On The Complexity Of Energy Efficient Pairwise Calibration In Embedded Sensors

Hüseyin Akcan*

Dept. of Software Engineering, Izmir University of Economics 35330, Balçova, Izmir, Turkey

Abstract

Technological advances in nanotechnology enabled the use of microelectromechanical systems (MEMS) in various application areas. With the integration of various sensor devices into MEMS, autonomously calibrating these sensors become a major research problem. When performing calibration on real-world embedded sensor network deployments, random errors due to internal and external factors alter the calibration parameters and eventually effect the calibration quality in a negative way. Therefore, during autonomous calibration, calibration paths which has low cost and low error values are preferable. To tackle the calibration problem on embedded wireless sensor networks, we present an energy efficient and minimum error calibration model, and also prove that due to random errors the problem turns into an NP-complete problem. To the best of our knowledge this is the first time a formal proof is presented on the complexity of an iterative calibration based problem when random errors are present in the measurements. We also conducted heuristic tests using genetic algorithm to solve the optimization version of the problem, on various graphs. The NP-completeness result also reveals that more research is needed to examine the complexity of calibration in a more general framework in real-world sensor network deployments.

Keywords: NP-completeness, heuristic algorithms, embedded sensor networks.

1. Introduction

Recent advances in sensor technology enabled low cost, small sized embedded devices to be integrated into our daily life. Advanced nanoscale electronics integrated in microelectromechanical systems (MEMS) are getting increasingly common every day. The advances in the manufacturing technology also triggered the use of these devices embedded in smart nodes called sensor nodes, which eventually formed into networks of sensors connected through wireless communication. Today sensor networks have a wide application area, from remote temperature monitoring [1] to fault diagnosis [2]. Due to manufacturing defects, or caused by environmental conditions over time, each sensor needs to be calibrated [3, 4]. Traditionally, calibration is done in controlled environments, such as laboratories equipped with specialized calibration hardware based on well known standards. The calibration can be done by physically adjusting the hardware, or in a non-intrusive way by adjusting the parameters of the sensor.

Calibration is known to correct only the systematic errors in measurements. However, systematic errors are not the only type of error observed in real-world deployments. The measurement errors are classified as systematic and random errors. Each measurement has unpredictable random errors due to environmental noise, precision of the equipment, or manufacturing defects in the sensors. Therefore, in real-world deployments, random measurement errors are inevitable. Furthermore, these random measurement errors interfere with the calibration process and alter the calibration parameters.

[5, 6, 7] report results of calibration in real-world sensor network deployments. Buonadonna *et al.*[5] states calibration as one of the most challenging tasks in real-world sensor deployments. The challenges can be summarized as difficulty of calibrating a massive number of sensors and inconveniences at physically accessing the sensors as they may be deployed in harsh or even hostile environments. Moreover, the sensors are presumed to stay active for long periods of times after deployment, and therefore expected to be calibrated periodically due to environmental conditions or internal defects. For these reasons traditional calibration methods are not directly applicable to sensor networks. In an attempt to solve this problem, parametric calibration methods have been proposed [8, 9, 10, 11, 12]. Calibration is also investigated on mobile sensor networks [13, 14], and source localization in acoustic sensing platforms [15].

In *parametric calibration*, a calibration function is defined, that maps the output value of a target sensor to a reference sensor's reported value by adjusting the parameters of the target sensor. The process of calibrating one sensor against a reference sensor by using a calibration function is also known as *pairwise calibration* [9]. Pairwise calibration is performed among closeby sensor pairs so that the correlation among their sensor readings when

*Corresponding author. Tel: +90 232 488 8287, Fax: +90 232 488 8475

Email address: huseyin.akcan@ieu.edu.tr (Hüseyin Akcan)

observing the same event is exploited to perform the calibration. In wireless sensor networks the process of using pairwise calibration iteratively in such a way that sensors already calibrated are used to calibrate uncalibrated sensors is known as *iterative calibration* [12]. The main advantage of iterative calibration over reference broadcasting is that iterative calibration does not require all the sensors in the network to observe the same event at the same time.

In Section 2 we discuss the consequences of iterative calibration on sensor networks, and present a calibration model that minimizes the maximum error due to calibration by using the minimum energy on a wireless sensor network. In Section 3 we present our genetic algorithm as a heuristic solution to the given problem, and in Section 4 we show the results of the various experiments we conducted with the genetic algorithm. Finally, we present the conclusion in Section 5.

2. The Problem Definition

Sensor networks are subject to environmental conditions, and they need to be calibrated periodically, therefore the pairwise calibration, which depends on one-to-one wireless communication among neighbour sensors, should be done in an energy efficient way. Our main objective in this paper is to perform energy efficient calibration in sensor networks while minimizing the post-calibration error of each sensor. To succeed in our objective, we have to provide two guarantees during our pairwise calibration: (1) the calibration path should span all the sensors by using the minimum energy, (2) throughout the calibration, the maximum error introduced should be minimized. We further discuss the reasonability of these two assumptions below.

