A Fast Tri-individual Memetic Search Approach for the Distance-based Critical Node Problem

Yangming Zhou[a,b], Gezi Wang[c], Jin-Kao Hao[d], Na Geng[a,b], Zhibin Jiang[a,b,*]

[a] *Data-Driven Management Decision Making Lab, Shanghai Jiao Tong University, Shanghai 200030, China*
[b] *Sino-US Global Logistics Institute, Antai College of Economics and Management, Shanghai Jiao Tong University, Shanghai 200030, China*
[c] *Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China*
[d] *Department of Computer Science, Université d'Angers, Angers 49045, France*

European Journal of Operational Research, Nov. 2022
https://doi.org/10.1016/j.ejor.2022.11.039

Abstract

The distance-based critical node problem involves identifying a subset of nodes in a graph such that the removal of these nodes leads to a residual graph with the minimum distance-based connectivity. Due to its NP-hard nature, solving this problem using exact algorithms has proved to be highly challenging. Moreover, existing heuristic algorithms are typically time-consuming. In this work, we introduce a fast tri-individual memetic search approach to solve the problem. The proposed approach maintains a small population of only three individuals during the whole search. At each generation, it sequentially executes an inherit-repair recombination operator to generate a promising offspring solution, a fast betweenness centrality-based late-acceptance search to find high-quality local optima, and a simple population updating strategy to maintain a healthy population. Extensive experiments on both real-world and synthetic benchmarks show our method significantly outperforms state-of-the-art algorithms. In particular, it can steadily find the known optimal solutions for all 22 real-world instances with known optima in only one minute, and new upper bounds on the re-

*Corresponding author
Email addresses: `yangming.zhou@sjtu.edu.cn` (Yangming Zhou), `y80200226@mail.ecust.edu.cn` (Gezi Wang), `jin-kao.hao@univ.angers.fr` (Jin-Kao Hao), `gengna@sjtu.edu.cn` (Na Geng), `zbjiang@sjtu.edu.cn` (Zhibin Jiang)

maining 22 large real-world instances. For 54 synthetic instances, it finds new upper bounds on 36 instances, and matches the previous best-known upper bounds on 15 other instances in ten minutes. Finally, we investigate the usefulness of each key algorithmic ingredient.

Keywords: Combinatorial optimization; Critical node problem; Distance-based connectivity; Metaheuristic; Memetic search.

---

1. Introduction

Graphs or networks arise in a variety of application areas due to their elegance and inherent ability to logically describe important interactive relations. In a number of situations, they are greatly affected by a small fraction of influential nodes whose removal would significantly degrade certain network functionality. To identify these important nodes (referred to as critical nodes), critical node detection problems (CNDPs) [9, 39, 20, 47, 50, 11] have been proposed and widely studied across a variety of domains. Take epidemic control as an example, the decision-maker is often interested in identifying a limited number of people to be vaccinated to reduce the overall transmissibility of disease virus [14, 19]. To monitor carbon dioxide emissions, the decision-maker attempts to find the critical paths and nodes that contribute strongly to carbon emissions embodied in transmission [44, 45].

The critical node problem (CNP) is a well-known representative CNDP [9]. It aims to minimize the pairwise connectivity of the residual graph via the removal of a limited subset of nodes from the original graph. The pairwise connectivity measure essentially counts the number of pairs of nodes connected by a path. As indicated in [32], the existence of a path between two nodes may not be sufficient, and the path length should also be considered as well (as short as possible). Indeed, considering the path length is especially relevant to model practical applications in supply chain networks, communication networks, and transportation networks, where a distance-based connectivity measure needs to be optimized. Correspondingly, distance-based critical node problems have been studied in recent years [42, 7, 22, 32].

The distance-based critical node problem (DCNP) considered in this work is a computationally challenging NP-hard problem [42]. Compared to CNP, few efforts have been devoted to developing efficient algorithms for DCNP in the literature. In addition, existing heuristic algorithms are time-consuming [4, 32]. To enrich the solution approaches of DCNP, we propose a fast tri-individual memetic search (FTMS) approach. Our contributions are two fold.

- The proposed fast tri-individual memetic search approach is characterized by the use of a small population of only three individuals. At each generation, it sequentially executes an inherit-repair recombination operator (to generate an offspring solution), a fast betweenness centrality-based late-acceptance search procedure (to perform local optimization), and a simple population updating strategy (to manage the population).
- We conduct extensive experiments on both real-world and synthetic benchmarks to evaluate FTMS. Comparative results show that FTMS competes well with state-of-the-art algorithms in terms of both solution quality and computation time. In particular, it can steadily and quickly find the optimal solutions for all 22 real-world instances with known optima in only one minute, and new upper bounds for 22 large real-world instances. It significantly outperforms state-of-the-art algorithms on 54 synthetic instances, finding new upper bounds on 36 instances and matching previous best-known upper bounds on 15 other instances in ten minutes.

The rest of this paper is organized as follows. After an introduction of DCNP in Section 2, we conduct a brief overview of previous studies on both CNP and DCNP in Section 3. Section 4 presents the proposed FTMS method for DCNP. Sections 5 and 6 are devoted to performance comparisons and experimental analyses, respectively. Finally, we conclude in Section 7.

2. Problem Description

Given an undirected graph $G = (V, E)$ with $n = |V|$ nodes (vertices) and $m = |E|$ edges, and a positive integer B (i.e., budgetary constraint), DCNP aims to remove a subset of nodes of cardinality at most B to minimize a distance-based connectivity metric, i.e.,

$$\min_{S \subseteq V} \quad \sum_{i,j \in V \setminus S, i<j} c_{ij} \cdot \psi(d(i,j)) \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in S} w_i \leq B \tag{2}$$

where $d(i, j)$ is the distance (i.e., the length of the shortest path) between nodes i and j in the residual graph $G[V \setminus S]$, $\psi : \mathbb{Z}_+ \cup \{+\infty\} \to \mathbb{R}$ represents a distance-based connectivity metric, $c_{ij} \in \mathbb{R}_+$ is the cost associated with the connection between the pair of nodes i and j, $w_i \in \mathbb{R}_+$ is the weight of node $i \in V$. For any two disconnected nodes i and j in $G[V \setminus S]$, we have $d(i,j) = +\infty$.

The distance-based connectivity measure $\psi(\cdot)$ of a graph is assumed to be a function of the actual pairwise distances between nodes in the remaining graph (e.g., global efficiency, Harary index, characteristic path length, Wiener index) [42] instead of simply knowing whether nodes are connected or not in the classic CNP. Several DCNPs were studied in the literature based on different distance-based connectivity measures, e.g., minimizing the total number of pairs of nodes connected by a hop distance of at most k; minimizing the Harary index, or equivalently, the efficiency of the graph; minimizing the sum of power functions of distances in the graph; maximizing the generalized Wiener index, and maximizing the shortest path between two given nodes in the graph.

In this work, we consider the general case of DCNP, i.e., minimizing the number of node pairs connected by a path of length at most k. Correspondingly, the distance-based pairwise connectivity metric is defined as follows:

$$\psi(d(i,j)) = \begin{cases} 1, & d(i,j) \leq k, \\ 0, & \text{otherwise}, \end{cases} \tag{3}$$

where $d(i,j)$ is the distance (i.e., the length of the shortest path) between nodes i and j in the residual graph $G[V \setminus S]$, and k is a given positive integer indicating a maximal distance limit. It means that only when the hop distance between a pair of nodes does not exceeds k, they can be considered as a truly connected node pair. Here, we focus on the unweighted version of DCNP, i.e., $c_{ij} = 1$, $\forall i,j \in V$ and $i < j$. Correspondingly, the objective function of DCNP can be further described as follows:

$$f(S) = \sum_{i,j \in V \setminus S, i<j} \psi(d(i,j)) \quad S \subseteq V \tag{4}$$

When $k = 1$, DCNP reduces to minimizing the number of remaining edges, which is also known as the maximum coverage problem (MCP) [10]. The goal of MCP is to find a subset $S \subseteq V$ with a fixed number of nodes and the number of edges covered by S is maximized, which is equivalent to minimize the number of remaining edges. When $k \geq n-1$, DCNP reduces to minimizing the total number of connected node pairs, which is the classic CNP [9, 48]. Both MCP and CNP are known to be NP-hard [9, 10].

3. Related Work

3.1. Previous Studies on CNP

Detecting critical nodes in complex network is a challenging combinatorial optimization problem. Due to its NP-hard nature, it has attracted much attention and considerable efforts have been dedicated to address this important problem [9, 16, 35, 38, 1, 47]. Existing algorithms can be divided into two categories: exact and heuristic algorithms.

Exact algorithms are known to guarantee the optimality of their obtained solution. Arulselvan et al. [9] presented the first integer programming (IP) model for CNP and solved it with the CPLEX solver. Di Summa et al. [16] further proposed two improved IP models, and solved them within the framework of the branch-and-cut method. They could find optimal solutions for small sparse instances with up to 150 nodes. Veremyev et al. [41] developed an improved compact IP model for CNP, which was able to optimally solve CNP instances with up to 1200 nodes. They further proposed a general IP model for CNP and its variants [40], which could provide optimal solutions only for medium instances up to an instance with 1612 nodes and 2106 edges in affordable computation time. Ventresca and Aleman [36] proposed a randomized rounding algorithm for the cardinality-constrained CNP, which achieved a $1/(1-\theta)$-approximation. These exact or approximation algorithms are practical only on instances of limited sizes, typically with no more 5000 nodes.

To handle large instances, heuristic algorithms are good alternatives to provide approximate solutions in a reasonable computation time. Two categories of heuristic algorithms have been proposed in the literature. The first category is single solution-based methods, which manipulate only a single candidate solution, such as constructive heuristics [29, 37, 1], simulated annealing [34], variable neighborhood search [6, 33] and iterated local search [6, 43]. Another category is population-based methods, which maintain a population of candidate solutions that are manipulated and evolved during the search process, such as population-based incremental learning [34], genetic algorithm [5], path-relinking [30], memetic algorithm [48] and its variant named variable population memetic algorithm [47]. Compared to single solution-based methods, population-based methods show better performance for CNP. To our knowledge, most of the best-known results available in the literature were achieved by memetic algorithms. However, they suffer from the need of managing a large population to maintain the diversity of the search, making them typically time-consuming, in particular when they are applied to large and very large instances with at least several thousands of nodes.

3.2. Previous Studies on DCNP

Like CNP, DCNP is also a computationally challenging NP-hard problem [42]. Compared to CNPs, much less efforts have been made on DCNP. Existing algorithms for DCNP include exact and heuristic algorithms.

To solve optimally DCNP, Verremyev et al. [42] proposed two exact algorithms. The first one relies on an integer programming formulation (IP) with additional preprocessing and modeling enhancements. However, for some types of distance-based metrics, the model size grows quickly with the number of nodes, thus the standard solver like CPLEX can only handle very small instances. For solving larger instance, the same authors developed a truncate-and-resolve algorithm (TRA) that iteratively solved a series of simplified IPs to obtain an optimal solution of the original problem. Experimental results showed TRA significantly outperforms (by at least an order of magnitude) the algorithm based on the initial IP formulation. TRA can optimally solve instances with about 500 nodes. With the help of variable fixing rules, sparse instances with 1500 nodes are also solved. However, TRA has two main drawbacks. On the one hand, it is sensitive to the diameter of the graph, and it would be time-consuming for instances with large diameter. On the other hand, it is sensitive to the length of the edges, and it performs badly on instances with the large length of the edges. To overcome the above issues, Hooshmand et al. [22] proposed a new IP model with simple structure and solved it by an efficient Bender decomposition algorithm, which is better than TRA in terms of computational time. This algorithm is not sensitive to the graph diameter nor length of the edges. Recently, Salemi and Buchanan [32] introduced two new IP models (i.e., thin and path-like). Comparative results between their models and the IP model of [42] showed that these three models were equivalent in strength when the objective coefficients were non-negative, but the thin model was the strongest generally. Although the thin model generally has an exponential number of constraints, it admits an efficient separation routine used in a branch-and-cut algorithm. Alozie et al. [3] presented a new path-based model (PBM) for DCNP, where separation heuristics and valid inequalities that exploit the structure of the problems were used to enhance the model. Besides general graphs, Aringhieri et al. [8] presented dynamic programming algorithms for DCNP over special graph classes such as paths, trees, and series-parallel graphs.

