Applied Soft Computing Journal 73 (2018) 83–95

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

# A genetic algorithm based solution to the Minimum-Cost Bounded-Error Calibration Tree problem

Hüseyin Akcan

*Department of Software Engineering, Izmir University of Economics, Izmir, Turkey*

## HIGHLIGHTS

- The MBCT problem is a spanning tree problem with two objectives.
- MBCT minimizes the spanning tree cost and bounds the maximum post-calibration skew.
- GAWES is proposed as a novel genetic algorithm based solution to the MBCT problem.
- Main novelty of GAWES is using extreme efficient solutions in the genetic algorithm.
- Experiments confirm that GAWES is superior to the state of the art.

ARTICLE INFO

*Article history:*
Received 11 October 2017
Received in revised form 20 June 2018
Accepted 9 August 2018
Available online xxxx

*Keywords:*
Genetic algorithm
Wireless sensor networks
Energy efficiency
Calibration tree

ABSTRACT

Sensors in wireless sensor networks are required to be self-calibrated periodically during their prolonged deployment periods. In calibration planning, employing intelligent algorithms are essential to optimize both the efficiency and the accuracy of calibration. The Minimum-Cost Bounded-Error Calibration Tree (*MBCT*) problem is a spanning tree problem with two objectives, minimizing the spanning tree cost and bounding the maximum post-calibration skew. The decision version of the *MBCT* problem is proven to be NP-Complete. In this paper, the *GAWES* algorithm is presented as a novel genetic algorithm based solution to the optimization version of the *MBCT* problem. *GAWES* adopts extreme efficient solution generation within the genetic algorithm to improve the search quality. It is demonstrated through experimentation that *GAWES* is superior to the existing state of the art algorithm, both in energy efficiency and calibration accuracy.

## 1. Introduction

Technological advances in Micro-Electro-Mechanical Systems (MEMS) have brought the network of self-configurable widely deployed sensor nodes, as wireless sensor networks, in our daily lives. Equipped with low power radios, nodes in wireless sensor networks are able to perform various sensing tasks and facilitate an ad hoc network to aggregate and extract useful data from the deployed environment. As a consequence of their flexible design and wireless operability, wireless sensor networks are capable of performing tasks that are not suitable or affordable for humans, such as remote area monitoring [1], underwater monitoring [2], and deployments in hazardous environments [3,4]. To tackle these deployment constraints, wireless sensor networks are designed to operate in a self-configurable manner where manual configuration is not a viable option. As each individual sensor unit operates on low power battery, energy efficiency is the most essential constraint for all the algorithms that need to be developed for sensor networks. The total lifetime of the sensor network depends on the lifetime of each individual sensor node in the network. Therefore, algorithms deployed on wireless sensor networks should not only use less power, but also be well distributed to avoid energy depletion on a single node.

Periodic calibration of each individual sensor is a critical problem in wireless sensor networks. As manual calibration is not an option after deployment, these sensors should self-calibrate themselves using nearby sensors as references. However, sensors need to communicate during calibration and wireless communication is one of the most energy consuming tasks for a sensor node. Therefore, an efficient and accurate self-calibration algorithm is essential for sensors that are deployed in remote areas for extended periods of time.

Calibration in wireless sensor networks poses many challenges [5–9]. A list of these challenges includes the inability to physically access sensors in most scenarios, their massive number,

*E-mail address:* huseyin.akcan@ieu.edu.tr.

https://doi.org/10.1016/j.asoc.2018.08.013
1568-4946/© 2018 Elsevier B.V. All rights reserved.

and their energy constraints. To overcome these difficulties, researchers have proposed methods to calibrate sensors without any human intervention using peer based iterative calibration [10–14]. However, iterative calibration algorithms also introduce a new set of challenges to the sensor network community. One such major challenge is to calibrate the network to achieve minimum calibration error using minimum energy in exchange.

Minimum-Cost Bounded-Error Calibration Tree (*MBCT*) problem [15] is based on iterative calibration of nodes in wireless sensor networks. In these networks, energy usage is a tight constraint. Therefore, calibrating the sensor network by using the minimum communication cost and yet get a reasonable accuracy on the calibration is a critical problem. However, *MBCT* problem is generic enough to be applied to other domains. In its abstract form, the problem optimizes the cost of the spanning tree, as well as the cost of reaching from each vertex to the root of the tree, where each vertex has an associated cost value.

The main contribution of this paper is a novel genetic algorithm based solution to the optimization version of the *MBCT* problem. In this work, a method to find the extreme efficient solutions after the crossover stage of the proposed genetic algorithm is employed. Consequently, the search is more efficiently directed to the ideal point that minimizes both the energy usage and the calibration error. As a result, through experimentation, this paper demonstrates that the proposed algorithm outperforms the existing state of the art in terms of both energy efficiency and calibration accuracy.

The rest of the paper is organized as follows, related work is presented in Section 2. In Section 3 *MBCT* problem definition is presented. Section 4 outlines extreme efficient solution calculation. In Section 5 the proposed genetic algorithm based solution is given, and experimental results are presented in Section 6. Finally Section 7 concludes the paper.

## 2. Related works

Calibration in sensor networks is an essential task and each sensor needs to be calibrated periodically [6,8]. Results of real world tests for calibration are reported in [7,9]. As sensors are expected to operate prolonged amounts of time after deployment, efficient periodic calibration is a critical task for the lifetime of the network. The challenges with respect to periodic calibration in wireless sensor networks are reported in [5].

Researchers have proposed parametric calibration methods [10–14] as an alternative to traditional calibration methods. In parametric calibration, a calibration function is used to map the reported output values of the reference sensor to the input values of the current sensor. Therefore, each sensor can self-calibrate parametrically based on a presumed reference sensor nearby, without any physical interaction.

There are various representations in the literature for encoding spanning trees in evolutionary algorithms. These methods can be listed as Characteristic Vectors [16], Predecessor Coding [17], Prüfer Numbers [18], Blob Code [19], Link-and-Node Biasing [17], Network Random Keys [20,21], and Edge-Sets [22]. A comparative analysis of these methods are presented in [22] on various criteria including locality, heritability, feasibility, and time-and-space complexity. It is reported in [22] that Edge-Sets is superior to the rest of the methods listed. [22] also discusses three different methods in order to create random spanning trees for initializing the population and performing the crossover. The methods are listed as PrimRST, KruskalRST, and RandomWalkRST. PrimRST uses a modified version of Prim's spanning tree algorithm and is biased towards creating star topologies. RandomWalkRST uses a random walk based strategy and is biased to create path like topologies. KruskalRST uses a modified version of Kruskal's spanning tree algorithm and creates trees that are in between star and path like topologies. The *GAWES* algorithm proposed in this paper uses Edge-Sets representation as chromosome encoding, and KruskalRST method to populate the initial population and perform crossover.

The first definition and complexity result of *MBCT* problem appeared in [15]. A genetic algorithm based heuristic algorithm (*GA*) is also proposed in [15] to solve the optimization version of the *MBCT* problem. The efficiency of *GA* is evaluated using various fitness functions on a set of randomly generated graphs. *GA* uses Edge-Sets representation for chromosome encoding and a modified version of Kruskal minimum spanning tree algorithm (KruskalRST) is used to create random chromosomes and perform crossover. The suggested parameters for *GA* algorithm is reported in [15] as an iteration size of 50 000, mutation rate of 0.1, and initial population size of 400.

*MBCT* problem seeks an efficient answer to the bicriteria spanning tree problem, where one needs to minimize both edge cost and maximum post-calibration skew of the spanning tree. In this sense, *MBCT* has similarities to hop constrained [23] and rooted distance constrained [24] spanning tree problems. In hop constrained spanning tree problem, the objective is to minimize the spanning tree cost given that the hop distance of each node to the root is less than some predefined constant. Similarly, in rooted distance constrained spanning tree problem, the objective is to find the minimum cost spanning tree given that delay of each node, associated with each edge, is less than a predefined distance value. In both hop constrained and rooted distance constrained spanning tree problems, both of the costs that needs to be minimized are on the edges. In *MBCT*, having the second cost on the vertex changes the definition and the characteristic of the problem. Therefore, *MBCT* problem clearly distinguishes itself from existing hop constrained and rooted distance constrained spanning tree problems.

## 3. The problem definition

The Minimum-Cost Bounded-Error Calibration Tree problem was first defined in [15]. Formal definition of the *MBCT* problem was stated in [15] as:

**Definition 3.1** (*MBCT [15]*)**.** "Given a wireless sensor network modeled as an undirected graph $G(V, E)$, and a designated reference node $r \in V$, where each $e \in E$ is assigned distance values $d_e > 0$, and each $v \in V$ is associated with a maximum random measurement error $\epsilon_v$, the *MBCT* problem is defined as finding a spanning tree over $G$ rooted at $r$ with total edge cost not greater than a constant $C > 0$, while the post-calibration skew of each sensor $v \in V$ is bounded by a positive constant $k$".

The *MBCT* problem was shown to be NP-complete in [15], and a genetic algorithm based heuristic was proposed in the same work for the optimization version of the problem. The optimization version of the *MBCT* problem minimizes both the total cost value and the post-calibration skew.