(1) In this paper, we are dealing with calibration in a sensor network setting. One of the main properties of a sensor network is that it is composed of multiple sensor devices (could go up to massive amounts), and once deployed they should perform their sensing task without further human intervention, as accessing the sensors may not be possible for all application scenarios. As the sensors are expected to stay active for long amounts of times, they are also expected to be calibrated periodically due to external or internal factors. The calibration process, due to the above mentioned reasons, has to be a self-calibration process, and once the calibration is performed the process should cover all the sensors and should be done in an energy efficient way as the sensors in general are low on battery power. Therefore we claim that the first assumption is a reasonable one.

(2) In a sensor network setting, we assume multihop communication between sensors. This is a widely accepted assumption for sensor networks, as for large networks the radio range of nodes does not cover all the nodes, or such communication is costly compared to multihop communication. In a multihop communication network, the calibration is done in an iterative manner [12]. In iterative calibration all sensors are calibrated based on a fixed sensor which might be pre-calibrated or not. Therefore, calibration follows a path, or a tree structure. On a sensor graph, there are multiple ways to realize this path, and each path contributes differently to the post-calibration error of each sensor. The aim of assumption (2) is to minimize the post-calibration error of each sensor introduced as a result of the calibration process.

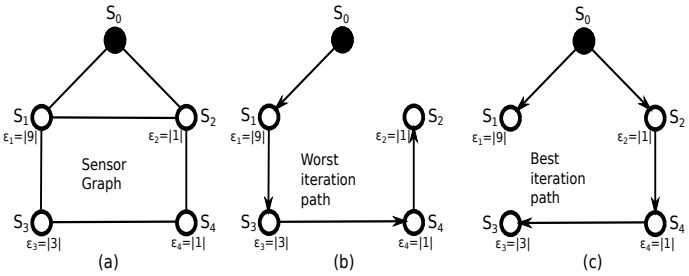


Figure 1: Motivational example showing the sensor graph (a), and two alternative calibration paths (b) and (c).

2.1. Motivational example

To further clarify the subject we would like to first give an example on simple pairwise sensor calibration and then give an example on various iteration paths. Formal definitions of some of the terms used in here will be presented in Section 2.2. First, assume that we have two sensors x_1 and x_2 at hand, with absolute maximum measurement errors of ϵ_1 and ϵ_2, respectively. Also assume that x_1 is already calibrated against a reference sensor, and we would like to calibrate sensor x_2 using the readings from sensor x_1. During calibration, the reading we get from x_1 will be in a range $[x_1-\epsilon_1, x_1+\epsilon_1]$, and the x_2 reading will be again in range $[x_2-\epsilon_2, x_2+\epsilon_2]$. The calibration function for x_2 will map the reading of x_2 to the reading of x_1. Therefore, in the worst case if x_1 reports the reading as $x_1-\epsilon_1$ and x_2 reports the reading as $x_2+\epsilon_2$, or similarly x_1 reports the reading as $x_1+\epsilon_1$ and x_2 reports the reading as $x_2-\epsilon_2$, the post-calibration skew of sensor x_2 will be $|\epsilon_1+\epsilon_2|$, which is the sum of the absolute maximum measurement errors.

Figure 1 presents an example on the effects of iteration paths on the calibration error. In Figure 1(a) one can see the sensor graph with five sensor nodes where the edges represent direct communication links among the sensors. The edges also represent that the vertices attached to the edges are close enough to calibrate each other. S_0 is assumed to be the pre-calibrated sensor, and the remaining sensors will be calibrated based on this sensor. Also for each sensor the absolute maximum measurement error is given as ϵ_i values. For brevity, we can assume that all the edge distances are equal to each other. The problem now is to find an iteration path or ordering, such that the post-calibration skew of each sensor and the total calibration path costs will both be minimized. As the edge distances are equal to each other, we reduce the problem to only minimizing the post-calibration skew in this example.

In Figure 1(b) a possible solution for the iteration path is presented, where the arrows represent pairwise calibration among sensors. As a result of the calibration, the post-calibration skew values of the sensors are the sum of the absolute maximum errors of each node and can be given as: $\xi_1 = |9|$, $\xi_2 = |14|$, $\xi_3 = |12|$, $\xi_4 = |13|$, where the maximum post-calibration skew becomes $|14|$ for the overall network.

Figure 1(c) shows another possible iteration path on the same sensor graph. The post-calibration skew of the sensors on this iteration path then becomes: $\xi_1 = |9|$, $\xi_2 = |1|$, $\xi_3 = |5|$, $\xi_4 = |2|$, where the maximum post-calibration skew this time is $|9|$. It is obvious from the example that the iteration path in Figure 1(c) is a better alternative compared to the one in Figure 1(b), and different paths contribute differently to the post-calibration skew of the sensors. Therefore, even though we cannot control the random error in measurements, we can adjust the path in such a way that the maximum post-calibration skew introduced in calibration is minimized for the whole network.

In the above example we demonstrated to optimize the post-calibration skew for the network, where the spanning tree costs of all the calibration paths were equal to each other. However, the problem that we propose in this paper focuses on minimizing both the post-calibration skew and the spanning tree cost at the same time and therefore is an intractable problem. We give formal definition of the problem and the intractability proof below.

