BUILDING ENERGY-EFFICIENT BROADCAST TREES USING A GENETIC ALGORITHM IN WIRELESS SENSOR NETWORKS

by

MIN KYUNG AN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2007

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervising professor, Dr. Ramez Elmasri, for his guidance and giving me an opportunity to work on this thesis. I appreciate my committee members, Dr. Leonidas Fegaras and Dr. Yonghe Liu, for their time and valuable suggestions. I would also like to thank my friend, K. Park, for constantly motivating me and for his advice during the course of master's studies. Special thanks should be given to my parents and my love, H. Kim, for their powerful support and encouragement throughout my academic career.

This work was supported by the Korean Science and Engineering Foundation Grant funded by the Korea government (MOST) (No.2005-215-C00025).

July 13, 2007

ABSTRACT

BUILDING ENERGY-EFFICIENT BROADCAST TREES USING A GENETIC ALGORITHM IN WIRELESS SENSOR NETWORKS

Publication No. ______

Min Kyung An, MS

The University of Texas at Arlington, 2007

Supervising Professor: Dr. Ramez Elamsri

In wireless broadcast environment, data can be transmitted to several nodes by a single transmission. The nodes in that environment have limited energy resources; therefore we need to reduce the energy consumption when they broadcast data to all nodes to prolong the lifetime of the networks. We call this problem the MPB (minimum power broadcast) problem, and solving the problem has been shown to be an NP-Complete problem [2], [11]. We focus on finding 'near-optimal' solutions for the problem.

An algorithm for constructing the minimum power broadcast trees, named BIP (broadcast incremental power) algorithm, was first proposed by Wieselthier et al. in [1], and several other algorithms also have been proposed by researchers so far. They use the broadcast nature of the problem to optimize energy consumption.

We propose an alternate search based paradigm wherein the minimum broadcast tree is found using a genetic algorithm, which is used to find an approximate solution to avoid the NP-Completeness problems. We start by using the BIP algorithm and the MST (Minimum Spanning Tree) algorithm to create an initial search space in our genetic algorithm and we evolve the initial broadcast trees in the space to get more energy-efficient broadcast tree.

Through the simulations, the genetic algorithm achieved up to 20.60% improvement over the other broadcasting algorithms including the traditional BIP algorithm.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Recent advances in wireless sensor networks have led to the development of sensor nodes. Even though these sensor nodes are very tiny and often have short distances, they have the capability to communicate with the other sensor nodes which are in their radio range and forward the data to the other sensor nodes, until they reach their destinations. Although recent sensor nodes have a remarkable improvement over the traditional sensor nodes, they still have limited energy resources. Therefore, the energy efficiency of routing and data collection becomes critical. On this score, it is definitely important to reduce the communication range and amount of data as much as we can so that the lifetime of nodes can be prolonged.

One common paradigm for communication among the sensor nodes having this kind of drawback is broadcasting, and energy-efficiency is a crucial aspect in constructing broadcast trees. Here, the MPB (minimum power broadcast) problem is addressed, and unfortunately finding an optimal solution to the MPB problem is shown to be an NP-Completeness problem in [2] and [11]. Therefore, many studies are devoted to find near optimal solutions of the MPB problem. Especially, reducing the number of redundant transmissions, and reducing the energy consumed by detecting and receiving have been studied to solve the MPB problem.

In this thesis, we use a genetic algorithm to find approximate solutions to optimization and search problems. We find several feasible initial solutions for the problem, and evolve the solutions to get the ‘near-optimal’ solution which yields less power consumption than other known solutions.

1.2 Goals of the Thesis

The contributions of this thesis are:

- Building a broadcast tree structure over wireless sensor networks which are maintained by the base station to reduce the total energy consumption
- Proposal of techniques for using a genetic algorithm to optimize the initial trees and find better minimum power broadcast tree
- Comparison with the other algorithms using other kinds of evolutionary algorithms

1.3 Outline of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we overview wireless sensor networks and broadcasting in the sensor networks. We also give an overview of a genetic algorithm, which is a search technique and one of the evolutionary algorithms to find approximate solutions to optimization. In Chapter 3, we introduce the procedure of our genetic algorithm used to build broadcast trees in the wireless sensor networks. In Chapter 4, we implement and evaluate the genetic algorithm, comparing it with the traditional BIP algorithm. In Chapter 5, we compare the evaluation result of the genetic algorithm with the other related techniques which are also used to build broadcast trees. Finally, Chapter 6 contains the conclusions.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we discuss the background for our genetic algorithm to build broadcast trees, and introduce other algorithms which have been studied and proposed by other researchers so far.

