CLEI Journal, Volume 29, Issue 1, Paper 9, February 2026

A Simulated Annealing algorithm using the Node-depth Phylogenetic structure for the Degree-Constrained Minimum Spanning Tree problem

José Flavio Lopes
Universidade Federal de Alfenas, Department of Computer Science,
Alfenas, Brazil, 37133-840
jose.flavio@sou.unifal-mg.edu.br

and

Iago Augusto Carvalho
Universidade Federal de Alfenas, Department of Computer Science,
Alfenas, Brazil, 37133-840
iago.carvalho@unifal-mg.edu.br

Abstract

The degree-constrained minimum spanning tree problem (DCMST) is an NP-hard optimization problem defined on connected weighted graphs. It consists of computing a minimum-cost spanning tree of a graph whose nodes have degrees smaller than or equal to a predefined constant D. This paper proposes a new heuristic algorithm for solving the DCMST, namely the Node-depth Phylogenetic-based Simulated Annealing heuristic (NPE-SA). The proposed algorithm implements the classic Simulated Annealing (SA) metaheuristic using the Node-depth Phylogenetic-based Encoding (NPE), a powerful data structure for indirectly representing spanning trees. Computational experiments performed on two sets of classic instances from the literature demonstrated that NPE-SA outperforms the best metaheuristic from the literature when solving the proposed DCMST instances. Furthermore, it outperforms the Node-depth Phylogenetic-based CLONALG heuristic, another heuristic that employs the NPE data structure

Keywords: Degree-constrained minimum spanning tree problem, Simulated Annealing, Node-depth Phylogenetic-based Encoding, Hybrid Genetic Algorithm, CLONALG.

1 Introduction

Let $G = (N, E, \mathcal{W})$ be a connected graph, where N is the set of nodes and E is the set of edges. Furthermore, $\mathcal{W} : E \to \mathbb{R}_+$ is a function that associates every edge $e \in E$ to a positive weight value w_e. The minimum spanning tree problem (MST) consists of computing a spanning tree of G whose sum of its edge weights is minimal. This problem can be used when one is interested in constructing a minimum-weight structure that connects three or more physically distant points, finding applications in communications, circuit designs, transportation networks, and cluster analysis, among other areas. It can be solved by polynomial-time algorithms, such as Prim's and Kruskal's [1].

Several MST variants have one or more additional characteristics. They are usually NP-complete, e. g., the degree-constrained minimum spanning tree problem [2], the minimum-cost bounded-error minimum spanning tree problem [3], and the leaf-constrained minimum spanning tree problem [4]. Their complex structures need to be addressed by sophisticated exact techniques, such as column-generation algorithms [5], constraint programming [6], or branch-and-bound methods [7]. In addition, approximate solutions for these problems can be quickly obtained using heuristic and metaheuristic algorithms [8].

This paper studies the degree-constrained minimum spanning tree problem (DCMST). This problem is defined on a weighted connected graph $G = (N, E, \mathcal{W})$ as previously defined along with an additional parameter D that constrains the maximum degree of a node in the solution. The objective of DCMST is

to find a minimum weight spanning tree of G whose nodes have degrees smaller than or equal to D. This problem is NP-hard if $D \geq 2$ [9].

DCMST finds applications in network design and transportation systems. For instance, it can be used to design road intersections with a maximum number of connections [10] or to model the limited number of connections of a network switch in the design of communication networks [11]. Furthermore, it also finds applications in VLSI design [12] and backplane wiring among pins [13], among other areas.

In this paper, we propose the Node-depth Phylogenetic-based Simulated Annealing heuristic (NPE-SA), a new and efficient heuristic for the DCMST. NPE-SA combines the classic Simulated Annealing (SA) metaheuristic [14] and the Node-depth Phylogenetic-based Encoding (NPE) [15], which is a powerful data structure that indirectly represents spanning trees.

NPE is more suitable for combining with the SA than other data structures, such as the edge-set encoding, the link-and-node encoding, or the characteristic vectors encoding, because it has a perfect coverage of the search space and is not biased towards certain spanning tree structures. Furthermore, one can easily model some additional problem constraints within NPE, such as the maximum node degree of the DCMST, and has a good locality, which is important for later stages of the SA algorithm. We refer the reader to the work of Carvalho and Ribeiro [16] for a discussion on spanning tree encodings for evolutionary algorithms.

The objective of NPE-SA is to find good quality approximated solutions for the DCMST in a reasonable computational time. Computational experiments evaluate our heuristic using classic DCMST instances from the literature. We show that NPE-SA outperforms the other two heuristics for the DCMST, including the best-known heuristic in the literature for this problem.

The remainder of this paper is organized as follows. Section 2 reviews related works. Section 3 presents the NPE-SA, which is evaluated in Section 4. Finally, Section 5 draws the concluding remarks of our paper.

2 Related work

Several ad-hoc heuristics for the DCMST were proposed in the literature. Narula and Ho [17] proposed primal and dual heuristics for this problem, while Savelsbergh and Volgenant [10] proposed two general primal heuristics. Additionally, Boldon, Deo, and Kumar [18] presented a heuristic based on Prim's algorithm.

Exact algorithms for the DCMST were also proposed in the literature [19, 20]. Initially, Caccetta and Hill [19] proposed a branch-and-cut algorithm based on subtour elimination constraints. This branch-and-cut algorithm was later improved by Bicalho, Cunha, and Lucena [20]. The authors proposed a branch-and-cut-and-price algorithm for DCMST, whereas the rows were generated by separating subtour inequalities using a similar strategy to that of Caccetta and Hill [19]. Furthermore, the columns were inserted using Lagrangian algorithms.

A large number of metaheuristic algorithms were also proposed for the DCMST. The most common metaheuristics are based on genetic algorithms (GA) [21, 22, 23, 24, 11]. The initial study by Zhou and Gen [21] proposed a GA based on Prüfer-numbers encoding [25]. Later, Knowles and Corne [22] presented a GA based on a multi-dimensional array encoding, while Raidl and Julstrom [23] proposed a similar algorithm based on the weight encoding. Two GA variants with the classic edge-set encoding were proposed by Raidl and Julstrom [24], whereas one of the variants implemented a heuristic within their evolutionary operators. Finally, Singh and Sundar [11] proposed a hybrid GA (HGA) that combines a steady-state GA and local search strategies. The latter algorithm was demonstrated to be the best metaheuristic in the literature for solving DCMST.