To solve DCNP approximately, Aringhieri et al. [7] performed a preliminary analysis and provided some suggestions on designing heuristic algorithms. Alozie et al. [4] presented a centrality-based heuristic (CBH) algorithm for DCNP, which combined a backbone-based crossover procedure to generate an offspring solution and a centrality-based neighborhood search to improve it. However, it is computationally expensive, which makes it unpractical for solving hard and large DCNP instances. To enrich the heuristic solution approaches for DCNP, we are devoted to proposing an efficient heuristic algorithm to solve DCNP.

4. Fast Tri-individual Memetic Search for DCNP

This section presents the fast tri-individual memetic search (FTMS) approach for DCNP. It starts with the solution representation and evaluation, followed by the overall framework and a detailed introduction of each algorithmic module.

4.1. Solution Representation and Evaluation

Given a graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges, and an integer budget B, any subset $S \subseteq V$ with no more than B nodes is a feasible candidate solution of DCNP, i.e., $|S| \leq B$. Let S ($|S| < B$) be a candidate solution, it is easy to verify that any solution $S' \leftarrow S \cup \{v\}$ with one more node $v \in V \setminus S$ is a feasible solution no worse than S, i.e., $f(S') \leq f(S)$. Since B is the largest integer for a solution to be feasible, we can safely consider only candidate solutions S with exactly B nodes. Therefore, we represent a solution of DCNP as $S = \{v_{S(1)}, v_{S(2)}, \ldots, v_{S(B)}\}$ where $S(i)$ ($1 \leq i \leq B$) is the index of i-th node in S. Therefore, the search space Ω contains all possible subsets $S \subseteq V$ such that $|S| = B$.

Given a candidate solution S, its objective function value $f(S)$ can be directly computed according to equation (4), which counts the total number of node pairs connected by a hop distance equal to or less than k in the residual graph $G[V \setminus S]$. This evaluation requires $O(n^3)$ time even by the fastest algorithm. To reduce its complexity, an alternative method is applied based on the k-depth breadth first search (BFS) tree built for each node in $O(mn)$. As indicated in [4], this time complexity can be significantly improved to $O(b^k)$ when the depth of BFS tree is limited to k, where b denotes the branching factor of the tree.

4.2. Overall Framework

FTMS is a hybrid evolutionary algorithm and follows the general memetic algorithm (MA) approach [28] that combines local search and population-based search. MAs have been successfully applied to solve many NP-hard problems, such as the graph coloring problem [26], the orienteering problem with hotel selection [18], the critical node problem [48], the maximum diversity problem [46], and the soft-clustered vehicle routing problem [49] and the multiple traveling repairman problem with profits [31]. The canonical memetic algorithms often maintain a relatively large population (at least 10 individuals) during the search. It is very time-consuming to manage such a large population. Recently, some effort have been made to develop efficient memetic algorithms based on small population.

Inspired by [27, 17], FTMS maintains a small population of only three individuals, whose rationale is explained in Section 4.3. The overall framework of FTMS is shown in Algorithm 1. FTMS consists of four main modules: population initialization, inherit-repair recombination (IRR), betweenness centrality-based late-acceptance search (BCLS) and population updating. Once the small initial population is built by the population initialization procedure, the algorithm enters a loop to form a number of generations. At each generation, a promising offspring solution is first generated by the IRR operator, and it is then improved to a high-quality local optimum by the fast BCLS procedure. Then the improved offspring solution is used to update the population. The process repeats until a given stopping condition is satisfied, e.g., the computation time reaches a given time limit or the number of generations exceeds an allowable maximal generation count.

4.3. Population Initialization

FTMS starts its search from a small initial population of only three distinct individuals, i.e., $P = \{S_1, S_2, S_3\}$. As centrality metrics evaluate the importance of a node for various definitions of importance [7], S_1, S_2 and S_3 are constructed based on three different centrality metrics:

- Degree centrality (χ_D) is a simple count of the total number of edges incident to a target node. It cannot recognize a difference between quantity and quality.
- Katz centrality (χ_K) [23] computes the centrality for a target node based on the centrality of its neighbors. It is a generalization of the eigenvector centrality.

Algorithm 1: Pseudo-code of FTMS

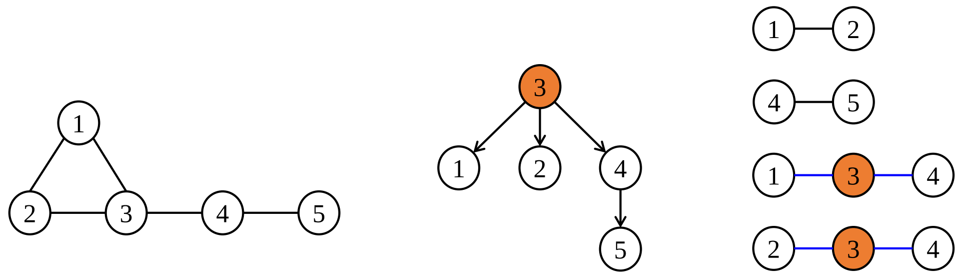
Input: A DCNP instance with the budgetary constraint B, maximal idle iteration count $\hat{\zeta}$, maximal idle generation count $\hat{\xi}$ and selection probability ϕ

Output: The best solution S^* found

```
// build an initial population
P = {S_1, S_2, S_3} ← PopulationInitialization()
S* ← arg min_{∀S_i∈P} f(S_i)
ξ ← 0 /*idle generation count*/
while Stopping condition is not met do
    // generate an offspring solution by an IRR operator
    S' ← IRR(P)
    // improve it by a local search
    S ← BCLS(S', ζ̂, φ)
    if f(S) < f(S*) then
        S* ← S
    // update the population
    P ← PopulationUpdating(P, S, ξ, ξ̂)
return The best solution S* found
```

- Betweenness centrality (χ_B) [12] is a measure of centrality in a graph based on shortest paths. It measures the fraction of shortest paths passing through a target node. The target node would have a high betweenness centrality if it appears in many shortest paths. As indicated in [12], all betweenness centrality values can be computed in $O(mn)$ under hop-based distances and in $O(mn+n^2 \log n)$ under edge-weighted distances.

These three centralities are popular measures, and they have been widely used to evaluate the importance of a node [24]. Degree centrality is the simplest centrality measure to compute, which can be quickly obtained in $O(n)$. For both Katz and betweenness centralities, we consider their special cases: k-Katz centrality and k-betweenness centrality. The former ranks nodes based on the size of the k-depth BFS tree rooted at each node, while the latter ranks nodes according to the number of their direct offspring summed over all generated k-depth BFS trees. Once the BFS trees have been built, k-Katz and k-betweenness centralities can be computed in $O(n)$ and $O(mn)$, respectively [13].



(a) An undirected graph (b) A tree rooted at node 3 (c) Short paths

Figure 1: An Illustrative Example of Degree Centrality, k-Katz Centrality and k-Betweenness Centrality)

To well understand the above-mentioned centrality metrics, we illustrate them with an example shown in Figure 1. Figure 1(a) presents an undirected graph G of five nodes, Figure 1(b) is a tree rooted at node 3, and Figure. 1(c) lists four shortest paths whose length does not exceed k and endpoint does not contain node 3. Given node 3 and the maximal hop distance $k = 2$, its degree centrality value equals the number of neighboring nodes, i.e., $\chi_D = 3$. The Katz centrality value of node 3 counts the number of nodes whose distance to node 3 does not exceed k, which is equivalent to reducing the size of the tree rooted at node 3 by one, i.e., $\chi_K = 4$. The betweenness centrality value of node 3 is the sum of the ratios of the number of shortest paths (marked as blue) passing through node 3 and corresponding distance does not exceed k to the total number of shortest paths whose distance does not exceed k, i.e., $\chi_B = 2$.

From an empty initial solution S_1, we first sort all nodes in G according to their degree centrality χ_D values in descending order, and then each node is iteratively added to S_1 with the probability δ $(0.5 \leq \delta < 1)$ until S_1 contains B nodes, i.e., $|S_1| = B$. Correspondingly, S_2 and S_3 can be constructed based on the k-Katz centrality and k-betweenness centrality, respectively.

4.4. Inherit-repair Recombination

At each generation, FTMS employs an inherit-repair recombination (IRR) operator to construct a promising offspring solution. IRR shares similar ideas with backbone-based crossovers used in [48, 4]. It works in two stages: random inheritance and greedy repair stages.

Definition 1. *(First Backbone). First backbone is a set of common nodes shared by all three solutions (i.e., S_1, S_2 and S_3), which can be formally defined as $\mathcal{R}_{st} = S_1 \cap S_2 \cap S_3$.*

Definition 2. *(Second Backbone). Second backbone consists of common nodes shared by only two solutions, which is formally defined as $\mathcal{R}_{nd} = ((S_1 \cap S_2) \cup (S_1 \cap S_3) \cup (S_2 \cap S_3)) \setminus \mathcal{R}_{st}$.*

Definition 3. *(Third Backbone). Third backbone is composed of nodes that belong to only one solution, which can be formally defined as $\mathcal{R}_{rd} = (S_1 \cup S_2 \cup S_3) \setminus (\mathcal{R}_{st} \cup \mathcal{R}_{nd})$.*

Given the three distinct solutions S_1, S_2 and S_3 in P, their nodes can be divided into three subsets denoted by the first backbone, second backbone and third backbone, as shown in Definitions 1-3. According to the definitions, three kinds of backbones can be identified from S_1, S_2 and S_3, and they form the set of all nodes in P, i.e., $S_1 \cup S_2 \cup S_3 = \mathcal{R}_{st} \cup \mathcal{R}_{nd} \cup \mathcal{R}_{rd}$. In addition, we define the set of nodes that does not belong to any solution of P as the non-backbone, i.e., $\mathcal{R}_{no} = V \setminus (S_1 \cup S_2 \cup S_3)$.

At the random inheritance stage, a partial solution S_0 is first obtained by directly inheriting all nodes of $\mathcal{R}_{st}$, i.e., $S_0 \leftarrow \mathcal{R}_{st}$. If the size of S_0 is less than floor$(\eta \cdot B)$, at each step, a node is randomly added to S_0 from a chosen backbone until $|S_0| \geq$ floor$(\eta \cdot B)$, where $0 < \eta < 1$ is a proportional factor. At each step, a backbone is selected based on two pre-defined factors θ $(0.5 \leq \theta < 1)$ and φ $(0.5 \leq \varphi < 1)$. Specifically, the second backbone $\mathcal{R}_{nd}$ is selected with the probability θ, and the probabilities to select the third backbone $\mathcal{R}_{rd}$ and non backbone $\mathcal{R}_{no}$ are $(1-\theta)\varphi$ and $(1-\theta)(1-\varphi)$, respectively.