First, we describe what the wireless sensor network is, and we introduce how to broadcast data from the source node (sink node) to all the other nodes in the network.

An important fact is that the building of broadcast trees which broadcast data to all the nodes in the network in minimum energy consumption has been shown to be NP-Complete problem in [2] and [11]. The problem of constructing the tree structures is called the MPB (minimum power broadcast) problem. Therefore, we are not able to find the best solution for this problem using an efficient algorithm. We are only able to find 'near-optimal' solutions using several methods, which have been proposed to avoid the NP-Complete problems like the MPB problem. One of them is to use a genetic algorithm, and we briefly introduce the algorithm which we have used and modified to solve the MPB problem, later in this chapter.

2.1 Broadcasting in Wireless Sensor Networks

2.1.1. Wireless Sensor Networks (*WSN*)

Recent advances in wireless communications have led to the development of sensor nodes. Wireless sensor networks consist of a large number of these static or mobile sensor nodes. Even though these sensor nodes are very tiny and have short communication distance to their vicinity, they have the capability to communicate with other sensor nodes which are in their radio ranges and forward the data to the other sensor nodes until it reaches their destinations. Newly developed sensor nodes have remarkable improvement over the traditional sensor nodes, and are used in wireless sensor networks with their properties for sensing, data processing, and communicating.

One of the features of wireless sensor networks is that the position of the sensor nodes doesn’t need to be predetermined. That is why they are deployed in random positions. This feature means that the algorithms and protocols related to the sensor networks must have self-organizing capabilities. Another feature of wireless sensor networks is that the sensor nodes cooperate with each other. Each sensor node partially processes the data which the node needs to be responsible for, and forwards the processed partial data to the other nodes responsible for aggregation and further transmission.

The sensor networks with these features are applied to a area of applications. Examples of the application areas are health, military, home monitoring and so on. In health and home monitoring, sensor nodes can be used to monitor disabled patients, and in military applications, they can be used for command, control, communication, computing, intelligence, surveillance, reconnaissance, and targeting system. In other application areas, we can use them to manage inventory, to monitor product quality, disaster rescue, or monitor the environment.

In general, each sensor node in a sensor network has limited energy resources; therefore, the energy efficiency becomes important. On this score, it is definitely important to reduce the communication range and amount of data as much as we can so that the life of nodes can be prolonged. Consequently, we introduce how to build broadcast trees with minimum energy consumption in this thesis.

2.1.2. Minimum Power Broadcast (*MPB*) *Problem in WSN*

Broadcasting is a method to allow all nodes to share the data efficiently with all the other nodes in the wireless sensor networks. Due to the limited energy resources, as we mentioned in the previous section, energy-efficiency is a crucial aspect in constructing the broadcast tree structures. We call this problem the MPB (minimum power broadcast) problem. Given an identified sink node and node constellation, the MPB is to minimize the total power consumption when the nodes in the networks are connected together, and they communicate with the other remaining nodes.

So far many studies are devoted to solve the MPB problem. Especially, reducing the number of redundant transmissions, and reducing the energy consumed by detecting and receiving have been studied to solve the MPB problem. Wieseltheir *et al* proposed the broadcast incremental power (BIP) algorithm by utilizing the so-called WMA (*wireless broadcast advantage* or *wireless multicast advantage*) [1], [7], [8] to solve the problem. This assumes that the nodes are equipped with omnidirectional antennas, and if a transmission is directed from node *i* to node *j*, then the signal will be received by all nodes which are within the transmission range of a communication from node *i* to node *j*. The BIP algorithm is derived from Prim's MST (minimum spanning tree) [26] algorithm and it adds one node to the broadcast tree at a time starting from the sink node. A node is added only if it has the minimal incremental cost to be connected to node of the other nodes that are already in the tree. Even though WMA (wireless multicast advantage) [1], [7], [8] provides energy savings, optimal solution of the MPB problem is shown to be NP-Complete in [2] and [11]. A number of approximate algorithms, therefore, have been proposed. Stojmenovic *et al*. [12] proposed internal nodes based broadcasting, Mark *et al*. [4] proposed evolutionary approach to building broadcast tree using the viability lemma, and Cartigny *et al*. [13] proposed a localized algorithm. The Ant Colony System approach (ACS) algorithm which is compared with my algorithm over BIP [1], [8] was also introduced by Mark *et al* [5], and many heuristic approximate methods are described in Das *et al*. [15], [16].