Additionally, SA was previously employed to solve DCMST in the literature. Krishnamoorthy, Ernst, and Sharaiha [26] contrasted several metaheuristic algorithms for DCMST, including a SA algorithm. However, their SA was outperformed by other metaheuristics in terms of number of best solutions found and average optimality gap. Knowles and Corne [22] also contrasted a SA for DCMST with other metaheuristics. However, their SA was outperformed by a genetic algorithm presented in the same work.

Other metaheuristics for this problem are based on Ant Colony Optimization (ACO) approaches [27, 28, 29], Particle Swarm Optimization (PSO) methods [30, 31], and local search methods [32, 33, 34, 35]. Among the ACO-based metaheuristics, the most prominent of them is that of Bui, Deng, and Zrncic [29] which employed local search strategies within the ACO framework. Regarding the PSO-based metaheuristics, the one proposed by Ernst [31] outperformed the method given by Binh and Nguyen [30]. The former algorithm employed a Lagrangian method within PSO to calculate lower bounds on the objective, which demonstrated to further improve the PSO convergence. Lastly, regarding the local search methods, there is no direct comparison between the results of the proposed algorithms. Thus, we cannot draw any conclusions regarding their performance.

A closely related problem is the min-degree constrained minimum spanning tree problem (mDCMST). In this problem, one is interested in computing a minimum-cost spanning tree of a graph whose non-leaf

nodes have a degree greater than or equal to K. Similarly to DCMST, this related problem is also NP-hard in the case that $K = 2$ [9] and in the case that $K \geq 3$ [36]. The most recent heuristic for this problem is the hybrid artificial bee colony algorithm (hABC) [37]. This algorithm employs local search strategies within the artificial bee colony algorithm, which outperforms all other heuristics from the literature for this problem. Furthermore, Cunha, Simonetti, and Lucena [38] presented a Branch-and-Cut algorithm for the mDCMST that demonstrated to be the best exact algorithm for this problem.

Another closely related problem is the bi-objective degree-constrained minimum spanning tree problem (B-DCMST), which was first studied by Knowles and Corne [39]. In B-DCMST, one minimizes both the weight of the spanning tree and the maximum degree of its nodes. This problem is also NP-hard as it is a generalization of the DCMST [9]. Knowles and Corne [39] proposed an instance generator for the B-DCMST and solved it using an evolutionary algorithm denominated archived elitist steady-state evolutionary algorithm. Later, Goldbarg, Souza, and Goldbarg [40] developed a particle swarm optimization algorithm that obtained the best results so far for the B-DCMST.

A third closely related problem is the Bounded-degree Minimum-radius Spanning Tree problem Ghosh and their collaborators [41]. This problem also has two objectives. The first is to minimize the maximum degree of the nodes of the tree, and the second is to minimize the maximum distance between their nodes. This problem was only solved through approximate algorithms, whereas the best of them is the (4-4)-bicriteria approximation rate of An and Cho [42].

Another related problem is the multi-constrained minimum spanning tree problem [43]. In this NP-hard problem, the objective is to find a minimum-cost spanning tree subject to two additional constraints: (i); the maximum degree D of the nodes; and (ii) the budget c for constructing the network. The authors proposed a branch-and-bound exact algorithm for this problem.

A last related problem is the Degree-Constrained k-Minimum Spanning Tree Problem [44]. This problem consists of finding a minimum cost subtree of G formed with at least k vertices where the degree of each vertex is less than or equal to an integer value $d \leq k - 2$. This problem, as presented by Adasme and Firoozabadi [44], is NP-Complete. The authors proposed three mathematical programming-based algorithms for the problem and two heuristics: an ant colony optimization approach and a variable neighborhood search. Computational experiments contrasted the performance of the exact algorithms and demonstrated that the ant colony optimization heuristics outperformed the variable neighborhood search ones for the proposed instances.

Despite the volume of work in the literature, a good-quality SA was never proposed to solve DCMST. Furthermore, no work employed the NPE data structure, which obtained excellent results for the minimum-cost bounded-error minimum spanning tree problem [16]. Therefore, in this work we implement and evaluate the NPE-SA, contrasting its results with those of the HGA of Singh and Sundar [11] and an extension of the node-depth phylogenetic-based artificial immune system (NPE-NSAIS) of Carvalho and Ribeiro [16].

3 The Node-depth Phylogenetic-based Simulated Annealing heuristic for DCMST

This section proposes the NPE-SA for DCMST. As stated in Section 1, this heuristic is a classic SA algorithm built using the NPE data structure [15]. First, Section 3.1 describes the SA algorithm. Next, Sections 3.2 and 3.3 present the NPE data structure along with its operators. Finally, Section 3.4 combines SA and NPE in the NPE-SA algorithm and shows how it was used to solve DCMST.

3.1 The Simulated Annealing algorithm

The Simulated Annealing algorithm (SA) [14] is a widely employed metaheuristic in both linear and integer optimization. It is one of the best-known metaheuristics for solving difficult global optimization problems [45]. This algorithm mimics the physical annealing process of metals, in which a metal piece is heated in a controlled manner and then cooled. It was proposed after the Metropolis algorithm [46] and developed to overcome the drawbacks of the Monte-Carlo approach of the latter method.

The pseudo-code of SA is presented in Algorithm 1. Without loss of generality, this pseudo-code represents the usage of SA for optimizing a minimization problem, as is the case of DCMST. It receives as input an initial solution (or state) s_0 and the number of solutions L generated in each iteration. Besides that, it also receives as input the initial temperature c and the cooling factor $\alpha \in [\,0, 1\,]$. Initially, line 1 sets s_{best} as the initial given solution. The loop from line 2 to 9 represents the iterations of the method, which is performed until c is smaller than a small constant $\epsilon = 0.01$. The loop within lines 3 and 8 generates L different solutions in every iteration of SA. Line 4 computes a new solution s'_k using a local search neighborhood operator with s_k as starting point. Then, on line 5, if the objective function value f of the solution s'_k is smaller than that of s_k, we persist the former solution in line 6. Else, in line 7, SA still accepts a non-improving solution with

Algorithm 1: Simulated Annealing

input : s_0, c, L, α
output: final state s_{best}

```
1  s_best ← s_0
2  while c > ε do
3      for l ← 0 to L do
4          s_k ← neighbor(s_best)
5          if (f(s_k) < f(s_best)) then
6              s_best ← s_k
7          else if ( (f(s_best) − f(s_k)) / (k_B c) > U(0,1) ) then
8              s_best ← s_k
9      c ← c α
10 return s_best
```

probability $\frac{f(s_k)-f(s'_k)}{k_B\, c}$, where k_B is the Boltzmann constant. Line 9 updates the temperature c using the cooling factor α. Finally, the best solution found s_{best} is returned in the last line.