At the greedy repair stage, the partial offspring solution S_0 is repaired by greedily adding a node from the remaining nodes (i.e., $V \setminus S_0$) until a feasible offspring solution is obtained. In particular, a node u is selected if it leads to the best improvement to $f(S)$, i.e., $u \leftarrow \arg\max\{f(S_0) - f(S_0 \cup \{v\})\}$, $\forall v \in V \setminus S_0$.

The IRR operator treats the common nodes shared by parent solutions as good elements, and aims to transmit the common nodes into the partial offspring solution at the random inherit stage. To ensure the quality of the offspring solution, the greedy repair stage of the IRR operator greedily repairs the partial solution to a feasible solution. The IRR operator can also be considered as an improved backbone-based crossover of [4]. It distinguishes itself from the backbone-based crossover by the node selection strategy. In particular, the backbone-based crossover adopts a hybrid node selection strategy by randomly or greedily selecting a node with a probability, while IRR employs a random and greedy node selection strategy in the inheritance and repair phases, respectively. Given the way that IRR recombines two parent solutions to obtain an offspring solution, this operator ensures simultaneously the role of search diversification and intensification of the FTMS algorithm.

4.5. Betweenness Centrality-based Late-acceptance Search

FTMS integrates a fast local optimization procedure called betweenness centrality-based late-acceptance search (BCLS) (see Algorithm 2). At the beginning, a node sequence $\mathcal{L}$ is constructed based on the k-betweenness centrality of nodes in the residual graph $G[V \setminus S]$, which takes time $O(mn + n \log n)$. Specifically, all nodes in $G[V \setminus S]$ are first sorted according to their k-betweenness centrality values in descending order. Then, the top-λ nodes are stored in the linked list $\mathcal{L}$, where $\lambda = B + \max\{5, \text{floor}(0.2 * B)\}$ like [4]. The remaining nodes are added into $\mathcal{L}$ in a random way. Once the node sequence $\mathcal{L}$ is obtained, BCLS enters a loop to iteratively perform node exchanges between S and $\mathcal{L}$ (lines 5-24). Each exchange operation is realized in two steps with the *add* and *remove* operators. The former aims to add a node u of $\mathcal{L}$ into S, while the latter tries to greedily remove a node v from S. For each head node u of $\mathcal{L}$, we add it into S with a pre-defined selection probability ϕ ($0.5 \leq \phi < 1$). Once node u is added into S, we greedily remove a node v from S in $O(Bb^k)$, which minimally deteriorates the objective function $f(S)$. With probability $1 - \phi$, node u is re-inserted into an intermediate position (i.e., $\mathcal{L}.begin() + 5$) of $\mathcal{L}$ (line 24). The whole computational complexity of BCLS is $O(mn + n \log n + \tilde{\zeta} Bb^k)$, where $\tilde{\zeta}$ is the number of iterations (i.e., iteration count).

BCLS distinguishes itself from the centrality-based neighborhood search (CNS) [4] in two aspects. On the one hand, BCLS always selects a node to insert into S according to the k-betweenness centrality only. On the other hand, BCLS adopts a linked list data structure to represent a node sequence (i.e., $\mathcal{L}$) obtained based on the k-betweenness centrality, which delays the acceptance of a node. These features ease the neighborhood operations during the search.

4.6. Population Updating

To maintain the diversity of the three individual population, a simple pool updating strategy is employed in FTMS. Once an improved offspring

Algorithm 2: Pseudo-code of BCLS

Input: A solution S, maximal idle iteration count $\hat{\zeta}$ and selection probability ϕ
Output: A improved solution S^*

$S^* \leftarrow S$
// construct a node sequence $\mathcal{L}$
Initialize $\mathcal{L}$ based on the k-betweenness centrality
$\zeta \leftarrow 0$ /*idle iteration count*/
while $\zeta < \hat{\zeta}$ **do**
    // record the head node of $\mathcal{L}$
    $u \leftarrow \mathcal{L}.front$
    // remove the head node from $\mathcal{L}$
    $\mathcal{L}.pop(u)$
    Generate a random decimal $r \in [0, 1]$
    if $r < \phi$ **then**
        $S \leftarrow S \cup \{u\}$
        $v \leftarrow \arg\min_{w \in S}\{f(S \setminus \{w\}) - f(S)\}$
        $S \leftarrow S \setminus \{v\}$
        // add node v to the end of $\mathcal{L}$
        $\mathcal{L}.push(v)$
        if $f(S) < f(S^*)$ **then**
            $S^* \leftarrow S$
            $\zeta \leftarrow 0$
        else
            $\zeta \leftarrow \zeta + 1$
    else
        // insert node u into a new position of $\mathcal{L}$
        $\mathcal{L}.insert(\mathcal{L}.begin() + 5, u)$
return An improved solution S^*

solution is obtained, it is accepted or discarded according to the population updating strategy, shown in Algorithm 3.

Given an improved offspring solution S and the current population P, if S is the same as an individual of P, we discard it. Otherwise, S replaces the worst individual S_w of P under two conditions, i.e., either S is better than the worst individual S_w in P or the idle update count ξ reaches the

Algorithm 3: Pseudo-code of Population Updating

Input: Population P, an improved offspring S, idle update count ξ and maximal idle update count $\hat{\xi}$

Output: An updated population P

if *S does not exist in P* **then**
    // identify the worst individual
    $S_w \leftarrow \arg\max_{\forall S_i \in P} f(S_i)$
    // replace the worst one
    if $f(S) < f(S_w)$ *or* $\xi > \hat{\xi}$ **then**
        $P \leftarrow P \cup \{S\} \setminus \{S_w\}$
        $\xi \leftarrow 0$
    else
        $\xi \leftarrow \xi + 1$
else
    $\xi \leftarrow \xi + 1$
return An updated population P

allowable maximal idle update count $\hat{\xi}$.

4.7. Computational Complexity of FTMS

To analyze the computational complexity of FTMS, we consider four main modules of Algorithm 1. FTMS starts the search from a small population P of three distinct solutions generated by the population initialization procedure of the time complexity of $O(mn + n \log n)$, where $n = |V|$ and $m = |E|$ present the total number of nodes and edges in G, respectively.

For each subsequent generation, FTMS sequentially executes the IRR operator, BCLS procedure, and population updating strategy. An offspring solution can be obtained in $O(B\eta + b^k + B(1-\eta)nb^k)$ by the IRR operator, where $0 < \eta < 1$ is a proportional factor to control the size of the partial solution at the random inheritance stage of IRR, k and b denote the depth and branching factor of the BFS tree, respectively. Then, the BCLS procedure is applied to improve S by performing neighborhood search around it, which can be finished in $O(mn + n \log n + \tilde{\zeta} Bb^k)$, where $\tilde{\zeta}$ is the iteration count of BCLS. Once an improved offspring solution is found, the population is updated with it in $O(B)$ time. Therefore, the total time complexity of FTMS is $O(B\eta + b^k + B(1-\eta)nb^k + mn + n \log n + \tilde{\zeta} Bb^k)$ for each generation.

5. Computational Experiments

5.1. Benchmark Instances

Our computational experiments were conducted on benchmark instances used in recent studies [42, 3, 4]. In addition, 11 large real-world CNP instances [5] are first adapted for DCNP. These instances consists of two categories: real-world and synthetic benchmarks.

- **Real-world benchmark** are divided into two categories: R1 and R2. The former is composed of 11 real-world networks selected from the Pajek and UCINET[1] datasets. The latter consists of 11 large instances selected from CNP instances[2].

- **Synthetic benchmark** are further classified into two groups: S1 and S2. The former contains 21 instances (i.e., Barabasi-Albert, Erdos-Renyi, and uniform random graphs) generated by using NetworkX random graph generators [21], while the latter includes two original CNP instances (i.e., FF250 and WS250a) and 10 new instances. These new instances share the same sizes and orders as the original benchmark instances in [34].

The main characteristics of both real-world and synthetic instances are presented in Table 1. Following [4], we solve each instance of the synthetic benchmark S2 with a given B value, while for the remaining instances, we solve each instance under two different budgetary constraints, i.e., $B = \text{floor}(0.05n)$ and $B = \text{floor}(0.1n)$.

5.2. Experimental Settings

Our algorithms[3] are implemented in the C++ programming language and complied with gcc 8.1.0 and the flag "-O0". All experiments are carried out on a computer equipped with an AMD Ryzen 7 5800U processor with 1.9 GHz and 16 GB RAM operating under the Windows 10 system. In following experiments, we set the largest hop distance as three, i.e., $k = 3$. It is an appropriate hop distance for most of benchmark instances, as suggested in [42, 34].

[1] http://vlado.fmf.uni-lj.si/pub/networks/data/

[2] http://individual.utoronto.ca/mventresca/cnd.html

[3] Our programs and results are available at https://github.com/YangmingZhou/DCNPs

Table 1: Characteristics of Benchmark Instances

Real-world Instances				Synthetic Instances			
Name	n	m	Density(%)	Name	n	m	Density(%)
Hi_tech	33	91	0.172	ba1	100	475	0.096
Karate	34	78	0.139	ba2	100	900	0.182
Mexican	35	117	0.197	er1(3)	80	474	0.150
Sawmill	36	62	0.098	er1(6)	80	476	0.151
Chesapeake	39	170	0.229	er1(9)	80	447	0.141
Dolphins	62	159	0.084	er2(3)	200	982	0.049
Lesmiserable	77	254	0.087	er2(6)	200	1018	0.051
Santafe	118	200	0.029	er2(9)	200	1017	0.051
Sanjuansur	75	155	0.056	gnm1	200	1000	0.050
LindenStrasse	232	303	0.011	gnm2	300	1500	0.033
USAir97	332	2126	0.039	gnm3	300	2000	0.045
yeast	2018	2705	0.001	FF250	250	514	0.017
Ham1000	1000	1998	0.004	BA250	250	1225	0.039
Ham2000	2000	3996	0.002	BA500	500	2475	0.020
Ham3000a	3000	5999	0.001	BA1000	1000	4975	0.010
Ham3000b	3000	5997	0.001	ER250	250	1190	0.038
Ham3000c	3000	5996	0.001	ER500	500	2570	0.021
Ham3000d	3000	5993	0.001	ER1000	1000	999	0.002
Ham3000e	3000	5996	0.001	WS250a	250	1246	0.040
Ham4000	4000	7997	0.001	WS250b	250	1250	0.040
Ham5000	5000	6594	0.001	WS500	500	2500	0.020
powergrid	4941	6594	0.001	GNM250	500	1250	0.010
				GNM500	1000	2500	0.005

5.3. Parameter Tuning

Our experimental results are obtained by executing FTMS algorithm with the parameter settings provided in Table 2. To determine the suitable parameter values, we resort to the well-known automatic parameter configuration tool called IRACE [25].

Table 2: Parameter Settings of FTMS

Parameter	Description	Candidate Values	Final Value	Section
δ	Probability to Add a Node to an Initial Solution	{0.5,0.6,0.7,0.8,0.9}	0.8	4.3
η	Proportion Factor	{0.1,0.3,0.5,0.7,0.9}	0.9	4.4
θ	First Factor to Select a Backbone	{0.5,0.6,0.7,0.8,0.9}	0.5	4.4
φ	Second Factor to Select a Backbone	{0.5,0.6,0.7,0.8,0.9}	0.9	4.4
$\hat{\zeta}$	Maximal Idle Iteration Count	{50,100,150,200,250}	150	4.5
ϕ	Probability to Add Node u to S	{0.5,0.6,0.7,0.8,0.9}	0.8	4.5
$\hat{\xi}$	Maximal Idle Generation Count	{3,5,7,9,11}	5	4.6

For each parameter, it requires some candidate values as input, as shown in the column "Candidate Values" of Table 2. The best parameter configuration is provided in the column of "Final Value". During the parameter tuning, we run IRACE with the default settings, and set the total time budget as 2000 executions. The whole experiments are conducted on eight representative instances with different sizes that are selected from both real-world and synthetic benchmarks, i.e., USAir97, ba2(6), er1(3), er2(9), gnm2(3), gnm3(9), ER250, and WS500. For each instance, it is solved with the time limit $\hat{t} = 60$ seconds.

