A Node-depth Phylogenetic-based Artificial Immune System for Multi-objective Network Design Problems

Article *in* Swarm and Evolutionary Computation · November 2019

DOI: 10.1016/j.swevo.2019.01.007

2 authors:

Iago Augusto Carvalho
Universidade Federal de Alfenas
44 PUBLICATIONS **100** CITATIONS
SEE PROFILE

Marco Ribeiro
Federal University of Minas Gerais
10 PUBLICATIONS **48** CITATIONS
SEE PROFILE

A Node-depth Phylogenetic-based Artificial Immune System for Multi-objective Network Design Problems

Iago A. Carvalho[a,*], Marco A. Ribeiro[a]

[a]*Department of Computer Science, Universidade Federal de Minas Gerais*

Abstract

Network Design Problems (NDP) constitute a traditional class of combinatorial optimization problems. They usually rely on finding an optimal tree on a graph that respects the particular constraints of the problem at hand. When using evolutionary algorithms to solve NDP, one can use specific encodings to represent a tree. A newly proposed tree encoding in the literature is the Node-depth Phylogenetic-based encoding (NPE), which possess small time and space complexity, among other desirable characteristics. In this work, we propose the first metaheuristic on the literature that implements the NPE, the Node-depth Phylogenetic-based Non-dominated Sorting Artificial Immune System (NPE-NSAIS). We assess the quality of the proposed algorithm by solving a bi-objective NDP, the Minimum-Cost Bounded-Error Calibration Tree problem (MBCT). Our results reveal that the NPE-NSAIS outperforms the state-of-the-art algorithm for the MBCT. Moreover, we show that the NPE-NSAIS outperforms three other NSAIS algorithms that employ

*Corresponding author

Email addresses: iagoac@dcc.ufmg.br (Iago A. Carvalho), marcoantonio@dcc.ufmg.br (Marco A. Ribeiro)

URL: https://iagoac.github.io/ (Iago A. Carvalho), http://marcoribeiro.me/ (Marco A. Ribeiro)

different encodings, the NSGA-II, and the SPEA2.

Keywords: Artificial Immune Systems, Solution Encodings, Network Design Problems, Node-depth Phylogenetic-based Encoding, Bi-objective Optimization

---

1. Introduction

Evolutionary Algorithms (EA) have been widely applied to solve combinatorial optimization problems [1, 2]. A particular class of such problems are the Network Design Problems (NDP). NDP rely on a spanning tree representation, which imposes additional difficulties for EA's [3]. Thus, a crucial aspect when building an EA to a NDP is the choice of its solution's encoding scheme [4].

A proper solution's encoding scheme should have small time and space complexity, good feasibility, locality, and heritability, among other aspects [5]. Several encoding schemes for NDP were proposed in the literature. One can classify these encodings into two groups: direct and indirect [4].

Direct encodings, such as the Edge-set Encoding [5], represents a tree as the set of its edges. Thus, one can apply the evolutionary operators directly into this edge set. On the other hand, indirect encodings, such as the Network Random Keys Encoding [6] and the Node-depth Encoding [7], uses a genotype-phenotype mapping where the representation is the solution's genotype, and the decoded solution is the phenotype. This mapping is done through a decoding algorithm, whose complexity should be the smaller as possible [4].

A newly proposed spanning tree encoding is the Node-depth Phylogenetic-

based Encoding (NPE) [8], an indirect spanning tree representation based on the idea of phylogenetic tree reconstruction [9]. The NPE presents desirable characteristics, such as small space and time complexity, good feasibility and high locality [8]. The authors proposed two mutation operators for the NPE. Despite its features, no EA using NPE was introduced in the literature.

The main objective of this work is to propose and evaluate the first EA using the NPE. Provided that the literature does not supply a recombination operator, it cannot be adequately embedded into classical EA such as Genetic Algorithms [10] or Swarm Evolutionary Algorithms [11]. Therefore, we propose the Non-dominated Sorting Artificial Immune System (NSAIS), a hybrid metaheuristic for multi-objective problems that employs concepts from the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [12] and from the Clonal Selection Algorithm (CLONALG) [13, 14]. Provided that NSAIS lies on mutating several copies of a solutions' pool, we can appropriately embed the NPE on it, resulting on the Node-depth Phylogenetic-based Non-dominated Sorting Artificial Immune System (NPE-NSAIS). To assess the quality of the NPE-NSAIS, we employ it to solve the Minimum-Cost Bounded-Error Calibration Tree problem (MBCT) [15], a bi-objective wireless sensors network optimization problem.

We organize the remainder of this paper as follows. Section 2 reviews the most prominent EA's spanning tree encodings. Section 3 presents the NPE and its operators. We formally introduce the NSAIS in Section 4. Next, Section 5 presents the MBCT, which is used on Section 6 to evaluate the NSAIS embedded with different encodings. Section 5 also compares the NSAIS with the NSGA-II [12] and the SPEA2 [16]. Finally, the last section

draws the concluding remarks of this paper.

2. Spanning tree encodings for evolutionary algorithms

EA's performance strongly correlates with its parameter's setting and solutions' encoding scheme [17]. The former adapts an EA to particular classes of problems [18] and controls the exploration vs. exploitation ratio [19]. The latter defines how it internally represent solutions and the way that evolutionary operators interact with them [4].

A proper solution encoding should consider several aspects [5]:

- *Space*: The chromosome representation cannot require an exponential space to represent a solution.
- *Time*: The recombination, mutation and evaluation operators should be computationally tractable. When representing spanning trees, the evaluation may include decoding a chromosome to identify the spanning tree it represents.
- *Feasibility*: All chromosomes need to represent a feasible solution. Besides that, the recombination and mutation operators also should generate feasible solutions.
- *Coverage*: The encoding scheme should be able to represent all feasible solutions, thus encompassing all the search space. Moreover, an optimal solution needs to be reachable through the iterative application of the evolutionary operators.

- *Bias*: All solutions should be equally likely represented. Moreover, the encoding scheme cannot favor one tree structure among others.
- *Locality*: The mutation operator should produce chromosomes that are similar to its parent, i. e., mutated solutions need to contain mostly the edges of its parent.
- *Heritability*: The recombination operator should produce an offspring that combines both parents substructure, similar to the locality aspect.
- *Constraints handling*: The chromosome decoding should be able to enforce problem-specific constraints, such as maximum/minimum node degree, maximum tree depth, or a bounded number of leaf nodes.
- *Hybrids*: The encoding's evolutionary operators should be able to incorporate problem-dependent heuristics.
- *Graph density*: The encoding scheme should be able to represent complete, dense, and sparse graphs.

In the remaining of this section we present the most prominent spanning tree encoding schemes from the literature on the light of these aspects. All encodings consider an undirected graph $G = (N, E)$, with $|N| = n$ nodes and $|E| = m$ edges. Raidl and Julstrom [5] present a summary of most of these encoding schemes. Moreover, Rothlauf's book [4] presents a detailed discussion of all encoding schemes proposed in the literature up to the time of its publication.

2.1. Characteristic Vectors Encoding

A characteristic vector is a binary structure whose positions represent graph edges [3, 20, 21]. Its decoding time complexity is linear on m. Despite its desirable time complexity, characteristic vectors have poor feasibility. On a complete graph, it is possible to build $2^{\frac{n(n-1)}{2}}$ solutions as binary permutations of such vector. However, only 2^{n-2} distinct spanning tree exists, as given by Cayley's formula [22]. Thus, several unfeasible encoding exists, and penalization or repairment strategies should be used, negatively affecting the encoding's locality and heritability.

2.2. Predecessor Encoding

A Predecessor Encoding designates an arbitrary node to represent the root of the tree [4]. Thus, it uses an integer vector $pred$ of size n where each node point to its predecessor. Thus, if $pred_i = j$, then node j is adjacent to node i. It possesses an $\mathcal{O}(n)$ time decoding algorithm. However, in this scheme, recombination and mutation operators can generate unfeasible solutions, which requires penalization [23] or repairment [24] strategies. Therefore, its locality and heritability aspects are also affected.

2.3. Prüfer Number Encoding

A Prüfer (or Pruefer) Number Encoding is based on Cayley's formula [22]. It performs an inverse mapping between spanning trees on G and vectors of length $n - 2$ over integers labeling the nodes. This encoding lacks of locality and heritability [5, 25]. Its encoding and decoding algorithms require $\mathcal{O}(n \log n)$. Moreover, they cannot be easily used on incomplete graphs, and its hybridization with ad-hoc heuristics is difficult [5, 25].

Several specializations of Prüfer number exists. The most prominent of them are the Dandelion Code, the Happy Code, and the Blob Code [26]. Both Dandelion Code and Blob Code achieve higher locality than the original Prüfer Number Encoding [27]. Other Prüfer Number's specializations are the Modified Happy Code [28], which retains better locality and heritability than Happy Code, the Stack-Queue Encoding [29], and the Rainbow Code [30].

2.4. Link-and-Node Biased Encoding

Another encoding is the Link-and-Node Biased Encoding, proposed by Palmer and Kershenbaum [3]. It encodes a spanning tree as a numeric vector of size n whose positions associate to the graph's nodes, in such a way that any numeric vector represents a valid chromosome. Furthermore, it employs the Prim's algorithm to decode the solution. Therefore its time complexity is $\Theta(m \log n)$. The link-and-node biased encoding can handle problem constraints with an additional complexity cost, depending on the constraint. Despite that, it cannot represents all possible spanning trees. Nevertheless, one can overcome this limitation by using a vector representing the edge weights [31]. However, such strategy increases its space complexity from n to $\frac{n(n+1)}{2}$.

2.5. Edge-set Encoding

The Edge-set Encoding directly represents a spanning tree as a set of their edges [5]. Consequently, it possesses an $\mathcal{O}(n)$ space and time complexity when implemented using hash tables. Additionally, it has good locality and heritability and can be hybridized with ad-hoc heuristics. However, it is biased towards minimum spanning trees, since its operators use Kruskal's

or Prim's algorithms [32]. Finally, one can easily apply it to represent both complete, dense, and sparse graphs without additional cost.

2.6. Network Random Keys Encoding

The Network Random Keys Encoding [6] is based on the idea of using random keys to encode permutations [33]. It employs an m-dimensional vector v_e, where each position corresponds to an edge on the graph, and contains a randomly generated real number. The decoding algorithm consists in sorting the weight vector and applying Kruskal's algorithm considering the values on v_e as the edge weights. Therefore, its time complexity is $\mathcal{O}(m \log n)$, bounded by the complexity of the sorting algorithm. Besides, since it uses a Minimum Spanning Tree algorithm, the network random keys encoding has a strong bias towards star-like trees, e. g., trees with low diameter and depth [5].