The main characteristic of SA is to accept non-improving solutions, i. e., solutions with worse objective function value than the best solution previously found by the algorithm. In the first iterations of SA, the value of the temperature c is high, which makes it possible to accept non-improving solutions in lines 9 and 10. Thus, SA can focus on exploring the search space in its initial iterations. However, as the value of c diminishes through the iterations of the algorithm, it becomes harder to accept non-improving solutions. Thus, in later iterations, SA focuses on improving the best solution found s_{best} by working as a local search algorithm. We refer the reader interested in a deeper discussion about SA and their operators to the work of Delahaye, Chaimatanan, and Mongeau [45].

3.2 The Node-depth Phylogenetic encoding

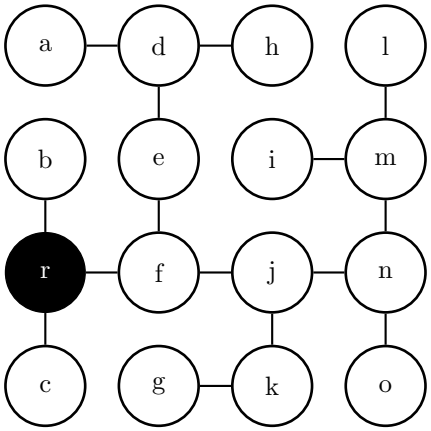
The Node-depth Phylogenetic-based Encoding (NPE) [15] is a data structure for indirectly representing spanning trees. NPE possesses good coverage and feasibility, which are desired characteristics for representing this class of problems [16]. This data structure can also be easily adapted to incorporate additional problem constraints, such as the maximum number of hops and the maximum degree of the tree nodes.

3.2.1 Solution representation

NPE is an indirect spanning tree encoding. It is represented by an ordered list of pairs $\mathcal{S} = [\, n_j, d_j \,]$, where n_j is a node and d_j denotes the node's depth on the tree. It is worth noting that $|\mathcal{S}| = |N|$, i. e., there exists an entry in $\mathcal{S}$ for every node of the graph. Figure 1 shows an example of a spanning tree of a graph (Figure 1a) and its corresponding NPE (Figure 1b). In this figure, r is the root node of the tree. One may observe that the NPE is computed by applying a Depth-First Search algorithm into the spanning tree [15].

3.2.2 Decoding operator

The NPE decoding algorithm iterates the ordered list of pairs $\mathcal{S}$ from the second to the last position. It assign edges between nodes n_j and n_i of $\mathcal{S}$ if $j < i$, $d_i - d_j = 1$, and $i - j$ is minimum. Algorithm 2 presents the pseudocode of the decoding algorithm. It receives as input a NPE $\mathcal{S}$ and returns a set of edges $E' \subseteq E$ that represents a spanning tree of G. Line 2 initializes a set of edges E' that will contain the edges of the graph, while Line 2 initializes a lookup table that associates depths d_i to the last node n_i found at this depth. As the first node is the root of the tree, the lookup table is initialized with r at its first position, i. e., at index 0, and inserts the root node at depth 0 on the lookup table. The loop from Line 2 to Line 2 iterates over all other indices of $\mathcal{S}$ except for the first one, because it contains no parents and was already inserted on the lookup table in Line 2. Line 2 finds the predecessor n_j of n_i by using the lookup table to retrieve the last node found at depth $d_i - 1$. Line 2 creates an edge between n_j and n_i and stores it on E'. Finally, Line 2 inserts n_i on the lookup table at position d_i (which corresponds to the depth of n_i). This operation replaces the previous node found at position d_i if necessary. The time complexity of the decoding algorithm is $\mathcal{O}(|N|)$ [15].



(a)

Node	r	b	f	e	d	a	h	j	n	m	l	i	o	k	g	c
Depth	0	1	1	2	3	4	4	2	3	4	5	5	4	3	4	1

(b)

Figure 1: A spanning tree with root r (a) and its corresponding NPE (b)

Algorithm 2: NPE decoding algorithm

input : NPE $\mathcal{S}(n, d)$
output: set containing $|\mathcal{S}| - 1$ edges

$E' \leftarrow \varnothing$
$lookup \leftarrow [\, n_0 \,]$
for $i \leftarrow 1$ **to** $|\mathcal{S}| - 1$ **do**
  $j \leftarrow lookup_{[\, d_i - 1 \,]}$
  $E' \leftarrow E' \cup \{\, (n_j, n_i) \,\}$
  $lookup[\, d_i \,] \leftarrow n_i$
return E'

3.3 Mutation operators

NPE employs two different mutation operators: (*i*) the Subtree Pruning and Regrafting on NPE (SPRN); and (*ii*) the Tree Bisection and Reconnection on NPE (TBRN). This section presents the concepts of both operators. The reader interested in implementation details of SPRN and TBRN are referred to the works by Lima and their collaborators [15] and Carvalho and Ribeiro [16].

Both mutation operators have two phases. The first is to remove one edge of the tree, thus resulting in two connected subgraphs. This edge removal is performed by selecting one node $\delta \in N$ at random (except the root node) that will be considered the root of subgraph G_2. Thus, it constructs a subgraph rooted at δ by traversing the NPE from n_δ to n_γ, such that $d_\gamma < d_\delta$. Every node in this traversal, excluding d_γ, is inserted in G_2. Additionally, G_2 also contains all edges from the NPE that connect two nodes in G_2. Finally, it randomly removes an existing edge in the NPE from γ to another node $\psi \in N : \psi \notin G_2$. One may observe that, in this point, subgraph G_1 is also constructed containing all nodes and edges that are not in G_2 and all edges that were previously contained in the NPE $\big($except edge $(\gamma, \psi)\big)$.

This first phase is demonstrated in Figure 2. Figure 2a represents a graph and Figure 2b displays a possible spanning tree of this graph along with its NPE. In this example, we remove edge $(\, r, d \,)$, as displayed in Figure 2c. This operation results in two subgraphs G_1 and G_2, whereas the subgraph G_1 contains the root node and is represented within a dotted border, while subgraph G_2 is contained within a dashed border.

The second phase of the mutation operators is to reconnect both resulting subgraphs as a new spanning tree. The operators differ from each other in how this reconnection is performed. The second phase of the mutation operators are presented below.