5.4. Compared with State-of-the-art Algorithms

To evaluate the performance of FTMS, we experimentally compare it with three state-of-the-art (SOTA) algorithms, i.e., the exact algorithm called path-based model (PBM) [3], the centrality-based heuristic (CBH) algorithm [4] and its multi-start version, i.e., multi-start centrality-based heuristic (MCBH). Since the source codes of CBH and MCBH are not available to us, we have re-implemented them in C++. To guarantee a fair comparison, we execute each algorithm on the same computational platform with the following stopping conditions. For each benchmark instance, we run each algorithm ten times with the time limit. Since there is no an available time limit in the literature, we determine the time limit based on the preliminary results. Specifically, we execute CBH with the time limit $\hat{t} = 3600$ seconds at each run. For each instance of real-world benchmark R1 and synthetic benchmark S1, we run both MCBH and FTMS under the time limit $\hat{t} = 60$ seconds for each run. For each instance of real-world benchmark R2 and synthetic benchmark S2, we adopt the time limit $\hat{t} = 600$ seconds. Note that the results of PBM are directly obtained from [3], which were achieved under the time limit $\hat{t} = 3600$ seconds.

To analyze the comparative results, we resort to the well-known Wilcoxon signed rank test [15] to check the significant difference on each comparison indicator between two compared algorithms. At a significance level of 0.05, algorithm X is significantly better than algorithm Y if its p-value is no more than 0.05.

5.4.1. Results on Real-world Benchmark Instances

Comparative results between FTMS and the state-of-the-art algorithms on real-world benchmark R1 with $B = \text{floor}(0.05n)$ and $B = \text{floor}(0.10n)$ are summarized in Table 3 and Table 4, respectively. In these tables, column 1 presents the instance name (Instance), column 2 indicates the optimal solutions (f^*) of PBM, reported in [3]. Columns 3-5 show the results of CBH, including the best result ($\hat{f}$) found during 10 runs, the average result ($\bar{f}$) and average computation time ($\bar{t}$) in seconds needed to reach the best result at each run. Correspondingly, columns 6-8 and 9-11 present the results of MCBH and FTMS, respectively. In addition, we count the number of instances in which FTMS finds better (#Wins), equal (#Ties) and worse (#Loses) results compared to each reference algorithm. The last row provides the p-values of the Wilcoxon signed ranks test.

Table 3: Comparison of FTMS and SOTA Algorithms on Real-world Benchmark R1 with $B = \text{floor}(0.05n)$

	PBM	CBH			MCBH*			**FTMS**		
Instance	f^*	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
Hi_tech	**397***	**397**	**397.0**	0.2	**397**	**397.0**	0.1	**397**	**397.0**	0.1
Karate	**324***	**324**	**324.0**	0.2	**324**	**324.0**	0.1	**324**	**324.0**	0.1
Mexican	**527***	**527**	**527.0**	0.3	**527**	**527.0**	0.1	**527**	**527.0**	0.1
Sawmill	**215***	**215**	**215.0**	0.2	**215**	**215.0**	0.1	**215**	**215.0**	0.1
Chesapeake	**696***	**696**	**696.0**	0.3	**696**	**696.0**	0.1	**696**	**696.0**	0.1
Dolphins	**820***	**820**	**820.0**	1.3	**820**	**820.0**	0.2	**820**	**820.0**	0.1
Lesmiserable	**930***	**930**	**930.0**	1.9	**930**	**930.0**	0.1	**930**	**930.0**	0.1
Santafe	**305***	**305**	**305.0**	1.2	**305**	**305.0**	0.1	**305**	**305.0**	0.1
Sanjuansur	**803***	**803**	**803.0**	0.9	**803**	**803.0**	0.1	**803**	**803.0**	0.1
LindenStrasse	**1054***	**1054**	1057.8	5.9	**1054**	**1054.0**	1.2	**1054**	**1054.0**	0.1
USAir97	**10623***	**10623**	10697.2	269.7	**10623**	**10623.0**	23.2	**10623**	**10623.0**	5.4
#Wins|Ties|Loses	0|11|0	0|11|0	2|9|0	11|0|0	0|11|0	0|11|0	3|8|0	–	–	–
#p-value	1.0e0	1.0e0	5.0e-1	–	1.0e0	1.0e0	–	–	–	–

* Optimal results obtained by the exact algorithm reported in [3] within 3600 seconds.
⋆ MCBH denotes a multi-start version of CBH.

From Table 3, we observe that all 11 real-world instances can be optimally solved by almost all three heuristic algorithms, i.e., CBH, MCBH and FTMS. In particular, FTMS can steadily find the optimal solutions for all 11 instances in the shortest time. For USAir97 instance, FTMS obtains the optimal solution in 5.4 seconds, against 269.7 and 23.2 seconds for CBH and MCBH, respectively. Compared to MCBH, FTMS finds better results on 3 instances and the same results on the remaining instances in terms of $\bar{t}$. FTMS achieves better results than CBH on all 11 instances in terms of $\bar{t}$. At a significance level of 0.05, there is no significant performance difference among CBH, MCBH and FTMS in terms of both $\hat{f}$ and $\bar{f}$ on real-world benchmark R1.

We can obtain similar observations from Table 4. FTMS is able to steadily find the optimal solutions on all real-world instances except USAir97. For USAir97 instance, all three heuristic algorithms can find the optimal solution, but FTMS also obtains the smallest $\bar{f}$ value. Although all 11 real-world instances can be optimally solved by each heuristic algorithm, FTMS can reach them in the shortest computation time with the highest success rate. At a significance level of 0.05, FTMS significantly outperforms CBH in terms of both $\bar{f}$ and $\bar{t}$, and it is also significantly better than MCBH in terms of $\bar{t}$. Results from both Tables 3 and 4 show FTMS competes favorably with SOTA algorithms in terms of both solution quality and computation time.

To further evaluate the performance of FTMS, we experimentally compare FTMS with SOTA on real-world benchmark R2. Detailed comparative

Table 4: Comparison of FTMS and SOTA Algorithms on Real-world Benchmark R1 with $B = \text{floor}(0.1n)$

	PBM	CBH			MCBH*			**FTMS**		
Instance	f^*	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
Hi_tech	**293***	**293**	294.8	0.5	**293**	**293.0**	0.9	**293**	**293.0**	0.1
Karate	**147***	**147**	150.9	0.4	**147**	**147.0**	0.6	**147**	**147.0**	0.1
Mexican	**358***	**358**	**358.0**	0.6	**358**	**358.0**	0.1	**358**	**358.0**	0.1
Sawmill	**135***	**135**	**135.0**	0.2	**135**	**135.0**	0.1	**135**	**135.0**	0.1
Chesapeake	**512***	**512**	515.2	1.1	**512**	**512.0**	0.1	**512**	**512.0**	0.1
Dolphins	**583***	**583**	591.7	2.5	**583**	**583.0**	1.3	**583**	**583.0**	0.1
Lesmiserable	**323***	**323**	**323.0**	2.9	**323**	**323.0**	0.1	**323**	**323.0**	0.1
Santafe	**116***	**116**	**116.0**	2.8	**116**	**116.0**	0.1	**116**	**116.0**	0.1
Sanjuansur	**457***	**457**	457.2	2.0	**457**	**457.0**	1.2	**457**	**457.0**	0.1
LindenStrasse	**429***	**429**	431.7	12.7	**429**	**429.0**	5.2	**429**	**429.0**	0.1
USAir97	**3100***	**3100**	3219.1	423.6	**3100**	3180.1	32.2	**3100**	**3110.1**	6.7
#Wins|Ties|Loses	0|11|0	0|11|0	7|4|0	11|0|0	0|11|0	1|10|0	6|5|0	–	–	–
#p-value	1.0e0	1.0e0	**1.6e-2**	–	1.0e0	1.0e0	–	–	–	–

* Optimal results obtained by exact algorithm reported in [3] within 3600 seconds.
⋆ MCBH denotes a multi-start version of CBH.

results between FTMS and SOTA algorithms on real-world benchmark R2 with $B = \text{floor}(0.05n)$ and $B = \text{floor}(0.10n)$ are summarized in Tables 5 and 6, respectively. From them, we observe that FTMS also performs very well on real-world benchmark R2. In particular, FTMS finds the best results in terms of $\hat{f}$ for all instances of R2. In terms of $\bar{f}$, FTMS finds the best results for all instances except for Hamilton1000. At a significance level of 0.05, FTMS significantly outperforms CBH and MCBH in terms of both $\hat{f}$ and $\bar{f}$.

5.4.2. Results on Synthetic Benchmark Instances

To further evaluate the performance of FTMS, we show computational results on the two sets of synthetic instances. Comparative results between FTMS and SOTA algorithms on synthetic benchmark S1 are summarized in Tables 7-8 with the same information as in the previous section.

As we can see from Table 7, FTMS reaches an excellent performance. It finds new upper bounds for 12 instances, and matches the previous upper bounds for 6 out of the remaining 9 instances. At a significance level of 0.05, we observe that FTMS significantly outperforms CBH in terms of all performance indicators (i.e. $\hat{f}$, $\bar{f}$ and $\bar{f}$). FTMS is also significantly better than MCBH in terms of both $\bar{f}$ and $\bar{t}$. While for $\hat{f}$, there is no significant difference between FTMS and MCBH.

From Table 8, we observe that FTMS also demonstrates a good performance. It finds new upper bounds on 15 instances, and matches the previous

Table 5: Comparison of FTMS and SOTA Algorithms on Real-world Benchmark R2 with $B = \text{floor}(0.05n)$

	CBH			MCBH*			**FTMS**		
Instance	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
yeast1	4231	4262.4	44.5	4222	4263.0	44.2	**4197**	**4197.0**	35.3
Hamilton1000	18185	18206.0	9.8	**18181**	**18189.4**	281.8	**18181**	18204.0	13.3
Hamilton2000	37086	37151.9	77.7	37050	37138.2	81.3	**37021**	**37040.3**	102.5
Hamilton3000a	55449	55499.9	266.3	55443	55506.6	268.5	**55402**	**55413.3**	234.7
Hamilton3000b	55354	55392.6	255.7	55379	55416.7	234.5	**55318**	**55343.4**	223.6
Hamilton3000c	55566	55614.5	254.0	55571	55624.6	331.2	**55514**	**55544.5**	359.3
Hamilton3000d	55477	55513.2	254.0	55447	55634.5	453.4	**55404**	**55425.7**	243.5
Hamilton3000e	56177	56220.7	259.7	56145	56275.2	346.1	**56104**	**56156.7**	282.7
Hamilton4000	74956	75075.9	572.5	74931	75020.1	550.1	**74893**	**74938.1**	489.2
Hamilton5000	98538	98819.5	0.8	98427	98792.3	0.8	**93253**	**93324.8**	442.6
powergrid	20584	21141.8	409.9	20563	20623.2	363.3	**20533**	**20546.6**	248.9
#Wins|Ties|Loses	11|0|0	11|0|0	–	10|1|0	10|0|1	–	–	–	–
#p-value	**9.8e-4**	**9.8e-4**	–	**2.0e-3**	**2.0e-3**	–	–	–	–

* MCBH denotes a multi-start version of CBH.