2.7. Node-depth Encoding

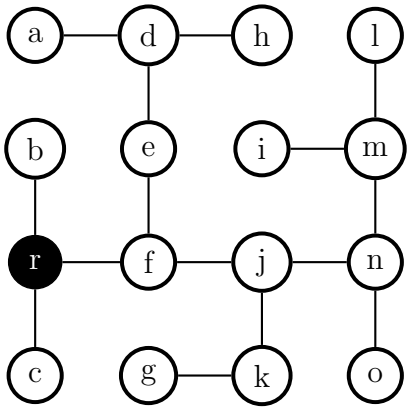
The Node-depth Encoding [7] is an indirect encoding focused on representing spanning forests. It represents each spanning tree as an arranged list of pairs of n positions, where each pair contains the node number and its depth on the tree. Thus, it has a $\mathcal{O}(n)$ space complexity. Moreover, its decoding can be done in linear time using a strategy presented by Delbem et al. [7]. It has good coverage and feasibility and can represent both complete and sparse graphs. Nevertheless, its initialization operator maintans a bias towards star-like structures [34]. A work by de Lima et al. [35] showed that its recombination operators are also biased toward star-like structures but have good heritability.

2.8. Node-depth-degree Encoding

The Node-depth-degree Encoding [36] is a generalization of the Node-depth Encoding [7, 34] that deals with forests. It represents each tree as an arranged list of triples of n positions, where the first two are the same as in the Node-depth Encoding, and the third represents the node's degree. This last item helps improve the time complexity on its operators. As in the Node-depth Encoding, its decoding algorithm has $\mathcal{O}(n)$ time complexity. However, its operators generate new forests in $\mathcal{O}(\sqrt{n})$, on average. This encoding also retains high locality, feasibility, and coverage. Moreover, it can represent both complete, dense, and sparse graphs. To the best of our knowledge, there is no study regarding it biases, heritability, and locality in the literature.

2.9. Node-depth Phylogenetic-based Encoding

The Node-depth Phylogenetic-based Encoding [8] is another extension of the Node-depth Encoding [7, 34], but focused on representing single trees. It represents its chromosome in the same way as the Node-depth Encoding does. Therefore, it also has good coverage and feasibility. The decoding algorithm has $\mathcal{O}(n)$ time complexity. Besides, its mutation and initialization operators are not biased and have high locality [8]. However, the literature does not provide recombination operators for this encoding. We will describe the Node-depth Phylogenetic-based encoding and its operators in details on the following section.



(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 at node r (a) and its corresponding NPE (b).

3. The Node-depth Phylogenetic-based Encoding

The Node-depth Phylogenetic-based Encoding (NPE) [8] is an indirect spanning tree encoding. It consists of an ordered list of pairs $[\, n_x, d_x \,]$, where n_x is a node and d_x is its depth on the tree. Figure 1 shows an example of a spanning tree of a graph (Figure 1a) and its corresponding NPE (Figure 1b), where node r is the root node of the tree. One can construct the corresponding NPE by applying a Depth-First Search algorithm into the spanning tree [8].

In the remaining of this section, we describe the operators of the NPE encoding. First, we present its decoding operator, which translates a NPE into a spanning tree. Then, we show two NPE mutation operators: (i) the Subtree Pruning and Regrafting on NPE (SPRN); and (ii) the Tree Bisection and Reconnection on NPE (TBRN). Both operators were derived from sim-

Algorithm 1: NPE decoding.

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

```
1 edges ← ∅
2 lookup ← [ n_0 ]
3 for i ← 1 to |S| − 1 do
4 |   j ← lookup[ d_i − 1 ]
5 |   edges ← edges ∪ { (n_j, n_i) }
6 |   lookup[ d_i ] ← n_i
7 return edges
```

ilar methods on Phylogenetic Tree Reconstruction (PTR). A complete and most detailed description of the three operators, along with some analysis regarding their bias and locality can be found in de Lima et al. [8].

3.1. Decoding operators

The NPE decoding algorithm traverses the corresponding array from the second to the last position. It assigns an edge between nodes n_j and n_i, if $j < i$, $d_i - d_j = 1$, and $i - j$ is minimum. Algorithm 1 presents the pseudocode of the decoding algorithm. It receives as input a NPE $\mathcal{S}$ representing a spanning tree of a graph $G = (N, E)$. Lines 1 and 2 initialize the structures used by the algorithm. The former initializes a set *edges* that will hold the edges of the graph. The latter initializes a lookup-table associating depths d_i to the last node n_i found at depth d_i. Since the first node is the root of the tree, the lookup-table is initialized as a list containing this node at its first position, i. e., at index 0. Line 3 iterates over all indices on $\mathcal{S}$, except the root, because it contains no parents and was already inserted on the lookup-table. Line 4 finds the predecessor n_j of n_i by using the lookup-table to retrieve the

Algorithm 2: SPRN Operator.

input : NPE $\mathcal{S}(n, d)$
output: mutated NPE $\mathcal{S}'(n', d')$

$i_p \leftarrow \mathcal{U}\{1,\ |\mathcal{S}| - 1\}$
$e_p \leftarrow i_p + 1$
while $e_p < |\mathcal{S}|$ **and** $d_{e_p} > d_{i_p}$ **do**
    $e_p \leftarrow e_p + 1$
$i_a \leftarrow$index of random neighbor of n_{i_p} not in $[\, i_p, e_p \,)$
$n' \leftarrow \{$ nodes on n from 0 to $(i_a - 1)$ excluding the range $[\, i_p, e_p \,) \}$
$d' \leftarrow \{$ depths on d from 0 to $(i_a - 1)$ excluding the range $[\, i_p, e_p \,) \}$
for $j \leftarrow i_p$ **to** $e_p - 1$ **do**
    $n' \leftarrow n' \cup \{\, n_j \,\}$
    $d' \leftarrow d' \cup \{\, d_j - d_{i_p} + d_{i_a} + 1 \,\}$
$n' \leftarrow n' \cup \{$ nodes on n from i_a to $|\mathcal{S}| - 1$ excluding the range $[\, i_p, e_p \,) \}$
$d' \leftarrow d' \cup \{$ depths on d from i_a to $|\mathcal{S}| - 1$ excluding the range $[\, i_p, e_p \,) \}$
return $\mathcal{S}'(n', d')$

last node found at depth $d_i - 1$. Line 5 creates an edge between n_j and n_i, and stores it on the edges' set. Finally, Line 6 inserts n_i on the lookup table at its depth d_i, replacing the previous one found at d_i if necessary. As stated before, the decoding algorithm time complexity is $\mathcal{O}(n)$.

3.2. Mutation operators

The NPE uses two different mutation operators: (*i*) the Subtree Pruning and Regrafting on NPE; and (*ii*) the Tree Bisection and Reconnection on NPE. In this section, we present those operators and how they affect a NPE chromosome.

3.2.1. Subtree Pruning and Regrafting on NPE

The Subtree Pruning and Regrafting on NPE (SPRN) operator is based on the Subtree Prune and Regraft (SPR) method for rearranging trees in

PTR [8]. This operator works exactly as the Node-depth Encoding mutation operator 1 [7] when applied on a single tree, but with improvements on speed.

Algorithm 2 describes how the SPRN works. Line 1 starts by selecting the index i_p of the node on $\mathcal{S}$ to be removed by generating a random index using a discrete uniform distribution. It excludes the root node since it cannot be pruned. Lines 2 to 4 find the end e_p of the range of nodes started at i_p that represents the subtree that will be pruned and regrafted, i. e., reinserted. Line 5 selects a random neighbor i_a of n_{i_p}, outside the range $[\, i_p, e_p \,)$, that will receive the new edge. The next steps consist in inserting the nodes on the mutated NPE. The first step (Lines 6 and 7) copies the nodes and depths from $\mathcal{S}$ to $\mathcal{S}'$, until it reaches i_a, skipping any node in the range $[\, i_p, e_p \,)$. The second step (Lines 8 to 10) inserts the nodes and depths on the pruned subtree into $\mathcal{S}'$. It fixes each node's depth by finding its relative depth $(d_j - d_{i_p} + 1)$ and adding it to the new depth of the subtree's root (d_{i_a}). Finally, the third step (Lines 11 and 12) copies the remaining nodes and depths from $\mathcal{S}$ to $\mathcal{S}'$.

We created an example on Figures 2 and 3 to help visualize better how does the SPRN operator works. In all the images, the root node r is highlighted in black. Edges being added to the tree have a wider stroke and the nodes linked by them are in bold on the NPE. Edges to be added are dashed. Empty slots on the NPE contain a single dash. Figure 2a shows a graph, while Figure 2b displays the tree to be mutated and its respective NPE. Figure 2c selects $(\, r, d \,)$ as the removed edge, meaning that $i_p = 4$ and $e_p = 8$. It also presents both subtrees resulting from the edge removal, highlighting the r's subtree with a dotted border and d's subtree with a dashed border. Besides the removed edge, d only connects to c outside its subtree, thus we

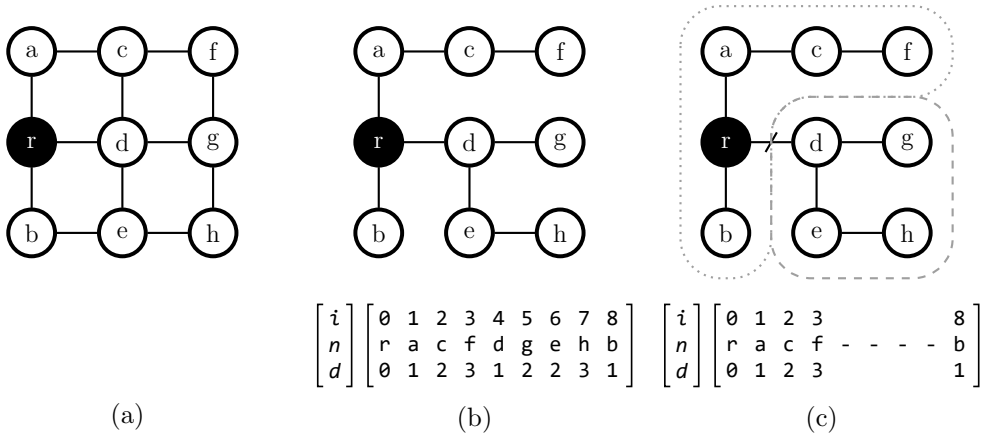


Figure 2: Example of a graph (a), one of its spanning trees to be mutated (b) and the subtrees after the removal of an edge (r, d) to be replaced by one of NPE's operators (c).

choose $i_a = 2$. Figure 3a shows the first step of the SPRN algorithm, where it inserts the nodes with indices from 0 to i_a into the mutated NPE. Figure 3b displays the second step, where it inserts d's subtree into the mutated NPE, updating the weights. Finally, Figure 3c shows the third step, where it inserts the nodes with indices from $i_a + 1$ until $|\mathcal{S}| - 1$, excluding the range $[\, i_p, e_p \,)$, into the mutated NPE.

3.2.2. Tree Bisection and Reconnection on NPE

The Tree Bisection and Reconnection on NPE (TBRN) operator is based on the Tree Bisection and Reconnection (TBR) method for rearranging trees in PTR [8]. This operator works exactly as the second Edge-set based Encoding mutation operator [5], and as the Node-depth Encoding mutation operator 2 [7] when applied on a single tree, but with improvements on speed.