Fig. 1 (a) presents an example graph, where each node $i$ has an associated maximum calibration error ($\epsilon_i$), and each edge $j$ has an associated cost $c_j$. $S_0$ is marked as the pre-calibrated sensor with zero calibration error. Post-calibration skew of each node is the sum of the absolute maximum errors of each node along the path to node $S_0$. For example, in Fig. 1 (b), node $S_2$ is connected to $S_0$ through a path passing through node $S_3$. Therefore, the post-calibration skew of node $S_2$ is then the sum of the calibration errors of nodes $S_2$, $S_3$, and $S_0$, which is equal to $|\xi_2| = 2 + 8 + 0 = 10$. Fig. 1 (b) shows the minimum spanning tree based on edge costs, where the total cost is 5, post-calibration skew of each node is given as $|\xi_1| = 7$, $|\xi_2| = 10$, $|\xi_3| = 8$, $|\xi_4| = 9$, $|\xi_5| = 1$, and the maximum post-calibration skew is 10. Fig. 1 (c) shows

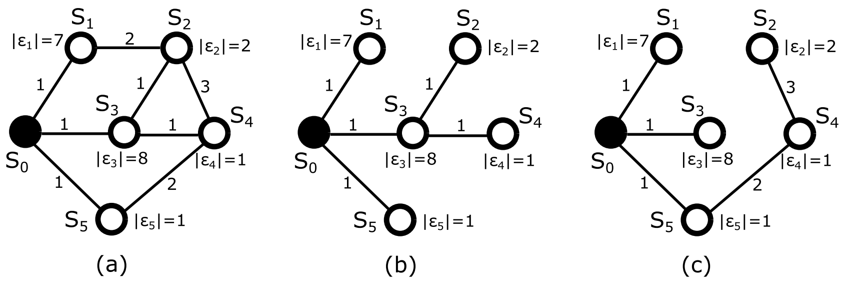


**Fig. 1.** Example graph (a), minimum spanning tree based on edge costs (b), spanning tree based on minimum maximum post-calibration skew and edge costs (c).

the spanning tree with the minimum possible maximum post-calibration skew and the minimum total cost, where the total cost is 8, post-calibration skew of each node this time is given as $|\xi_1| = 7$, $|\xi_2| = 4$, $|\xi_3| = 8$, $|\xi_4| = 2$, $|\xi_5| = 1$, and the maximum post-calibration skew is 8. As the absolute calibration error of node $S_3$ is 8, the maximum post-calibration skew cannot get any lower than this value. This example clearly demonstrates that both the total cost and the calibration accuracy can be controlled in a network by adjusting the topology of the spanning tree.

This paper proposes a new genetic algorithm based solution to the optimization version of *MBCT* problem. The novel part of the newly proposed algorithm is the use of extreme efficient solutions within the genetic algorithm. Details of the extreme efficient solution generation are described in Section 4 and the genetic algorithm based solution is presented in Section 5.

## 4. Extreme efficient solution generation

Extreme efficient solution is defined in [25] as:

**Definition 4.1** (*Extreme Efficient Solution [25]*). “In a multiobjective linear program,

$min\,\{Cx : Ax = b, x \geq 0\}$ ,

the set of extreme efficient solutions are the integer solutions of the linear program

$$min \left\{ \sum_{j=1,\dots,Q} \lambda_j c^j x : Ax = b, x \geq 0 \right\},$$

where, $\sum_{j=1}^{Q} \lambda_j = 1, \lambda_j > 0, j = 1, \dots, n$”.

[26,27] present methods to create new extreme efficient solutions using existing extreme efficient solutions on multi-criteria spanning tree problems. The *SearchXP* algorithm proposed in this paper adapts the extreme efficient solution generation methods in [26,27] to the *MBCT* problem. Given two existing extreme efficient solutions $S_1(x_1, y_1)$ and $S_2(x_2, y_2)$, a new extreme efficient solution is the furthest away point to the line passing over $S_1$ and $S_2$, which is a minimum spanning tree of the graph assigned the costs for each edge $e$ as $f(e) = f^1(e)(S_1^y - S_2^y) + f^2(e)(S_2^x - S_1^x)$. For an edge $e$, $f^1(e)$ represents the cost of edge $e$, $S_1^y$ and $S_2^y$ represent the value of the second criteria of the extreme efficient solutions $S_1$ and $S_2$. Again $f^2(e)$ represents the error of the outgoing vertex on edge $e$ and $S_1^x$ and $S_2^x$ represent the value of the first criteria of the extreme efficient solutions $S_1$ and $S_2$. The pseudocode of the *SearchXP* algorithm is presented in Fig. 2. In *SearchXP* algorithm, a queue of extreme efficient solution pairs is created and initiated with the given solutions $S_1$ and $S_2$. As long as the queue is not empty, the algorithm iteratively selects a new pair from the queue. The formula in line 6 is used to modify the costs and assign these new edge costs to the graph to create a single criteria graph. In line 7, a modified version of Prim's minimum spanning tree algorithm is used to solve the single criteria problem resulting a new extreme efficient solution $S$ with an accompanying spanning tree. If $S$ is not equal to $S_a$ and $S_b$, and new solution pairs $\{S_a, S\}$ and $\{S, S_b\}$ have not been seen before, these extreme efficient solution pairs are added to the queue. Upon completion, the *SearchXP* algorithm returns a list of all the extreme efficient solutions evaluated along with their accompanying spanning trees.

The proposed genetic algorithm uses the *SearchXP* extreme efficient solution generation algorithm after the crossover stage. The crossover candidates are treated as extreme efficient solutions to generate new extreme efficient solutions, thus new chromosomes. Newly generated extreme efficient solutions span the area between the two best crossover candidate solutions, hence, guide the search to the ideal point at which both criteria are minimized.

## 5. Heuristic solution

In this section, details of the proposed genetic algorithm based solution to the *MBCT* problem is described, which is named Genetic Algorithm With Extreme Solutions (*GAWES*) throughout the paper. In the optimization version of the *MBCT* problem, the objective is to minimize both the total edge cost and the post-calibration skew of the final spanning tree. Therefore, optimization version of the problem has two criteria. The main novelty of *GAWES* is in combining the existing extreme efficient solutions to create new extreme efficient solution after the crossover stage of the genetic algorithm. *GAWES* algorithm's search capacity is greatly enhanced by generating multiple extreme efficient solutions from the best chromosomes. As a result, superior solutions compared to the state of the art [15] are achieved. Details of the *GAWES* algorithm are described below in six stages, chromosome encoding, chromosome generation, initiating a population, crossover, calculating extreme efficient solutions, and mutation.

### 5.1. Chromosome encoding

Each chromosome in *GAWES* represents a valid spanning tree solution to the problem. Therefore, chromosomes are encoded using the Edge-Sets [22] representation as a list of edges given as vertex pairs of the input graph. Since a spanning tree on a graph of $|V|$ vertices has $|V - 1|$ edges, each chromosome in *GAWES* has a fixed size of $|V - 1|$.

### 5.2. Chromosome generation

During the crossover and mutation phases, altered chromosomes might not represent a valid spanning tree. Therefore, the chromosome generation function is used in these stages to recreate a valid spanning tree from the given altered chromosome. In the chromosome generation phase, a modified version of the Kruskal minimum spanning tree algorithm (KruskalRST [22]) is used. Contrary to the Kruskal algorithm, the non cycle creating edges within the chromosome is kept, and the rest of the edges are randomly added to the chromosome without any ordering until a valid spanning tree is generated.

```
SEARCHXP(GRAPH $G$, SOLUTION $S_1$, SOLUTION $S_2$)
1: $List < Solution > \leftarrow \emptyset$
2: $Q \leftarrow \emptyset$
3: Q.enqueue($\{S_1, S_2\}$)
4: while Q not empty do
5:     $\{S_a, S_b\}$ = Q.dequeue()
6:     Compute new costs $f^a(e)(S_a^y - S_b^y) + f^b(e)(S_b^x - S_a^x)$
7:     Solve MST based on new costs S=(x,y)
8:     if $(S \neq S_a) \wedge (S \neq S_b)$ then
9:         $List < Solution > .add(S)$
10:        if Solution pair $\{S_a, S\}$ not seen before then
11:            Q.enqueue($\{S_a, S\}$)
12:        if Solution pair $\{S, S_b\}$ not seen before then
13:            Q.enqueue($\{S, S_b\}$)
14: return $List < Solution >$
```

**Fig. 2.** The pseudocode of the *SearchXP* algorithm.

## 5.3. Initiating a population

The population is initiated with random chromosomes by generating random spanning trees and encoding them as chromosomes. The population size is controlled by a parameter within the *GAWES* algorithm.

## 5.4. Crossover

In the crossover stage, candidates are selected using Roulette Wheel Selection algorithm [28]. From the candidate parent chromosomes, a new child chromosome is created. The newly created child chromosome is inserted into the population, and the chromosome with the worst fitness value is removed from the population. In the crossover stage, the child gets each of its genes from one of the parents with equal probability. Therefore, as a result, a random chromosome is created from the best fitting parents. However, the crossover stage described does not guarantee that the resulting child chromosome be a valid spanning tree, therefore an acceptable solution to the *MBCT* problem. In order to fix the chromosome, the chromosome generation process described in Section 5.2 is applied, which converts the child chromosome to a spanning tree solution.

## 5.5. Calculating extreme efficient solutions

The main novelty in *GAWES*, compared to the existing state of the art genetic algorithm (*GA*) [15], is the use of extreme efficient solutions. In *GAWES*, after the crossover stage the two parent chromosomes are treated as extreme efficient solutions. New extreme efficient solutions are created from these two parent solutions using the *SearchXP* algorithm described before in Section 4. The newly generated solutions then are added to the population. Therefore, at each iteration new extreme efficient solutions encoded as chromosomes are added to the population. The addition of these efficient solutions enhances the borders of the search towards the ideal point, depicted as the minimum of both criteria.