Table 6: Comparison of FTMS and SOTA Algorithms on Real-world Benchmark R2 with $B = \text{floor}(0.1n)$

	CBH			MCBH*			**FTMS**		
Instance	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
yeast1	1397	1421.1	56.1	1397	1421.1	51.9	**1380**	**1382.1**	75.5
Hamilton1000	11873	11917.0	19.3	11884	11929.5	20.6	**11850**	**11862.9**	36.7
Hamilton2000	24532	24604.9	147.3	24526	24619.4	147.6	**24430**	**24475.4**	264.4
Hamilton3000a	36710	40990.2	60.6	36304	36393.3	466.0	**36175**	**36223.9**	568.4
Hamilton3000b	36187	36301.2	523.6	36231	36872.0	452.7	**36081**	**36128.9**	559.5
Hamilton3000c	36591	36670.2	494.6	36537	38453.1	331.7	**36425**	**36503.7**	548.1
Hamilton3000d	36525	36588.2	521.5	36478	37544.8	456.5	**36362**	**36430.9**	565.8
Hamilton3000e	36786	36907.6	500.8	36822	36943.5	486.6	**36697**	**36748.9**	564.3
Hamilton4000	56742	57301.2	0.5	56890	57350.8	0.5	**49772**	**49876.2**	567.3
Hamilton5000	71467	71875.9	0.8	71083	71697.0	0.7	**62366**	**62595.2**	597.2
powergrid	10836	10943.3	598.5	10709	10828.3	593.0	**10667**	**10681.3**	531.0
#Wins|Ties|Loses	11|0|0	11|0|0	–	11|0|0	11|0|0	–	–	–	–
#p-value	**9.8e-4**	**9.8e-4**	–	**9.8e-4**	**9.8e-4**	–	–	–	–

* MCBH denotes a multi-start version of CBH.

Table 7: Comparison of FTMS and SOTA Algorithms on Synthetic Benchmark S1 with $B = \text{floor}(0.05n)$

	PBM		CBH			MCBH[⋆]			**FTMS**		
Instance	LB	UB	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
ba1(3)	**4275**[*]	**4275**	**4275**	**4275.0**	8.5	**4275**	**4275.0**	0.7	**4275**	**4275.0**	0.1
ba1(6)	**4278**[*]	**4278**	**4278**	4278.4	9.8	**4278**	**4278.0**	0.2	**4278**	**4278.0**	0.1
ba1(9)	**4193**[*]	**4193**	**4193**	**4193.0**	7.7	**4193**	**4193.0**	0.2	**4193**	**4193.0**	0.1
ba2(3)	4384	**4461**	4465	**4465.0**	20.9	4465	**4465.0**	0.1	4465	**4465.0**	0.1
ba2(6)	**4369**[*]	**4369**	4436	4453.4	19.3	4371	4409.4	19.4	4371	**4371.0**	1.6
ba2(9)	4371	**4463**	4465	**4465.0**	18.3	4465	**4465.0**	0.1	4465	**4465.0**	0.1
er1(3)	2798	**2835**	2842	2843.4	9.6	**2835**	2838.5	25.0	**2835**	**2835.0**	3.9
er1(6)	2799	**2835**	**2835**	2839.6	11.2	**2835**	**2835.0**	4.0	**2835**	**2835.0**	0.5
er1(9)	**2814**[*]	**2814**	2847	2847.7	8.4	**2814**	2819.9	16.7	**2814**	**2815.6**	6.8
er2(3)	15990	16955	16842	16897.3	101.5	16823	16837.4	27.4	**16818**	**16829.2**	13.4
er2(6)	16026	16930	16887	16930.0	91.5	16833	16838.6	25.4	**16829**	**16831.4**	12.6
er2(9)	15970	16954	16899	16900.9	129.1	**16761**	16787.3	20.9	**16761**	**16764.9**	19.1
gnm1(3)	15972	16771	16706	16716.6	136.4	**16638**	**16649.5**	27.6	**16638**	16657.2	6.7
gnm1(6)	16209	17062	16975	16977.4	104.6	**16965**	**16965.2**	27.6	**16965**	16967.4	19.0
gnm1(9)	16099	16958	**16843**	16860.1	89.0	**16843**	**16843.0**	9.8	**16843**	16844.5	7.7
gnm2(3)	34015	36803	**35332**	**35334.7**	281.4	**35332**	35346.8	32.7	**35332**	35337.4	17.6
gnm2(6)	33701	36445	35236	35245.4	303.4	**35191**	35219.1	30.8	35203	**35215.9**	30.2
gnm2(9)	33782	36641	35331	35350.8	353.9	**35298**	35309.2	26.7	35303	**35303.0**	17.2
gnm3(3)	36403	40229	39978	39982.0	627.3	39620	39702.0	39.2	**39555**	**39615.6**	44.5
gnm3(6)	36557	40217	39848	39876.1	681.6	39490	39546.2	34.9	**39334**	**39440.0**	52.1
gnm3(9)	36258	40176	39852	39880.5	605.1	39548	39597.5	38.8	**39544**	**39578.7**	59.0
#Wins|Ties|Loses	–	12|6|3	13|8|0	16|4|1	–	5|14|2	12|6|3	–	–	–	–
#p-value	–	**2.2e-3**	**1.5e-3**	**4.2e-4**	–	3.5e-1	**5.0e-2**	–	–	–	–

[*] Optimal results obtained by exact algorithm reported in [3] within 3600 seconds.
[⋆] MCBH denotes a multi-start version of CBH.

Table 8: Comparison of FTMS and SOTA Algorithms on Synthetic Benchmark S1 with $B = \text{floor}(0.1n)$

	PBM		CBH			MCBH⋆			**FTMS**		
Instance	LB	UB	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
ba1(3)	**3330***	**3330**	**3330**	**3330.0**	13.6	**3330**	**3330.0**	0.8	**3330**	**3330.0**	0.2
ba1(6)	**3390***	**3390**	**3390**	3391.0	14.1	**3390**	**3390.0**	0.7	**3390**	**3390.0**	0.2
ba1(9)	**3328***	**3328**	**3328**	3328.4	13.6	**3328**	**3328.0**	2.3	**3328**	**3328.0**	0.4
ba2(3)	3716	3987	4005	4005.0	46.1	4002	4004.3	11.0	**3916**	**3991.2**	20.9
ba2(6)	3718	3916	3955	3955.9	44.5	3916	3918.8	18.0	**3909**	**3910.5**	20.8
ba2(9)	3702	3986	4004	4004.0	48.6	4002	4004.2	9.9	**3916**	**3949.7**	18.8
er1(3)	2395	**2474**	2535	2538.7	20.5	**2474**	**2474.6**	17.8	**2474**	2475.8	1.0
er1(6)	2395	**2482**	2485	2519.1	22.9	**2482**	2482.3	19.2	**2482**	**2482.1**	1.4
er1(9)	2378	**2452**	2528	2539.0	20.6	**2452**	**2456.5**	24.1	**2452**	2457.5	2.8
er2(3)	12469	14886	14332	14360.0	213.5	**14326**	**14341.7**	31.3	14331	14344.9	21.7
er2(6)	12575	15052	14364	14380.2	217.2	14262	14329.6	39.6	**14225**	**14235.7**	14.6
er2(9)	12414	15038	14397	14434.0	255.7	**14333**	14368.3	30.2	14347	**14363.5**	22.4
gnm1(3)	12517	14730	14193	14256.3	252.2	**14161**	**14173.1**	34.4	**14161**	14177.2	17.2
gnm1(6)	12601	14658	14399	14419.0	221.4	**14393**	14420.5	40.0	**14393**	**14399.4**	15.1
gnm1(9)	12565	14803	14195	14211.3	178.1	**14184**	14186.3	26.1	**14184**	**14184.4**	13.5
gnm2(3)	25877	28978	**28715**	28734.4	637.4	**28715**	28723.9	32.8	**28715**	**28715.0**	22.7
gnm2(6)	25262	30635	28554	28629.7	647.9	28541	28584.3	32.9	**28540**	**28540.4**	38.9
gnm2(9)	25765	30805	28868	28922.1	709.9	28872	28912.0	37.0	**28823**	**28833.3**	52.5
gnm3(3)	28541	35847	35158	35198.5	1539.6	34903	35014.2	37.9	**34733**	**34886.5**	56.6
gnm3(6)	27307	35501	35000	35079.6	1353.8	34635	34762.4	41.8	**34552**	**34596.6**	57.0
gnm3(9)	28698	35704	34785	34837.7	1323.1	**34505**	34725.1	40.1	34522	**34641.0**	60.0
#Wins|Ties|Loses	–	15|6|0	17|4|0	20|1|0	–	8|10|3	14|3|4	–	–	–	–
#p-value	–	**6.5e-4**	**2.9e-4**	**9.0e-5**	–	**5.0e-2**	**2.5e-3**	–	–	–	–

* Optimal results obtained by exact algorithm reported in [3] within 3600 seconds.
⋆ MCBH denotes a multi-start version of CBH.

best-known upper bounds on the remaining six instances. At a significance level of 0.05, FTMS significantly outperforms the state-of-the-art algorithms in terms of all performance indicators (i.e., $\hat{f}$, $\bar{f}$ and $\bar{t}$).

Finally, comparative results between FTMS and SOTA algorithms on synthetic benchmark S2 are summarized in Tables 9. It is worth noting that each instance is solved with a fixed B value like in [4]. Both MCBH and FTMS are executed with a longer time limit, i.e., $\hat{t} = 600$ seconds.

Table 9: Comparison of FTMS and SOTA Algorithms on Synthetic Benchmark S2

		PBM		CBH			MCBH*			**FTMS**		
Instance	B	LB	UB	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
FF250	13	**1587***	**1587**	**1587**	1598.2	16.7	**1587**	**1587.0**	0.9	**1587**	**1587.0**	0.1
BA250	25	**13772***	**13772**	**13772**	13788.3	137.6	**13772**	13773.2	24.5	**13772**	**13772.0**	7.5
BA500	50	**24847***	**24847**	**24847**	**24847.0**	1104.7	25316	31325.3	18.1	**24847**	**24847.0**	25.2
BA1000	100	16071	316735	59178	60488.9	3600.0	58651	72576.7	475.4	**58493**	**58528.2**	234.9
ER250	25	17959	22288	19894	19931.6	326.5	**19870**	**19870.0**	133.2	**19870**	19872.4	58.2
ER500	50	30846	79482	68062	68129.2	3189.6	**68012**	68098.9	222.3	**68012**	**68020.8**	145.5
ER1000	100	70494	221831	173538	174326.1	3600.0	175773	177637	0.3	**170351**	**170671.5**	380.7
WS250a	70	1039	2319	2034	2056.8	728.7	1889	**1919.4**	255.0	**1882**	1919.5	27.9
WS250b	25	14586	15223	**15020**	15044.7	316.1	**15020**	**15022.6**	384.8	**15020**	15029.3	97.4
WS500	50	25907	53729	51460	51567.2	3483.4	51500	51539.3	246.3	**51422**	**51434.7**	126.1
GNM250	25	18638	22711	20967	20984.5	453.8	**20944**	**20944.0**	151.7	**20944**	**20944.0**	64.8
GNM500	50	29462	78001	**65775**	65892.9	2883.0	65857	65925.0	185.5	**65775**	**65816.4**	206.3
#Wins|Ties|Loses	–	–	9|3|0	7|5|0	11|1|0	–	6|6|0	7|2|3	–	–	–	–
#p-value	–	–	**2.0e-4**	**4.4e-4**	**8.9e-5**	–	**6.5e-4**	**4.6e-4**	–	–	–	–

* Optimal results obtained by exact algorithm reported in [3] within 3600 seconds.
⋆ MCBH denotes a multi-start version of CBH.