3.3.1 Subtree Pruning and Regrafting on NPE

SPRN is based on the Subtree Prune and Regraft (SPR) method for rearranging trees in PTR [15]. SPRN works similarly as the Node-depth Encoding mutation operator 1 of Delbem and their collaborators [47]

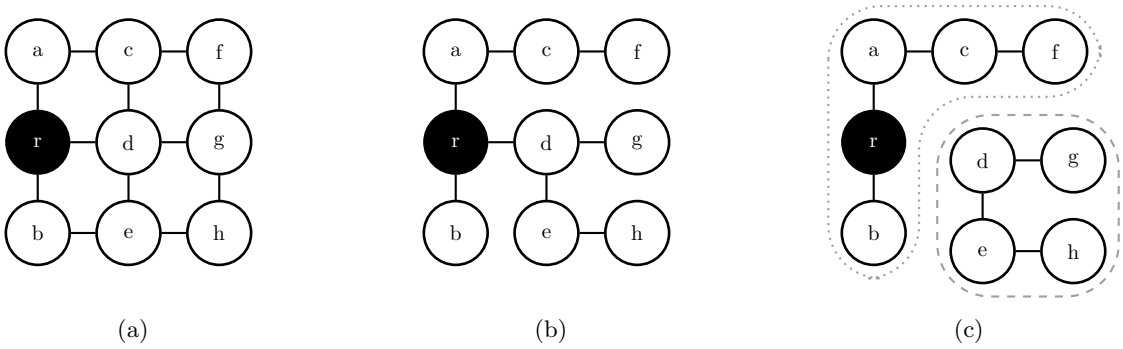


Figure 2: Example of a graph (a), one of its spanning trees to be mutated (b) and the subtrees after the removal of an edge (r, d) to be replaced by one of NPE's operators (c).

when employed on a single tree. However, this implementation has a speed improvement over its predecessor operator, achieving a time complexity of $\mathcal{O}(|N|)$ [15]. The idea of SPRN is to remove a branch from the tree and reconnect it to a different node without modifying the relative depths between the nodes of the branch.

The SPRN reconnection operator focuses on reconnecting G_1 and G_2 using node δ as the connection point. Thus, it randomly selects an edge $(i, \delta) \in E : i \in G_1 \setminus \{\psi\}$ and inserts it into the graph. This operation ensures that the resulting structure is a valid spanning tree and that it is different from the original one. Finally, the depths of the nodes are corrected and the complete NPE is reconstructed.

3.3.2 Tree Bisection and Reconnection on NPE

TBRN operator has a time complexity of $\mathcal{O}(|N|)$ and is based on the Tree Bisection and Reconnection (TBR) method to rearrange trees in PTR [15]. This operator works exactly as the second Edge-set based Encoding mutation operator [24] and the Node-depth Encoding mutation operator 2 [47] when applied on a single tree, but with speed improvements. The idea of TBRN is to remove a branch of the tree and reconnect it to a different node. However, differently from SPRN, the TBRN operator focuses on modifying the relative depths between the nodes of the branch.

The TBRN reconnection operator focuses on reconnecting G_1 and G_2 using a node $\rho \in G_2$, such that $\rho \neq \delta$, as the connection point. Thus, it randomly selects an edge $(i, \rho) \in E : i \in G_1$ and inserts it into the graph. As SPRN operator, TBRN also ensures that the resulting structure is a valid spanning tree and that it is different from the original one. Finally, the depths of the nodes are updated and the complete NPE is reconstructed.

3.4 The proposed heuristic

The NPE-SA for DCMST implements the SA algorithm using the NPE data structure. The pseudo-code of this algorithm is shown in Algorithm 3. It receives all parameters of the SA algorithm given in Algorithm 1 as input. Additionally, it also receives as input the parameter $tbrn_p \in [0, 1]$, which indicates the probability of employing the TBRN mutation operator. This parameter also controls the probability of applying the SPRN mutation operator, which is given by $1 - tbrn_p$. The output of this algorithm is the best solution found s_{best}. One can observe that NPE-SA employs a single NPE as the solution. Therefore, its space complexity is linearly bounded by the number of nodes in the input graph, as stated in Section 3.2.1.

NPE-SA implements the TBRN and the SPRN mutation operators as the neighbor function of SA. Thus, Line 3 of Algorithm 3 replaces Line 1 of Algorithm 1. In the NPE-SA algorithm, we generate a new solution s_k using the TBRN mutation operator with probability $tbrn_p$. Otherwise, we generate a new solution s_k using the SPRN operator.

One may observe that the SPRN and the TBRN mutation operators, as presented in Section 3.3, can generate spanning trees that do not respect the degree constraint of the DCMST. Thus, we deal with this problem, we modify DCMST's objective function to include a static penalty [48] when an invalid solution was produced by NPE-SA. This penalty factor remains constant during the entire evolutionary process and the penalty factor increases with the number of violated constraints, e. g., the number of nodes with a degree greater than D in the DCMST [49]. As TBRN and SPRN operators insert only one edge at a time, the maximum number of violated constraints by NPE-SA is two.

Algorithm 3: The NPE-SA for DCMST

input : $s_0, c, L, \alpha, tbrn_p$
output: final state s_{best}

1 $s_{best} \leftarrow s_0$
2 **while** $c > \epsilon$ **do**
3 **for** $l \leftarrow 0$ **to** L **do**
4 **if** $tbrn_p > \mathcal{U}(0,1)$ **then**
5 $s_k \leftarrow \text{TBRN}(s_{best})$
6 **else**
7 $s_k \leftarrow \text{SPRN}(s_{best})$
8 **if** $(f(s_k) < f(s_{best}))$ **then**
9 $s_{best} \leftarrow s_k$
10 **else if** $\left(\frac{f(s_{best})-f(s_k)}{k_B\, c} > \mathcal{U}(0,1)\right)$ **then**
11 $s_{best} \leftarrow s_k$
12 $c \leftarrow c\,\alpha$
13 **return** s_{best}

4 Computational experiments

The computational experiments were performed on a single core of an Intel Core i5-7200u with 2.5 Ghz clock and 8 Gb of RAM, running the Ubuntu Linux version 23.04 operating system. NPE-SA was implemented in Python 3 from scratch. The stopping criteria for NPE-SA were the value of c (as described in Line 3 of Algorithm 3); a total of 100.000 objective function evaluations; and a total of 500 consecutive generations without improvement in the objective function value.