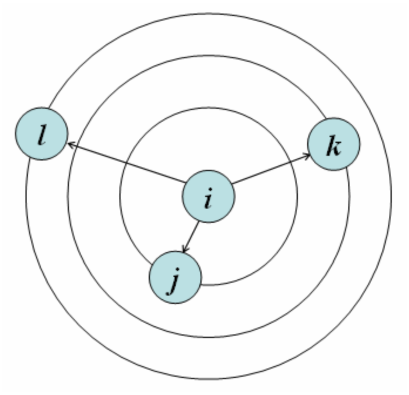
2.1.3. Construction of Minimum-Energy Broadcast Tree

In this section, we review the well-known traditional broadcasting algorithm, BIP (broadcasting incremental power), which is proposed by Wieseltheir *et al* [1], [8]. It has been compared with newer algorithms for the MPB problem by many researchers. In this thesis, we also compare our algorithm for building broadcast tree with this traditional famous algorithm.

2.1.3.1 Introduction of Broadcast Incremental Power (BIP) Algorithm

The BIP is the algorithm for the formation of energy-efficient broadcast trees and the broadcast properties in wireless networks are incorporated in this BIP. The base of BIP is the “node-based” nature of wireless networks, because “link-based” models, which were previously proposed, do not reflect the properties of the all-wireless network environment. The tree built by the BIP is rooted at the source node (sink) and the final object of the BIP is to let the source node reach all of the desired destinations.

During the BIP operation, we assume that the node locations are fixed, and the channel conditions don't change in the wireless network. And the BIP reflects the broadcast nature; when omnidirectional antennas are used, every transmission by a node can be received by all nodes which are within its communication radio range. Figure 2.1 shows this broadcast nature. The transmission power required at node *i* to reach its neighbors, *j*, *k*, and *l*, $P_{i(j,k,l)}$, is the maximum power of P_{ij}, P_{ik}, P_{il}. In contrast with the "link-based" network, so-called wired network in which the $P_{i(j,k,l)}$ should be the sum of the costs to the individual nodes, some energy can be saved in this broadcast nature.



$$P_{i(j,k,l)} = max\ \{ P_{ij}, P_{ik}, P_{il} \}$$

Figure 2.1 Broadcast Nature

2.1.3.2. The Broadcast Incremental Power (BIP) Algorithm

In this section, we briefly introduce the BIP operation with a simple example. We construct a minimum-power broadcast tree, rooted at the source. We can not guarantee that it is the best tree for broadcasting; however, we say that it is the near optimal solution of the MPB problem.

Given sensor nodes in a network as shown in Figure 2.2, node 1 is the source node, and we start the operation with the node. We start determining the node which the source node can reach with the minimum expenditure of power. In Figure 2.2(a), node 2 is added to the tree because it is the nearest one from the source node. Here, we realize that the source node 1 has minimum expenditure of power to reach node 2.

In step 2 with Figure 2.2(b), we determine which “new” node needs to be added to the tree at minimum *additional* cost. There are two nodes, node 1 and node 2, which we can be chosen as the node which increases its broadcast power. If the node 1 is selected, then the node 1 should choose node 3, because it is the next nearest node which node 1 can reach. When node 3 is added as the “new” node to the tree, the transmission power of the node 1 is c3, not (c1 + c3) with the WMA. Here is the *incremental* cost associated with increasing power of node 1 from a level sufficient to reach node 2 to a level sufficient to reach node 3, and we get the incremental cost of node 1, (c3 – c1). If node 2 is selected as the node to increase its power, it should choose node 5, which it can reach with minimum expenditure of power. In this case, the incremental cost of node 2 is equal to the power c2, because node 2 has not transmitted yet. Now we compare those two incremental costs of the two nodes, and we choose the node whose incremental cost is smaller than another one. Node 1 has smaller incremental cost than node 2, and it adds node 3 to the tree.