2.2. Formal Definition of the Problem

Pairwise calibration in sensor networks can be modeled as a pre-order traversal of a spanning tree, where the parent sensor is calibrated first (assuming the root is pre-calibrated), and the children are calibrated based on the parent. An *iteration path* for calibration of a sensor j is defined as the path from the root to sensor j on the spanning tree. As calibration is expected to be done periodically, an energy efficient spanning tree that can also minimize the maximum post-calibration error is required. We call the problem as *Minimum-Cost Bounded-Error Calibration Tree* problem. The formal definition of the problem is stated below.

Definition 2.1 (***Calibration function***)**.** *Given a sensor j with a nominal sensor reading x'_j, an absolute maximum random measurement error of $|\epsilon_j|$ and a pre-calibrated reference sensor r, the calibration function $F_j(x_j)$ maps the reading x_j, such that $x'_j \in [x_j - \epsilon_j, x_j + \epsilon_j]$, to the output of sensor r.*

Definition 2.2 (***Post-calibration skew***)**.** *The post-calibration skew (ξ_j) of sensor j for a given measurement is the difference between the calibrated value of sensor j and the actual value x. As such $\xi_j = |x - F_j(x_j)|$. The post-calibration skew of a network with n sensors is then given by $\max_{k=1..n}(\xi_k)$.*

Theorem 2.3. *Let r be a reference sensor, and P be an iteration path from r to sensor $j \in [1 \dots n]$. The post-calibration skew of sensor j is then $\xi_j \leq \sum_{k \in P} \epsilon_k$.*

Proof. The proof is easy to show by induction on the number of sensors on an iteration path. Assume a path with $n+1$ sensors, where sensors are numbered from 0 to n based on their distance from reference sensor 0. For the base case when $k = 1$ assume that sensor 0 is the reference sensor with no random error, sensor 1 is calibrated against the reported value of sensor 0 using calibration function F_1

$$F_1(x_1) - \epsilon_1 \leq x \leq F_1(x_1) + \epsilon_1,$$

where x is the ground truth value. The post-calibration skew for sensor 1 is then,

$$\xi_1 = |x - F_1(x_1)| \leq \epsilon_1. \tag{1}$$

It is easy to see from Eq. 1 that the base case holds. Let us assume by the inductive hypothesis that the theorem holds for $1 \leq k \leq n$ so that,

$$\xi_k = |x - F_k(x_k)| \leq \sum_{j=1}^{k} \epsilon_j. \tag{2}$$

We can show that the theorem holds for $k+1$ by writing the calibration function F_{k+1} for sensor $k+1$ against the reported value of sensor k as:

$$F_{k+1}(x_{k+1}) - \epsilon_{k+1} \leq F_k(x_k) \leq F_{k+1}(x_{k+1}) + \epsilon_{k+1}. \tag{3}$$

If we rewrite Eq. 2 and add $-x$ to the inequality, we get,

$$-x - \sum_{j=1}^{k} \epsilon_j \leq -F_k(x_k) \leq -x + \sum_{j=1}^{k} \epsilon_j. \tag{4}$$

If we sum Eq. 3 and 4 and add $x - F_{k+1}(x_{k+1})$ to the inequality, we get,

$$-\sum_{j=1}^{k+1} \epsilon_j \leq x - F_{k+1}(x_{k+1}) \leq \sum_{j=1}^{k+1} \epsilon_j, \tag{5}$$

so that the post-calibration skew of $k+1$ is then,

$$\xi_{k+1} = |x - F_{k+1}(x_{k+1})| \leq \sum_{j=1}^{k+1} \epsilon_j,$$

which completes the proof. □

As a direct outcome of Theorem 2.3 we can say that the post-calibration skew is dependent on the iteration path P of calibration, as the random errors of the sensors on the iteration path P effect the post-calibration skew. Therefore different paths create different post-calibration skew values.

Definition 2.4 (***Calibration Cost***)**.** *The pairwise calibration cost is the wireless communication cost of sensor j with sensor i at distance $d_{i,j}$. The total calibration cost is the sum of all pairwise calibration costs over the entire network.*

The decision version of the *Minimum-Cost Bounded-Error Calibration Tree* problem can be stated as:

Definition 2.5 (***MBCT***)**.** *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 ϵ_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.*

We can show that $MBCT$ is NP-complete by a reduction from the *Exact 3-Cover* problem which is shown to be NP-complete in [16]. In this proof, we follow a similar reduction as in [17].