We contrasted the proposed NPE-SA with the Hybrid Genetic Algorithm (HGA) of Singh and Sundar [11], the best metaheuristic in the literature for DCMST. Additionally, we evaluated NPE-SA against a modified version of the Node-depth Phylogenetic-based CLONALG algorithm (NPE-CLONAL). This algorithm is a single-objective variant of the Node-depth Phylogenetic-based Non-dominated Sorting Artificial Immune System (NPE-NSAIS) of Carvalho and Ribeiro [16], a metaheuristic based on the NPE data structure for optimizing multi-objective network design problems. AS DCMST is a single-objective problem, NPE-CLONAL replaced the non-dominated sorting operators of NPE-NSAIS with the original CLONALG mechanisms [50]. NPE-CLONAL is also suitable for solving DCMST, as this problem can be considered a network design problem, such as stated in Section 1.

HGA and NPE-CLONAL were also implemented in Python 3 from scratch. To perform a fair comparison, both algorithms employed two stopping criteria: (*i*) a total of 100.000 objective function evaluations; and (*ii*) a total of 500 consecutive generations without improvement in the objective function value.

4.1 Instance's description

This work employed two classic instance sets for DCMST that were previously used in the computational experiments of Singh and Sundar [11]:

- **CRD**: In this set, the nodes are randomly positioned using a uniform distribution into a two-dimensional plane and the edge's cost are equal to the Euclidean distance between them.
- **SYM**: In this set, the nodes are randomly positioned using a uniform distribution into a five-dimensional plane. As in CRD set, the edge's cost is equal to the Euclidean distance between them.

Both instance sets are complete graphs. We built CRD instances with 50, 60, 70, 80, 90, and 100 nodes. Additionally, SYM instances were constructed with 50, 55, 60, 65, and 70 nodes. The higher the number of nodes, the more difficult is to solve DCMST. We highlight that this number of nodes was choose after the computational experiments of Singh and Sundar [11], which is the work that originally proposed the HGA.

We proposed 10 different instances for each combination of instance set and number of nodes. Therefore, our computational experiments evaluated a total of 60 CRD instances and 50 SYM instances. All instances were developed using a different seed for the Mersenne Twister pseudo-random number generator [51].

Finally, we evaluated DCMST for $D = \{\, 2, 3, 4, 5 \,\}$. Thus, for every proposed instance, we generated four different copies with a different value of D. Consequently, our computational experiments evaluated a total of $(60 + 50) \cdot 10 \cdot 4 = 4400$ different benchmark instances.

4.2 Hyperparameter optimization

The optimal parameters of HGA were drawn from its original work [11]. For NPE-SA and NPE-CLONAL, we performed a hyperparameter optimization experiment using the Optuna package [52], a newly proposed hyperparameter optimization framework. The parameters of the algorithms were optimized using the Tree-structured Parzen Estimator, which is the default Optuna's hyperparameter optimization algorithm.

The hyperparameter optimization experiment was carried out in five additional instances of each instances set. These instances were randomly generated with different numbers of nodes and values of D. Therefore, we ensure NPE-SA and NPE-CLONAL parameters are not biased towards the proposed set of instances. The results of the hyperparameter optimization experiment were drawn after 30 independent runs with different seeds for the algorithm pseudo-random number generator.

The results of the hyperparameter optimization experiment are reported in Table 1. The first column gives the name of the algorithm, while the second column presents the parameter being optimized. The third column displays the range of tests, i. e., the minimum and maximum values considered for each parameter. Finally, the fourth column presents the best parameter found in this experiment. Additionally, for NPE-SA, we fixed $L = 1$. These optimized parameters were employed in the remaining of this work.

Table 1: Hyperparameter optimization for NPE-CLONAL and NPE-SA

Algorithm	Parameter	Range	Optimal value
NPE-CLONAL	Number of clones	$[30, 90] \in \mathbb{N}$	80
	$tbrn_p$	$[0, 1] \in \mathbb{R}$	0.2956
	Population size	$[5, 25] \in \mathbb{N}$	16
NPE-SA	Initial temperature c	$[10, 20] \in \mathbb{R}$	13.3634
	$tbrn_p$	$[0, 1] \in \mathbb{R}$	0.1005
	Cooling factor α	$[0, 1] \in \mathbb{R}$	0.9860

4.3 Comparison of the algorithms

Tables 2 to 5 compares HGA, NPE-CLONAL, and NPE-SA for the DCMST for values of $D = 2, 3, 4$, and 5, respectively. The first column of each table shows the instance set, while the second column gives the number of nodes of the instances. The third and fourth columns display, respectively, the average and the standard deviation of HGA for 30 independent runs with different seeds for the algorithm pseudo-random number generator. The same information is given for NPE-CLONAL in the fifth and sixth columns and for NPE-SA in the last two columns. All algorithms ran in less than 5 seconds. Therefore, due to the inexpressive computational effort needed to run the algorithms, we opted not to include their running time in our computational experiments.

Table 2: Results for $D = 2$

		HGA [11]		NPE-CLONAL		NPE-SA	
Set	Nodes	Average	Dev	Average	Dev	Average	Dev
CRD	50	1621.0	77.2	2534.3	117.8	825.8	72.3
	60	2151.4	87.1	2934.6	145.5	1003.7	93.2
	70	2647.3	89.7	3614.4	134.7	1194.5	121.3
	80	2791.1	96.4	3998.9	172.9	1398.1	120.8
	90	3605.9	237.4	4530.2	180.9	1661.9	114.8
	100	4133.0	127.7	5012.3	181.8	1893.0	158.4
SYM	50	3021.9	384.4	3681.1	346.8	2062.7	443.6
	55	3277.6	453.1	3937.5	504.4	2150.5	399.4
	60	3683.1	487.3	4379.9	394.3	2368.1	458.3
	65	4034.6	451.1	4720.0	370.7	2641.4	511.1
	70	4414.2	484.3	5167.2	557.9	2775.6	544.0

One can see from these results that NPE alone was not able to provide better solutions for DCMST, as NPE-CLONAL found worse results than HGA and NPE-SA for all evaluated instances. On the other hand, NPE-SA demonstrated to surpass HGA for all of the evaluated instance sets, achieving a smaller average

Table 3: Results for $D = 3$

		HGA [11]		NPE-CLONAL		NPE-SA	
Set	Nodes	Average	Dev	Average	Dev	Average	Dev
CRD	50	950.4	58.5	2327.1	151.5	503.0	29.7
	60	1256.8	82.4	2743.7	156.2	559.3	18.4
	70	1590.6	82.5	3315.1	95.0	581.4	17.1
	80	1888.7	86.3	3795.9	150.6	634.3	16.2
	90	2275.8	115.9	4296.2	149.9	684.6	18.6
	100	2683.1	113.3	4814.6	176.0	768.9	22.1
SYM	50	2262.1	387.6	3577.5	329.2	1471.9	345.4
	55	2450.1	449.6	3763.2	505.8	1557.5	392.9
	60	2757.8	465.2	4166.1	495.6	1661.1	394.8
	65	3031.4	454.5	4531.2	347.5	1728.3	383.3
	70	3336.9	472.4	4968.7	507.8	1842.3	424.9