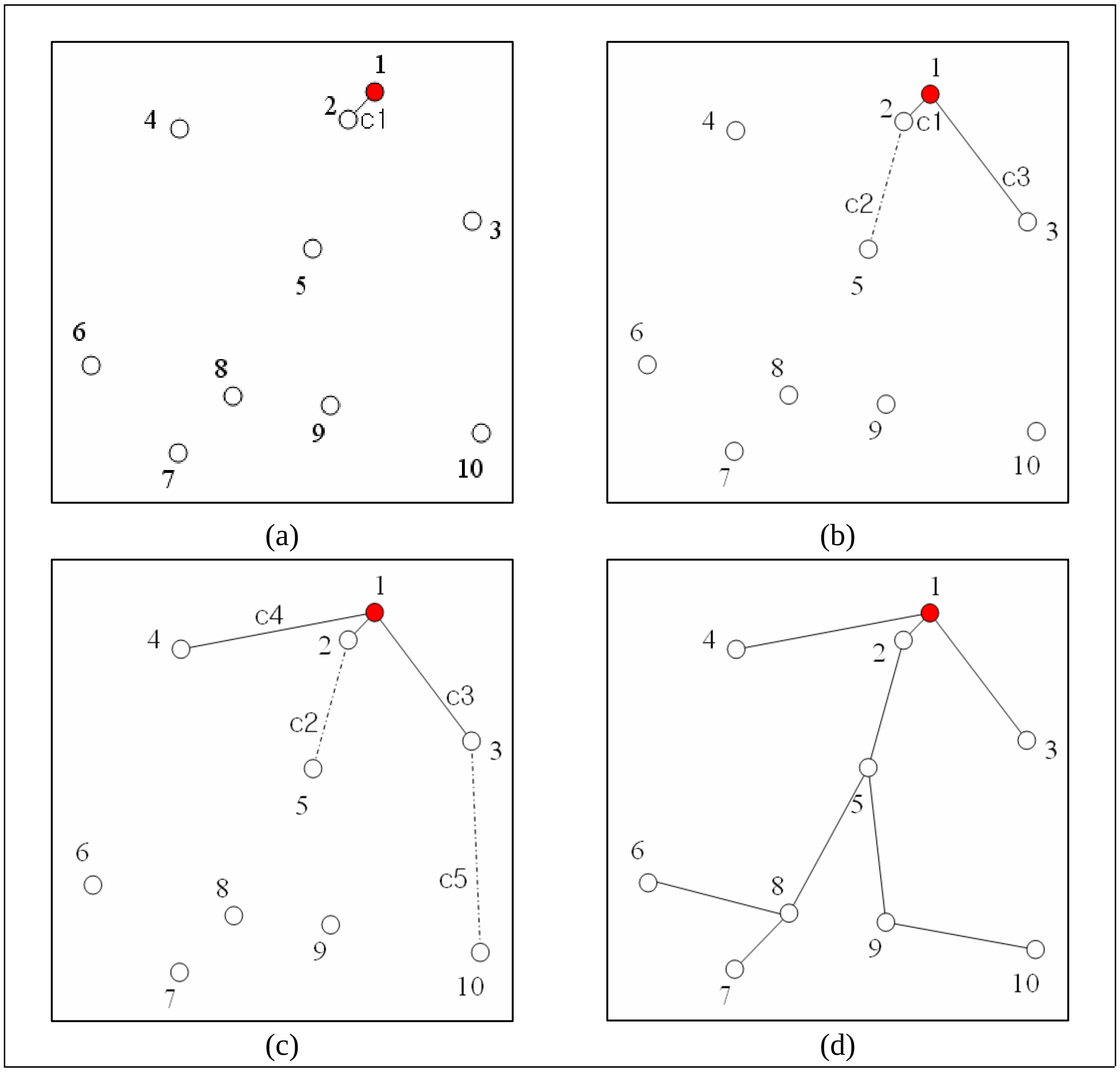


Figure 2.2 An Example of BIP Operation

(a) Step 1: Starting from the source node
(b) Step 2: Incremental Cost of (Node 1) = c3 – c1, (Node 2) = c2
(c) Step 3: Incremental Cost of (Node 1) = c4 – c3, (Node 2) = c2, (Node 3) = c5
(d) Step 4: Final broadcast tree

In step 3 with Figure 2.2(c), there are three nodes which we consider to operate. The incremental cost of node 1 is (c4 – c3), incremental cost of node 2 is c2, and incremental cost of node 3 is c5. Node 1 increases its power to reach the third node in its radio range, so it adds node 4 to the tree, because it has the smallest incremental cost.