Definition 2.6 (***Exact 3-Cover***)**.** *Given a set $Y = \{y_1, ..., y_q\}$ of 3-element subsets of a set $X = \{x_1, ..., x_{3p}\}$, and $q \geq p$, does there exist a subset $Y' \subset Y$ of pairwise disjoint sets such that $\bigcup_{y \in Y'} y = X$?*

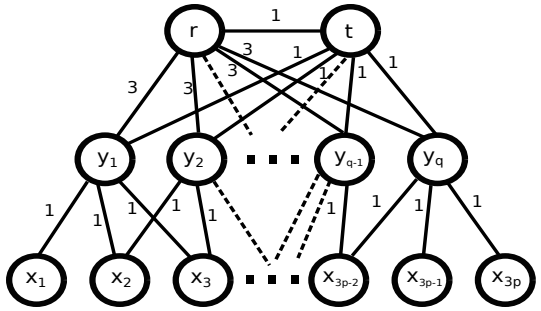


Figure 2: An instance of $MBCT$ obtained from an instance of *Exact 3-Cover*. Link costs are given, and ϵ_v values are equal to ϵ for each sensor.

Lemma 2.7. *$MBCT$ is in NP.*

Proof. Given a solution $T(V, E')$, $E' \subseteq E$, to the $MBCT$ problem, it is easy to verify in polynomial number of steps that $T(V, E')$ is a spanning tree, $\sum_{e \in E'} d_e \leq C$, and $\forall\ v \in V : \xi_v \leq k$. □

Given an arbitrary instance of *Exact 3-Cover*, an instance $G(V, E)$ of $MBCT$ with r as reference node can be constructed in polynomial time as shown in Figure 2 by following the steps below.

$$
\begin{aligned}
V &= r \cup t \cup X \cup Y \\
E &= (r,t) \cup \{(r, y_i) : y_i \in Y\} \cup \\
&\quad \{(t, y_i) : y_i \in Y\} \cup \\
&\quad \{(y_i, x_j) : x_j \in y_i, \text{ and } y_i \in Y, x_j \in X\} \\
d_e &= \begin{cases} 3 & e \in \{(r, y_i) : y_i \in Y\} \\ 1 & \text{otherwise} \end{cases} \\
\epsilon_v &= \epsilon \quad \forall v \in V \\
k &= 2\epsilon \\
C &= 5p + q + 1
\end{aligned}
$$

We can now prove the sufficiency and necessity parts as below.

Lemma 2.8. *If Exact 3-Cover has a solution then $MBCT$ has a solution.*

Proof. For a feasible solution $Y' \subset Y$ of *Exact 3-Cover*, there are exactly p $y_i \in Y'$ terms each containing 3 disjoint x_j terms, such that a solution rooted at r in the corresponding instance of $MBCT$ has x_j nodes connected to exactly p non-leaf y_i nodes, each also directly connected to r, while the remaining leaf $y_i \in Y - Y'$ connect to r through t. Therefore, the total post-calibration skew of each node is bounded by 2ϵ, and the total edge cost of the spanning tree is $3p + |Y'| * 3 + |Y - Y'| * 1 + 1 = 5p + q + 1 = C$. □

Lemma 2.9. *If $MBCT$ has a solution then Exact 3-Cover has a solution.*

Proof. For a feasible solution $T(V, E')$ of $MBCT$ rooted at r, the $\{x_j, j = 1, \ldots, 3p\}$ appear as leaf nodes, and each x_j connects to only one $\{y_i, i = 1, \ldots, q\}$, where these non-leaf y_i nodes are also connected to node r. The remaining leaf y_i nodes are connected to r through t. It is clear that the maximum post-calibration skew on $T(V, E')$ is equal to 2ϵ. The edge cost of $T(V, E')$ is not greater than C if and only if the number of non-leaf y_i nodes are equal to p, such that the sum of edge costs for all x_js, plus the non-leaf y_is, plus the leaf y_is and node t are equal to:

$$C = 3p * 1 + p * 3 + (q - p) * 1 + 1 = 5p + q + 1.$$

Therefore, for the instance of *Exact 3-Cover*, the non-leaf y_i nodes form a feasible solution of 3 element disjoint subsets that cover set X. □

Theorem 2.10. *The $MBCT$ problem is NP-complete.*

Proof. Directly follows from Lemmas 2.7, 2.8, and 2.9. □

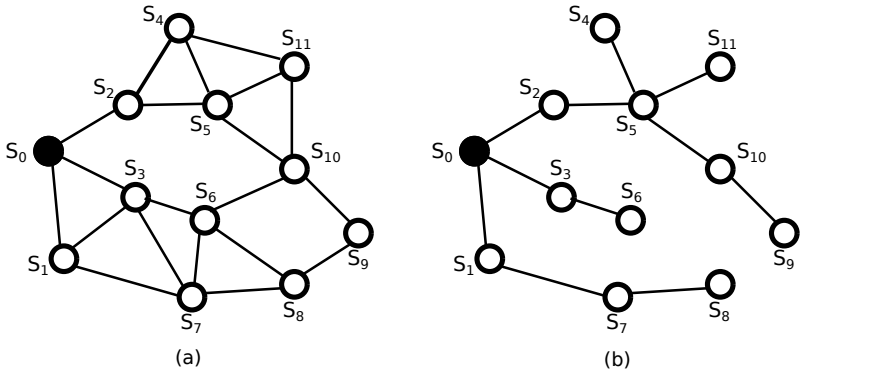


Figure 3: Example graph (a), spanning tree of the graph (b), and the chromosome encoding showing the edges (c).