Table 4: Results for $D = 4$

		HGA [11]		NPE-CLONAL		NPE-SA	
Set	Nodes	Average	Dev	Average	Dev	Average	Dev
CRD	50	958.6	64.0	2226.8	103.7	500.3	28.9
	60	1344.1	71.8	2728.4	113.5	549.2	16.5
	70	1670.4	112.2	3224.2	92.2	589.4	16.7
	80	2083.3	120.1	3702.4	168.7	640.1	18.1
	90	2547.7	125.6	4139.4	165.6	690.6	13.4
	100	2926.8	162.4	4656.8	178.6	751.5	16.6
SYM	50	2032.8	348.2	3562.4	339.1	1460.7	331.9
	55	2338.0	442.5	3735.3	498.5	1537.3	384.2
	60	2591.7	472.8	4156.6	406.7	1645.8	390.7
	65	2922.1	390.8	4568.1	462.0	1711.1	379.9
	70	3128.6	501.1	4795.0	470.0	1818.8	414.0

Table 5: Results for $D = 5$

		HGA [11]		NPE-CLONAL		NPE-SA	
Set	Nodes	Average	Dev	Average	Dev	Average	Dev
CRD	50	989.5	75.6	2196.2	130.4	499.0	28.8
	60	1223.7	73.5	2704.5	147.0	544.9	16.8
	70	1524.8	66.3	3125.7	106.9	583.1	16.0
	80	1878.8	72.3	3635.8	122.2	636.4	16.6
	90	2250.3	100.8	4118.0	173.7	677.0	9.8
	100	2664.6	99.1	4638.6	169.0	739.1	15.4
SYM	50	2088.6	337.3	3539.6	375.1	1459.6	333.5
	55	2246.2	438.5	3676.5	474.6	1538.4	383.0
	60	2561.1	433.9	4092.3	592.2	1651.0	393.2
	65	2766.4	403.3	4362.5	322.3	1718.3	380.5
	70	3113.4	466.0	4757.3	515.7	1827.5	416.3

objective function value. A similar conclusion is drawn regarding the standard deviations of the algorithms. In all tables, NPE-SA demonstrated to be a more stable algorithm, achieving a smaller deviation on the 30 different runs than HGA and NPE-CLONAL, whereas NPE-CLONAL had the largest standard deviation in most of the cases.

We observed in the computational experiments that the main drawback of the three heuristics is to build and maintain valid solutions, as their evolutionary operators tend to generate spanning trees with one

or more nodes with degrees higher than D. Thus, we believe that NPE-SA outperformed both HGA and NPE-CLONAL because it evolves a single solution, a characteristic of SA algorithms. Therefore, NPE-SA focuses on a single good-quality solution and does not waste computation in searching non-valid spanning trees, as is the case of HGA and NPE-CLONAL. Furthermore, the cooling mechanism of the SA algorithm helps in avoiding local optima early and focusing on a single good-quality solution later.

4.3.1 Statistical analysis

One can see from Tables 2 to 5 that NPE-SA outperformed the other algorithms in terms of mean results. Therefore, it indicates that NPE-SA is the best among the algorithms evaluated in optimizing the proposed DCMST benchmark objective functions. To assess this observation, an experimental analysis of the data was carried out following the statistical methodology of Derrac and their collaborators [53]. This methodology has three steps, as described below. All steps assume a significance level $\alpha = 0.05$, i.e., the null hypothesis is rejected if a p-value smaller or equal to 0.05 is obtained. We analyzed the data of Tables 2 to 5 separately. Thus, we could draw independent conclusions for all values of D.

In the first step, a Shapiro-Wilk normality test is applied to verify if the average fitness values obtained by the algorithms follow a normal distribution. For all values of D, the Shapiro-Wilk test found a p-value of 0.001 for all algorithms, thus indicating that the data does not follow a normal distribution. Therefore, a non-parametrical statistical test will be employed in the next steps.

The second step was to apply a Friedman's Test to verify whether the average fitness values of one of the heuristics statistically differ from that of another. The null hypothesis is that the three algorithms achieve the same average results. The input data for the Friedman's test is ranked according to the methodology proposed by Carvalho [54]. The Friedman's test obtained a p-value smaller than 0.05, for all values of D, thus rejecting the null hypothesis and indicating that exist a significant difference between the average results values of, at least, two of the algorithms evaluated in this work.

In the third step, we applied a post-hoc test for Friedman's, namely the Nemenyi's test. This statistical test compares the results of multiple algorithms and evaluates the pair of hypothesis

$$\begin{cases} H_0: & \mu_i \leq \mu_j \\ H_1: & \mu_i \neq \mu_j, \end{cases} \qquad \forall (\mu_i, \mu_j) \in W,$$

whereas $W = \{\mu_{\text{HGA}}, \mu_{\text{NPE-CLONAL}}, \mu_{\text{NPE-SA}}\}$, such that $\mu_{\text{HGA}}, \mu_{\text{NPE-CLONAL}}$, and $\mu_{\text{NPE-SA}}$ are respectively the average ranking obtained by HGA, NPE-CLONAL, and NPE-SA in the second step. The null hypothesis (H_0) affirms that the average rankings μ_i and μ_j of all pairs of algorithms do not significantly differ from each other. Thus, the null hypothesis affirms that all algorithms have similar average rankings. On the other hand, the alternative hypothesis (H_1) suggests that the average rankings given by an algorithm μ_i statistically differ from those of another algorithm μ_j.

This post-hoc test rejected the null hypothesis and indicated that $\mu_{\text{NPE-SA}} \leq \mu_j : j \in W \setminus \mu_{\text{NPE-SA}}$. Thus, we can affirm that there exists a significant difference between the average rankings of NPE-SA and the other two evaluated in this work. Additionally, the post-hoc test also demonstrated that HGA outperformed NPE-CLONAL for all values of D. Finally, we conclude, for all values of D, that NPE-SA is the best among the evaluated heuristics in solving the proposed benchmark objective functions.