## 5.6. Mutation

In the mutation phase, with a given low probability, a random edge is removed from the solution chromosome. Again, to fix the chromosome, the chromosome generation function defined in Section 5.2 is used. The mutation rate is controlled with a parameter in the *GAWES* algorithm.

```
POPULATESOLUTIONS(SOLUTION $S_1$, SOLUTION $S_2$)
1: $List < Solution >$ = SEARCHXP(G, $S_1$, $S_2$)
2: for Each S in $List < Solution >$ do
3:     $C_S$ = CreateChromosome(S)
4:     Population.add($C_S$)
```

**Fig. 3.** The pseudocode of the *PopulateSolutions* function.

**Table 1**
Distributions of the parameters used in the datasets.

| | Dataset_A | Dataset_B | Dataset_C |
|---|---|---|---|
| Graph size | 25–1000 | 100–500 | 100 |
| Node degree dist. | normal | uniform | normal or exponential |
| Edge weight dist. | normal | uniform | normal or exponential |
| Calib. error dist. | uniform | uniform | normal or exponential |

The pseudocode of the *PopulateSolutions* function and the *GAWES* algorithms are presented in Figs. 3 and 4, respectively. The *PopulateSolutions* function generates new extreme efficient solutions from the two given solutions and adds these new solutions to the population. *PopulateSolutions* function is called in lines 7 and 13 in *GAWES*. In line 7, extreme efficient solutions generated from the two solutions that optimize the first and the second criteria are added to the population. In line 13, two solutions depicted by the two parent chromosomes during the crossover phase are used to create new solutions.

In lines 2–4 of the *GAWES* algorithm, the population is randomly created. In lines 5–7, new solutions are generated from the existing two extreme efficient solutions and added to the population. *GAWES* algorithm runs for a fixed number of iterations defined as a parameter. In line 10, Roulette Wheel Selection algorithm is used to select two parent chromosomes. Crossover is performed in line 12, and new extreme efficient solutions are created in line 13. Mutation is called, if necessary, in lines 14–15, and chromosome

```
GAWES(GRAPH G)
 1: /* Initiate the population */
 2: for L=1 to PopulationSize do
 3:     C = CreateRandomChromosome(G)
 4:     Population.add(C)
 5: $S_1$ = Solution based on first criteria
 6: $S_2$ = Solution based on second criteria
 7: PopulateSolutions($S_1$,$S_2$)
 8: BestFitnessScore ← ∞
 9: for I=1 to NumberOfIterations do
10:     {$C_1$,$C_2$} = RouletteWheelSelection(Population)
11:     $C_{Worst}$ = SelectWorstChromosome(Population)
12:     $C_{New}$ = Crossover($C_1$,$C_2$)
13:     PopulateSolutions($C_1.sol$,$C_2.sol$)
14:     if RandomNumber < MutationProbability then
15:         $C_{New}$ = Mutation($C_{New}$)
16:     /* ChromosomeGeneration makes sure $C_{New}$
                                    is a spanning tree */
17:     $C_{New}$ = ChromosomeGeneration($C_{New}$,G)
18:     BestFitnessScore = min(BestFitnessScore,
                                    GetFitnessScore($C_{New}$))
19:     Population.add($C_{New}$)
20:     Population.remove($C_{Worst}$)
21: return BestFitnessScore
```

**Fig. 4.** The pseudocode of the *GAWES* algorithm.

generation routine is called to create a valid spanning tree from the new chromosome in line 17. Best fitness score is updated in line 18, new chromosome is inserted into the population in line 19, and the worst chromosome found in line 11 is removed from the population in line 20.

## 6. Experimental results

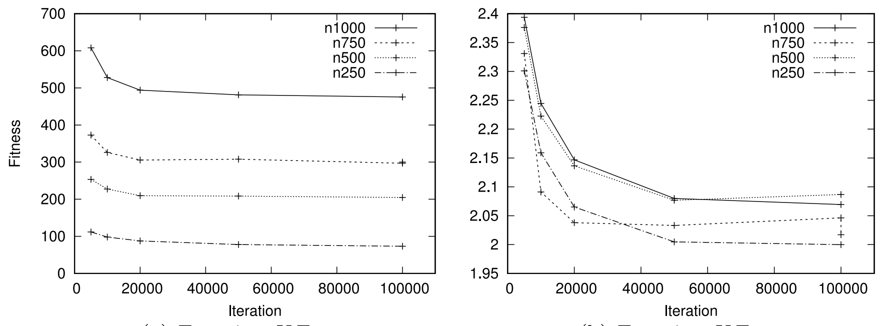
In this section, results of the simulations conducted to compare the performance of *GAWES* with the state of the art genetic algorithm (*GA*) [15] are presented. Both algorithms solve the optimization version of the *MBCT* problem and the objective is to minimize both the total cost and the post-calibration skew of the spanning tree. The experimental results clearly demonstrate that *GAWES* is superior to *GA*, especially in larger graphs.

The organization of the experimental results section can be outlined as follows. In Section 6.1, properties of the datasets and the fitness functions used in the experiments are described. Parameter tuning for *GAWES* is discussed in Section 6.2, and the results of the simulations comparing *GAWES* and *GA* are presented in Section 6.3.

### 6.1. Datasets and fitness functions

The experiments are conducted on three different datasets with various node degree, edge weight, and calibration error distributions. Each dataset includes multiple graph instances with different sizes. The names and the parameters of the graphs used in the experiments are presented in Table 1. During the construction of the graph instances, only positive integers are used as node degree, edge weight, and calibration error values.

*Dataset_A* is the dataset used in [15]. The graph sizes in *Dataset_A* range from 25 to 1000. Node degree distribution is selected as normal distribution with an average of 5 nodes and a standard deviation of 2 nodes. Edge weight distribution is also selected as normal distribution with an average of 10 and standard deviation of 5. For calibration error, uniform distribution is used with a minimum value of 1 and maximum value of 10. To further evaluate the performance of *GAWES* and *GA*, in addition to the existing dataset, two new datasets with various configurations are created in this paper. The parameters of the new *Dataset_B* and *Dataset_C* are also presented in Table 1. In *Dataset_B*, graphs with sizes varying from 100 to 500 are created. The node degree distribution in this dataset is set as uniform with minimum 4 and maximum 8 nodes. Edge weight distribution is selected as uniform also with values between 1 and 10. A uniform distribution is used for calibration error of nodes having values of minimum 1 and maximum 10. In *Dataset_C*, graphs with 100 nodes are created with two different random distributions for edge weight, node degree, and calibration error. Node degree distribution is selected as normal or exponential. In normal distribution the average value is selected as 8 and the standard deviation is 4. In exponential distribution $\lambda$ is selected as 1 and the result is scaled with a scalar value 8. Edge weight distribution is selected as normal or exponential. In normal distribution the average value is selected as 8 and the standard deviation as 4. In exponential distribution $\lambda$ is selected as 1 and the result is scaled with a scalar value of 8.



(a) Function $XF_A$ (b) Function $XF_B$

**Fig. 5.** *y*-axis displays the fitness value of the corresponding fitness functions $XF_A$ (a), and $XF_B$ (b) for various number of iterations of *GAWES*.

**Table 2**
Parameters and formulas of fitness functions used.

| Parameter name | Explanation |
|---|---|
| *MIN_COST* | Minimum spanning tree cost |
| *MAX_COST* | Maximum spanning tree cost |
| *MIN_ERROR* | Minimum post-calibration skew |
| *MAX_ERROR* | Maximum post-calibration skew |
| *norm_cost* | $\frac{(cost - MIN\_COST) \times 100}{(MAX\_COST - MIN\_COST)}$ |
| *norm_error* | $\frac{(error - MIN\_ERROR) \times 100}{(MAX\_ERROR - MIN\_ERROR)}$ |
| **Fitness function name** | **Fitness Function Formula** |
| $F_A$, $XF_A$ | $\frac{(cost - MIN\_COST)^2}{MIN\_COST} + \frac{(error - MIN\_ERROR)^2}{MIN\_ERROR}$ |
| $F_B$, $XF_B$ | $0.1 \times norm\_cost + 0.9 \times norm\_error$ |

Calibration error distribution is selected as normal or exponential. In normal distribution the average value is selected as 0 and the standard deviation is selected as 3. In exponential distribution $\lambda$ is selected as 1 and the result is scaled with a scalar value of 10.

For each graph, *MIN_COST* represents the minimum spanning tree cost of the graph based on edge costs only and *MIN_ERROR* represents the minimum error tree cost of the graph based on post-calibration skew values only. Therefore, for each graph, the ideal solution to achieve is to find a spanning tree with {*MIN_COST*, *MIN_ERROR*} parameters. Again, *MAX_COST* and *MAX_ERROR* represent the upperbounds for the total cost and post-calibration skew for each graph. In order to perform a fair comparison among the algorithms, these parameters are used to calculate two different fitness functions for each algorithm, inline with the ones used in [15]. The fitness functions and the parameters used in these fitness functions are presented in Table 2.