3. Heuristic Solution

In this section, we present the genetic algorithm [18] applied to solve the optimization version of the *MBCT* problem. In the optimization version, the objective is to minimize both the post-calibration skew and the total calibration costs. The genetic algorithm is designed in five main stages, chromosome encoding, creating an initial population, crossover, mutation, and fix chromosome routine. Below we describe the details of each stage.

3.1. Chromosome encoding

In our genetic algorithm each chromosome is encoded as a list of edges of a valid spanning tree for the sensor graph. As spanning tree has $V-1$ edges for V sensors, the size of each chromosome is fixed and equal to $V-1$. Figure 3 shows an example graph, its spanning tree and the chromosome encoding. As seen in Figure 3 (c), each chromosome is encoded in terms of the edges of the spanning tree. Each $\{S_i, S_j\}$ pair in the chromosome represents an edge between nodes S_i and S_j in the graph, and forms a single gene in the chromosome.

3.2. Creating an initial population

The initial population is created randomly. From the input graph, various random spanning trees are created, and each spanning tree is encoded as a chromosome and included in the initial population. The size of the initial population is controlled as a parameter, which we cover in the experiments section.

3.3. Crossover

Crossover is the stage where new child chromosomes are created from the fittest parents in the population. In the genetic algorithm, we use roulette wheel selection [19] to pick the two chromosomes to crossover. Roulette wheel selection algorithm selects the chromosomes randomly based on their fitness values, where the fittest chromosomes have a higher chance to be selected for crossover. During the crossover stage a boolean vector of size $V-1$ is created, and this vector is filled randomly such that, for each gene in the chromosomes, there is an associated boolean value in the vector. For all the true boolean values the gene of the first parent is selected, and for all the false boolean values in the vector the gene of the second parent is selected. Even though this algorithm performs a random crossover among the two parent chromosomes, it does not ensure a valid spanning tree encoding for the newly created child chromosome. A process called Fix Chromosome is applied, as described below in Section 3.5, to convert the chromosome encoding to a valid spanning tree.

3.4. Mutation

Mutation is used in genetic algorithms to avoid local optimal results. Mutation is applied with a low mutation probability. In the mutation phase, a randomly selected edge is removed from the chromosome. Removing an edge from the chromosome disconnects the spanning tree, therefore the chromosome has to be reorganized to represent a valid spanning tree. The reorganization of the chromosome is done in the Fix Chromosome section below.

3.5. Fix chromosome

The objective of the Fix Chromosome function is to ensure the chromosome represents a valid spanning tree, therefore the chromosome is altered until the objective is met. In order to do so in the Fix Chromosome function we use the existing edges in the chromosome, remove the edges that create cycles, and add random edges to the spanning tree until the spanning tree has $V-1$ edges. This process is designed similar to Kruskal minimum spanning tree algorithm, but without ordering the edges based on their weights.

We designed the genetic algorithm to be generic enough so that it allows the injection of different fitness functions. The results of using these fitness functions are evaluated in the experiments section below.

```
GeneticAlgorithm(Graph G)
 1: /* Fill the population with random chromosomes */
 2: for L=1 to PopulationSize do
 3:    C = CreateNewRandomChromosome(G)
 4:    Population.add(C)
 5: BestFitnessScore ← ∞
 6: /* Run for NumberOfIterations */
 7: for I=1 to NumberOfIterations do
 8:    {C1,C2} = RouletteWheelSelection(Population)
 9:    CWorst = SelectWorstChromosome(Population)
10:    CNew = Crossover(C1,C2)
11:    if RandomNumber < MutationProbability then
12:       CNew = Mutation(CNew)
13:    /* FixChromosome makes sure CNew is a spanning tree */
14:    CNew = FixChromosome(CNew,G)
15:    BestFitnessScore = min(BestFitnessScore,
                              GetFitnessScore(CNew))
16:    Population.add(CNew)
17:    Population.remove(CWorst)
18: return BestFitnessScore
```

Figure 4: The pseudo-code of the genetic algorithm.

The pseudo-code of the genetic algorithm is presented in Figure 4. The input for the genetic algorithm is the sensor graph, and other optional parameters such as initial population size, number of iterations and mutation probability. Lines 2 to 4 show the code for new chromosome creation and random initialization of the population. Line 5 initializes the best fitness score, which is the outcome of the algorithm and returned in line 18. Lines 7 to 17 show the code for the main iteration of the genetic algorithm. The candidate parent chromosomes are selected using roulette wheel selection in line 8, and the worst chromosome based on the fitness value is selected in line 9. The crossover is performed in line 10, and with a low probability the mutation is performed in lines 11 and 12. The FixChromosome function, as described above in Section 3.5 is called in line 14. If the new chromosome has a lower fitness value than the best chromosome found so far, the best fitness score value is updated in line 15. The new chromosome is added to the population in line 16, and the worst chromosome is removed from the population in line 17, keeping the size of the population constant throughout the iterations. Finally in line 18 the best fitness score found in the genetic algorithm is reported back.