Algorithm 3 describes how the TBRN works. As on the SPRN, Line 1

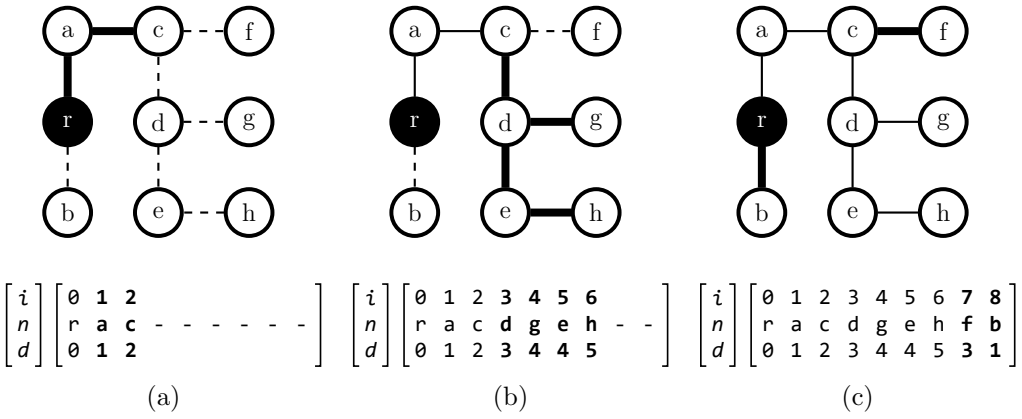


Figure 3: SPRN operator's steps.

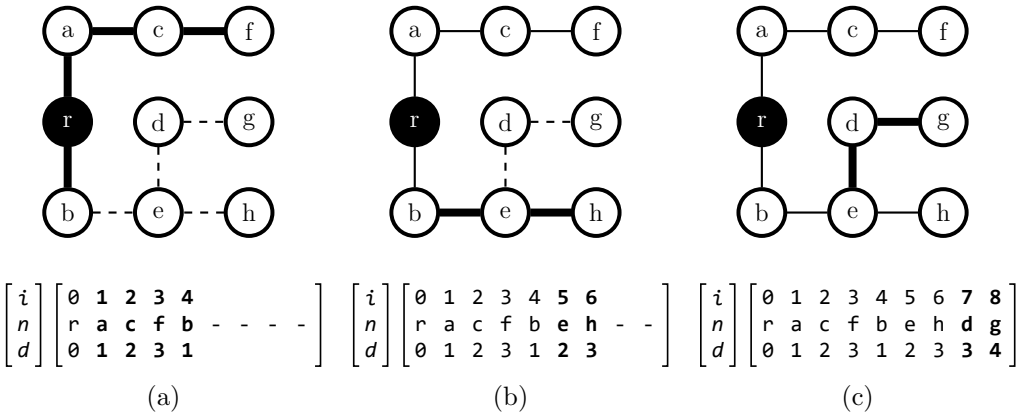
randomly selects the index i_p of the node on $\mathcal{S}$ to be removed using a discrete uniform distribution, ignoring the root node. Lines 2 to 6 find the end e_p and of the range of nodes started at i_p that represents the subtree that will be pruned and regrafted. They also build a set R_p containing all the nodes on the subtree. Line 7 selects a random edge with an endpoint $r \in R_p$ and the other $a \in n \setminus R_p$. This edge will reconnect both subtrees. The next steps consist in inserting the nodes on the mutated NPE. The first step (Lines 10 and 11) copies the nodes and depths from $\mathcal{S}$ to $\mathcal{S}'$, until it reaches i_a, skipping the range $[\, i_p, e_p \,)$. The second step (Lines 12 to 25) reinserts the range R_p into $\mathcal{S}'$. It starts from the subtree rooted at n_{i_r}, and iteratively inserts the parent subtrees, in a bottom-up fashion, ignoring the nodes already added. It stops after entirely inserting the subtree rooted at n_{i_p}. Finally, the third step (Lines 26 and 27) copies the remaining nodes and depths from $\mathcal{S}$ to $\mathcal{S}'$. One can notice that this operator executes a similar operation as the Edge-set encoding mutation.

As in the SPRN, we created an example on Figure 4 for the TBRN starting

Algorithm 3: TBRN Operator.

input : NPE $\mathcal{S}(n, d)$
output: mutated NPE $\mathcal{S}'(n', d')$

$i_p \leftarrow \mathcal{U}\{1,\ |\mathcal{S}| - 1\}$
$e_p \leftarrow i_p + 1$
$R_p \leftarrow \{\, n_{i_p} \,\}$
while $e_p < |\mathcal{S}|$ **and** $d_{e_p} > d_{i_p}$ **do**
    $R_p \leftarrow R_p \cup \{\, n_{e_p} \,\}$
    $e_p \leftarrow e_p + 1$
$(r, a) \leftarrow$ random edge with first endpoint in R_p and last in $(n \setminus R_p)$
$i_r \leftarrow$ index of node r on n
$i_a \leftarrow$ index of node a on n
$n' \leftarrow \{\,$ nodes on n from 0 to $(i_a - 1)$ excluding the range $[\, i_p, e_p \,)\,\}$
$d' \leftarrow \{\,$ depths on d from 0 to $(i_a - 1)$ excluding the range $[\, i_p, e_p \,)\,\}$
$i_x \leftarrow 0$
$par \leftarrow i_r$
$R_x \leftarrow \varnothing$
while $i_x \neq i_p$ **do**
    $i_x \leftarrow par$
    $par \leftarrow$ first index on $\mathcal{S}$ smaller than i_x with depth equal to $(d_{i_x} - 1)$
    $e_x \leftarrow i_x$
    do
        if $e_x \notin R_x$ **then**
            $R_x \leftarrow R_x \cup \{\, e_x \,\}$
            $n' \leftarrow n' \cup \{\, n_{e_x} \,\}$
            $d' \leftarrow d' \cup \{\, d_{e_x} - d_{par} + d_{i_x} + 1 \,\}$
        $e_x \leftarrow e_x + 1$
    while $e_x < |\mathcal{S}|$ **and** $d_{e_x} > d_{i_x}$
$n' \leftarrow n' \cup \{\,$ nodes on n from i_a to $|\mathcal{S}| - 1$ excluding the range $[\, i_p, e_p \,)\,\}$
$d' \leftarrow d' \cup \{\,$ depths on d from i_a to $|\mathcal{S}| - 1$ excluding the range $[\, i_p, e_p \,)\,\}$
return $\mathcal{S}'(n', d')$



(a) (b) (c)

Figure 4: TBRN operator's steps.

from the configuration on Figure 2. It follows the same conventions stated on the SPRN. Different from the SPRN, the TBRN can choose any edge connecting a node on d's subtree and a node on r's subtree. Thus, the possibilities are $\{(d, r), (d, c), (e, b), (g, f)\}$. In this example, we choose (e, b) as the edge added, therefore, $i_r = 6$ and $i_a = 8$. Figure 4a shows the first step of the algorithm, where it inserts all nodes with indices from 0 to i_a, excluding the range $[i_p, e_p)$. The second step takes two iterations to insert all nodes on the pruned subtree into the mutated NPE, displayed, respectively, on Figure 4b and Figure 4c. The former adds the subtree rooted at e into the mutated NPE. The latter inserts the subtree rooted at d, ignoring the nodes already added. Finally, the third step does nothing in this example since i_a already points to the end of $\mathcal{S}$.

4. The Non-dominated Sorting Artificial Immune System

In this section, we describe the proposed Non-dominated Sorting Artificial Immune System (NSAIS) algorithm. In Section 6, we evaluate four

variants of the NSAIS, each one embedding a different encoding: (*i*) the Network Random-Keys (NRK-NSAIS); (*ii*) the Edge-set Encoding (ESE-NSAIS); (*iii*) the Node-depth Encoding (NDE-NSAIS); and (*iv*) the Node-depth Phylogenetic-based Encoding (NPE-NSAIS).

The NSAIS is a hybrid metaheuristic between the NSGA-II [12] and the CLONALG [13, 14], hence being an instance of an Artificial Immune System [37]. It employs the NSGA-II's *Non-dominated Sorting*, *Crowding-Distance Assignment*, and *Crowded-Comparison Operator*. In addition, it uses the clonal selection theory [38] and the concept of antigen-driven affinity [39] as a metaheuristic optimization algorithm, similar to the CLONALG. Thus, NSAIS inherits CLONALG's *Cloning* and *Hypermutation* operators. Finally, the NSAIS's *Selection Operator* hybridizes the selection operators of both algorithms.

Algorithm 4 presents a pseudo-code of the NSAIS. It receives two parameters as input: the size of the solution's set $psize \in \mathbb{N}_{>0}$; and the cloning scale $cs \in \mathbb{N}_{>0}$, which determines the number of copies generated of each solution on the cloning operator. Line 1 creates the initial solution's set $\mathcal{P}$, which is evaluated on Line 2. The loop on Lines 3 to 8 executes until a stopping criterion is met. Each iteration of this loop corresponds to one generation of the NSAIS. First, Line 4 clones the current solution's set by through the CLONALG's cloning operator. Then, Line 5 apply the hypermutation operator, and Line 6 evaluates the new computed solution's set. Finally, Line 7 selects a subset of the cloned solutions to undergo to the next generation. The algorithm stops and returns the resulting solution's set on Line 8. In the remaining of this section, we detail each of the NSAIS operators.

Algorithm 4: Non-dominated Sorting Artifical Immune System

input : $psize \leftarrow$ population size
input : $cs \leftarrow$ cloning scale
output: list of $psize$ individuals

```
initialize(P, psize)
evaluate(P)
while not stop-condition do
    P' ← clone(P, cs)
    hypermutate(P')
    evaluate(P')
    P ← select(P', psize)
return P
```

4.1. Solution evaluation

Initially, NSAIS evaluates the fitness of the solution's set. Then, NSAIS employs the NSGA-II's *Non-dominated Sorting* procedure. Thus, the next step consists in computing the *Nondomination Fronts* and assigning a *Crowding Distance* to each solution. We describe those operations below.

Let $\mathcal{Z} = \{ Z_1, \ldots, Z_n \}$ be the set of objective functions of the problem at hand. Additionally, let $\mathcal{P} = \{ X_1, \ldots, X_{psize} \}$ be the current solution's set. We denote as X_i^j the value of objective $Z_j \in \mathcal{Z}$ achieved by solution $X_i \in \mathcal{P}$. A solution $X_i \in \mathcal{P}$ dominates solution $X_k \in \mathcal{P}$ if X_i^j is better or equal than X_k^j for all objectives $Z_j \in \mathcal{Z}$, and X_i^j is strictly better than X_k^j for at least one objective $Z_j \in \mathcal{Z}$. Otherwise, we say that both solutions are *non-dominated* among them. We represent this domination relationship between two solutions as $X_i \prec X_k$, i. e., solution X_i dominates solution X_k.