## 6.2. Parameter tuning for GAWES

In this section, experiments are conducted to find the best initial population size and mutation probability parameters for *GAWES*. All the experiments are conducted on *Dataset_A*. The graphs used in the experiments have sizes 250, 500, 750, and 1000. Each parameter tuning experiment is conducted 10 times and the average results are reported. In the first experiment, *GAWES* algorithm is tested with various iterations to find the best iteration count. The results of the experimentation on all four graphs for the fitness functions $XF_A$ and $XF_B$ are presented in Fig. 5. In this experiment, the initial population size is set to 200, mutation probability is set to 0.05, and the change of fitness values with iteration counts varying from 5000 to 100 000 is reported. Fig. 5 clearly demonstrates that the fitness value decreases with increasing iteration counts. Consequently, increasing the iteration count increases the runtime of the algorithm.

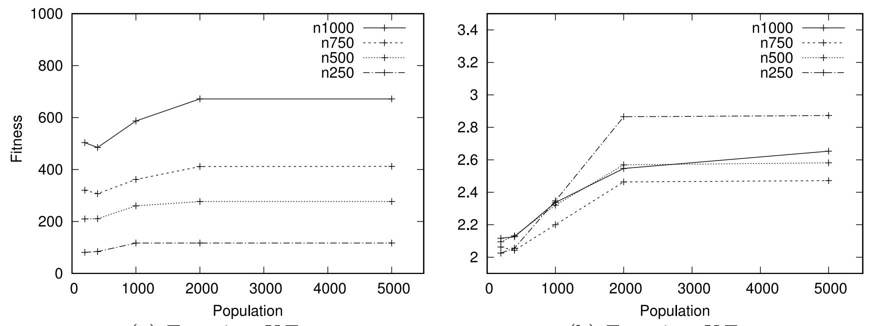
In the next experiment, *GAWES* is tested with various initial population sizes ranging from 200 to 5000. The iteration count for this experiment is selected as 100 000, and the mutation probability is selected as 0.3. It can be observed from Fig. 6 that the fitness values increase for population sizes larger than 400.

Fig. 7 shows the effect of using various mutation probabilities on the fitness value of the *GAWES* algorithm. In this experiment, the iteration count is selected as 40 000 and the initial population size is selected as 1000. As seen from the figure, increasing the mutation probability slightly increases the fitness value, therefore is not preferred.

Initial experiments on parameter tuning suggest a population size of 400 and mutation probability of 0.1, along with an iteration count between 50 000 and 100 000. In order to identify the best combination of values a new experiment is conducted. In this new experiment all combinations of four different mutation rate values (0.001, 0.005, 0.05, 0.1), three different iteration counts (20 000, 50 000, 100 000), and three different population size values (200, 400, 800) are tested on four different graphs. All experiments are conducted 100 times and the average values and the standard deviations are reported in Table 3. The best average values for each fitness function on each graph is reported as bold. Based on the results, parameter set of mutation rate 0.001, iteration count of 100 000, population size of 400 has the best results on 7 out of 8 fitness functions. Examining mutation rates, the results of 0.001 and 0.005 are close to each other, while 0.001 has more best results. In all experiments with iteration count 100 000, population size of 400 gives better results than 200 and 800.

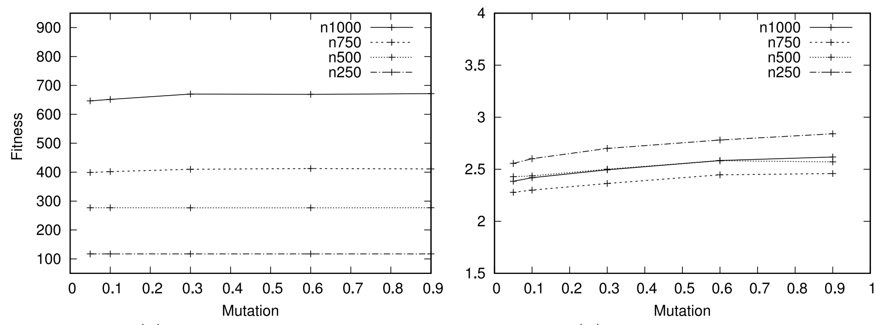
## 6.3. Comparing GAWES with GA

We are now ready to give the performance comparison results of *GAWES* and *GA* over the three existing datasets. Fitness functions, input graphs, iteration counts (or CPU runtimes), initial population sizes and, mutation probabilities are given as input parameters for both genetic algorithms. The details of these parameters are described in Table 4. In the performance comparisons, two different parameter sets are used for *GAWES*. The first set, depicted as *GAWES*, uses the optimal parameters of *GA* as reported in [15] with initial population size selected as 400 and the mutation probability is set to 0.1. The second set, depicted as *GAWES**, uses the best parameters selected in the parameter tuning experiments in Section 6.2 with initial population size selected as 400 and the mutation probability selected as 0.001. On each iteration, *GAWES* investigates extreme efficient solutions and uses more CPU time than *GA*. Therefore, to be fair in comparison, experimentation for both algorithms are done for a fixed amount of time for each graph type and the results are reported. For this reason, in each experiment, the experimentation times are the same for both algorithms, however, number of iterations vary. Each experiment is conducted 100 times and the minimum, maximum, average results



(a) Function $XF_A$ (b) Function $XF_B$

**Fig. 6.** *y*-axis displays the fitness value of the corresponding fitness functions $XF_A$ (a), and $XF_B$ (b) for various population sizes of *GAWES*.



(a) Function $XF_A$ (b) Function $XF_B$

**Fig. 7.** *y*-axis displays the fitness value of the corresponding fitness functions $XF_A$ (a), and $XF_B$ (b) for various mutation probabilities of *GAWES*.