From Table 9, we find that FTMS also show an excellent performance on synthetic benchmark S2. Specifically, it attains new upper bounds for 9 instances, and matches the previous best-known upper bounds for the remaining three instances. At a significance level of 0.05, it is significantly better than the state-of-the-art algorithms in terms of all performance indicators, i.e., $\hat{f}$, $\bar{f}$ and $\bar{t}$. These results show FTMS competes well with SOTA algorithms.

5.5. Results of FTMS with a Long Time Limit $\hat{t} = 600$ Seconds

To further study the behavior of FTMS algorithm, we report the results of FTMS with a long time limit $\hat{t} = 600$ seconds. As observed from Tables 3-7, FTMS is able to steadily find the optimal solutions for all instances of the real-world benchmark R1 in only 60 seconds. It also can reach the known optimal results of the first three instances of the synthetic benchmark S1 both with $B = \text{floor}(0.05n)$ and $B = \text{floor}(0.1n)$. Therefore, our experiment focuses on the remaining instances that have not been optimally solved by FTMS with a 100% success rate.

Comparative results between FTMS with $\hat{t} = 60$ and $\hat{t} = 600$ seconds are summarized in Tables 10-11. In these tables, column 1 gives the instance name (Instance), columns 2-3 present the lower bounds (LB) and upper bounds (UB), respectively. Columns 4-6 provide the results of FTMS under $\hat{t} = 60$, including $\hat{f}$, $\bar{f}$ and $\bar{t}$. Correspondingly, columns 7-9 show the results of FTMS under $\hat{t} = 600$ seconds. In addition, we calculate the number of instances on which FTMS finds a better (#Wins), equal (#Ties) and worse (#Loses) results in terms of both $\hat{f}$ and $\bar{f}$ compared to each reference algorithm. The last row provides the p-values of the Wilcoxon signed ranks test.

Table 10: Results of FTMS on Synthetic Benchmark S1 with $B = \text{floor}(0.05n)$

	PBM		FTMS ($\hat{t} = 60$s)			FTMS ($\hat{t} = 600$s)		
Instance	LB	UB	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
ba2(3)	4384	**4461**	4465	**4465.0**	0.0	4465	**4465.0**	0.0
ba2(6)	**4369**	**4369**	4371	**4371.0**	1.6	4371	**4371.0**	2.0
ba2(9)	4371	**4463**	4465	**4465.0**	0.0	4465	**4465.0**	0.0
er1(3)	2798	**2835**	**2835**	**2835.0**	3.9	**2835**	**2835.0**	4.8
er1(6)	2799	**2835**	**2835**	**2835.0**	0.5	**2835**	**2835.0**	0.8
er1(9)	**2814**	**2814**	**2814**	2815.6	6.8	**2814**	**2814.0**	12.7
er2(3)	15990	16955	**16818**	16829.2	13.4	**16818**	**16818.0**	15.3
er2(6)	16026	16930	**16829**	**16831.4**	12.6	**16829**	16834.0	22.8
er2(9)	15970	16954	**16761**	16764.9	19.1	**16761**	**16761.0**	37.8
gnm1(3)	15972	16771	**16638**	16657.2	6.7	**16638**	**16638.0**	8.1
gnm1(6)	16209	17062	**16965**	16967.4	19.0	**16965**	**16967.1**	28.2
gnm1(9)	16099	16958	**16843**	**16844.5**	7.7	**16843**	**16844.5**	11.8
gnm2(3)	34015	36803	**35332**	**35337.4**	17.6	**35332**	**35337.4**	115.1
gnm2(6)	33701	36445	35203	35215.9	30.2	**35191**	**35198.2**	115.6
gnm2(9)	33782	36641	35303	35303.0	17.2	**35298**	**35302.0**	38.5
gnm3(3)	36403	40229	39555	39615.6	44.5	**39473**	**39555.9**	84.5
gnm3(6)	36557	40217	39334	39440.0	52.1	**39315**	**39350.9**	149.4
gnm3(9)	36258	40176	39544	39578.7	59.0	**39371**	**39460.5**	94.0
#Wins|Ties|Loses	–	12|3|3	5|13|0	10|7|1	–	–	–	–
#p-value	–	**1.3e-3**	**2.7e-2**	**3.9e-2**	–	–	–	–

From Table 10, we observe that FTMS improves its results under a longer time limit $\hat{t} = 600$ seconds. In particular, it finds new upper bounds for 12 instances, and matches the previous upper bounds for 3 out of the remaining 6 instances. Compared to FTMS with $\hat{t} = 60$ seconds, FTMS with $\hat{t} = 600$ seconds demonstrates a better performance in terms of both $\hat{f}$ and $\bar{f}$. At a significance level of 0.05, FTMS with $\hat{t} = 600$ seconds significantly outperforms FTMS with $\hat{t} = 60$ seconds in terms of both $\hat{f}$ and $\bar{f}$. Moreover, we find FTMS quickly converges to local optimum in no more than 150 seconds for all instances.

As we can see from Table 11, FTMS also improves its performance on synthetic benchmark S1 with $B = \text{floor}(0.1n)$. It finds new upper bounds for 15 instances and matches the previous upper bounds on the remaining three instances. At a significance level of 0.05, FTMS with $\hat{t} = 600$ seconds

Table 11: Results of FTMS on Synthetic Benchmark S1 with $B = \text{floor}(0.1n)$

	PBM		FTMS ($\hat{t} = 60$s)			FTMS ($\hat{t} = 600$s)		
Instance	LB	UB	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
ba2(3)	3716	3987	**3916**	3991.2	20.9	**3916**	**3944.4**	76.6
ba2(6)	3718	3916	**3909**	3910.5	20.8	**3909**	**3909.0**	52.5
ba2(9)	3702	3986	**3916**	3949.7	18.8	**3916**	**3916.0**	68.3
er1(3)	2395	**2474**	**2474**	2475.8	1.0	**2474**	**2474.0**	5.3
er1(6)	2395	**2482**	**2482**	**2482.1**	1.4	**2482**	2482.2	1.1
er1(9)	2378	**2452**	**2452**	2457.5	2.8	**2452**	**2453.7**	3.9
er2(3)	12469	14886	14331	14344.9	21.7	**14320**	**14333.6**	59.9
er2(6)	12575	15052	**14225**	14235.7	14.6	**14225**	**14225.0**	21.0
er2(9)	12414	15038	14347	14363.5	22.4	**14344**	**14347.5**	54.9
gnm1(3)	12517	14730	**14161**	14177.2	17.2	**14161**	**14162.2**	27.8
gnm1(6)	12601	14658	**14393**	14399.4	15.1	**14393**	**14394.5**	57.7
gnm1(9)	12565	14803	**14184**	14184.4	13.5	**14184**	**14184.0**	16.6
gnm2(3)	25877	28978	**28715**	**28715.0**	22.7	**28715**	**28715.0**	49.5
gnm2(6)	25262	30635	**28540**	**28540.4**	38.9	**28540**	28540.8	83.6
gnm2(9)	25765	30805	**28823**	28833.3	52.5	**28823**	**28831.0**	160.3
gnm3(3)	28541	35847	34670	34813.7	76.5	**34581**	**34676.2**	243.3
gnm3(6)	27307	35501	34552	34596.6	57.0	**34418**	**34449.8**	177.1
gnm3(9)	28698	35704	34522	34641.0	60.2	**34466**	**34498.0**	193.8
#Wins|Ties|Loses	–	15|3|0	5|13|0	15|1|2	–	–	–	–
#p-value	–	**4.3e-4**	**2.7e-2**	**3.2e-4**	–	–	–	–

is also significantly better than FTMS with $\hat{t} = 60$ seconds in terms of both $\hat{f}$ and $\bar{f}$. These results confirm that FTMS is able to find still better results under a long time limit. Furthermore, we find FTMS converges to local optimum in no more than 250 seconds for all instances.

6. Experimental Analysis

In this section, we perform additional experiments to gain a deeper understanding of FTMS. In particular, we perform four groups of experiments: 1) to compare the run-time distributions of FTMS and the state-of-the-art algorithm MCBH, 2) to evaluate the benefit of the IRR operator, 3) to investigate the effectiveness of the BCLS procedure, and 4) to analyze the randomness of FTMS. Our experimental analyses are conducted on the eight representative instances used for parameter tuning.

6.1. Run-time Distributions of FTMS and MSCH

To further evaluate FTMS, we resort to the time-to-target (TTT) plots [2] to analyze the run-time distributions of both FTMS and MCBH on the eight representative instances. A TTT plot is a useful tool for algorithm performance comparison. To produce a TTT plot, we run each algorithm 100 times, and record the computation time to obtain a solution at least as good as a given target value. After sorting them in ascending order, a probability p_i is associated with the i-th computation time t_i. A TTT plot is obtained by plotting these points $(t_i, p_i), i = 1, 2, \ldots, 100$. Figure 2 presents the TTT plots of FTMS and MCBH, where the target value of each instance is given in the parentheses behinds its instance name.

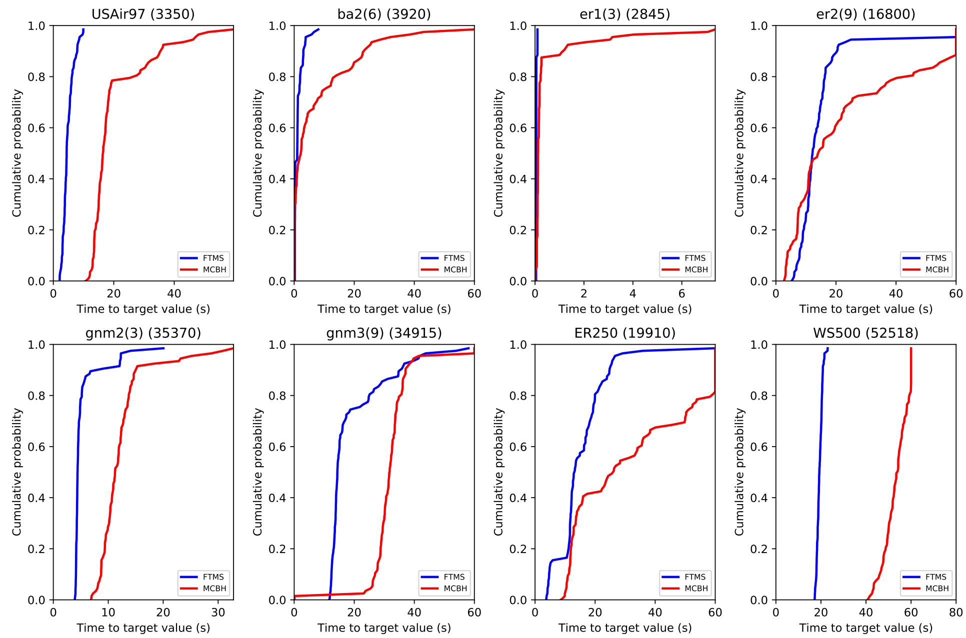


Figure 2: Run-time Distributions of FTMS and MCBH Algorithms

From Figure 2, we can observe that FTMS is more likely to find a target solution faster than MCBH. For USAir97 instance, it shows that the probability of finding the target value 3350 at most 20 seconds is approximately 60% for MCBH, while it is 100% for FTMS. Taking the gnm2(3) instance as another example, we find that the probabilities of finding the target value 35370 at most 10 seconds are about 20% and 90% for MCBH and FTMS, respectively. Similar observations apply to the other six instances. These results clearly show that FTMS outperforms MCBH.