Finally, as shown in Figure 2.2(d), we can get the final broadcast tree by repeating this procedure until all nodes are included in the tree.

The BIP is motivated from Prim's algorithm for the MSTs, in the sense that new nodes are added to the tree until all the nodes in the network are added to the tree. Unlike Prim's algorithm [26] using unchanging link costs, however, the BIP updates the costs at each step dynamically with the fact that the cost to add a new node is the *incremental* cost.

We compare the BIP and some other algorithms for the MPB problem with our genetic algorithm, and show which algorithm creates more efficient broadcast trees.

2.2 Genetic Algorithm

In this section, we describe the basics of the genetic algorithm which can be used to find optimal solutions to NP-Complete problems by optimization and search.

2.2.1 Introduction of the Genetic Algorithm

Genetic algorithm is motivated by Darwin's theory about evolution [20]. This algorithm is a particular class of evolutionary algorithm which uses *initialization*, *crossover*, and *mutation* operators to solve optimizing problems. Many different initialization, crossover, and mutation operators have been proposed so far, and we will introduce the most popular operation methods in section 2.2.2, and proposed operations for our own genetic algorithm, which we will be proposed here, are going to be introduced in Chapter 3. In those operations, there is a *population* of abstract representations of *candidate solutions*. The solutions can be represented in binary as string of 0s and 1s [21], but there are also many other different ways to represent the candidate solutions, and these will be introduced in section 2.2.3. Representation methods for our own genetic algorithm also will be presented in Chapter 3.

Figure 2.3 and 2.4 shows the high-level processing steps of the genetic algorithm. Before we initialize a population, we need one more process, encoding, to represent the solutions. Once a population is generated, we evaluate the all the candidate solutions, which will be also called initial solutions, in the population. Here, we use a *fitness function*, which is defined over the genetic representation and measures the quality of the represented solutions to evaluate the solutions. With these initial candidate solutions, we operate selection of parental solutions, crossover of the pairs of parental solutions, mutation of the new solution, which will be called offspring, and then we evaluate the offspring as to whether it has the capability to be included in the population compared with currently existing solutions. If it is evaluated as a better solution to be used, then it replaces one of the solutions of the population. We repeat this procedure until the *termination condition* is satisfied.

2.2.2 Procedures of a Genetic Algorithm

We have studied the basics of the genetic algorithm so far. Now we describe the procedures of the genetic algorithm in detail. The genetic algorithm process consists of the following steps: initialization, evaluation, selection, crossover, mutation, replacement, and termination.

```
procedure genetic algorithm;
begin
    initialize population with randomly generated candidate solutions;
    evaluate each candidate solution;
    while (TERMINATION CONDITION not satisfied) do
        select parents;
        crossover pairs of parents to create a offspring;
        mutate the offspring;
        evaluate the new candidate;
        replace  the new candidate generating a new population
    end while
end;
```

Figure 2.3 The General Pseudo-code for a Genetic Algorithm

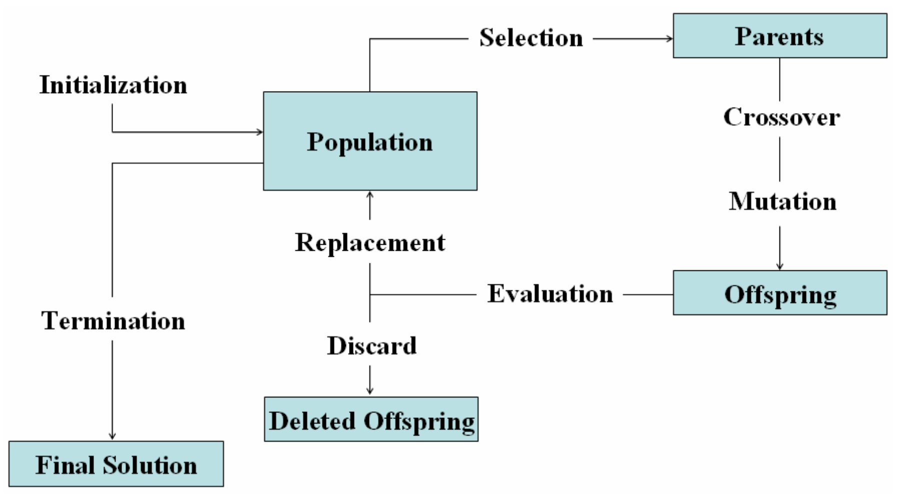


Figure 2.4 The General Scheme of a Genetic Algorithm

2.2.2.1 Initialization Phase

In the first step, many initial individual candidate solutions are generated randomly to form an initial population. Traditionally, the population size, which means how many initial candidate solutions are there, is also generated randomly covering the entire range of possible solutions [21]. However, it has been studied that a very big size of population usually does not improve the performance of a genetic algorithm. The population with 20 to 30 initial candidate solutions has been considered as good population sizes [22].