**Table 3**
Results of parameter tuning experiments for *GAWES*.

| Mut | Iter | Pop | 250 | | | | 500 | | | | 750 | | | | 1000 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $XF_A$ | | $XF_B$ | | $XF_A$ | | $XF_B$ | | $XF_A$ | | $XF_B$ | | $XF_A$ | | $XF_B$ | |
| | | | Avg | Std | Avg | Std | Avg | Std | Avg | Std | Avg | Std | Avg | Std | Avg | Std | Avg | Std |
| 0.001 | 20000 | 200 | 85.96 | 4.14 | 2.06 | 0.03 | 211.10 | 5.60 | 2.12 | 0.03 | 306.02 | 8.39 | 2.04 | 0.03 | 492.08 | 18.36 | 2.11 | 0.05 |
| 0.001 | 20000 | 400 | 103.63 | 4.74 | 2.16 | 0.04 | 226.53 | 5.88 | 2.22 | 0.02 | 316.64 | 6.69 | 2.10 | 0.02 | 522.38 | 13.80 | 2.22 | 0.04 |
| 0.001 | 20000 | 800 | 117.25 | 0.00 | 2.68 | 0.07 | 276.92 | 0.24 | 2.47 | 0.02 | 409.68 | 3.89 | 2.33 | 0.02 | 660.34 | 9.91 | 2.44 | 0.02 |
| 0.001 | 50000 | 200 | 79.05 | 4.59 | 2.02 | 0.04 | 206.49 | 5.04 | 2.09 | 0.03 | 303.71 | 9.14 | 2.02 | 0.03 | 481.65 | 20.58 | 2.07 | 0.04 |
| 0.001 | 50000 | 400 | 83.00 | 3.91 | 2.02 | 0.03 | 204.92 | 5.20 | 2.11 | 0.02 | 293.20 | 6.83 | 2.03 | 0.02 | 467.53 | 13.88 | 2.08 | 0.03 |
| 0.001 | 50000 | 800 | 114.98 | 3.16 | 2.24 | 0.05 | 243.97 | 7.62 | 2.26 | 0.02 | 325.92 | 7.08 | 2.13 | 0.02 | 547.16 | 21.47 | 2.22 | 0.04 |
| 0.001 | 100000 | 200 | 74.13 | 4.06 | 2.00 | 0.04 | 206.43 | 4.86 | 2.09 | 0.03 | 303.30 | 9.67 | 2.02 | 0.03 | 477.43 | 22.51 | 2.07 | 0.06 |
| 0.001 | 100000 | 400 | **73.87** | 3.25 | **1.97** | 0.03 | **199.53** | 4.32 | **2.08** | 0.02 | 287.67 | 8.00 | **2.01** | 0.02 | **451.88** | 14.49 | **2.04** | 0.03 |
| 0.001 | 100000 | 800 | 95.23 | 6.12 | 2.08 | 0.04 | 219.71 | 5.86 | 2.14 | 0.02 | 285.60 | 7.75 | 2.03 | 0.02 | 475.15 | 18.28 | 2.08 | 0.04 |
| 0.005 | 20000 | 200 | 85.98 | 4.12 | 2.06 | 0.03 | 210.63 | 5.24 | 2.11 | 0.03 | 307.90 | 8.94 | 2.04 | 0.03 | 491.53 | 17.18 | 2.11 | 0.05 |
| 0.005 | 20000 | 400 | 104.39 | 4.20 | 2.17 | 0.04 | 225.78 | 5.36 | 2.21 | 0.02 | 317.98 | 6.89 | 2.11 | 0.02 | 522.34 | 13.24 | 2.22 | 0.04 |
| 0.005 | 20000 | 800 | 117.25 | 0.00 | 2.70 | 0.07 | 276.87 | 0.41 | 2.48 | 0.03 | 408.57 | 5.14 | 2.33 | 0.02 | 661.66 | 9.99 | 2.44 | 0.02 |
| 0.005 | 50000 | 200 | 78.15 | 3.95 | 2.02 | 0.04 | 207.27 | 4.85 | 2.09 | 0.03 | 303.78 | 9.03 | 2.02 | 0.03 | 483.68 | 18.65 | 2.08 | 0.06 |
| 0.005 | 50000 | 400 | 83.49 | 3.66 | 2.03 | 0.03 | 205.35 | 5.54 | 2.11 | 0.02 | 293.54 | 6.45 | 2.02 | 0.02 | 466.75 | 14.61 | 2.08 | 0.03 |
| 0.005 | 50000 | 800 | 114.84 | 3.30 | 2.23 | 0.04 | 242.84 | 7.51 | 2.26 | 0.02 | 327.38 | 6.55 | 2.13 | 0.02 | 550.89 | 18.97 | 2.22 | 0.03 |
| 0.005 | 100000 | 200 | 75.24 | 4.23 | 2.00 | 0.04 | 206.36 | 4.93 | **2.08** | 0.03 | 302.81 | 9.92 | 2.03 | 0.03 | 476.87 | 20.64 | 2.08 | 0.06 |
| 0.005 | 100000 | 400 | 73.89 | 3.53 | **1.97** | 0.04 | 199.81 | 4.34 | **2.08** | 0.02 | 289.87 | 8.36 | **2.01** | 0.02 | 452.41 | 13.29 | **2.04** | 0.03 |
| 0.005 | 100000 | 800 | 96.16 | 6.27 | 2.08 | 0.04 | 220.69 | 6.72 | 2.13 | 0.02 | **284.89** | 6.13 | 2.03 | 0.02 | 472.37 | 18.02 | 2.09 | 0.03 |
| 0.05 | 20000 | 200 | 86.04 | 4.47 | 2.06 | 0.03 | 211.35 | 5.55 | 2.12 | 0.03 | 308.69 | 10.16 | 2.05 | 0.03 | 498.12 | 17.74 | 2.12 | 0.05 |
| 0.05 | 20000 | 400 | 107.05 | 4.39 | 2.19 | 0.04 | 229.79 | 6.01 | 2.24 | 0.02 | 325.92 | 6.79 | 2.12 | 0.02 | 533.19 | 15.22 | 2.25 | 0.03 |
| 0.05 | 20000 | 800 | 117.24 | 0.05 | 2.71 | 0.07 | 276.91 | 0.31 | 2.49 | 0.03 | 409.60 | 4.50 | 2.34 | 0.02 | 664.29 | 7.83 | 2.45 | 0.02 |
| 0.05 | 50000 | 200 | 79.31 | 4.36 | 2.02 | 0.03 | 207.30 | 4.46 | 2.09 | 0.04 | 303.59 | 9.04 | 2.03 | 0.03 | 481.55 | 16.64 | 2.08 | 0.05 |
| 0.05 | 50000 | 400 | 85.68 | 3.98 | 2.04 | 0.03 | 207.35 | 5.46 | 2.12 | 0.02 | 296.82 | 7.88 | 2.03 | 0.02 | 472.65 | 14.19 | 2.09 | 0.03 |
| 0.05 | 50000 | 800 | 116.81 | 1.33 | 2.27 | 0.04 | 248.79 | 6.92 | 2.28 | 0.02 | 337.01 | 7.37 | 2.15 | 0.02 | 556.14 | 17.45 | 2.25 | 0.03 |
| 0.05 | 100000 | 200 | 76.29 | 3.92 | 2.00 | 0.03 | 204.14 | 4.74 | 2.09 | 0.03 | 302.96 | 9.26 | 2.03 | 0.03 | 475.03 | 16.15 | 2.07 | 0.05 |
| 0.05 | 100000 | 400 | 75.07 | 3.40 | 1.98 | 0.03 | 200.58 | 4.29 | **2.08** | 0.02 | 291.98 | 7.22 | **2.01** | 0.02 | 453.18 | 14.54 | 2.05 | 0.03 |
| 0.05 | 100000 | 800 | 99.97 | 5.43 | 2.10 | 0.04 | 223.19 | 6.05 | 2.15 | 0.02 | 290.64 | 6.45 | 2.03 | 0.02 | 481.48 | 17.60 | 2.11 | 0.04 |
| 0.1 | 20000 | 200 | 89.27 | 4.68 | 2.07 | 0.03 | 212.09 | 4.97 | 2.13 | 0.03 | 314.10 | 8.64 | 2.05 | 0.03 | 499.21 | 16.58 | 2.16 | 0.05 |
| 0.1 | 20000 | 400 | 108.70 | 3.70 | 2.21 | 0.04 | 234.50 | 6.16 | 2.25 | 0.02 | 332.02 | 7.30 | 2.14 | 0.02 | 544.94 | 14.83 | 2.27 | 0.04 |
| 0.1 | 20000 | 800 | 117.24 | 0.05 | 2.73 | 0.07 | 276.93 | 0.18 | 2.50 | 0.03 | 410.24 | 3.96 | 2.36 | 0.02 | 667.88 | 5.68 | 2.47 | 0.02 |
| 0.1 | 50000 | 200 | 81.17 | 3.95 | 2.02 | 0.03 | 206.24 | 4.48 | 2.09 | 0.03 | 306.79 | 9.42 | 2.04 | 0.03 | 485.65 | 17.62 | 2.09 | 0.05 |
| 0.1 | 50000 | 400 | 87.30 | 4.32 | 2.06 | 0.03 | 208.93 | 6.83 | 2.13 | 0.02 | 300.37 | 8.50 | 2.04 | 0.02 | 481.05 | 13.20 | 2.11 | 0.03 |
| 0.1 | 50000 | 800 | 116.86 | 1.18 | 2.30 | 0.04 | 255.12 | 5.85 | 2.29 | 0.02 | 345.61 | 7.21 | 2.17 | 0.02 | 567.60 | 17.11 | 2.29 | 0.03 |
| 0.1 | 100000 | 200 | 77.04 | 3.91 | 1.99 | 0.03 | 204.6 | 5.67 | **2.08** | 0.03 | 304.62 | 10.22 | 2.03 | 0.04 | 473.67 | 16.63 | 2.07 | 0.06 |
| 0.1 | 100000 | 400 | 76.95 | 3.51 | 2.00 | 0.03 | 201.37 | 4.22 | 2.09 | 0.02 | 292.30 | 7.52 | **2.01** | 0.02 | 461.68 | 13.98 | 2.05 | 0.03 |
| 0.1 | 100000 | 800 | 102.61 | 4.76 | 2.13 | 0.04 | 227.40 | 5.16 | 2.16 | 0.02 | 297.75 | 7.38 | 2.05 | 0.02 | 489.18 | 19.83 | 2.14 | 0.03 |

**Table 4**
Parameters of *GAWES* and *GA*.

| Parameter name | Description | Range |
|---|---|---|
| Fitness function | Fitness function used in the algorithms as described in Table 2 | $F_A$, $F_B$, $XF_A$, $XF_B$ |
| Runtime | Fixed runtime in seconds for each algorithm | $(graph\_size \times 2)$ seconds |
| Iteration | Number of iterations | varies with runtime |
| Population size | Initial population size | 400 |
| Mutation probability | Genetic algorithm mutation probability | 0.1(*GA*, *GAWES*), 0.001(*GAWES**) |