The non-dominated sorting algorithm divides the solution's set $\mathcal{P}$ into nondomination fronts $\mathcal{F} = \{ F_1, \ldots, F_m \}$, such that all solutions in a given front are non-dominated among them [12]. Furthermore, all solutions in

front F_i are dominated by a solution in front F_{i-1}. Therefore, the best solutions (the non-dominated solutions) are located in front F_1. We assign a *Nondomination* rank R_i to each solution, which corresponds to the number of the front where it is located.

Finally, we compute the *Crowding Distance* C_i for all solutions X_i, which is an estimation of the density of solutions on a given solution's vicinity. This is done by summing the area of the cuboid whence the solution is inserted, for each objective. We highlight that the crowding distance C_i of a given solution $X_i \in F_j$ only takes into account the solutions contained in the same front. Those values are used by the NSAIS selection operator in order to obtain a most diverse solution's set.

4.2. Cloning

The NSAIS cloning operator aims at generating several copies of the solutions in $\mathcal{P}$. It computes a new solution's set $\mathcal{P}'$, which is a matrix $A_{|X|\times(cs+1)}$ containing the original solutions plus cs copies of each one.

4.3. Hypermutation

The NSAIS hypermutation operator is dependent on the embedded encoding. We employ the mutation operators as a *Proportional Hypermutation Operator*, where the number of mutations applied into a solution is equal to its nondomination rank. Let R_i be the nondomination rank of solution $X_i \in \mathcal{P}$. The number of mutations m applied to the copies of X_i is equal to R_i. Therefore, we execute a single mutation into copies of individuals in front F_1. On the other hand, we apply m mutations into copies of individuals from the front F_m. This operator ensures that NSAIS intensifies the search

into desirable solutions while using the worst solutions to explore the search space.

4.4. Selection

The selection operator is applied into the resulting solution's set $\mathcal{P}'$ to select the *psize* solutions that will undergo to the next generation. Let X_i and X_k be two solutions in $\mathcal{P}'$. The NSAIS selection operator uses the NSGA-II's *Crowded-Comparison Operator*, which ranks the solutions of $\mathcal{P}'$ according the following rules.

1. If $R_i < R_k$, then X_i is ranked as better than X_k
2. If $R_i = R_k$ and $C_i > C_k$, then X_i is ranked as better than X_k

In both cases, we also say that $X_i \prec X_k$. Thereby, the selection operator chooses the *psize* best ranked solutions of $\mathcal{P}'$ to form the next generation's solution set whilst discarding the remaining solutions.

5. The Minimum-Cost Bounded-Error Calibration Tree problem

We assess the performance of the NPE-NSAIS by solving a newly proposed bi-objective Wireless Sensor Networks (WSN) problem, called Minimum-Cost Bounded-Error Calibration Tree problem (MBCT). The MBCT is an $\mathcal{NP}$-Hard optimization problem introduced by Akcan [15] and originally solved through a Genetic Algorithm using an Edge-set Encoding and the weighted-sum method [40].

The MBCT arises from the need of sensors' periodically calibration due to manufacturing errors or environmental conditions changes over time [41, 42].

In real-world deployments of WSN, one can calibrate sensors against a reference one using a *Pairwise Calibration* function [43]. This calibration process is usually performed in an iterative fashion [44]. Initially, a well-adjusted sensor calibrates its neighbors. Then, their neighbors calibrate uncalibrated sensors that are close by to them. This iterative process continues until it calibrates the whole network.

This process must consider two aspects. The first aspect is the distance between the calibrated sensor and its neighbors. As greater the distance between sensors, more energy is dispensed by this process. Therefore, to minimize the WSN's energy consumption, a sensor should communicate with the closest possible other [45, 46]. The second aspect is the maximum post-calibration error. The calibration of a sensor node may not be flawlessly done, thus generating a post-calibration error [43]. Considering the iterative nature of the WSN calibration, this error can be carried out to others, thus impairing the network accuracy [44].

Given a WSN, the goal of the MBCT is to compute a communication tree that possesses two objective functions. The first is to minimize the distance between sensors involved in the calibration process. The second is to minimize the maximum post-calibration error achieved by iterative calibrating sensors using this tree.

We formally define the MBCT problem as follows. Let $G = (N, E)$ be a graph that represents a WSN, where the nodes in N represent sensors and the edges in E indicate possible communication links between two sensors. Furthermore, each edge $e \in E$ holds a cost c_e representing the distance between sensors. Moreover, each node $n \in N$ carries a given maximum post-

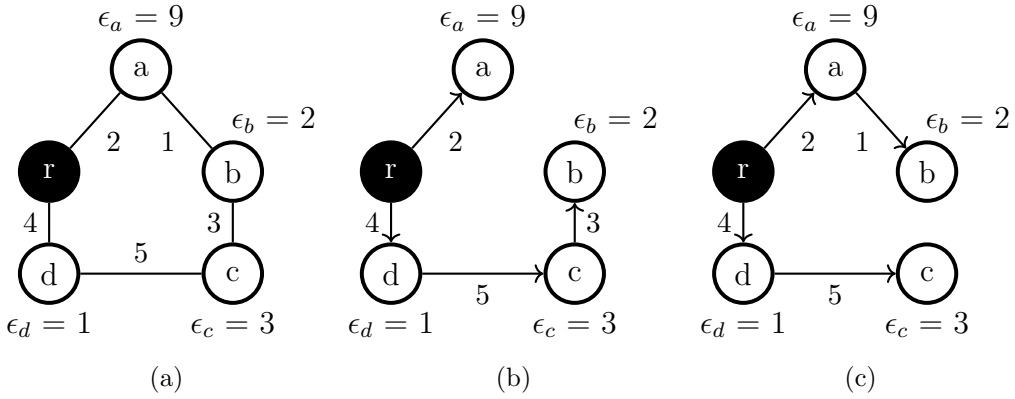


Figure 5: Example of a MBCT instance (a) and two feasible solutions for the presented instance (b) and (c). The black node represents the root of the tree. In addition, the edge direction symbolizes the orientation of the pairwise calibration among sensors.

calibration error ϵ_n. Given a root node $r \in N$, the goal of the MBCT is to compute a spanning tree T of G rooted in r that minimizes both objectives described below.

The first objective is solely the sum of the edge costs in T. The second objective is the longest path from the r to the other nodes under T, using their maximum post-calibration error as distance. For the sake of brevity, we will denote the first objective as Z_1 and the second objective as Z_2 in the remaining of the text.

Figure 5 shows a graph and two of its possible spanning trees to illustrate how to compute both objectives. Figure 5a shows a MBCT instance representing a WSN with 5 nodes and 5 edges. Figures 5b and 5c give two feasible solutions for this instance. We assume that node r represents a calibrated sensor. One can see from Figure 5b that its Z_1 value is $2+4+5+3 = 14$. In addition, it comprises two calibration paths. The first path transverse nodes $\{r, a\}$ and its maximum post-calibration error is $\epsilon_a = 9$, while the second

path transverse nodes $\{ r, d, c, b \}$ and its maximum post-calibration error is $\epsilon_d + \epsilon_c + \epsilon_b = 1 + 3 + 2 = 6$. Therefore, $Z_2 = \max\{ 6, 9 \} = 9$. Similarly, for the solution presented in Figure 5c, we have $Z_1 = 2 + 1 + 4 + 5 = 12$ and $Z_2 = \max\{ (\epsilon_a + \epsilon_b), (\epsilon_d + \epsilon_c) \} = \max\{ 11, 4 \} = 11$.

Despite having two objectives, Akcan [15] originally solved the MBCT using a mono-objective Genetic Algorithm (GA) through a weighted-sum method [40]. However, in this work, we tackle the MBCT using *Evolutionary Multi-Objective* (EMO) algorithms.

6. Computational experiments

In this section, we performed a set of computational experiments to evaluate the NSAIS when solving the MBCT. We evaluated four versions of the NSAIS, where each version employs a different encoding: (*i*) the Network Random-Keys NSAIS (NRK-NSAIS); (*ii*) the Edge-set Encoding NSAIS (ESE-NSAIS); (*iii*) the Node-depth Encoding NSAIS (NDE-NSAIS); and (*iv*) the Node-depth Phylogenetic-based Encoding NSAIS (NPE-NSAIS). In addition, we compare the NSAIS algorithms with a Genetic Algorithm for the MBCT [15], two variations of the NSGA-II [12]: (*i*) a Network Random Keys Encoding NSGA-II (NRK-NSGA) and (*ii*) an Edge-set Encoding NSGA-II (ESE-NSGA), and two variations of the SPEA2 [16]: (*i*) a Network Random Keys Encoding SPEA2 (NRK-SPEA); and (*ii*) an Edge-set Encoding SPEA2 (ESE-SPEA).

The computational experiments were performed on a single core of an AMD Opteron$^{\text{TM}}$ 6376 processor with 2.3 GHz, running under the Linux operating system. To implement the algorithms, we used `C++17` compiled

through GNU's `g++ 8.2`. Additionally, we used Mersenne Twister as the pseudo-random number generator [47]. We limited the running time of all algorithms to 120 seconds (2 minutes). The statistical tests were carried out on the `R 3.3.0`.

We used two instances' sets in the computational experiments. The first set is exactly the seven instances used by Akcan [15] for tuning his GA for the MBCT. We generated the second set in the same way as Akcan did. They consist on random graphs with $|N| \in \{ 25, 50, 100, 250, 500, 750, 1000 \}$ whose nodes' degree follow a binomial distribution $\mathcal{B}(\mu, \sigma^2)$, where $\mu \in \{ 3, 5, 10 \}$ and $\sigma^2 = 1$. Furthermore, we randomly generated the weight of the edges through a binomial distribution $\mathcal{B}(10, 2)$ and the post-calibration error from a discrete uniform distribution $\mathcal{U}\{1, 20\}$. For each possible $(|N|, \mu)$ pair, we generated 10 instances using different seeds for its pseudo-random number generator. Therefore, the second instances' set comprises a total of 210 instances.

We evaluate the algorithms using the *Hypervolume* method [48]. It computes the area under the non-dominated front F_1 as shown in Figure 6. Therefore, the higher the hypervolume value, then better is the solution's quality on the non-dominated front. Initially, we compute the nadir point by separately solving each objective at optimality as proposed by Akcan [15]. Therefore, for Z_1, we find the minimum spanning tree using Kruskal's algorithm. For Z_2, we compute shortest path tree of G by employing an adapted Dijkstra's algorithm that considers the nodes' weights instead of the edges' weights as the path cost. The nadir point receives coordinates (x_n, y_n), where x_n denotes the post-calibration error of the minimum spanning tree and y_n

represents the edge's cost of the shortest path tree. Next, we compute an augmented non-dominated front F' by adding both the minimum spanning tree and the shortest path tree points to F_1.