5 Conclusions and future work

This paper proposed the Node-depth Phylogenetic-based Simulated Annealing heuristic (NPE-SA) for the degree-constrained minimum spanning tree problem (DCMST). The NPE-SA incorporates the traditional Simulated Annealing metaheuristic with the Node-depth Phylogenetic-based Encoding, an efficient data structure for representing spanning trees. Computational experiments performed on classic instances from the literature demonstrated the capabilities of our method, whereas it outperformed the Hybrid Genetic Algorithm (HGA) [11] and a Node-depth Phylogenetic-based CLONAL algorithm (NPE-CLONAL), which is an adaptation of the Node-depth Phylogenetic-based Artificial Immune System algorithm [16] for a single objective. Furthermore, computational experiments demonstrated that the NPE data structure cannot guarantee good results when solving network design problems. Therefore, it strengthens the need to develop appropriate search mechanisms to obtain good-quality solutions for DCMST.

Future work may have three focuses. The first is to further improve NPE-SA for the DCMST or embed it into a matheuristic for this problem, using similar strategies as those presented in Maniezzo, Stützle, and Voß [55]. The second is to generalize NPE-SA for other constrained spanning tree problems from the literature. As can be seen in Section 3.4, the proposed heuristic can be easily adapted to other problems by modifying the penalty function. The third focus is to build and validate a new instance repository

for constrained minimum spanning tree problems, following the recent work of Carvalho and Coco [56]. The objective is to build a repository and instance generator similar to that of Osaba, Villar-Rodriguez, and Romero [57] for packing problems and that of Uchoa and their collaborators [58] for vehicle routing problems.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, the Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq), and the Fundação de Amparo à Pesquisa do Estado de Minas Gerais - Brasil (FAPEMIG).

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Minimum spanning trees," in *Introduction to Algorithms*, 4th ed. MIT Press Cambridge, 2022, pp. 585–603.

[2] A. M. de Almeida, P. Martins, and M. C. de Souza, "Min-degree constrained minimum spanning tree problem: complexity, properties, and formulations," *International Transactions in Operational Research*, vol. 19, no. 3, pp. 323–352, 2012.

[3] H. Akcan, "On the complexity of energy efficient pairwise calibration in embedded sensors," *Applied Soft Computing*, vol. 13, no. 4, pp. 1766 – 1773, 2013.

[4] N. Deo and P. Micikevicius, "A heuristic for a leaf constrained minimum spanning tree problem," in *Congressus Numerantium*. Utilitas Mathematica, 1999, pp. 61–72.

[5] L. Gouveia, A. Paias, and D. Sharma, "Modeling and solving the rooted distance-constrained minimum spanning tree problem," *Computers & Operations Research*, vol. 35, no. 2, pp. 600–613, 2008.

[6] T. F. Noronha, C. C. Ribeiro, and A. C. Santos, "Solving diameter-constrained minimum spanning tree problems by constraint programming," *International Transactions in Operational Research*, vol. 17, no. 5, pp. 653–665, 2010.

[7] I. A. Carvalho and M. A. Ribeiro, "An exact approach for the minimum-cost bounded-error calibration tree problem," *Annals of Operations Research*, vol. 287, no. 1, pp. 109–126, 2020.

[8] A. E. Ezugwu, A. K. Shukla, R. Nath, A. A. Akinyelu, J. O. Agushaka, H. Chiroma, and P. K. Muhuri, "Metaheuristics: a comprehensive overview and classification along with bibliometric analysis," *Artificial Intelligence Review*, vol. 54, pp. 4237–4316, 2021.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability*. Freeman San Francisco, 1979, vol. 174.

[10] M. Savelsbergh and T. Volgenant, "Edge exchanges in the degree-constrained minimum spanning tree problem," *Computers & Operations Research*, vol. 12, no. 4, pp. 341–348, 1985.

[11] K. Singh and S. Sundar, "A hybrid genetic algorithm for the degree-constrained minimum spanning tree problem," *Soft Computing*, vol. 24, no. 3, pp. 2169–2186, 2020.

[12] A. B. Kahng and G. Robins, *On optimal interconnections for VLSI*. Springer Science & Business Media, 1994, vol. 301, ch. 2, pp. 16–63.

[13] R. Ravi, M. V. Marathe, S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, "Approximation algorithms for degree-constrained minimum-cost network-design problems," *Algorithmica*, vol. 31, pp. 58–78, 2001.

[14] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[15] T. W. de Lima, A. C. B. Delbem, A. da Silva Soares, F. M. Federson, J. B. A. L. Junior, and J. V. Baalen, "Node-depth phylogenetic-based encoding, a spanning-tree representation for evolutionary algorithms. part i: Proposal and properties analysis," *Swarm and Evolutionary Computation*, vol. 31, pp. 1–10, 2016.

[16] I. A. Carvalho and M. A. Ribeiro, "A node-depth phylogenetic-based artificial immune system for multi-objective network design problems," *Swarm and Evolutionary Computation*, vol. 50, p. 100491, 2019.

[17] S. C. Narula and C. A. Ho, "Degree-constrained minimum spanning tree," *Computers & Operations Research*, vol. 7, no. 4, pp. 239–249, 1980.

[18] B. Boldon, N. Deo, and N. Kumar, "Minimum-weight degree-constrained spanning tree problem: Heuristics and implementation on an simd parallel machine," *Parallel Computing*, vol. 22, no. 3, pp. 369–382, 1996.

[19] L. Caccetta and S. P. Hill, "A branch and cut method for the degree-constrained minimum spanning tree problem," *Networks: An International Journal*, vol. 37, no. 2, pp. 74–83, 2001.

[20] L. H. Bicalho, A. S. Da Cunha, and A. Lucena, "Branch-and-cut-and-price algorithms for the degree constrained minimum spanning tree problem," *Computational Optimization and Applications*, vol. 63, pp. 755–792, 2016.

[21] G. Zhou and M. Gen, "A note on genetic algorithms for degree-constrained spanning tree problems," *Networks: an International Journal*, vol. 30, no. 2, pp. 91–95, 1997.

[22] J. Knowles and D. Corne, "A new evolutionary approach to the degree-constrained minimum spanning tree problem," *IEEE Transactions on Evolutionary computation*, vol. 4, no. 2, pp. 125–134, 2000.

[23] G. R. Raidl and B. A. Julstrom, "A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem," in *Proceedings of the 2000 ACM symposium on Applied computing-Volume 1*, 2000, pp. 440–445.

[24] ——, "Edge sets: an effective evolutionary coding of spanning trees," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 225 – 239, 6 2003.