4. Experiments

In this section we present the results of our experimental evaluation for the optimization version of the $MBCT$ problem. In the optimization version of the $MBCT$ problem, the objective is to find a spanning tree with respect to a given reference node over a given graph, such that the total edge cost of the spanning tree and the post-calibration skew of each sensor will be minimized at the same time. We first describe the parameters used in the genetic algorithm and the parameters of the graphs used in the experimentation in Section 4.1. Later, in Section 4.2 we present the results of applying genetic algorithm to the $MBCT$ problem for various parameters and fitness functions.

4.1. Experimental Setup

We conducted our experiments on various sized randomly generated graphs. The details of the graphs are presented in Figure 5.

Graph name	**#of nodes**	**Avg. node degree**	**Std. node degree**
n25	25	5	2
n50	50	5	2
n100	100	5	2
n250	250	5	2
n500	500	5	2
n750	750	5	2
n1000	1000	5	2

Figure 5: Graph names, number of nodes, average and standard deviation of the node degrees of the graphs used in the experimentation.

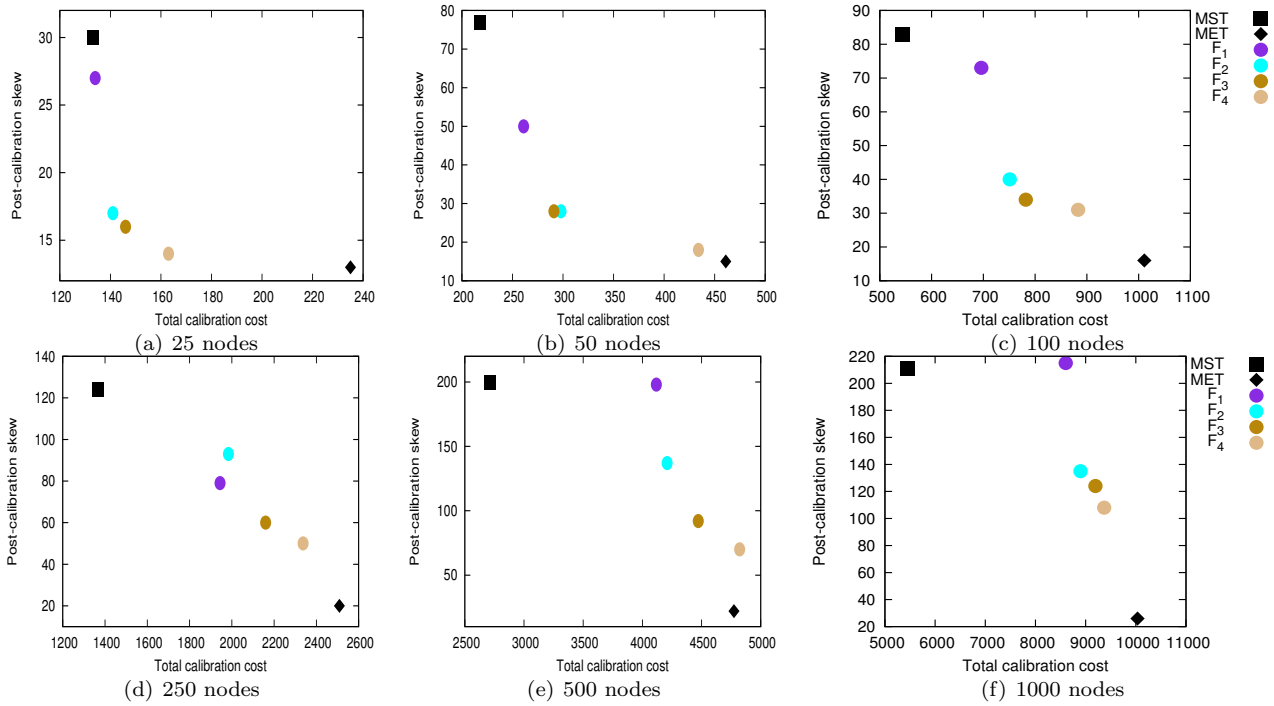
On each graph, we run polynomial time algorithms to find the minimum total calibration cost and the minimum post-calibration skew values. In order to calculate the minimum total calibration cost (MIN_COST) we run Kruskal minimum spanning tree algorithm. MIN_COST gives us a lowerbound for the total calibration cost of the graph. Similarly, we calculate the minimum post-calibration skew (MIN_ERROR) of the graph using a modified version of the Dijkstra's shortest path algorithm. MIN_ERROR gives us a lowerbound for the post-calibration skew of the graph. Similarly, MAX_COST and MAX_ERROR are the upperbounds for the total calibration cost and the post-calibration skew values for any given graph. These values are used to calculate the fitness functions used in the genetic algorithm, as shown in Figure 6.

