based construction mechanism for enhancing CEMCNP algorithm.

†This statistic has been discussed in [34]–[36], a value of less than 14% or 15% is usually an acceptable range suggested in [35] and [36]. Whereas in [34], it didn't give the upper limit.

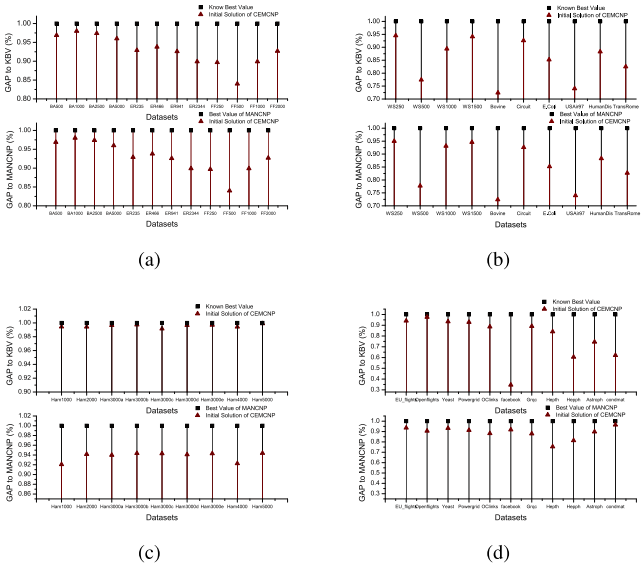


Fig. 9 High-quality initial solution by means of cluster expansion-based construction.

5. Conclusion

In this work, we proposed a new cluster expansion method for solving the critical node problem, our CEMCNP algorithm combines a leaves contraction mechanism, an incremental cluster expansion operation, a multi-start based diversified late acceptance local search, as well as an iterative dynamic evaluation strategy. Extensive evaluations show that CEMCNP running on 35 of total 42 benchmark instances are superior to the outcome of KBV, while holding the equivalent performance in comparison with the existing MANCNP and VPMS algorithms over 22 of total 42 graph models with fewer number of node exchange operations. In addition, this CEMCNP algorithm well improved the 3 previous best results out of the challenging instances. Analysis between CEMCNP and CEMCNP* reveals that our distinctive score weighting scheme can well speed up the local search process, so as to obtain better initial solutions.

Acknowledgments

This work is partially supported by the Key projects of science and technology research plan of Hubei Provincial Department of Education D20212201, Philosophy and social science research project of Hubei Provincial Department of Education 21Q213, and Hubei Provincial Natural Science Foundation of China No. 2020CFB306.

References

[1] S.P. Borgatti, "Identifying sets of key players in a network," IEMC '03 Proc. Managing Technologically Driven Organizations: The Human Side of Innovation and Change, pp.127–131, 2003.

[2] R. Cohen, S. Havlin, and D. ben-Avraham, "Efficient Immunization Strategies for Computer Networks and Populations," Physical Review Letters, vol.91, no.24, p.247901, 2003.

[3] A. Arulselvan, et al., "Managing network risk via critical node identification," Risk Management in Telecommunication Networks, 2007.

[4] A. Kumar, P.K. Gupta, and A. Srivastava, "A review of modern technologies for tackling COVID-19 pandemic," Diabetes & Metabolic Syndrome: Clinical Research & Reviews, vol.91, no.4, pp.569–573, 2020.

[5] A. Arulselvan, C.W. Commander, and L. Elefteriadou, "Detecting critical nodes in sparse graphs," Computers & Operations Research, vol.36, no.7, pp.2193–2200, 2008.

[6] M. Di Summa, A. Grosso, and M. Locatelli, "Complexity of the critical node problem over trees," Computers & Operations Research, vol.38, no.12, pp.1766–1774, 2011.

[7] Y. Atay, I. Koc, I. Babaoglu, and H. Kodaz, "Community detection from biological and social networks: A comparative analysis of metaheuristic algorithms," Applied Soft Computing, vol.50, pp.194–211, 2017.

[8] 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.

[9] B. Addis, M. Di Summa, and A. Grosso, "Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth," Discrete Appl. Math., vol.161, no.16–17, pp.2349–2360, 2013.

[10] M. Lalou and H. Kheddouci, "A polynomial-time algorithm for finding critical nodes in bipartite permutation graphs," Optim Lett, vol.13, pp.1345–1364, 2019.

[11] A. Aliabdi, A. Mohades, and M. Davoodi, "Constrained shortest path problems in bi-colored graphs: a label-setting approach," GeoInformatica, pp.1–19, 2019.

[12] B. Addis, M. Di Summa, and A. Grosso, "Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth," Discrete Appl. Math., vol.161, no.16, pp.2349–2360, 2013.

[13] J.L. Walteros, A. Veremyev, P.M. Pardalos, and E.L. Pasiliao, "Detecting critical node structures on graphs: A mathematical programming approach," Networks, vol.73, pp.48–88, 2013.

[14] D. Granata, G. Steeger, and S. Rebennack, "Network interdiction via a Critical Disruption Path: Branch-and-Price algorithms," Computers & Operations Research, vol.40, no.11, pp.2689–2702, 2013.

[15] T.N. Dinh, M.T. Thai, and H.T. Nguyen, "Bound and exact methods for assessing link vulnerability in complex network," Journal of Combinatorial Optimization, vol.28, no.1, pp.3–24, 2014.

[16] C. Areas, "An exact algorithm for the two-echelon capacitated vehicle routing problem," Operations Research, vol.61, no.2, pp.298–314, 2013.

[17] S.H. Yakhchali, "A path enumeration approach for the analysis of critical activities in fuzzy networks," Information Sciences, vol.204, no.20, pp.23–35, 2012.