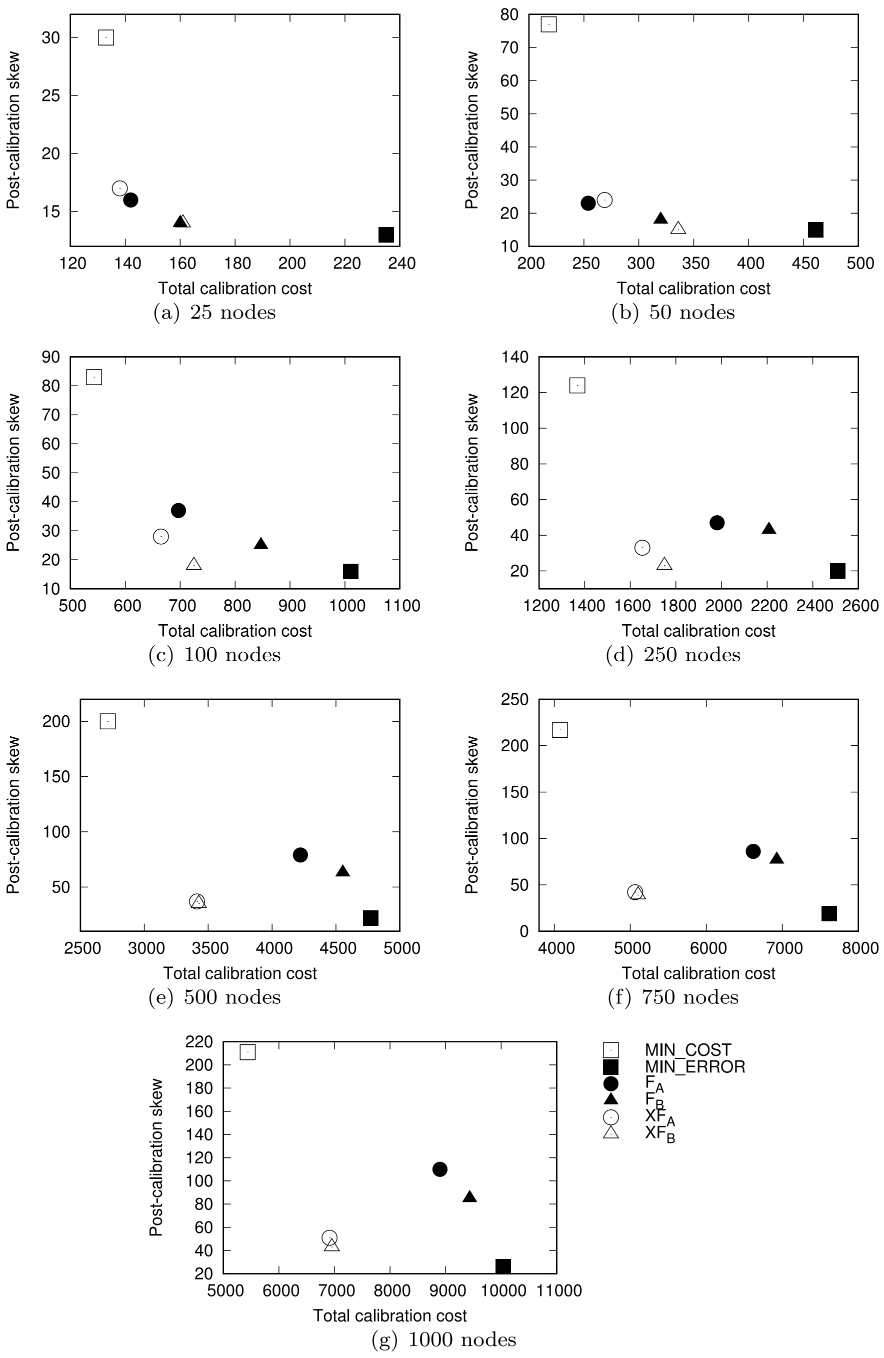
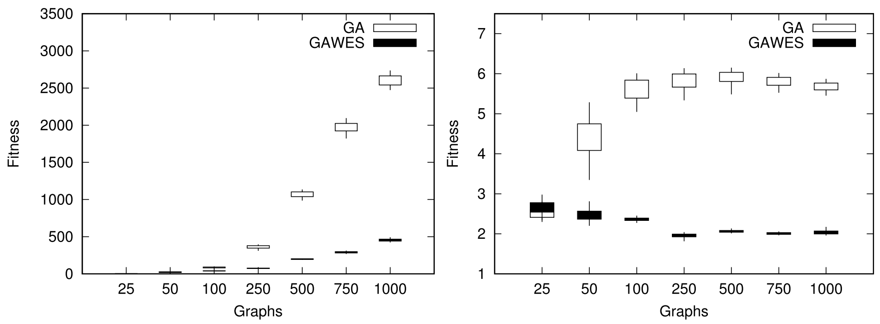


**Fig. 8.** Results of the GAWES and the GA algorithms for all fitness functions on each graph in *Dataset_A*.

and the standard deviations are reported. Experiments are run on a computer with 2 GHz Intel Xeon Gold 6138 CPU and 512 GB RAM.

Fig. 8 plots the results of the experimentation on *Dataset_A*. The best results of both algorithms for all fitness functions on each graph are displayed in respective figures. In each figure, left axis shows post-calibration skew and bottom axis shows total calibration cost for each spanning tree result. The values of the fitness functions $F_A$ and $F_B$ are given as the results of the *GA* algorithm while the values of $XF_A$ and $XF_B$ are given as the results of the *GAWES* algorithm. In the figures, a solution $x$ is superior to another solution $y$ if and only if both total calibration cost value and post-calibration skew value of $x$ is smaller than the respective



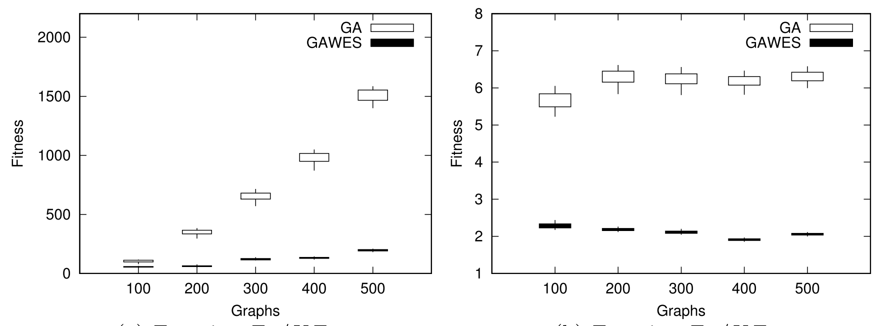
(a) Function $F_A/XF_A$ (b) Function $F_B/XF_B$

**Fig. 9.** Results of *GAWES* and *GA* on graphs in *Dataset_A*. The vertical lines mark the minimum and the maximum values and the boxes display the average values along with the standard deviations.

**Table 5**
Results of *GAWES* and *GA* on graphs in *Dataset_A*. Time values are given in seconds.

| | Graph size | | 25 | 50 | 100 | 250 | 500 | 750 | 1000 |
|---|---|---|---|---|---|---|---|---|---|
| | MIN_COST | | 133 | 218 | 543 | 1368 | 2714 | 4076 | 5439 |
| | MIN_ERROR | | 13 | 15 | 16 | 20 | 22 | 19 | 26 |
| | CPU TIME | | 50s | 100s | 200s | 500s | 1000s | 1500s | 2000s |
| GA | $F_A$ | Cost | 142 | 254 | 697 | 1981 | 4222 | 6618 | 8899 |
| | | Error | 16 | 23 | 37 | 47 | 79 | 86 | 110 |
| | | Avg $F_A$ | 1.67 | 22.18 | 86.42 | 362.83 | 1070.42 | 1973.48 | 2602.68 |
| | | Std $F_A$ | 0.20 | 3.14 | 5.95 | 17.74 | 31.37 | 49.96 | 61.46 |
| | $F_B$ | Cost | 160 | 320 | 847 | 2208 | 4554 | 6927 | 9434 |
| | | Error | 14 | 18 | 25 | 43 | 63 | 77 | 85 |
| | | Avg $F_B$ | 2.51 | 4.42 | 5.62 | 5.83 | 5.92 | 5.81 | 5.68 |
| | | Std $F_B$ | 0.09 | 0.33 | 0.22 | 0.16 | 0.11 | 0.10 | 0.09 |
| GAWES | $XF_A$ | Cost | 138 | 269 | 665 | 1653 | 3412 | 5065 | 6911 |
| | | Error | 17 | 24 | 28 | 33 | 37 | 42 | 51 |
| | | Avg $XF_A$ | 2.43 | 24.34 | 40.72 | 74.64 | 198.81 | 290.37 | 454.56 |
| | | Std $XF_A$ | 0.45 | 3.13 | 1.56 | 3.34 | 4.35 | 8.37 | 13.24 |
| | $XF_B$ | Cost | 161 | 336 | 725 | 1750 | 3426 | 5102 | 6952 |
| | | Error | 14 | 15 | 18 | 23 | 35 | 39 | 43 |
| | | Avg $XF_B$ | 2.66 | 2.47 | 2.36 | 1.96 | 2.06 | 2.01 | 2.03 |
| | | Std $XF_B$ | 0.12 | 0.10 | 0.03 | 0.03 | 0.02 | 0.02 | 0.04 |
| GAWES* | $XF_A$ | Cost | 141 | 259 | 661 | 1636 | 3413 | 5066 | 6891 |
| | | Error | 17 | 25 | 28 | 35 | 37 | 42 | 52 |
| | | Avg $XF_A$ | 2.50 | 22.30 | 40.77 | 71.34 | 198.62 | 287.96 | 449.22 |
| | | Std $XF_A$ | 0.39 | 3.11 | 1.64 | 3.28 | 4.10 | 7.25 | 14.44 |
| | $XF_B$ | Cost | 161 | 334 | 724 | 1688 | 3453 | 5092 | 6910 |
| | | Error | 14 | 15 | 18 | 27 | 33 | 39 | 43 |
| | | Avg $XF_B$ | 2.63 | 2.43 | 2.36 | 1.94 | 2.06 | 2.00 | 2.02 |
| | | Std $XF_B$ | 0.13 | 0.11 | 0.03 | 0.02 | 0.02 | 0.02 | 0.04 |

values of solution $y$. As seen in the figures, except the smallest graphs ($n$25 and $n$50), *GAWES* results are superior compared to *GA* results for fitness functions $F_A/XF_A$ and $F_B/XF_B$. As the graph size increases, the difference in the results also dramatically increase, where *GAWES* performs much better than *GA* for both fitness functions $F_A/XF_A$ and $F_B/XF_B$. The results are presented in more detail in Table 5, including the runtime in seconds for each run of the algorithms, average and standard deviation for all fitness values, and the *GAWES** results. Comparing *GAWES* with *GAWES**, it is observed that based on the average fitness values *GAWES** is superior on the datasets with size 50, 250, 750, and 1000. A more detailed result of the experiments including the minimum, maximum, average values and the standard deviations are presented in Fig. 9 for both fitness functions. The vertical lines mark the minimum and the maximum values and the boxes display the average values along with the standard deviations. Fig. 9(a) plots



(a) Function $F_A/XF_A$ (b) Function $F_B/XF_B$

**Fig. 10.** Results of *GAWES* and *GA* on graphs in *Dataset_B*. The vertical lines mark the minimum and the maximum values and the boxes display the average values along with the standard deviations.