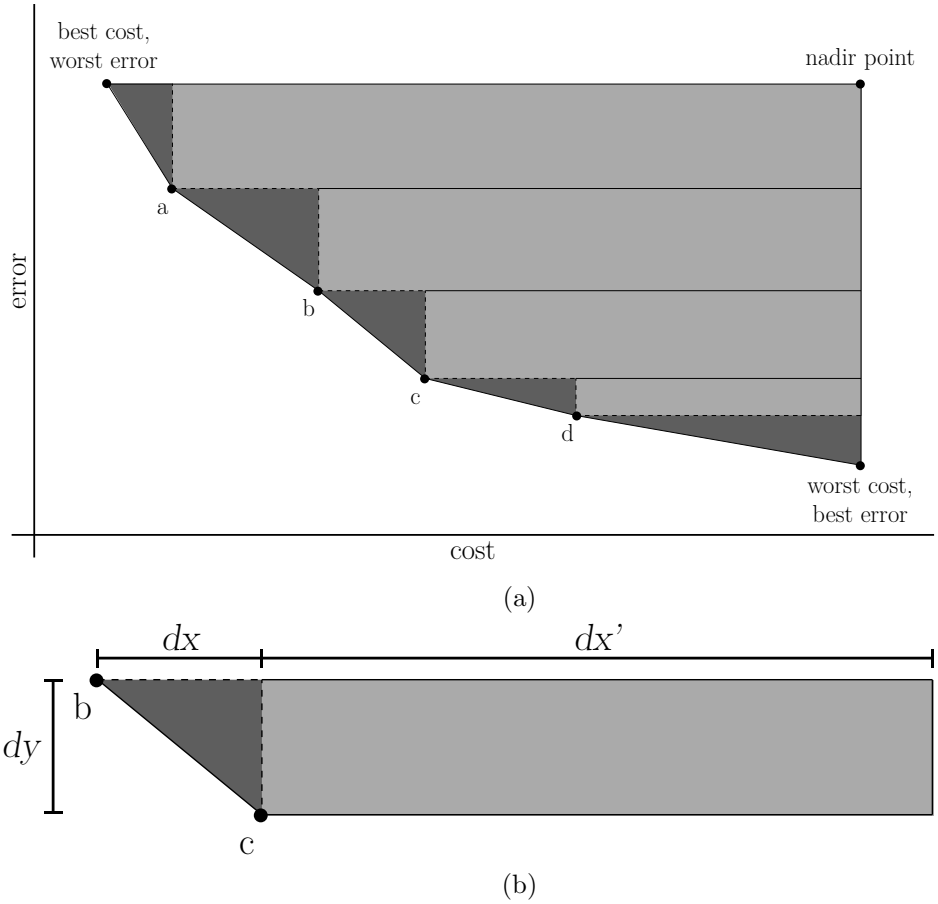


Figure 6: The area under front F' (a) and a zoom on F' indicating dx, dy, and dx' (b).

The area under F' is then computed as illustrated in Figure 6b. Let $b = (\, x_b, y_b \,)$ and $c = (\, x_c, y_c \,)$ be two consecutive points of F'. Furthermore, let $dx = |x_b - x_c|$ and $dy = |y_b - y_c|$ be the difference between the coordinates of points b and c, and $dx' = |x_n - dx|$ be the distance on the x-axis from the nadir point to $\max\{\, x_b, x_c \,\}$. We compute the area as $\frac{dx \times dy}{2} + dx' \times dy$, i. e.,

the area of the triangle (in a darker color) added to the rectangle's area (in a lighter color). The area under F' is the summation of the areas computed for all pair of consecutive points in F', sorted by cost.

6.1. Parameter tuning

The first experiment aimed to compute the optimal parameter set for all of the NSAIS and NSGA-II variants through a full factorial experiment. Since all algorithms rely on stochastic operators, we ran each parameter's combination 30 times with different initial random seeds and analyzed their average results. We performed this experiment on the seven instances used by Akcan [15] to optimize his GA parameters.

Table 1 shows the parameters of the algorithms along with their evaluated values. The first column shows the algorithm name while the second column displays its parameters' names and the intervals where they varied. The $psize$ parameter refers to the algorithm solution's set size, while cs represents the cloning scale, as defined in Section 4. For the NSGA-II variants, the mut_ρ serve as the probability of mutating an individual after crossover, and the $tsize$ refers to the tournament selection size. For the SPEA2 variants, we define $mut\rho$ and $psize$ are the same as defined above, while $asize$ represents the archive size [16]. For the NDE-NSAIS, the $op2_\rho$ indicates the probability of applying the NDE's mutation operator 2 [7]. Thus, one can determine the chance of using NDE's mutation operator 1 as the inverse of $op2_\rho$, i. e., $(1 - op2_\rho)$. Similarly, for the NPE-NSAIS, the $tbrn_\rho$ indicates the probability of applying the TBRN mutation. Thus, the chance of using the SPRN mutation is $(1 - tbrn_\rho)$. We highlight that we only evaluated the SPEA2 variants when $asize \leqslant psize$.

Table 1: Parameter's evaluated values for each algortihm.

Algorithm	**Parameters**
NRK-NSGA, ESE-NSGA	$psize \in \{ 10, 20, 30, 40, 50 \}$ $mut_\rho \in \{ 0.00, 0.05, 0.10, 0.15, 0.20 \}$ $tsize \in \{ 2, 3, 4, 5, 6 \}$
NRK-SPEA, ESE-SPEA	$psize \in \{ 10, 20, 30, 40, 50 \}$ $mut_\rho \in \{ 0.00, 0.05, 0.10, 0.15, 0.20 \}$ $asize \in \{ 10, 20, 30, 40, 50 \}$
NRK-NSAIS, ESE-NSAIS	$psize \in \{ 10, 20, 30, 40, 50 \}$ $cs \in \{ 1, 3, 5, 7, 10 \}$
NDE-NSAIS	$psize \in \{ 10, 20, 30, 40, 50 \}$ $cs \in \{ 1, 3, 5, 7, 10 \}$ $op2_\rho \in \{ 0.00, 0.25, 0.50, 0.75, 1.00 \}$
NPE-NSAIS	$psize \in \{ 10, 20, 30, 40, 50 \}$ $cs \in \{ 1, 3, 5, 7, 10 \}$ $tbrn_\rho \in \{ 0.00, 0.25, 0.50, 0.75, 1.00 \}$

Table 2 shows the results of this experiment. One can see that, on average, smaller solution's set sizes and clonal scales led to better hypervolume values. It indicates that the almost all algorithms greatly benefit from a higher number of generations than from a large number of individuals mutated each generation. Therefore, the best solution's set size for NRK-NSGA, ESE-NSGA, ESE-NSAIS, NDE-NSAIS, and NPE-NSAIS was $psize = 10$. Additionally, all NSAIS algorithms achieve better average hypervolume values with $cs = 3$. NPE-NSAIS was shown not to be much influenced by the TBRN probability parameter. However, the one who gives the smallest average fitness value was $tbrn_\rho = 0.75$, while the $op2_\rho$ parameter value which

Table 2: Best parameter's set for each algorithm.

Algorithm	**Parameters**	**Algorithm**	**Parameters**
NRK-NSGA	$psize = 30$ $mut_\rho = 0.15$ $tsize = 5$	ESE-NSGA	$psize = 10$ $mut_\rho = 0.15$ $tsize = 5$
NRK-SPEA	$psize = 40$ $mut_\rho = 0.20$ $asize = 40$	ESE-SPEA	$psize = 30$ $mut_\rho = 0.20$ $asize = 10$
NRK-NSAIS	$psize = 10$ $cs = 3$	ESE-NSAIS	$psize = 10$ $cs = 3$
NDE-NSAIS	$psize = 10$ $cs = 3$ $op2_\rho = 0.50$	NPE-NSAIS	$psize = 10$ $cs = 3$ $tbrn_\rho = 0.75$

gives the smallest average fitness for NDE-NSAIS was $op2_\rho = 0.50$. Finally, one can see that both the SPEA2 variants achieved a greater hypervolume value with larger population sizes and the highest mutation probability evaluated. However, the NRK-SPEA achieved their best results with a greater archive size, while the ESE-SPEA used a smaller archive size.

6.2. NSAIS heuristics evaluation

The second experiment compares the quality of NPE-NSAIS's solutions with the other NSAIS algorithms (NRK-NSAIS, ESE-NSAIS, and NDE-NSAIS). We ran this analysis on the second instance's set. As in the other tests, we executed each algorithm 30 times with different initial random seeds and analyzed their average hypervolume values. We ran all algorithms with their optimized parameter set, as computed in Section 6.1.

Table 3 displays the results for NRK-NSAIS, ESE-NSAIS, NDE-NSAIS, and NPE-NSAIS for all evaluated instances. It shows the average relative deviation of the hypervolume achieved by each algorithm regarding the best hypervolume value found by an algorithm for each instance, in percentage. The first column shows the instance size. The second column displays the average node degree. The third column reports the average relative deviation of the hypervolume achieved by NRK-NSAIS over the best hypervolume computed by any of the proposed algorithms, computed as $\frac{BEST-ALG}{BEST}$, where $BEST$ denotes the average maximum hypervolume found and ALG denotes the average hypervolume achieved by NRK-NSAIS. This column also reports the standard deviation of this same measure. The remaining columns show this same information for ESE-NSAIS, NDE-NSAIS, and NPE-NSAIS, respectively. The last line contains the hypervolume average relative deviation of all algorithms over the best computed hypervolumes. We recall that each line summarizes the results of 10 instances generated with the same parameters and different seeds for the pseudo-random number generator.

One can see from Table 3 that the NRK-NSAIS achieved the worst results among all NSAIS variants for all evaluated instance sets. When analyzing the instances' sizes separately, we observed that the NDE-NSAIS achieved the smaller hypervolume average relative deviation for instances with 25 nodes. However, the NPE-NSAIS performed better than other algorithms for all other instance sizes. Also, the NPE-NSAIS's hypervolume relative deviation only exceeds 1 % on one case. Finally, the NPE-NSAIS performs better than all other algorithms in instances with higher average node degree. It is worth noticing that it finds all of the best hypervolume values for instances with

Table 3: Relative deviation of the hypervolume per algorithm regarding the best hypervolume found per instance.