6.2. Superiority of the IRR Operator

FTMS uses the inherit-repair recombination operator to generate offspring solutions. To show its interest, we experimentally compare FTMS with a variant called FTMS′. FTMS′ is obtained from FTMS by replacing the IRR operator with the backbone-based crossover presented in [4].

Comparative results between FTMS and FTMS′ are summarized in Table 12. At its bottom, we calculate the number of instances for which FTMS finds better (#Wins), equal (#Ties) and worse (#Loses) results compared to FTMS′.

Table 12: Comparison between FTMS′ (with backbone-based crossover) and FTMS (with inherit-repair recombination operator)

		FTMS′			**FTMS**		
Instance	B	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
USAir97	33	**3100**	**3100.0**	20.5	**3100**	**3100.0**	11.4
ba2(6)	10	**3909**	3910.3	37.9	**3909**	**3909.7**	27.5
er1(3)	4	**2835**	**2835.0**	1.9	**2835**	**2835.0**	3.5
er2(9)	10	**16761**	16778.5	12.3	**16761**	**16764.9**	19.1
gnm2(3)	15	**35332**	35339.5	24.4	**35332**	**35338.3**	17.5
gnm3(9)	30	34559	34713.6	45.9	**34475**	**34621.9**	53.3
ER250	25	**19870**	**19872.4**	32.7	**19870**	**19872.4**	23.2
WS500	50	51532	51663.9	56.3	**51483**	**51639.0**	35.4
#Wins|Ties|Loses	–	2|6|0	5|3|0	7|0|1	–	–	–

From Table 12, we observe that FTMS performs better than FTMS′. FTMS finds better results on two instances, and the same results on the remaining six instances in terms of $\hat{f}$. In terms of $\bar{f}$, FTMS obtains better results on five instances and the same results on the remaining three instances. For the computation time, FTMS also remains competitive. These results confirm the interest of the inherit-repair recombination operator.

6.3. Effectiveness of the BCLS Procedure

To evaluate the effectiveness of the betweenness centrality-based late-acceptance search (BCLS), we compare FTMS with an alternative algorithm named FTMS″. FTMS″ is obtained from FTMS by replacing BCLS with the centrality-based neighborhood search (CNS) proposed in [4]. Comparative performances in terms of the best result and average result are presented in the left and right sides of Figure 3, respectively. The x-axis indicates the instance name, and y-axis displays the performance gaps. By treating FTMS″ as a baseline algorithm, we calculate their performance gap as $(f - \tilde{f})/\tilde{f} \times 100\%$, where f is the result of FTMS and $\tilde{f}$ is the result of FTMS″. A performance gap smaller than zero means that FTMS achieves a better result on the corresponding instance.

As we can see from the left side of Figure 3, FTMS has a better performance than FTMS″ in terms of the best result. In particular, it finds better

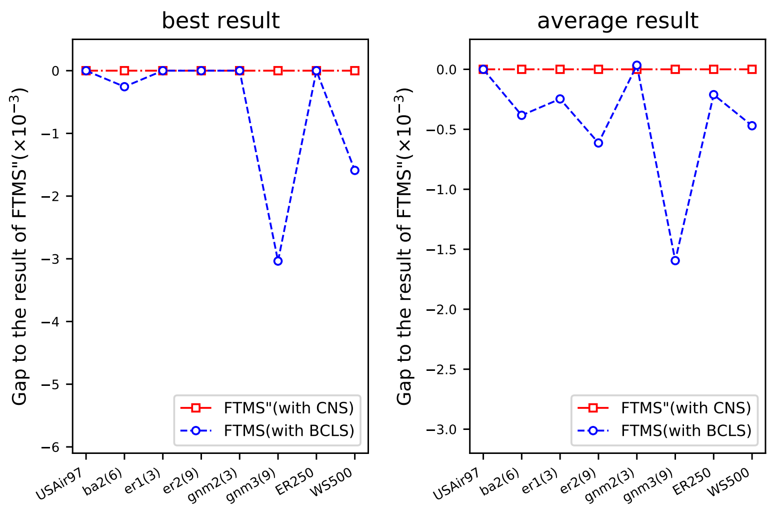


Figure 3: Comparison between FTMS$''$ (with centrality-based neighborhood search) and FTMS (with betweenness centrality-based late-acceptance search)

results on three instances and the same results on the remaining five instances. FTMS also demonstrates an excellent performance in terms of the average result, as shown in the right side of Figure 3. We can observe that FTMS is able to achieve better results on six instances and the same result on the remaining two instances. These results conform the effectiveness of BCLS.

6.4. Randomness Analysis of FTMS

Randomization is common in many implementations of metaheuristics. Our FTMS algorithm also integrates several randomized components, such as the IRR operator and the BCLS procedure. Taking BCLS as an example, we experimentally evaluate the interest of randomization. As shown in Algorithm 2, at each iteration, a head node u of $\mathcal{L}$ is added into S with a probability ϕ. Otherwise, node u is re-inserted into an intermediate position of $\mathcal{L}$. To show the merit of this randomized strategy in FTMS, we experimentally compare FTMS with an alternative version named FTMS$'''$ by setting $\phi = 1.0$. That is, a head node u is always added into S at each iteration of BCLS. Figure 4 describes the comparative performance of FTMS$'''$ and FTMS.

From Figure 4, we observe that FTMS performs better in terms of the best result on three instances than FTMS$'''$, and they have the same performance on the remaining five instances. In terms of the average result,

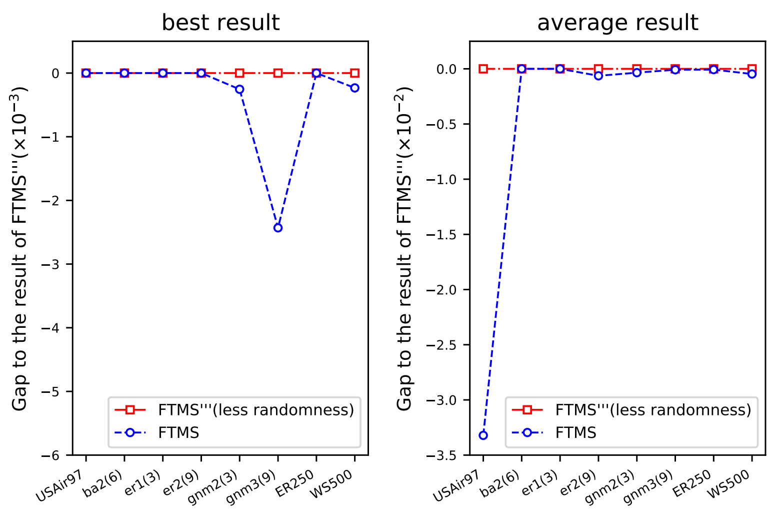


Figure 4: Comparison between FTMS‴ (with $\phi = 1.0$) and FTMS (with $\phi = 0.8$)

FTMS also shows a better performance by finding improved results on six instances and the same results on the remaining two instances as FTMS‴ does. These observations confirm the usefulness of randomization in FTMS.

7. Conclusion and Future Work

Detecting critical nodes in a complex network represents a class of challenging NP-hard problems. Considerable efforts have been devoted to developing efficient algorithms for critical node detection problems. However, few efforts have been made to solve the distance-based critical node problems, which aim to identify a subset of nodes in a network whose deletion minimizes the distance-based pairwise connectivity (i.e., the number of node pairs connected by a path of length at most k). To address it, we proposed a fast tri-individual memetic search (FTMS) method. FTMS is characterized by a small population of only three individuals, which relies on an inherit-repair recombination operator to generate a promising solution and a fast betweenness centrality-based late-acceptance search to find high-quality local optima during the search.

Extensive computational studies on both real-world and synthetic benchmarks show FTMS is highly competitive compared to state-of-the-art algorithms. In particular, FTMS is able to steadily find the optimal solutions for 22 real-world instances with known optima in only one minute, and new upper bounds for the remaining 22 large real-world instances. For 54 synthetic instances, FTMS also achieves excellent performance by finding 36 new upper bounds and matching 15 previous upper bounds. As future work, it is worthy of further improving FTMS based on other centrality measures, such as closeness centrality and eigenvector centrality. It is also interesting to adopt FTMS for solving other distance-based critical node detection problems.

Acknowledgment

We would like to thank the anonymous referees for their helpful comments and suggestions, which helped us to improve the presentation of the work. This work was supported in part by the National Natural Science Foundation of China under Grants 61903144, 71871142 and 71931007, and Startup Plan for New Young Teachers of Shanghai Jiao Tong University under Grant 22X010503820.

Appendix A. Influence of the Population Size

Unlike the general memetic algorithm [28], the FTMS algorithm maintains a small population of only three individuals. To investigate the influence of the population size on the performance of FTMS, we experimentally compare the three-individual FTMS with FTMS using 10 individuals (named FTMS_1). Detailed comparative results are summarized in Table A.13.

Table A.13: Comparison of FTMS Algorithms with Different Population Sizes

		FTMS_1 (with large population)			FTMS (with small population)		
Instance	B	$\hat{f}$	$\bar{f}$	$\bar{t}$	$\hat{f}$	$\bar{f}$	$\bar{t}$
USAir97	33	**3100**	3109.9	23.2	**3100**	**3100.0**	11.4
ba2(6)	10	**3909**	3910.9	28.7	**3909**	**3909.7**	27.5
er1(3)	4	**2835**	**2835.0**	2.4	**2835**	**2835.0**	3.5
er2(9)	10	**16761**	16765.6	30.8	**16761**	**16764.9**	19.1
gnm2(3)	15	35353	35353.0	12.9	**35332**	**35338.3**	17.5
gnm3(9)	30	34534	34662.4	55.9	**34475**	**34621.9**	53.3
ER250	25	**19870**	19881.3	35.7	**19870**	**19872.4**	23.2
WS500	50	51588	51782.0	33.5	**51483**	**51639.0**	35.4
#Wins|Ties|Loses	–	3|5|0	7|1|0	7|0|1	–	–	–

The results in Table A.13 show that FTMS performs better than FTMS_1 in terms of both the best and average results. In particular, FTMS obtains better results than FTMS_1 in terms of $\hat{f}$, including improved results on three instances and same results on the remaining five instances. In terms of $\bar{f}$, FTMS also has a better performance by finding improved results on all instances except er1(3) than FTMS_1, For er1(3), both FTMS and FTMS_1 find the same result. These results confirm the usefulness of the small population strategy used in FTMS.

Appendix B. Rationale behind the Population Initialization Strategy

The FTMS algorithm starts its search from a small population consisting of three distinct individuals, i.e., S_1, S_2 and S_3. Those three individuals are constructed based on the degree centrality, k-Katz centrality and k-betweenness centrality, respectively. To show the usefulness behind this population initialization strategy, we experimentally compare FTMS with six variants using different population initialization strategies as follows.

- FTMS_2: the initial population is built only based on the degree centrality;
- FTMS_3: the initial population is built only based on the k-Katz centrality;
- FTMS_4: the initial population is built only based on the k-betweenness centrality;
- FTMS_5: the initial population is built based on both the degree and k-Katz centralities;
- FTMS_6: the initial population is built based on both the degree and k-betweenness centralities;
- FTMS_7: the initial population is built based on both the k-Katz and k-betweenness centralities.