**Table 6**
Results of *GAWES* and *GA* on graphs in *Dataset_B*. Time values are given in seconds.

| Graph size | | | 100 | 200 | 300 | 400 | 500 |
|---|---|---|---|---|---|---|---|
| MIN_COST | | | 177 | 379 | 576 | 746 | 929 |
| MIN_ERROR | | | 18 | 19 | 18 | 23 | 21 |
| CPU TIME | | | 200s | 400s | 600s | 800s | 1000s |
| GA | $F_A$ | Cost | 275 | 675 | 1092 | 1484 | 1967 |
| | | Error | 39 | 54 | 62 | 80 | 92 |
| | | Avg $F_A$ | 104.84 | 350.49 | 655.21 | 983.01 | 1509.83 |
| | | Std $F_A$ | 7.25 | 15.60 | 25.47 | 33.18 | 43.14 |
| | $F_B$ | Cost | 354 | 830 | 1288 | 1806 | 2238 |
| | | Error | 30 | 43 | 49 | 55 | 71 |
| | | Avg $F_B$ | 5.66 | 6.30 | 6.24 | 6.19 | 6.31 |
| | | Std $F_B$ | 0.18 | 0.15 | 0.13 | 0.11 | 0.11 |
| GAWES | $XF_A$ | Cost | 261 | 496 | 816 | 1022 | 1306 |
| | | Error | 29 | 36 | 32 | 42 | 45 |
| | | Avg $XF_A$ | 56.03 | 61.09 | 120.22 | 131.91 | 196.19 |
| | | Std $XF_A$ | 3.07 | 3.53 | 5.89 | 4.41 | 5.89 |
| | $XF_B$ | Cost | 276 | 528 | 837 | 1062 | 1327 |
| | | Error | 21 | 29 | 28 | 35 | 41 |
| | | Avg $XF_B$ | 2.28 | 2.18 | 2.11 | 1.91 | 2.06 |
| | | Std $XF_B$ | 0.05 | 0.03 | 0.03 | 0.02 | 0.02 |
| GAWES* | $XF_A$ | Cost | 264 | 496 | 810 | 1020 | 1302 |
| | | Error | 29 | 34 | 31 | 42 | 45 |
| | | Avg $XF_A$ | 55.77 | 55.92 | 112.43 | 127.08 | 193.08 |
| | | Std $XF_A$ | 2.45 | 3.21 | 4.31 | 4.08 | 6.85 |
| | $XF_B$ | Cost | 276 | 584 | 854 | 1060 | 1315 |
| | | Error | 21 | 24 | 26 | 35 | 41 |
| | | Avg $XF_B$ | 2.25 | 2.18 | 2.07 | 1.90 | 2.04 |
| | | Std $XF_B$ | 0.04 | 0.03 | 0.03 | 0.02 | 0.03 |

the result for fitness function $F_A/XF_A$. For graph size 25, minimum, maximum, average, and standard deviation of *GA* are 1.30, 2.17, 1.67, and 0.20, respectively. Again for graph size 25, the *GAWES* results are 1.42, 3.61, 2.43, and 0.45, respectively. For graph size 50, minimum, maximum, average, and standard deviation of *GA* are 10.21, 27.78, 22.18, and 3.14, respectively. Again for graphs size 50, the *GAWES* results are 17.33, 32.08, 24.34, and 3.13, respectively. For graph size 100, *GA* has minimum 71.24, maximum 101.09, average 86.42 and standard deviation of 5.95. *GAWES* on graph size 100 has minimum 36.41, maximum 46.31, average 40.72 and standard deviation of 1.56. As seen in Fig. 9(a), the results of *GA* and *GAWES* for graphs 25 and 50 are close to each other, but for graphs with size 100 and larger *GAWES* clearly outperforms *GA* for fitness function $F_A/XF_A$. Examining fitness function $F_B/XF_B$ results in Fig. 9(b), for graph size 25, minimum, maximum, average, and standard deviation of *GA* are 2.30, 2.80, 2.51, and 0.09, respectively. Again for graphs size 25, the *GAWES* results are 2.34, 2.98, 2.66, and 0.12, respectively. As seen in Fig. 9(b), the results of *GA* and *GAWES* for graph size 25 are close to each other, however, for graphs with size 50 and larger *GAWES* clearly outperforms *GA* for

**Table 7**
Results of *GAWES* and *GA* on graphs in *Dataset_C*. Time values are given in seconds.

| | | Graph acronym | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Graph size | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| | | Calib. error dist. | exp. | exp. | exp. | exp. | normal | normal | normal | normal |
| | | Node degree dist. | exp. | exp. | normal | normal | exp. | exp. | normal | normal |
| | | Edge weight dist. | exp. | normal | exp. | normal | exp | normal | exp. | normal |
| | | MIN_COST | 156 | 313 | 179 | 334 | 173 | 318 | 147 | 310 |
| | | MIN_ERROR | 12 | 11 | 11 | 13 | 11 | 11 | 9 | 11 |
| | | CPU TIME | 200s | 200s | 200s | 200s | 200s | 200s | 200s | 200s |
| GA | $F_A$ | Cost | 261 | 430 | 299 | 472 | 285 | 450 | 259 | 432 |
| | | Error | 27 | 28 | 29 | 22 | 23 | 20 | 26 | 26 |
| | | Avg $F_A$ | 116.98 | 91.51 | 141.62 | 80.41 | 114.26 | 74.14 | 148.98 | 86.87 |
| | | Std $F_A$ | 10.45 | 6.99 | 11.34 | 6.05 | 10.67 | 4.72 | 11.80 | 6.27 |
| | $F_B$ | Cost | 570 | 613 | 565 | 574 | 492 | 572 | 499 | 635 |
| | | Error | 12 | 16 | 16 | 18 | 14 | 14 | 12 | 13 |
| | | Avg $F_B$ | 2.68 | 4.84 | 4.23 | 4.84 | 3.05 | 3.81 | 3.73 | 4.54 |
| | | Std $F_B$ | 0.20 | 0.22 | 0.19 | 0.19 | 0.19 | 0.18 | 0.23 | 0.21 |
| GAWES | $XF_A$ | Cost | 185 | 425 | 239 | 443 | 228 | 407 | 201 | 407 |
| | | Error | 17 | 19 | 19 | 28 | 14 | 17 | 15 | 19 |
| | | Avg $XF_A$ | 10.55 | 50.59 | 30.08 | 71.47 | 21.40 | 33.61 | 30.14 | 42.50 |
| | | Std $XF_A$ | 0.44 | 1.88 | 3.07 | 5.75 | 0.36 | 3.40 | 2.66 | 2.38 |
| | $XF_B$ | Cost | 222 | 508 | 356 | 514 | 301 | 446 | 257 | 446 |
| | | Error | 12 | 11 | 12 | 13 | 11 | 11 | 9 | 11 |
| | | Avg $XF_B$ | 0.38 | 1.95 | 1.33 | 1.96 | 0.71 | 1.37 | 0.65 | 1.44 |
| | | Std $XF_B$ | 0.02 | 0.05 | 0.03 | 0.04 | 0.03 | 0.04 | 0.04 | 0.03 |
| GAWES* | $XF_A$ | Cost | 193 | 421 | 238 | 449 | 231 | 410 | 196 | 406 |
| | | Error | 16 | 18 | 19 | 27 | 14 | 14 | 15 | 18 |
| | | Avg $XF_A$ | 10.64 | 49.03 | 27.15 | 71.20 | 21.43 | 35.58 | 28.84 | 42.19 |
| | | Std $XF_A$ | 0.05 | 2.72 | 1.54 | 6.91 | 0.17 | 4.05 | 3.22 | 2.64 |
| | $XF_B$ | Cost | 222 | 506 | 355 | 514 | 298 | 455 | 256 | 445 |
| | | Error | 12 | 11 | 12 | 13 | 11 | 11 | 9 | 11 |
| | | Avg $XF_B$ | 0.36 | 1.91 | 1.31 | 1.95 | 0.70 | 1.40 | 0.64 | 1.43 |
| | | Std $XF_B$ | 0.02 | 0.05 | 0.03 | 0.04 | 0.03 | 0.01 | 0.04 | 0.03 |

fitness function $F_B/XF_B$. As a summary it can be concluded that even though both algorithms are executed for the same amount of time, *GAWES* outperforms *GA* on all graphs in *Dataset_A* except the smallest two.

The best, average, and standard deviation results for *Dataset_B* are reported in Table 6. Experiment is conducted on graphs with sizes varying from 100 to 500. Node degree, edge weight, and calibration error distributions of all the graphs are uniform distributions. Fig. 10 plots the minimum, the maximum, the average values and the standard deviations for both fitness functions. The vertical lines mark the minimum and the maximum values and the boxes display the average values along with the standard deviations. As seen from Table 6 and Fig. 10, even though both *GAWES* and *GA* are given the exact same runtime, *GAWES* outperforms *GA* on all instances and for both fitness functions. Observing the average results in Table 6, the ratio of the average fitness results $F_A/XF_A$ increases from 1.87 on graph with size 100 to 7.70 on graph with size 500. For $F_B/XF_B$, the ratio increases from 2.48 to 3.06 for the same graphs. Based on these numbers, it is safe to conclude that even though *GAWES* is superior to *GA* on all graphs, the gap between the results increases even more with the graph size. Comparing *GAWES* and *GAWES** based on the results in Table 6, it is observed that *GAWES** is equal or superior to *GAWES* on all graphs in *Dataset_B*.