Size	**Avg. Degree**	**NRK-NSAIS**	**ESE-NSAIS**	**NDE-NSAIS**	**NPE-NSAIS**
25	3	0.75 ± 0.90	1.03 ± 1.28	1.15 ± 1.30	1.03 ± 0.90
	5	3.42 ± 1.73	0.27 ± 0.31	0.15 ± 0.22	0.49 ± 0.50
	10	8.25 ± 1.39	0.21 ± 0.34	0.18 ± 0.22	0.36 ± 0.45
50	3	4.32 ± 1.29	0.14 ± 0.27	0.30 ± 0.31	0.14 ± 0.13
	5	10.28 ± 1.05	0.37 ± 0.38	0.23 ± 0.35	0.17 ± 0.31
	10	22.18 ± 3.38	0.12 ± 0.16	0.21 ± 0.25	0.12 ± 0.12
100	3	14.19 ± 2.90	0.17 ± 0.16	0.30 ± 0.35	0.21 ± 0.28
	5	23.54 ± 3.62	0.28 ± 0.23	0.22 ± 0.26	0.05 ± 0.10
	10	38.63 ± 5.01	0.40 ± 0.24	0.22 ± 0.27	0.13 ± 0.16
250	3	39.41 ± 6.54	0.48 ± 0.44	0.40 ± 0.34	0.08 ± 0.19
	5	49.68 ± 4.34	0.64 ± 0.27	0.32 ± 0.28	0.03 ± 0.10
	10	56.59 ± 5.14	1.11 ± 0.63	0.50 ± 0.39	0.03 ± 0.08
500	3	58.73 ± 4.70	1.40 ± 0.48	0.77 ± 0.45	0.02 ± 0.07
	5	63.34 ± 4.89	2.61 ± 0.57	0.88 ± 0.46	0.09 ± 0.29
	10	68.08 ± 4.92	5.94 ± 1.07	1.83 ± 0.95	0.00 ± 0.00
750	3	63.97 ± 6.00	4.11 ± 0.91	1.63 ± 1.00	0.14 ± 0.44
	5	68.99 ± 8.40	6.37 ± 1.77	1.66 ± 2.04	0.84 ± 1.44
	10	67.85 ± 9.51	12.61 ± 3.71	4.78 ± 2.12	0.00 ± 0.00
1000	3	62.26 ± 6.00	9.59 ± 2.55	3.89 ± 2.93	0.25 ± 0.77
	5	71.90 ± 7.91	14.88 ± 3.28	3.65 ± 3.30	0.39 ± 1.23
	10	71.04 ± 8.10	23.10 ± 2.65	16.03 ± 6.01	0.00 ± 0.00
	Average	47.52 ± 23.94	6.21 ± 8.33	3.39 ± 5.90	0.09 ± 0.22

average node degree 10 and more than 500 nodes.

6.3. Comparison of NPE-NSAIS against GA

The third experiment compares the NPE-NSAIS with the Akcan's [15] GA, i. e., the best solver for the MBCT on the literature so far. Thus, we evaluate the NPE-NSAIS solutions in the light of the weighted-sum objectives proposed by Akcan's [15].

We present at Tables 4 and 5 four different objective functions for the MBCT. Table 4 shows the objective function parameters. The first column shows the parameter name, while the second column displays the methodology to compute each parameter.

Table 4: Weighted-sum objective function parameters [15].

Name	**Calculus**
$MIN\text{-}COST$	$\min(Z_1)$
$MAX\text{-}COST$	$\max(Z_1)$
$MIN\text{-}ERROR$	$\min(Z_2)$
$MAX\text{-}ERROR$	$\max(Z_2)$
$norm\text{-}cost$	$100 \times \frac{cost - MIN\text{-}COST}{MAX\text{-}COST - MIN\text{-}COST}$
$norm\text{-}error$	$100 \times \frac{error - MIN\text{-}ERROR}{MAX\text{-}ERROR - MIN\text{-}ERROR}$

Table 5 shows the four weighted-sum objective functions developed by Akcan [15] computed as combinations of the above parameters. The first column shows the objective function's name, while the second column indicates the importance of each parameter to the final result. One can see that O_1 focus on minimizing the tree's cost and slightly reduces the maximum post-calibration error. On the other hand, O_4 mainly minimizes the

maximum post-calibration error, giving little attention to the tree's cost. Finally, O_2 and O_3 give equal importance to both objectives, but O_3 does not normalize its inputs.

Table 5: MBCT objective functions [15].

Name	**Calculus**
O_1	$0.9 \times \textit{norm-cost} + 0.1 \times \textit{norm-error}$
O_2	$0.5 \times \textit{norm-cost} + 0.5 \times \textit{norm-error}$
O_3	$\frac{(cost - MIN\text{-}COST)^2}{MIN\text{-}COST} + \frac{(error - MIN\text{-}ERROR)^2}{MIN\text{-}ERROR}$
O_4	$0.1 \times \textit{norm-cost} + 0.9 \times \textit{norm-error}$

Despite proposing four weighted-sum objective functions, Akcan [15] only evaluates O_2 and O_3, since these functions tries to take equal importance to both objectives. Therefore, we also compare the NPE-NSAIS with the GA on functions O_2 and O_3.

Table 6 shows the results of this experiment. The first column shows the instance size. The second column displays the average node degree. The third column reports the average deviation of $NPE - NSAIS$ over GA for the weighted-sum objective function O_2. It is computed as $1 - \frac{NPE-NSAIS}{GA}$, where $NPE - NSAIS$ denotes the average objective value achieved by all solutions in the NPE-NSAIS nondominated fronts and GA represents the average value achieved by Akcan's [15] GA. Therefore, a positive number indicates that the NPE-NSAIS overcomes the GA. On the other hand, a negative number imply that the GA achieved a better average solution value than the NPE-NSAIS. The third column shows the same information for the weighted-sum objective function O_3.

Table 6: Average improvement of NPE-NSAIS over GA.

Size	**Avg. Degree**	$\mathbf{O_2}$	$\mathbf{O_3}$
25	3	$-49 \pm 23\,\%$	$-1111 \pm 800\,\%$
	5	$-65 \pm 26\,\%$	$-944 \pm 456\,\%$
	10	$-65 \pm 39\,\%$	$-796 \pm 522\,\%$
50	3	$-24 \pm 9\ \%$	$-386 \pm 119\,\%$
	5	$8 \pm 14\,\%$	$-127 \pm 83\ \%$
	10	$9 \pm 11\,\%$	$-150 \pm 87\ \%$
100	3	$52 \pm 7\ \%$	$-9 \pm 42\ \%$
	5	$52 \pm 6\ \%$	$8 \pm 24\ \%$
	10	$57 \pm 6\ \%$	$40 \pm 14\ \%$
250	3	$68 \pm 6\ \%$	$47 \pm 24\ \%$
	5	$67 \pm 3\ \%$	$49 \pm 11\ \%$
	10	$70 \pm 4\ \%$	$60 \pm 5\ \%$
500	3	$66 \pm 4\ \%$	$39 \pm 6\ \%$
	5	$62 \pm 4\ \%$	$39 \pm 9\ \%$
	10	$58 \pm 3\ \%$	$43 \pm 4\ \%$
750	3	$59 \pm 4\ \%$	$26 \pm 8\ \%$
	5	$51 \pm 6\ \%$	$30 \pm 6\ \%$
	10	$49 \pm 5\ \%$	$39 \pm 6\ \%$
1000	3	$57 \pm 4\ \%$	$21 \pm 10\ \%$
	5	$45 \pm 7\ \%$	$33 \pm 6\ \%$
	10	$42 \pm 5\ \%$	$35 \pm 4\ \%$

One can see from Table 6 that the NPE-NSAIS overcome the GA for all instances with 250 or more nodes. The NPE-NSAIS achieved an improvement up to 70 % on instances with 250 nodes and average degree 10. However, the GA overcomes the NPE-NSAIS on small-sized instances. This behavior is due the easy to solve those instances and the GA's design. Akcan's [15] GA evolves only one solution on each generation, thus imposing a great convergence pressure. Since those instances are small-sided (and thus easy to solve), the GA overcomes the NPE-NSAIS. However, as the instance size increases (and thus became harder to solve), the GA became trapped on local optima, and thus cannot converge properly. Therefore, the NPE-NSAIS and its more sophisticated strategies achieved better results on medium and large-sized instances.

6.4. Comparison of NSAIS against other Multi-objective Evolutionary Algorithms

The fourth and last experiment evaluates the NSAIS against the NSGA-II and the SPEA2 when all algorithms use the same encoding scheme. Thus, we evaluate both the ESE-NSAIS and the NPE-NSAIS versus two NSGA-II implementations (ESE-NSGA and NRK-NSGA) and two SPEA2 implementations (ESE-SPEA, NRK-SPEA). We ran this analysis on the second instance's set. As in the other tests, we executed each algorithm 30 times with different initial random seeds and analyzed their average hypervolume values. We ran all algorithms with their optimized parameter set, as computed in Section 6.1.

Table 7 displays the results of this experiment for all evaluated instances. It shows the average relative deviation of the hypervolume achieved by each

algorithm regarding the best hypervolume value found by an algorithm for each instance, in percentage. The first column shows the instance size. The second column displays the average node degree. The third column reports the average relative deviation of the hypervolume achieved by ESE-NSAIS over the best hypervolume computed by any of the proposed algorithms, computed as $\frac{BEST-ALG}{BEST}$, where $BEST$ denotes the average maximum hypervolume found and ALG denotes the average hypervolume achieved by ESE-NSAIS. This column also reports the standard deviation of this same measure. The remaining columns show this same information for ESE-NSGA, ESE-SPEA, NRK-NSAIS, NRK-NSGA, and NRK-SPEA, respectively. The last line contains the hypervolume average relative deviation of all algorithms over the best computed hypervolumes. As in Table 3, each line summarizes the results of 10 instances generated with the same parameters and different seeds for the pseudo-random number generator.

Table 7 shows that the NSAIS's algorithms perform better than the NSGA-II's and the SPEA2's when they use the same encoding. The NRK-NSGA and the NRK-SPEA achieved an average relative deviation of 51.99 % and 57.50 %, respectively, while the NRK-NSAIS achieved an average relative deviation of 47.52 %. This same conclusion holds for the case of the Edge-set Encoding, on which the ESE-NSGA, the ESE-SPEA, and the ESE-NSAIS achieved average relative deviation of 11.92 %, 31.44 %, and 6.21 %, respectively. Therefore, those result indicates that the NSAIS algorithm performs better than the NSGA-II and the SPEA2 when solving the MBCT on the proposed instances, even when using the same encoding. We highlight that the NPE-NSAIS outperforms both the ESE-NSAIS and the NRK-NSAIS, as

Table 7: Relative deviation of the hypervolume per algorithm regarding the best hypervolume found per instance.