Table B.14 summarizes the comparative results between FTMS and its six variants. At its bottom, we give the average value and average rank of both $\hat{f}$ and $\bar{f}$ performance indicators. We order these seven algorithms for each instance separately, the best performing algorithm obtaining the rank of 1, the second best rank of 2, and so on. In case of ties, average ranks are assigned. Finally, we obtain the average rank of each algorithm by averaging the ranks of all eight instances. The smaller the average rank, the better the algorithm. From the results, we observe that FTMS obtains the smallest average values and the smallest average ranks in terms of both $\hat{f}$ and $\bar{f}$. In

Table B.14: Comparison of FTMS Algorithms with Different Initialization Strategies

		FTMS_2		FTMS_3		FTMS_4		FTMS_5		FTMS_6		FTMS_7		**FTMS**	
Instance	B	$\hat{f}$	$\bar{f}$	$\hat{f}$	$\bar{f}$	$\hat{f}$	$\bar{f}$	$\hat{f}$	$\bar{f}$	$\hat{f}$	$\bar{f}$	$\hat{f}$	$\bar{f}$	$\hat{f}$	$\bar{f}$
USAir97	33	**3100**	3215.5	**3100**	3186.5	**3100**	3174.0	**3100**	3163.9	**3100**	3228.0	**3100**	3163.9	**3100**	**3100.0**
ba2(6)	10	**3909**	3910.0	**3909**	3910.0	**3909**	3910.2	**3909**	3910.7	**3909**	**3909.2**	**3909**	3910.1	**3909**	3909.7
er1(3)	4	**2835**	**2835.0**	**2835**	**2835.0**	**2835**	**2835.0**	**2835**	**2835.0**	**2835**	**2835.0**	**2835**	**2835.0**	**2835**	**2835.0**
er2(9)	10	**16761**	16763.7	**16761**	16770.7	**16761**	16766.7	**16761**	**16762.8**	**16761**	16768.6	**16761**	16767.6	**16761**	16764.9
gnm2(3)	15	35341	35351.8	**35332**	35343.1	35341	35348.2	**35332**	35347.3	35341	35349.4	**35332**	35344.9	**35332**	**35338.3**
gnm3(9)	30	34497	34668.0	34553	34683.5	34542	**34619.4**	34606	34665.8	34613	34648.0	34562	34636.0	**34475**	34621.9
ER250	25	**19870**	19874.1	**19870**	19876.4	**19870**	**19871.0**	**19870**	19875.2	**19870**	19874.8	**19870**	19874.8	**19870**	19872.4
WS500	50	51566	51791.0	51492	51632.6	**51466**	51594.0	51564	51816.2	51627	51760.4	51473	**51565.7**	51483	51639.0
avg.value		20984.9	21051.1	20981.5	21029.7	20978.0	21014.8	20997.1	21047.1	21007.0	21046.7	20980.3	21012.3	**20970.6**	**21010.2**
avg.rank		4.3	4.7	3.8	4.8	3.8	3.4	4.2	4.6	5.0	4.7	3.7	3.5	**3.3**	**2.4**

particular, FTMS finds the best solutions on all test instances except for WS500. For the WS500 instance, FTMS obtains the third best solution. These results prove the usefulness of our chosen population initialization strategy.

References

References

[1] Addis, B., Aringhieri, R., Grosso, A., Hosteins, P., 2016. Hybrid constructive heuristics for the critical node problem. Annals of Operations Research 238, 637–649.

[2] Aiex, R.M., Resende, M.G., Ribeiro, C.C., 2007. TTT plots: a perl program to create time-to-target plots. Optimization Letters 1, 355–366.

[3] Alozie, G.U., Arulselvan, A., Akartunalı, K., Pasiliao Jr, E.L., 2021. Efficient methods for the distance-based critical node detection problem in complex networks. Computers & Operations Research 131, 105254.

[4] Alozie, G.U., Arulselvan, A., Akartunalı, K., Pasiliao Jr, E.L., 2022. A heuristic approach for the distance-based critical node detection problem in complex networks. Journal of the Operational Research Society 73, 1347–1361.

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

[6] Aringhieri, R., Grosso, A., Hosteins, P., Scatamacchia, R., 2016b. Local search metaheuristics for the critical node problem. Networks 67, 209–221.

[7] Aringhieri, R., Grosso, A., Hosteins, P., Scatamacchia, R., 2016c. A preliminary analysis of the distance based critical node problem. Electronic Notes in Discrete Mathematics 55, 25–28.

[8] Aringhieri, R., Grosso, A., Hosteins, P., Scatamacchia, R., 2019. Polynomial and pseudo-polynomial time algorithms for different classes of the distance critical node problem. Discrete Applied Mathematics 253, 103–121.

[9] Arulselvan, A., Commander, C.W., Elefteriadou, L., Pardalos, P.M., 2009. Detecting critical nodes in sparse graphs. Computers & Operations Research 36, 2193–2200.

[10] Ausiello, G., Boria, N., Giannakos, A., Lucarelli, G., Paschos, V.T., 2012. Online maximum k-coverage. Discrete Applied Mathematics 160, 1901–1913.

[11] Baggio, A., Carvalho, M., Lodi, A., Tramontani, A., 2021. Multilevel approaches for the critical node problem. Operations Research 69, 486–508.

[12] Brandes, U., 2001. A faster algorithm for betweenness centrality. Journal of Mathematical Sociology 25, 163–177.

[13] Brandes, U., 2008. On variants of shortest-path betweenness centrality and their generic computation. Social Networks 30, 136–145.

[14] Charkhgard, H., Subramanian, V., Silva, W., Das, T.K., 2018. An integer linear programming formulation for removing nodes in a network to minimize the spread of influenza virus infections. Discrete Optimization 30, 144–167.

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

[16] Di Summa, M., Grosso, A., Locatelli, M., 2012. Branch and cut algorithms for detecting critical nodes in undirected graphs. Computational Optimization and Applications 53, 649–680.

[17] Ding, J., Lü, Z., Li, C., Shen, L., Xu, L., Glover, F.W., 2019. A two-individual based evolutionary algorithm for the flexible job shop scheduling problem, in: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, AAAI Press. pp. 2262–2271.

[18] Divsalar, A., Vansteenwegen, P., Sörensen, K., Cattrysse, D., 2014. A memetic algorithm for the orienteering problem with hotel selection. European Journal of Operational Research 237, 29–49.

[19] Doostmohammadian, M., Rabiee, H.R., Khan, U.A., 2020. Centrality-based epidemic control in complex social networks. Social Network Analysis and Mining 10, 32.

[20] Fan, C., Zeng, L., Sun, Y., Liu, Y., 2020. Finding key players in complex networks through deep reinforcement learning. Nature Machine Intelligence 2, 317–324.

[21] Hagberg, A., Swart, P., S Chult, D., 2008. Exploring network structure, dynamics, and function using NetworkX. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States). URL: https://www.osti.gov/servlets/purl/960616.

[22] Hooshmand, F., Mirarabrazi, F., MirHassani, S., 2020. Efficient benders decomposition for distance-based critical node detection problem. Omega 93, 102037.

[23] Katz, L., 1953. A new status index derived from sociometric analysis. Psychometrika 18, 39–43.

[24] Landherr, A., Friedl, B., Heidemann, J., 2010. A critical review of centrality measures in social networks. Business & Information Systems Engineering 2, 371–385.

[25] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58.

[26] Lü, Z., Hao, J., 2010. A memetic algorithm for graph coloring. European Journal of Operational Research 203, 241–250.

[27] Moalic, L., Gondran, A., 2018. Variations on memetic algorithms for graph coloring problems. Journal of Heuristics 24, 1–24.

[28] Neri, F., Cotta, C., 2012. Memetic algorithms and memetic computing optimization: A literature review. Swarm and Evolutionary Computation 2, 1–14.

[29] Pullan, W., 2015. Heuristic identification of critical nodes in sparse real-world graphs. Journal of Heuristics 21, 577–598.

[30] Purevsuren, D., Cui, G., Qu, M., Win, N.H., 2017. Hybridization of GRASP with exterior path relinking for identifying critical nodes in graphs. IAENG International Journal of Computer Science 44, 157–165.

[31] Ren, J., Hao, J.K., Wu, F., Fu, Z.H., 2023. An effective hybrid search algorithm for the multiple traveling repairman problem with profits. European Journal of Operational Research 304, 381–394.

[32] Salemi, H., Buchanan, A., 2022. Solving the distance-based critical node problem. INFORMS Journal on Computing 34, 1309–1326.

[33] de San Lázaro, I.M., Sánchez-Oro, J., Duarte, A., 2021. Finding critical nodes in networks using variable neighborhood search, in: Proceedings of Variable Neighborhood Search - 8th International Conference, ICVNS 2021, Abu Dhabi, United Arab Emirates, March 21-25, 2021, pp. 1–13.

[34] Ventresca, M., 2012. Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem. Computers & Operations Research 39, 2763–2775.

[35] Ventresca, M., Aleman, D., 2013. Evaluation of strategies to mitigate contagion spread using social network characteristics. Social Networks 35, 75–88.

[36] Ventresca, M., Aleman, D., 2014. A randomized algorithm with local search for containment of pandemic disease spread. Computers & Operations Research 48, 11–19.

[37] Ventresca, M., Aleman, D., 2015a. Efficiently identifying critical nodes in large complex networks. Computational Social Networks 2, 1–16.

[38] Ventresca, M., Aleman, D., 2015b. Network robustness versus multi-strategy sequential attack. Journal of Complex Networks 3, 126–146.

[39] Ventresca, M., Harrison, K.R., Ombuki-Berman, B.M., 2018. The bi-objective critical node detection problem. European Journal of Operational Research 265, 895–908.

[40] Veremyev, A., Boginski, V., Pasiliao, E.L., 2014a. Exact identification of critical nodes in sparse networks via new compact formulations. Optimization Letters 8, 1245–1259.

[41] Veremyev, A., Prokopyev, O.A., Pasiliao, E.L., 2014b. An integer programming framework for critical elements detection in graphs. Journal of Combinatorial Optimization 28, 233–273.

[42] Veremyev, A., Prokopyev, O.A., Pasiliao, E.L., 2015. Critical nodes for distance-based connectivity and related problems in graphs. Networks 66, 170–195.

[43] Wang, Z., Di, Y., 2022. Cluster expansion method for critical node problem based on contraction mechanism in sparse graphs. IEICE TRANSACTIONS on Information and Systems 105, 1135–1149.

[44] Zhang, Y., Chen, Q., Chen, B., Liu, J., Zheng, H., Yao, H., Zhang, C., 2020. Identifying hotspots of sectors and supply chain paths for electricity conservation in China. Journal of Cleaner Production 251, 119653.

[45] Zhao, Y., Cao, Y., Shi, X., Wang, S., Yang, H., Shi, L., Li, H., Zhang, J., 2021. Critical transmission paths and nodes of carbon emissions in electricity supply chain. Science of The Total Environment 755, 142530.

[46] Zhou, Y., Hao, J.K., Duval, B., 2017. Opposition-based memetic search for the maximum diversity problem. IEEE Transactions on Evolutionary Computation 21, 731–745.

[47] Zhou, Y., Hao, J.K., Fu, Z.H., Wang, Z., Lai, X., 2021a. Variable population memetic search: A case study on the critical node problem. IEEE Transactions on Evolutionary Computation 25, 187–200.

[48] Zhou, Y., Hao, J.K., Glover, F., 2019. Memetic search for identifying critical nodes in sparse graphs. IEEE Transactions on Cybernetics 49, 3699–3712.

[49] Zhou, Y., Kou, Y., Zhou, M., 2022. Bilevel memetic search approach to the soft-clusteredvehicle routing problem. Transportation Science URL: https://doi.org/10.1287/trsc.2022.1186.

[50] Zhou, Y., Wang, Z., Jin, Y., Fu, Z.H., 2021b. Late acceptance-based heuristic algorithms for identifying critical nodes of weighted graphs. Knowledge-Based Systems 211, 106562.