The experimentation results on *Dataset_C* are presented in Table 7 and Fig. 11. As denoted in Table 7, *Dataset_C* has graphs with all possible combinations of normal and exponential distributions for node degree, edge weight, and calibration error. Table 7 reports the input parameters of each dataset configuration along with the best, average and standard deviation results of the simulations on the datasets for each algorithm. Fig. 11 plots the minimum, the maximum, the average values and the standard deviations for both fitness functions. The vertical lines mark the minimum and the maximum values and the boxes display the average values along with the standard deviations. A thorough examination of the results reveals that in this dataset, except for graph *D* with fitness function $F_A/XF_A$, *GAWES* is superior to *GA* on all graphs and for both fitness functions. Comparing *GAWES* with *GAWES** for average values of fitness function $XF_A$, *GAWES** is superior on all graphs except *A*, *E* and *F*, and for fitness function $XF_B$ again *GAWES** is superior on all graphs except *F*.

Closely examining Table 7 for the results of the fitness function $XF_B$ reveals that $XF_B$ obtained the *MIN_ERROR* on 7 out of 8 graphs in the dataset. Therefore, use of $XF_B$ as a fitness function suits the applications for which minimizing the post-calibration skew is the primary objective and hence more important than minimizing the total cost.

When comparing fitness functions $XF_A$ and $XF_B$, it is clear that $XF_A$ balances total calibration cost and post-calibration skew while

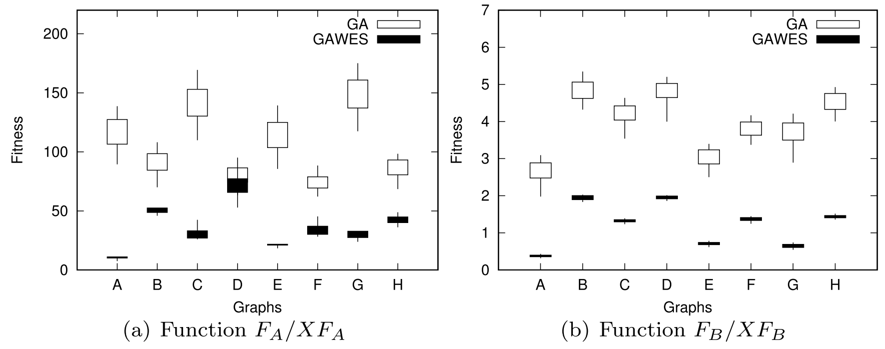


**Fig. 11.** Results of *GAWES* and *GA* on graphs in *Dataset_C*. The vertical lines mark the minimum and the maximum values and the boxes display the average values along with the standard deviations.

$XF_B$ minimizes post-calibration skew in exchange of higher total calibration costs. Therefore, the use of the best fitness function depends on the application at hand. *GAWES* is designed generic enough so that the algorithm can work with any fitness function generated based on total cost and post-calibration skew.

As a conclusion to the experimentation, the main definitive feature of *GAWES* compared to *GA* is the mechanism to generate and use extreme efficient solutions within the genetic algorithm. Therefore, the superiority of the results of *GAWES* over *GA* can be safely attributed to this feature. The extreme efficient solutions method depends on generating new chromosomes from existing chromosomes. These newly generated chromosomes, if exist, are extreme efficient solutions themselves, therefore guide the search towards the ideal point where both the cost and the error are minimized. Using a mechanism like this in the genetic algorithm improves the quality of the population and as a result generates superior solutions. One second benefit of using extreme efficient solutions is that it gives the genetic algorithm a new way of creating chromosomes in addition to regular crossover. [22] reports that random spanning tree creation for encoding new chromosomes can have bias problems towards a star like or a path like topology depending on the method used. Both *GA* and *GAWES* use Edge-Set chromosome encoding with KruskalRST as the algorithm to generate random chromosomes and to fix infeasible chromosomes. Therefore, as [22] reports, use of a single algorithm to create and fix chromosomes tends to create a bias in the population towards a certain type of tree topology. However, extreme efficient solutions use a completely different method to create new child chromosomes, therefore enriches the population by adding variety. As a summary, the use of extreme efficient solutions improves the quality based on these two benefits; by creating new extreme efficient solutions as child chromosomes, and by adding a new method alternative to crossover for generating new child chromosomes.

## 7. Conclusion

In this paper, a novel genetic algorithm based solution (*GAWES*) to the optimization version of the *MBCT* problem is proposed. The main novelty of *GAWES* is the use of extreme efficient solutions within the genetic algorithm. Experimental evaluation results on three different datasets confirm that *GAWES* algorithm is superior to the existing state of the art genetic algorithm both in energy efficiency and calibration accuracy. As a future work, *GAWES* algorithm will be applied to other bicriteria spanning tree problems.

## References

[1] I.-J. Su, C.-C. Tsai, W.-T. Sung, Area temperature system monitoring and computing based on adaptive fuzzy logic in wireless sensor networks, Appl. Soft Comput. 12 (5) (2012) 1532–1541.

[2] P. Braca, R. Goldhahn, G. Ferri, K.D. LePage, Distributed information fusion in multistatic sensor networks for underwater surveillance, IEEE Sens. J. 16 (11) (2016) 4003–4014.

[3] R.A. Lara-Cueva, R. Gordillo, L. Valencia, D.S. Benítez, Determining the main csma parameters for adequate performance of wsn for real-time volcano monitoring system applications, IEEE Sens. J. 17 (5) (2017) 1493–1502.

[4] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, M. Welsh, Deploying a wireless sensor network on an active volcano, IEEE Internet Comput. 10 (2) (2006) 18–25.

[5] P. Buonadonna, D. Gay, J. Hellerstein, W. Hong, S. Madden, Task: sensor network in a box, in: EWSN, Istanbul, Turkey, 2005, pp. 133–144.

[6] 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 Trans. Sen. Netw. 5 (25) (2009) 1–25.

[7] N. Ramanathan, T. Schoellhammer, E. Kohler, K. Whitehouse, T. Harmon, D. Estrin, Suelo: human-assisted sensing for exploratory soil monitoring studies. In: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys'09, Berkeley, California, 2009, pp. 197–210.

[8] A.B. Sharma, L. Golubchik, R. Govindan, Sensor faults: detection methods and prevalence in real-world datasets, ACM Trans. Sen. Netw. 6 (2010) 23:1–23:39.

[9] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, W. Hong, A macroscope in the redwoods, in: SenSys, San Diego, California, USA, 2005, pp. 51–63.

[10] L. Balzano, R. Nowak, Blind calibration of sensor networks, in: IPSN, pp. 79–88, Cambridge, Massachusetts, USA, 2007.

[11] V. Bychkovskiy, S. Megerian, D. Estrin, M. Potkonjak, A collaborative approach to in-place sensor calibration. In IPSN, pages 301–316, Palo Alto, CA, USA, 2003.

[12] M. Takruri, S. Challa, R. Chakravorty, Recursive bayesian approaches for auto calibration in drift aware wireless sensor networks, J. Netw. 5 (2010) 823–832.

[13] C. Taylor, A. Rahimi, J. Bachrach, H. Shrobe, A. Grue, Simultaneous localization, calibration, and tracking in an ad hoc sensor network, in: IPSN, Nashville, Tennessee, USA, 2006, pp. 27–33.

[14] K. Whitehouse, D. Culler, Calibration as parameter estimation in sensor networks, in: WSNA, Atlanta, Georgia, USA, 2002, pp. 59–67.

[15] H. Akcan, On the complexity of energy efficient pairwise calibration in embedded sensors, Appl. Soft Comput. 13 (4) (2013) 1766–1773.

[16] P. Piggott, F. Suraweera, Encoding graphs for genetic algorithms: An investigation using the minimum spanning tree problem, in: Progress in Evolutionary Computation, Berlin, Heidelberg, 1995, pp. 305–314.

[17] 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, Orlando, Florida, USA, 1994, pp. 379–384.

[18] H. Prüfer, Neuer beweis eines satzes über permutationen, Arch. Math. Phys. 27 (1918) 742–744.

[19] B.C. Julstrom, The blob code: A better string coding of spanning trees for evolutionary search, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) Workshop Program, San Mateo, California, 2001, pp. 256–261.

[20] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA J. Comput. 6 (2) (1994) 154–160.

[21] F. Rothlauf, D.E. Goldberg, A. Heinzl, Bad codings and the utility of well-designed genetic algorithms, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, Las Vegas, Nevada, USA, 2000, pp. 355–364.

[22] G.R. Raidl, B.A. Julstrom, Edge sets: an effective evolutionary coding of spanning trees, IEEE Trans. Evol. Comput. 7 (3) (2003) 225–239.

[23] L. Gouveia, L. Simonetti, E. Uchoa, Modeling hop-constrained and diameter-constrained minimum spanning tree problems as steiner tree problems over layered graphs, Math. Program. 128 (1–2) (2011) 123–148.

[24] L. Gouveia, A. Paias, D. Sharma, Modeling and solving the rooted distance-constrained minimum spanning tree problem, Comput. OR 35 (2) (2008) 600–613.
[25] M. Ehrgott, X. Gandibleux, A survey and annotated bibliography of multiobjective combinatorial optimization, OR Spektrum 22 (2000) 425–460.
[26] H.W. Hamacher, G. Ruhe, On spanning tree problems with multiple objectives, Ann. Oper. Res. 52 (4) (1994) 209–230.
[27] S. Steiner, T. Radzik, Computing all efficient solutions of the biobjective minimum spanning tree problem, Comput. OR 35 (1) (2008) 198–211.
[28] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.