Size	Avg. Degree	ESE-NSAIS	ESE-NSGA	ESE-SPEA	NRK-NSAIS	NRK-NSGA	NRK-SPEA
25	3	1.03 ± 1.28	0.60 ± 1.02	1.92 ± 1.63	0.75 ± 0.90	1.82 ± 1.30	8.45 ± 2.62
	5	0.27 ± 0.31	0.65 ± 0.66	1.55 ± 1.01	3.42 ± 1.73	6.14 ± 2.25	14.89 ± 3.70
	10	0.21 ± 0.34	0.84 ± 0.43	1.84 ± 0.92	8.25 ± 1.39	13.21 ± 2.19	24.51 ± 2.72
50	3	0.14 ± 0.27	0.96 ± 0.59	1.79 ± 0.68	4.32 ± 1.29	9.05 ± 1.20	22.41 ± 2.31
	5	0.37 ± 0.38	1.35 ± 0.78	2.26 ± 0.80	10.28 ± 1.05	17.44 ± 1.79	30.81 ± 3.26
	10	0.12 ± 0.16	1.32 ± 0.45	2.91 ± 0.85	22.18 ± 3.38	29.62 ± 3.82	40.65 ± 5.02
100	3	0.17 ± 0.16	0.97 ± 0.63	2.76 ± 1.08	14.19 ± 2.90	22.37 ± 5.39	37.20 ± 7.26
	5	0.28 ± 0.23	1.11 ± 0.28	3.48 ± 1.12	23.54 ± 3.62	31.97 ± 5.02	43.67 ± 5.88
	10	0.40 ± 0.24	1.51 ± 0.35	4.89 ± 1.20	38.63 ± 5.01	44.47 ± 5.78	53.04 ± 6.19
250	3	0.48 ± 0.44	1.97 ± 0.81	12.57 ± 3.14	39.41 ± 6.54	48.29 ± 8.22	58.50 ± 8.95
	5	0.64 ± 0.27	2.85 ± 0.74	18.40 ± 2.27	49.68 ± 4.34	57.14 ± 5.96	64.81 ± 5.39
	10	1.11 ± 0.63	4.03 ± 0.88	24.09 ± 2.98	56.59 ± 5.14	60.82 ± 5.49	64.86 ± 6.71
500	3	1.40 ± 0.48	5.58 ± 1.04	42.88 ± 2.97	58.73 ± 4.70	63.82 ± 5.11	70.17 ± 5.13
	5	2.61 ± 0.57	9.93 ± 1.39	48.79 ± 2.99	63.34 ± 4.89	68.09 ± 5.73	71.05 ± 5.67
	10	5.94 ± 1.07	14.86 ± 1.37	53.00 ± 3.42	68.08 ± 4.92	72.05 ± 5.67	73.14 ± 5.99
750	3	4.11 ± 0.91	14.76 ± 1.96	60.18 ± 3.87	63.97 ± 6.00	66.90 ± 5.41	73.02 ± 5.83
	5	6.37 ± 1.77	19.10 ± 1.72	63.89 ± 4.25	68.99 ± 8.40	73.58 ± 8.86	75.35 ± 8.21
	10	12.61 ± 3.71	25.77 ± 2.54	63.43 ± 5.27	67.85 ± 9.51	71.19 ± 8.83	71.84 ± 8.72
1000	3	9.59 ± 2.55	24.86 ± 2.48	66.58 ± 3.89	62.26 ± 6.00	67.28 ± 6.84	70.58 ± 6.26
	5	14.88 ± 3.28	28.17 ± 4.01	69.48 ± 2.98	71.90 ± 7.91	75.75 ± 7.31	78.38 ± 7.65
	10	23.10 ± 2.65	35.11 ± 2.19	69.92 ± 2.57	71.04 ± 8.10	72.54 ± 7.37	74.49 ± 8.63
	Average	6.21 ± 8.33	11.92 ± 12.95	31.44 ± 28.16	47.52 ± 23.94	51.99 ± 22.70	57.50 ± 18.87

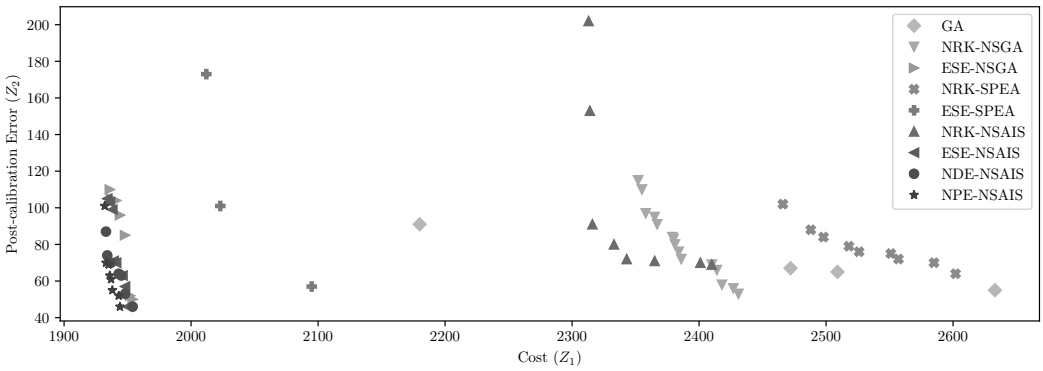
shown in Table 3. Thus, it indicates that the NPE-NSAIS is the best of the algorithm for the MBCT on the evaluated instances.

6.5. Pareto curves

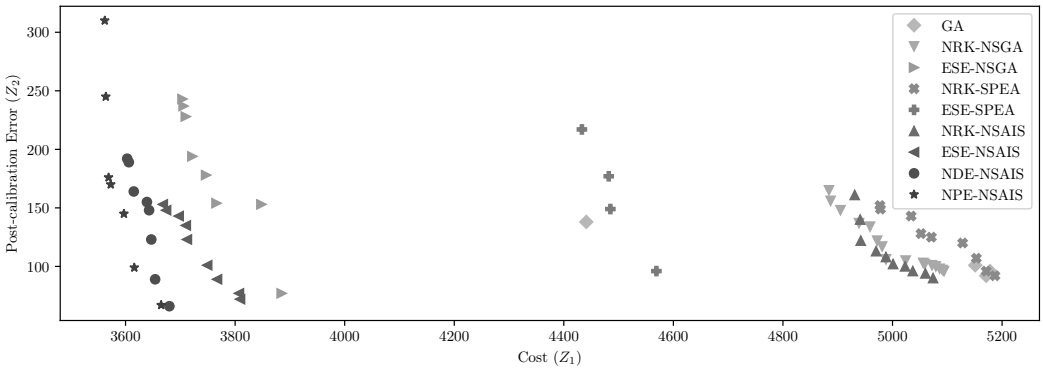
Figure 7 shows the fronts F_1 obtained by all of the evaluated algorithms on three instances with different characteristics. One can see that the NRK-NSGA, the NRK-NSAIS, and the NRK-SPEA fronts are close by them. The results obtained by the Akcan GA are also close to the ones achieved by the NRK-based algorithms. The NPE-NSAIS clearly found a greater hypervolume value than all other algorithms in Figures 7b and 7c. It illustrates the results reported on Table 3, whereas the NPE-NSAIS found smaller relative deviations as the instance size increases. Furthermore, one can trivially note that ESE-NSGA, ESE-NSAIS, NDE-NSAIS, and NPE-NSAIS outperforms GA, the state-of-the-art algorithm on the literature for solving the MBCT.

6.6. Statistical evaluation

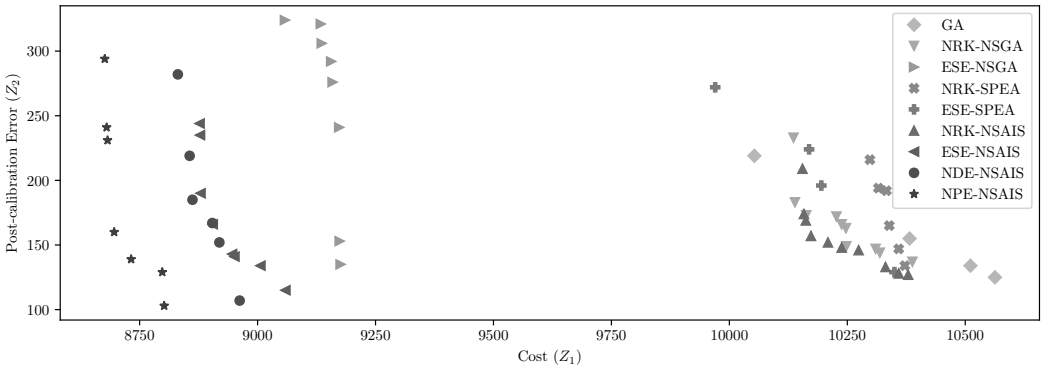
We evaluated the results of the experiments in Sections 6.2 and 6.4 through multiple one-sided paired statistical tests. Each test evaluates if the average hypervolume value achieved by the NPE-NSAIS is significantly greater than the hypervolume values obtained by the other algorithms. Thus, five one-sided paired tests were performed. They respectively compare the NPE-NSAIS with the NRK-NSGA, the ESE-NSGA, the NRK-NSAIS, the ESE-NSAIS, and the NDE-NSAIS. We did not include the GA in this test because it is not multi-objective, and thus it is not possible to compute its hypervolume.



(a) Instance with 250 nodes and average degree of 5.



(b) Instance with 500 nodes and average degree of 10.



(c) Instance with 1000 nodes and average degree of 3.

Figure 7: Pareto-fronts obtained by the evaluated algorithms.

All tests evaluate the hypothesis pair denoted by Equation (1), where μ_{npe} denotes the average hypervolume value achieved by the NPE-NSAIS and μ_{α} represents the average hypervolume value achieved by other algorithm. The null hypothesis (H_0) states that their average hypervolume values do not significantly differ. On the other hand, the alternative hypothesis (H_1) states that the NPE-NSAIS average hypervolume value is significantly greater than the average hypervolume of the compared algorithm.

$$\begin{cases} H_0 : \ \mu_{npe} = \mu_{\alpha} \\ H_1 : \ \mu_{npe} > \mu_{\alpha} \end{cases} \tag{1}$$

Initially, we evaluate the data assumptions to choose between a parametric or a non-parametric test. The Shapiro-Wilk normality test showed that the NPE-NSAIS results do not come from a normally distributed population ($p < 0.05$). Therefore, we employed the non-parametric paired Mann-Whitney-Wilcoxon test.

The statistical tests showed that the NPE-NSAIS average hypervolume value is significantly greater than all NSGA-II's and SPEA2's implementations, NRK-NSAIS, and ESE-NSAIS algorithm's average hypervolumes with 99 % of significance ($p < 0.01$). Yet, our test showed that the NPE-NSAIS average hypervolume value is also significantly greater than the NDE-NSAIS's, but with 95 % of significance ($p < 0.05$), which is due to the proximity of NPE's and NDE's operators. Therefore, we conclude that the NPE-NSAIS outperformed all other proposed algorithms for solving the MBCT on the evaluated instances.

7. Conclusions

This paper presented the Non-dominated Sorting Artificial Immune System (NSAIS), a hybrid metaheuristic based on the NSGA-II [12] and on the CLONALG [13, 14] metaheuristics. We embedded the NSAIS with the Node-depth Phylogenetic-based Encoding (NPE) [8], an encoding optimized for representing spanning trees on graphs, resulting in the NPE-NSAIS. This metaheuristic is intended to solve difficult network design optimization problems. The quality of the NPE-NSAIS was assessed by solving the Minimum-Cost Bounded-Error Calibration Tree problem (MBCT) [15], a bi-objective wireless sensor networks problem that arises from the periodical need of sensors calibration.

We compared the NPE-NSAIS with the state-of-the-art algorithm for the MBCT [15] and other three NSAIS embedded with other solution encodings: (*i*) the Network Random-Keys NSAIS (NRK-NSAIS); (*ii*) the Edge-set Encoding NSAIS (ESE-NSAIS); and (*iii*) the Node-depth Encoding NSAIS (NDE-NSAIS). In addition, we compared the NSAIS algorithms with two variations of the NSGA-II and two variations of the SPEA2: (*i*) a Network Random Keys Encoding NSGA-II (NRK-NSGA); (*ii*) an Edge-set Encoding NSGA-II (ESE-NSGA); (*iii*) a Network Random Keys Encoding SPEA2 (NRK-SPEA); and (*iv*) an Edge-set Encoding SPEA2 (ESE-SPEA).