Parameter name	**Parameter calculation**
MIN_COST	Minimum total calibration cost
MAX_COST	Upperbound for the total calibration cost of a spanning tree
MIN_ERROR	Minimum post-calibration skew
MAX_ERROR	Upperbound for the post-calibration skew
$norm\_cost$	$\frac{(cost-MIN\_COST)*100}{(MAX\_COST-MIN\_COST)}$
$norm\_error$	$\frac{(error-MIN\_ERROR)*100}{(MAX\_ERROR-MIN\_ERROR)}$

Function name	**Functions**
$F1$	$0.9 * norm\_cost + 0.1 * norm\_error$
$F2$	$0.5 * norm\_cost + 0.5 * norm\_error$
$F3$	$\frac{(cost-MIN\_COST)^2}{MIN\_COST} + \frac{(error-MIN\_ERROR)^2}{MIN\_ERROR}$
$F4$	$0.1 * norm\_cost + 0.9 * norm\_error$

Figure 6: Fitness functions used in the experimentation.

In order to evaluate the effects of the fitness functions, we tested our genetic algorithm with four different fitness functions. The details of the fitness functions are presented in Figure 6.



(a) 25 nodes (b) 50 nodes (c) 100 nodes

(d) 250 nodes (e) 500 nodes (f) 1000 nodes

Figure 7: Experiments showing the post-calibration skew and the total calibration cost of the four different fitness functions on various graphs.

4.2. Simulation results

In this section we present the simulation results of the various experiments we conducted. First, we tested four of the fitness functions on all the graphs presented in Figure 5. The results of the experiments can be seen in Figure 7. In all the figures, the minimum total calibration cost (MIN_COST) and its corresponding post-calibration skew value, and the minimum post-calibration skew value (MIN_ERROR) and its corresponding total calibration values are presented as MST and MET, respectively. The number of iterations, population size and the mutation probability are selected as 50000, 400, and 0.1 respectively. These parameters for the initial experiments are selected as average values based on our previous experimentation with the problem. The analytical selection process of the parameters are described in more detail below. All the experiments are conducted 10 times and the minimum fitness results are reported.

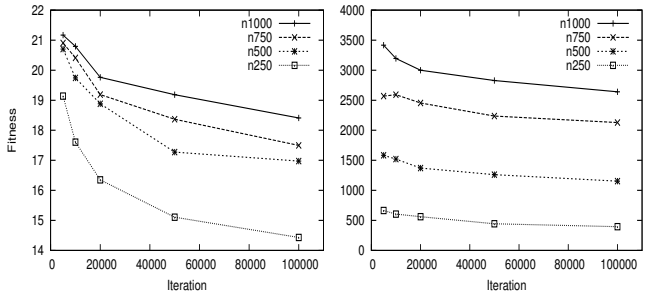
As seen in Figure 7, function $F1$ minimizes the calibration cost while sacrificing from the post-calibration skew value. In smaller graphs however, as in $n25$ (Fig.7(a)), the calibration cost of $F1$ is close to MST, while the post-calibration skew is lower than MST, which clearly outperforms the MST. Function $F4$ on the other hand tries to minimize the post-calibration skew in trade with higher calibration cost values. However, similar to $F1$, in smaller graphs $F4$ also can achieve post-calibration skew values close to MIN_ERROR with much smaller calibration cost values, which is a valid alternative to the MET. Functions $F2$ and $F3$ achieves a balance between the calibration cost and the post-calibration skew values. Both functions try to minimize both the calibration cost and the post-calibration skew values at the same time, therefore, these two functions are the main functions we are particularly be interested in this paper. As we can see again from Figure 7, for smaller graphs, $F2$ and $F3$ minimizes both the calibration cost and the skew values and the results generated are on the lower side of a line drawn between points MST and MET. However, for larger graphs, due to the intractable nature of the problem, minimizing both of the objectives at the same time is harder to do. This result alone hints that there is still room for research for minimizing both objectives at the same time, especially on larger graphs.

Parameter name	**Description**	**Tested range**
Fitness function	Fitness function used in the genetic algorithm (See Fig. 6)	$F2$, $F3$
Graph name	Name of the graph the test is conducted on (See Fig. 5)	$n250$, $n500$, $n750$, $n1000$
Iteration	The number of iterations for the genetic algorithm	5000 - 100000
Population Size	The size of the population for the genetic algorithm	200 - 5000
Mutation Probability	The mutation probability of the genetic algorithm	0.05 - 0.3

Figure 8: Genetic algorithm parameters used in experimentation

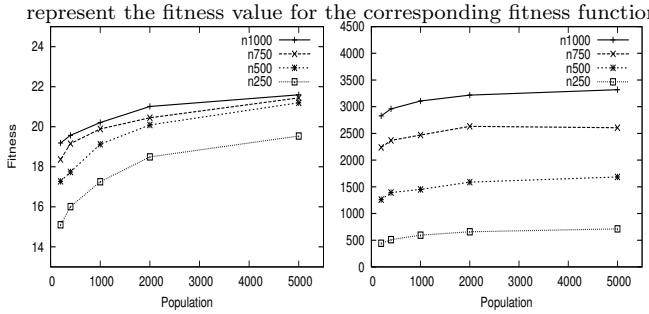
Based on the observation we make above, we conducted more tests to see the effects of the simulation parameters on the success of the genetic algorithms, specifically for fitness functions $F2$ and $F3$. Figure 8 lists the various parameters and their short description that we use in our simulations.