[18] B. Addis, R. Aringhieri, A. Grosso, and P. Hosteins, "Hybrid constructive heuristics for the critical node problem," Annals of Operations Research, vol.238, no.1-2, pp.637–649, 2016.

[19] M. Ventresca and D. Aleman, "A Fast Greedy Algorithm for the Critical Node Detection Problem," Lecture Notes in Computer Science, pp.603–612, 2014.

[20] T. Ren, Z. Li, Y. Qi, Y. Zhang, S. Liu, Y. Xu, and T. Zhou, "Identifying vital nodes based on reverse greedy method," Scientific Reports, vol.10, no.1, 2020.

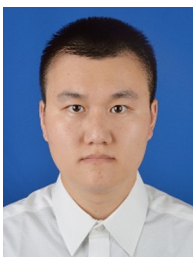
[21] D. Purevsuren and G. Cui, "Efficient heuristic algorithm for identifying critical nodes in planar networks," Computers & Operations Research, vol.106, pp.143–153, 2019.

[22] D. Purevsuren, et al., "Hybridization of GRASP with exterior path relinking for identifying critical nodes in graphs," IAENG International Journal of Computer Science, vol.44, no.2, pp.157–165, 2017.

[23] D. Purevsuren, G. Cui, and N.N.H. Win, "Heuristic algorithm for identifying critical nodes in graphs," Advances in Computer Science: an International Journal, vol.5, no.3, pp.1–4, 2016.

[24] H. Jhuge and J. Zhang, "Topological centrality and its e-science applications," Journal of the American Society for Information Science and Technology, vol.61, pp.1824–1841, 2010.
[25] Z. Wenping, W. Zhikang, and Y. Gui, "A novel algorithm for identifying critical nodes in networks based on local centrality," Journal of Computer Research and Development, vol.56, no.9, pp.1872–1880, 2019.
[26] W.E. Hart, J.E. Smith, and N. Krasnogor, "Recent Advances in Memetic Algorithms," Springer Berlin Heidelberg, 2005.
[27] C. Cotta, Handbook of Memetic Algorithms, Springer Berlin Heidelberg, 2012.
[28] Y.-H. Kim, Y. Yoon, and Z.W. Geem, "A comparison study of harmony search and genetic algorithm for the max-cut problem," Swarm and Evolutionary Computation, vol.44, 2018.
[29] P.R. De Oliveira Da Costa, S. Mauceri, P. Carroll, and F. Pallonetto, "A Genetic Algorithm for a Green Vehicle Routing Problem," Electronic Notes in Discrete Mathematics, vol.64, pp.65–74, 2017.
[30] O. Alp and E. Erkut, "An efficient genetic algorithm for the p-median problem," Annals of Operations Research, vol.122, pp.21–42, Sept. 2003.
[31] C. Moon, J. Kim, and G. Choi, "An efficient genetic algorithm for the traveling salesman problem with precedence constraints," European Journal of Operational Research, vol.140, no.3, pp.606–617, 2002.
[32] R. Aringhieri, A. Grosso, and P. Hosteins, "A Genetic Algorithm for a class of Critical Node Problems," Electronic Notes in Discrete Mathematics, vol.52, pp.359–366, 2016.
[33] 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.
[34] Y. Zhou, J.-K. Hao, Z.-H. Fu, Z. Wang, and X. Lai, "Variable Population Memetic Search: A Case Study on the Critical Node Problem," IEEE Trans. Evol. Comput., vol.25, no.1, pp.187–200, 2021.
[35] Y. Zhou, H. Jin-Kao, and G. Fred, "Memetic search for identifying critical nodes in sparse graphs," IEEE Trans. Cybern., pp.1–14, 2017.
[36] 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.
[37] Y. Zhou and J.-K. Hao, "A fast heuristic algorithm for the critical node problem," In Proc. Genetic and Evolutionary Computation Conference Companion (GECCO '17), Association for Computing Machinery, New York, NY, USA, pp.121–122, 2017.
[38] 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.
[39] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia, "VNS solutions for the Critical Node Problem," Electronic Notes in Discrete Mathematics, vol.47, pp.37–44, 2015.
[40] M. Ventresca and D. Aleman, "Efficiently identifying critical nodes in large complex networks," Computational Social Networks, vol.2.1, no.6, 2015.
[41] L. Chang, W. Li, and W. Zhang, "Computing a near-maximum independent set in linear time by reducing-peeling," In: Proc. SIGMOD 2017, pp.1181–1196, 2017.
[42] M. Namazi, C. Sanderson, M.A.H. Newton, M.M.A. Polash, and A. Sattar, "Diversified late acceptance search," in AI 2018: Advances in Artificial Intelligence - 31st Australasian Joint Conference, Wellington, New Zealand, Dec. 11-14, 2018, pp.299–311, 2018.
[43] Y. Zhou, Z. Wang, and Y. Jin, "Late acceptance-based heuristic algorithms for identifying critical nodes of weighted graphs," Knowledge-Based Systems, vol.211, 106562, 2021.

Zheng Wang received the Ph.D. degree in computer software theory from Huazhong University of Science and Technology in 2012. From 2012 to 2019, he was a senior engineer in Wuhan Maritime Communication Research Institute. Since 2019, he has been a lecturer with the school of Information and Communication Engineering, Hubei University of Economics. His research interests focused on the data gathering in wireless sensor networks, network optimization and intelligent optimization algorithms.



Yi Di received his Ph.D. degree in armament science and technology from Nanjing University of Science and Technology, China in 2018. From 2018 to 2019, he was a lecturer in Wuchang University of Technology. Since 2019, he has been a lecturer with the school of Information and Communication Engineering, Hubei University of Economics. His research interests include intelligent information processing and applications, Target recognition and tracking, intelligent system.