Computational experiments showed that the NSAIS outperformed both the NSGA-II and the SPEA2 when the algorithms employed the same encoding. Furthermore, the NPE-NSAIS statistically outperformed all the evaluated algorithms for the MBCT. Therefore, the NPE showed to be a better data structure for solving the MBCT than the Network Random Keys, the

Edge-Set, and the Node-Depth. Our results indicate that the NPE-NSAIS may be an advantageous option to solve other Network Design Problems though Evolutionary Algorithms.

Future works on this theme should focus on three themes. The first is to apply the NPE-NSAIS to solve other multi-objective network design optimization problems. The second is to implement different heuristic or metaheuristic algorithms using the NPE encoding and use them to solve similar problems. The third is to use the NSAIS to solve other classes of problems.

Acknowledgments

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 de Minas Gerais - Brasil* (FAPEMIG).

[1] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, Q. Zhang, Multiobjective evolutionary algorithms: A survey of the state of the art, Swarm and Evolutionary Computation 1 (1) (2011) 32 – 49, ISSN 2210-6502.

[2] F. Neumann, C. Witt, Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity, in: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO'12 Companion, ACM, New York, NY, USA, ISBN 978-1-4503-1178-6, 1035 – 1058, 2012.

[3] C. C. Palmer, A. Kershenbaum, Representing trees in genetic algorithms, in: Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, vol. 1, IEEE, 379 – 384, 1994.

[4] F. Rothlauf, Representations for Genetic and Evolutionary Algorithms, vol. 104 of *Studies in Fuzziness and Soft Computing*, Springer, Secaucus, NJ, USA, first edn., ISBN 354025059X, 2006.

[5] G. R. Raidl, B. A. Julstrom, Edge sets: an effective evolutionary coding of spanning trees, IEEE Transactions on Evolutionary Computation 7 (3) (2003) 225 – 239, ISSN 1089-778X.

[6] F. Rothlauf, D. E. Goldberg, A. Heinzl, Network Random Keys – A Tree Representation Scheme for Genetic and Evolutionary Algorithms, Evolutionary Computation 10 (1) (2002) 75 – 97.

[7] A. C. B. Delbem, A. de Carvalho, C. A. Policastro, A. K. O. Pinto, K. Honda, A. C. Garcia, Node-Depth Encoding for Evolutionary Algorithms Applied to Network Design, in: K. Deb (Ed.), Genetic and Evolutionary Computation – GECCO 2004, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-24854-5, 678 – 687, 2004.

[8] T. W. de Lima, A. C. B. Delbem, A. da Silva Soares, F. M. Federson, J. B. A. L. Junior, 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 31 (2016) 1 – 10, ISSN 2210-6502.

[9] J. Felsenstein, Inferring Phylogenies, vol. 2, Sinauer Associates Sunderland, MA, ISBN 9780878931774, 2004.

[10] M. Mitchell, An Introduction to Genetic Algorithms, MIT press, ISBN 9780262631853, 1998.

[11] J. Krause, J. Cordeiro, R. S. Parpinelli, H. S. Lopes, A Survey of Swarm Algorithms Applied to Discrete Optimization Problems, in: X.-S. Yang, Z. Cui, R. Xiao, A. H. Gandomi, M. Karamanoglu (Eds.), Swarm Intelligence and Bio-Inspired Computation, vol. 1 of *Elsevier insights*, chap. 7, Elsevier, Oxford, first edn., ISBN 978-0-12-405163-8, 169 – 191, 2013.

[12] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE transactions on evolutionary computation 6 (2) (2002) 182–197.

[13] L. N. De Castro, F. J. Von Zuben, The clonal selection algorithm with engineering applications, in: Proceedings of GECCO, vol. 2000, 36 – 39, 2000.

[14] L. N. de Castro, F. J. V. Zuben, Learning and optimization using the clonal selection principle, IEEE Transactions on Evolutionary Computation 6 (3) (2002) 239 – 251, ISSN 1089-778X.

[15] H. Akcan, On the complexity of energy efficient pairwise calibration in embedded sensors, Applied Soft Computing 13 (4) (2013) 1766 – 1773, ISSN 1568-4946.

[16] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm, Tech. Rep. TIK-Report

103, Eidgenössische Technische Hochschule Zürich (ETH), URL https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/145755/eth-24689-01.pdf, 2001.

[17] F. Rothlauf, D. E. Goldberg, A. Heinzl, Bad Codings and the Utility of Well-designed Genetic Algorithms, in: Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation, GECCO'00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860-708-0, 355 – 362, 2000.

[18] F. Lobo, C. Lima, Z. Michalewicz, Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence, Springer Berlin Heidelberg, ISBN 9783540694311, 2007.

[19] M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and Exploitation in Evolutionary Algorithms: A Survey, ACM Computing Surveys 45 (3) (2013) 35, ISSN 0360-0300.

[20] L. Davis, D. Orvosh, A. Cox, Y. Qiu, A Genetic Algorithm for Survivable Network Design, in: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860-299-2, 408 – 415, 1993.

[21] P. Piggott, F. Suraweera, Encoding graphs for genetic algorithms: An investigation using the minimum spanning tree problem, in: X. Yao (Ed.), Progress in Evolutionary Computation, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-49528-4, 305 – 314, 1995.

[22] A. Cayley, A Theorem on Trees, The Quarterly Journal of Mathematics 23 (1889) 376 – 378.

[23] M. Krishnamoorthy, A. T. Ernst, Y. M. Sharaiha, Comparison of Algorithms for the Degree Constrained Minimum Spanning Tree, Journal of Heuristics 7 (6) (2001) 587 – 611, ISSN 1572-9397.

[24] F. N. Abuali, R. L. Wainwright, D. A. Schoenefeld, Determinant Factorization: A New Encoding Scheme for Spanning Trees Applied to the Probabilistic Minimum Spanning Tree Problem, in: Proceedings of the 6th International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 15-19, 1995, 470 – 477, 1995.

[25] F. Rothlauf, 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, 238 – 244, 1999.

[26] S. Picciotto, How to encode a tree, Ph.D. thesis, University of California, San Diego, 1999.

[27] T. Paulden, D. K. Smith, Recent Advances in the Study of the Dandelion Code, Happy Code, and Blob Code Spanning Tree Representations, in: 2006 IEEE International Conference on Evolutionary Computation, IEEE, ISSN 1089-778X, 2111 – 2118, 2006.

[28] S. Caminiti, R. Petreschi, String Coding of Trees with Locality and Heritability, in: L. Wang (Ed.), Computing and Combinatorics, Springer

Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-31806-4, 251 – 262, 2005.

[29] N. Deo, P. Micikevicius, A New Encoding for Labeled Trees Employing a Stack and a Queue, Bulletin of the Institute of Combinatorics and its Applications 34 (2002) 77 – 85.

[30] T. Paulden, D. K. Smith, From the Dandelion Code to the Rainbow code: a class of bijective spanning tree representations with linear complexity and bounded locality, IEEE Transactions on Evolutionary Computation 10 (2) (2006) 108 – 123, ISSN 1089-778X.

[31] T. Gaube, F. Rothlauf, The Link and Node Biased Encoding Revisited: Bias and Adjustment of Parameters, in: E. J. W. Boers (Ed.), Applications of Evolutionary Computing, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-45365-9, 1 – 10, 2001.

[32] F. Rothlauf, On the Bias and Performance of the Edge-Set Encoding, IEEE Transactions on Evolutionary Computation 13 (3) (2009) 486 – 499, ISSN 1089-778X.

[33] J. C. Bean, Genetic Algorithms and Random Keys for Sequencing and Optimization, ORSA Journal on Computing 6 (2) (1994) 154 – 160.

[34] T. W. de Lima, F. Rothlauf, A. C. Delbem, The Node-depth Encoding: Analysis and Application to the Bounded-diameter Minimum Spanning Tree Problem, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO'08, ACM, New York, NY, USA, ISBN 978-1-60558-130-9, 969 – 976, 2008.

[35] T. W. de Lima, A. C. B. Delbem, R. L. Lima, G. P. Sabin, M. A. A. de Oliveira, Permutation-based Recombination Operator to Node-depth Encoding, Procedia Computer Science 80 (2016) 279 – 288, ISSN 1877-0509, international Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA.

[36] A. C. B. Delbem, T. W. de Lima, G. P. Telles, Efficient Forest Data Structure for Evolutionary Algorithms Applied to Network Design, IEEE Transactions on Evolutionary Computation 16 (6) (2012) 829 – 846, ISSN 1089-778X.

[37] C. A. C. Coello, V. Cutello, D. Lee, M. Pavone, Recent advances in immunological inspired computation, Engineering Applications of Artificial Intelligence 62 (2017) 302 – 303, ISSN 0952-1976.

[38] S. F. M. Burnet, The Clonal Selection Theory of Acquired Immunity, vol. 3 of *The Abraham Flexner lectures 2*, Vanderbilt University Press Nashville, 1959.

[39] K. Rajewsky, Clonal selection and learning in the antibody system, Nature 381 (1996) 751 – 758.

[40] R. T. Marler, J. S. Arora, The weighted sum method for multi-objective optimization: new insights, Structural and Multidisciplinary Optimization 41 (6) (2010) 853 – 862, ISSN 1615-1488.

[41] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, M. Srivastava, Sensor Network

Data Fault Types, ACM Transactions on Sensor Networks 5 (3) (2009) 25, ISSN 1550-4859.

[42] A. Mahapatro, P. M. Khilar, Fault Diagnosis in Wireless Sensor Networks: A Survey, IEEE Communications Surveys Tutorials 15 (4) (2013) 2000–2026, ISSN 1553-877X.

[43] V. Bychkovskiy, S. Megerian, D. Estrin, M. Potkonjak, A Collaborative Approach to In-Place Sensor Calibration, in: F. Zhao, L. Guibas (Eds.), Information Processing in Sensor Networks, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-36978-3, 301 – 316, 2003.

[44] K. Whitehouse, D. Culler, Calibration As Parameter Estimation in Sensor Networks, in: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, WSNA'02, ACM, New York, NY, USA, ISBN 1-58113-589-0, 59 – 67, 2002.

[45] L. M. Feeney, An Energy Consumption Model for Performance Analysis of Routing Protocols for Mobile Ad Hoc Networks, Mobile Networks and Applications 6 (3) (2001) 239 – 249, ISSN 1572-8153.

[46] T. Rault, A. Bouabdallah, Y. Challal, Energy efficiency in wireless sensor networks: A top-down survey, Computer Networks 67 (2014) 104 – 122, ISSN 1389-1286.

[47] M. Matsumoto, T. Nishimura, Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator, ACM Transactions on Modeling and Computer Simulation 8 (1) (1998) 3 – 30, ISSN 1049-3301.

[48] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms—a comparative case study, in: International conference on parallel problem solving from nature, Springer, 292–301, 1998.