[25] F. Rothlauf and D. Goldberg, "Tree network design with genetic algorithms–an investigation in the locality of the Pruefernumber encoding," in *Late Breaking Papers at the Genetic and Evolutionary Computation Conference*, 1999, pp. 238 – 244.

[26] M. Krishnamoorthy, A. T. Ernst, and Y. M. Sharaiha, "Comparison of algorithms for the degree constrained minimum spanning tree," *Journal of Heuristics*, vol. 7, no. 6, pp. 587 – 611, 11 2001.

[27] M. N. Doan, "An effective ant-based algorithm for the degree-constrained minimum spanning tree problem," in *2007 IEEE Congress on Evolutionary Computation*. IEEE, 2007, pp. 485–491.

[28] Y.-T. Bau, C. K. Ho, and H. T. Ewe, "Ant colony optimization approaches to the degree-constrained minimum spanning tree problem," *Journal of Information Science and Engineering*, vol. 24, no. 4, pp. 1081–1094, 2008.

[29] T. N. Bui, X. Deng, and C. M. Zrncic, "An improved ant-based algorithm for the degree-constrained minimum spanning tree problem," *IEEE Transactions on evolutionary Computation*, vol. 16, no. 2, pp. 266–278, 2012.

[30] H. T. T. Binh and T. B. Nguyen, "New particle swarm optimization algorithm for solving degree constrained minimum spanning tree problem," in *PRICAI 2008: Trends in Artificial Intelligence*, T.-B. Ho and Z.-H. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1077–1085.

[31] A. T. Ernst, "A hybrid lagrangian particle swarm optimization algorithm for the degree-constrained minimum spanning tree problem," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[32] C. C. Ribeiro and M. C. Souza, "Variable neighborhood search for the degree-constrained minimum spanning tree problem," *Discrete Applied Mathematics*, vol. 118, no. 1-2, pp. 43–54, 2002.

[33] W. Wamiliana, "Solving the degree constrained minimum spanning tree problem using tabu and modified penalty search methods," *Jurnal Teknik Industri: Jurnal Keilmuan dan Aplikasi Teknik Industri*, vol. 6, no. 1, pp. 1–9, 2004.

[34] M. Zahrani, M. J. Loomes, J. Malcolm, and A. A. Albrecht, "A local search heuristic for bounded-degree minimum spanning trees," *Engineering Optimization*, vol. 40, no. 12, pp. 1115–1135, 2008.

[35] P. Martins and M. C. de Souza, "Vns and second order heuristics for the min-degree constrained minimum spanning tree problem," *Computers & Operations Research*, vol. 36, no. 11, pp. 2969–2982, 2009.

[36] A. M. de Almeida, P. Martins, and M. C. Souza, "md-mst is np-hard for d ¿= 3," *Electronic Notes in Discrete Mathematics*, vol. 36, pp. 9–15, 2010.

[37] S. Ghoshal and S. Sundar, "Two approaches for the min-degree constrained minimum spanning tree problem," *Applied Soft Computing*, vol. 111, p. 107715, 2021.

[38] A. S. da Cunha, L. Simonetti, and A. Lucena, "A strong symmetric formulation for the min-degree constrained minimum spanning tree problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 237–244, 2016.

[39] J. D. Knowles and D. W. Corne, "Benchmark problem generators and results for the multiobjective degree-constrained minimum spanning tree problem," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 2001, pp. 424–431.

[40] E. F. G. Goldbarg, G. R. de Souza, and M. C. Goldbarg, "Particle swarm optimization for the bi-objective degree constrained minimum spanning tree," in *2006 IEEE International Conference on Evolutionary Computation*. IEEE, 2006, pp. 420–427.

[41] A. Ghosh, Ö. D. Incel, V. A. Kumar, and B. Krishnamachari, "Multichannel scheduling and spanning trees: Throughput–delay tradeoff for fast data collection in sensor networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1731–1744, 2011.

[42] M. K. An and H. Cho, "Construction of bounded-degree minimum-radius spanning trees for wsns," in *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2025, pp. 95–102.

[43] T. J. Kumar and P. Singamsetty, "An exact algorithm for multi-constrained minimum spanning tree problem," *International Journal of Mathematics in Operational Research*, vol. 12, no. 3, pp. 317–330, 2018.

[44] P. Adasme and A. Dehghan Firoozabadi, "Degree-constrained k-minimum spanning tree problem," *Complexity*, vol. 2020, no. 1, p. 7628105, 2020.

[45] D. Delahaye, S. Chaimatanan, and M. Mongeau, "Simulated annealing: From basics to applications," in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. Cham: Springer International Publishing, 2019, pp. 1–35. [Online]. Available: https://doi.org/10.1007/978-3-319-91086-4_1

[46] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[47] A. C. B. Delbem, A. de Carvalho, C. A. Policastro, A. K. O. Pinto, K. Honda, and A. C. Garcia, "Node-depth encoding for evolutionary algorithms applied to network design," in *Genetic and Evolutionary Computation – GECCO 2004*, K. Deb, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 678 – 687.

[48] A. Smith and D. Coit, "Constraint-handling techniques," in *Handbook of Evolutionary Computation*, T. Baeck, D. B. Fogel, and Z. Michalewicz, Eds. Boca Raton: CRC Press, 1997, pp. 344–377.

[49] E. Mezura-Montes and C. A. C. Coello, "Constraint-handling in nature-inspired numerical optimization: past, present and future," *Swarm and Evolutionary Computation*, vol. 1, no. 4, pp. 173–194, 2011.

[50] L. N. de Castro and F. J. V. Zuben, "Learning and optimization using the clonal selection principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 239 – 251, 6 2002.

[51] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3 – 30, 1 1998.

[52] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[53] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[54] I. A. Carvalho, "On the statistical evaluation of algorithmic's computational experimentation with infeasible solutions," *Information Processing Letters*, vol. 143, pp. 24–27, 2019.

[55] V. Maniezzo, T. Stützle, and S. Voß, *Matheuristics*. Springer, 2021.

[56] I. A. Carvalho and A. A. Coco, "Data, instance sets, and instances generator for the hop-constrained minimum spanning tree problem, the delay-constrained minimum spanning tree problem, and their bi-objective variants," *Data in Brief*, vol. 50, p. 109553, 2023.

[57] E. Osaba, E. Villar-Rodriguez, and S. V. Romero, "Benchmark dataset and instance generator for real-world three-dimensional bin packing problems," *Data in Brief*, vol. 49, p. 109309, 2023.

[58] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, no. 3, pp. 845–858, 2017.