In Figure 9 we observe how the number of iterations in



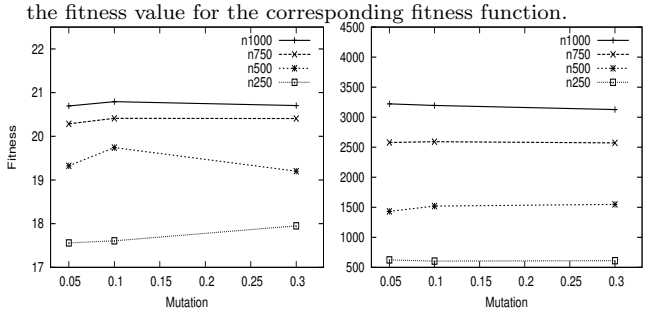
(a) Function $F2$ (b) Function $F3$

Figure 9: Change of the fitness value with number of iterations for functions F2 (a), and F3 (b). The y-dimension for both graphs represent the fitness value for the corresponding fitness function.



(a) Function $F2$ (b) Function $F3$

Figure 10: Change of the fitness value with population size for functions F2 (a), and F3 (b). The y-dimension for both graphs represent the fitness value for the corresponding fitness function.



(a) Function $F2$ (b) Function $F3$

Figure 11: Change of the fitness value with mutation probability for functions F2 (a), and F3 (b). The y-dimension for both graphs represent the fitness value for the corresponding fitness function.

the simulation change the fitness value for fitness functions $F2$ (in Figure 9(a)) and $F3$ (in Figure 9(b)). The population size and the mutation probability for this experiment is selected as 200 and 0.1, respectively. As we see from the figures, for all graphs, the fitness value decreases and we achieve better results as the number of iterations increases.

Similarly, in Figure 10, we observe the effects of the population size on the success of the genetic algorithm, for functions $F2$ (Fig. 10(a)) and $F3$ (Fig. 10(b)), on various graphs. The iteration number and the mutation probability for this experiment is selected as 50000 and 0.1, respectively. Fitness function $F2$ achieves better results with lower population sizes, although the population size effects the performance slightly when fitness function $F3$ is used.

Finally, in Figure 11, we changed the mutation probability in the genetic algorithm and observed the effects on the fitness value for fitness functions $F2$ (Fig. 11(a)) and $F3$ (Fig. 11(b)). The population size and the number of iterations for this experiment is selected as 200 and 10000, respectively. Even though for fitness function $F2$ the change in the mutation probability alters the fitness value slightly, overall based on our observation from the simulations we can conclude that the mutation probability does not have a significant effect on the solution of the $MBCT$ problem.

To summarize, in this section we presented the experimental results of the genetic algorithm to solve the optimization version of the $MBCT$ problem. We performed our tests for various graphs and genetic algorithm specific parameters. We observed that on relatively small graphs the algorithms clearly finds close to optimal results, however, for larger graphs due to the NP-hard nature of the problem, achieving close to optimal results are harder.

5. Conclusion

In this paper, we present an energy efficient and minimum error iterative calibration based model for wireless sensor networks and prove that in real-world deployments where random errors are inevitable, the problem turns into an NP-complete problem, independent from the magnitude of the random error. To the best of our knowledge this is the first time the complexity of an iterative calibration based model is analyzed for real-world sensor deployments. We also developed a genetic algorithm based heuristic solution to solve the optimization version of the $MBCT$ problem, and evaluated the algorithm on various graphs.

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, Applied Soft Computing 12 (2012) 1532 – 1541.

[2] J. A. Koupaei, M. Abdechiri, Sensor deployment for fault diagnosis using a new discrete optimization algorithm, Applied Soft Computing (2012) doi:10.1016/j.asoc.2012.04.026.

[3] 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 (2009) 25:1–25:29.

[4] 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.

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

[6] 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, pp. 197–210.

[7] 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, pp. 51–63.

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

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

[10] M. Takruri, S. Challa, R. Chakravorty, Recursive bayesian approaches for auto calibration in drift aware wireless sensor networks, Journal of Networks 5 (2010) 823–832.
[11] 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, pp. 27–33.
[12] K. Whitehouse, D. Culler, Calibration as parameter estimation in sensor networks, in: WSNA, Atlanta, Georgia, USA, pp. 59–67.
[13] E. Miluzzo, N. D. Lane, A. T. Campbell, R. Olfati-Saber, Calibree: A self-calibration system for mobile sensor networks, in: DCOSS, Santorini Island, Greece, pp. 314–331.
[14] C. Wang, P. Ramanathan, K. Saluja, Blindly calibrating mobile sensors using piecewise linear functions, in: SECON, pp. 1–9.
[15] L. Girod, M. Lukac, V. Trifa, D. Estrin, The design and implementation of a self-calibrating distributed acoustic sensing platform, in: SenSys, Boulder, Colorado, USA, pp. 71–84.
[16] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.
[17] H. Salama, D. Reeves, Y. Viniotis, The delay-constrained minimum spanning tree problem, in: Proc. of the Second IEEE Symposium on Computers and Communications,, pp. 699 –703.
[18] J. H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, 1975.
[19] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.