2.2.2.2 Evaluation Phase

In order to evaluate each individual initial candidate solution, we use an evaluation function which is commonly called a fitness function. We are able to decide which solution is good or better than the other solutions with this fitness function.

Initial Candidate Solutions	Qualities (Fitness values) with Fitness Function
s_1 = 1110011101	$f(s_1) = 7$
s_2 = 0101010110	$f(s_2) = 5$
s_3 = 0000111111	$f(s_3) = 6$
s_4 = 1110000010	$f(s_4) = 4$
s_5 = 1011110110	$f(s_5) = 7$
s_6 = 1110011111	$f(s_6) = 8$

Figure 2.5 An example of initial candidate solutions and fitness function

In Figure 2.5, there are six kinds of initial candidate solutions represented by string of 10 binary digits. The fitness function f of a candidate solution is defined as the number of 1s in its genetic code. If we are trying to maximize the function, then the initial solution s_6 will be the best solution of the population, because it has the largest value for fitness function f.

2.2.2.3 Selection Phase

As we mentioned in the section 2.2.1, candidate solutions are selected from the population to be parents for crossover. The individual candidate solutions are selected through a *fitness-based* process where the solutions whose fitness values are considered as fitter than the others are typically more likely to be selected [21]. With the simplest example, in Figure 2.5, if we decided to select two solutions from the population whose candidate solutions are sorted in descending order of their fitness value, then solutions whose fitness is 8 and 7 will be selected as parents.

The most popular selection method is roulette wheel selection. In this method, parents are selected according to their fitness. If one solution has better fitness, then the solution has more chances to be selected as parents. Figure 2.6 shows a simple example for roulette wheel selection. Imagine that we play marbles and we throw the marble on the roulette wheel, which is shown on the Figure 2.6, where each solution is assigned to a sector of the roulette wheel proportional to its fitness. Then the solution with bigger fitness will be more likely to be selected more times.

Many other methods about how to select parents are introduced in [23], for example, Boltzman selection, tournament selection, rank selection, steady state selection, and some others.

2.2.2.4 Crossover and Mutation Phase

The next step of the procedures of a genetic algorithm is to generate a new offspring by crossover of parental solutions which were selected by a selection method and by mutation operators. The new solution, named the offspring above, typically shares many of the characteristics of its parental solutions. This is from the Darwin's theory in which a child from parents usually has similar characteriscs of his/her parents,

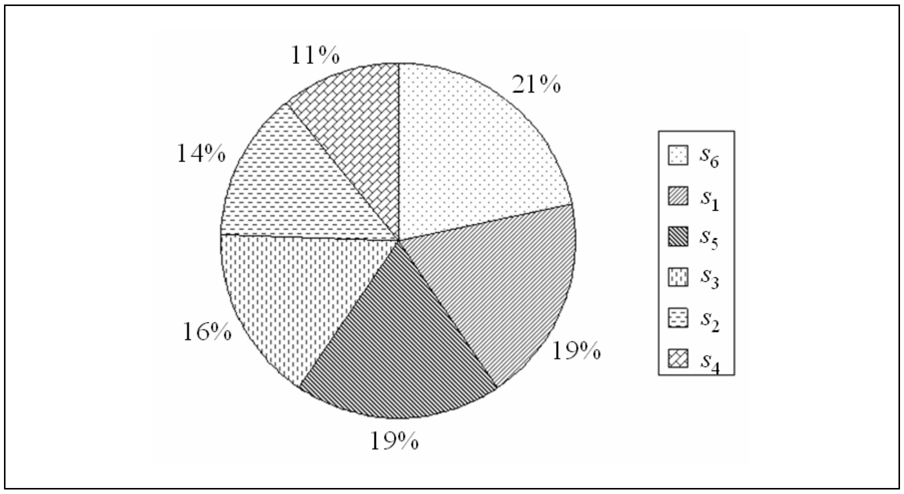


Figure 2.6 Roulette Wheel Selection

therefore, the new offspring may yield better fitness and finally the average fitness will have increased. That's why it is crucial that we need to try to get only good characteristics of parents to yield a new offspring whose fitness is better than its parents as much as we can. This feature can be called *heritability* [23] which means that offspring by crossover should represent solutions combining substructures of their parental solutions. The feature is the most important part of the genetic algorithm.

In order to show the traditional crossover and mutation method, let us consider the initial candidate solutions in Figure 2.5 again. The binary encoding crossover method [20], of course, is good for crossover and mutation of the solutions which are represented by binary strings. Figure 2.7 introduces two kinds of crossover methods, single point crossover and two point crossover. In a single-point crossover as shown in Figure 2.7 (a), one crossover point is assigned to each parent solution. The binary strings from the beginning of the first parent to the crossover point are copied to the offspring, and the binary strings from the crossover point to the end of the second parent

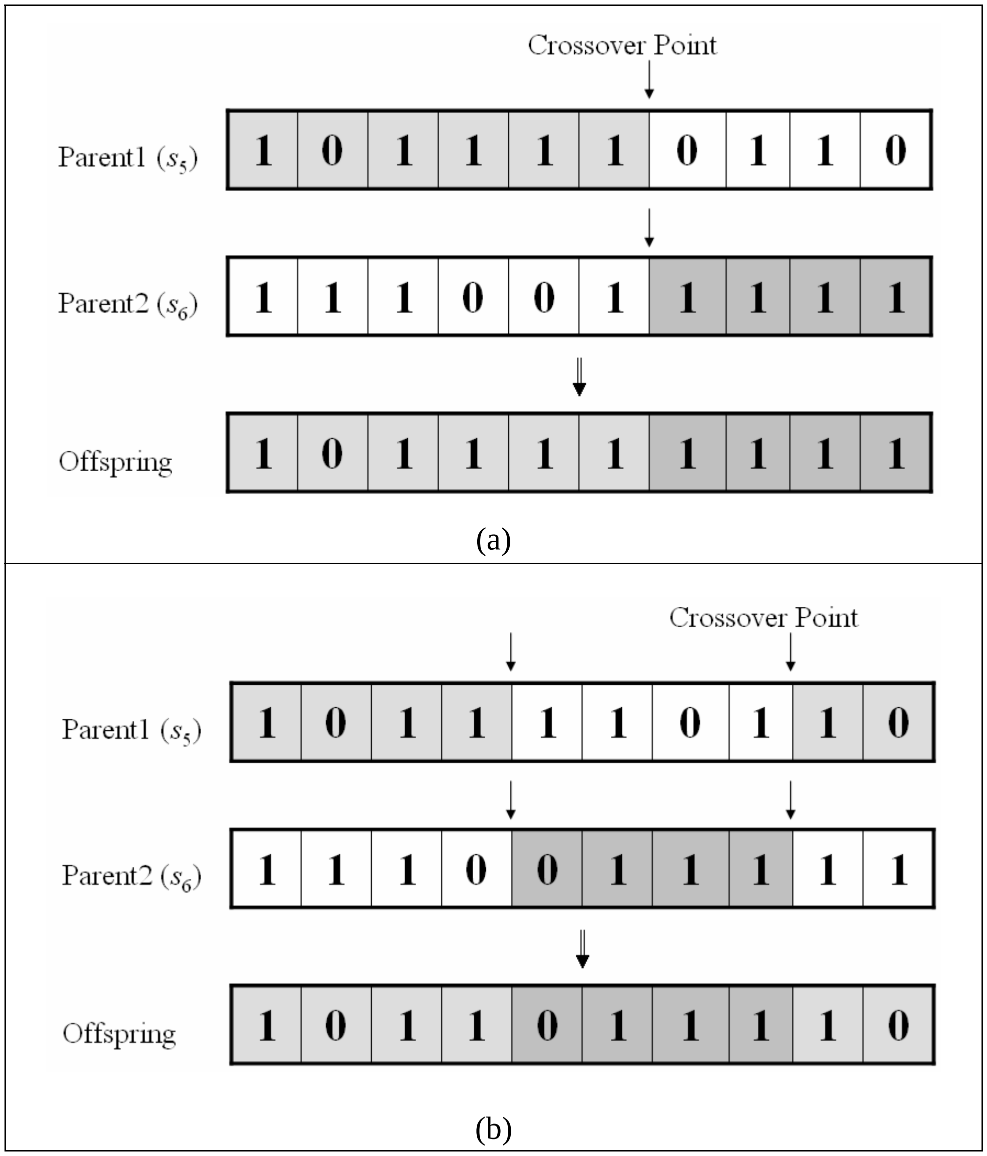


Figure 2.7 Examples of Binary Encoding Crossover Methods

(a) Single-Point Crossover (b) Two-Point Crossover

are copied to the rest of the offspring. In the two-point crossover as shown in Figure 2.7 (b), two crossover points are assigned to each parent solution. Then the first part of the offspring is filled with the binary strings from the beginning of the first parent to the crossover, the second part is filled with the binary strings from the first crossover point to the second crossover point of the second parent, and the rest part of the offspring is filled with the binary strings from the second crossover point to the rest of the first parent solution.

The binary encoding mutation method [20] is completed with the selected bits after the crossover operation. If we do the operation with the offspring which is made by the two-point crossover method, and there are selected bits as shown in Figure 2.8, then the selected bits are inverted.

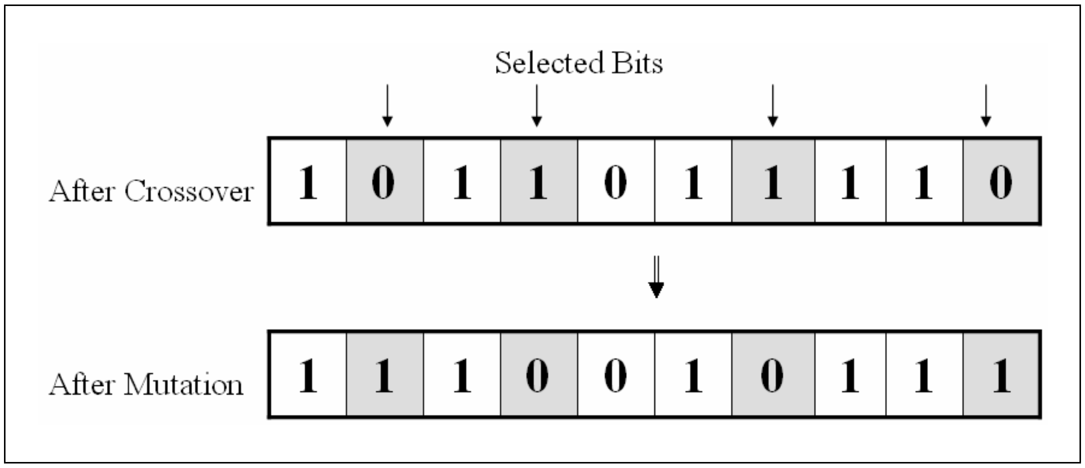


Figure 2.8 An example of Binary Encoding Mutation Method (Bit Inversion)

With these methods for representation of solutions, there are also many other crossover and mutation methods using permutation encoding, value encoding, tree encoding, and so on [20].

As a matter of fact, it is important to note that these operations, crossover and mutation, are representation dependent. The methods for crossover and mutation which have been introduced so far are suitable only for solutions represented by binary strings. If we represent solutions by tree structures which are the method we used to represent solutions, then other kinds of crossover and mutation operators are required. We will introduce the tree-based representation of solutions using edge-set which was the motivation of representation method of our work in Chapter 2.2.3.

There are basic recommendations for crossover and mutation operations in [20]. Crossover rate should be high; on the other hand, mutation rate should be very low. It would be better to have 80~95% of offspring rates of initial population size by crossover. In case of mutation, even though it is used to prevent all solutions in a population from falling into a local optimum of the solved problem, the reported best rate of offspring generated by mutation is only 0.5~1% of initial population size.

2.2.2.5 Replacement Phase

The main role of replacement operator is to place the new offspring, which is generated by the crossover and mutation, as a new member of the initial population. To decide which currently existing solutions need to be replaced or whether the new offspring needs to be replaced or not are based on the finesses of all the solutions. The solutions having better qualities, of course, should be kept while the solutions having worse qualities need to be replaced by the new offspring having better quality.

All of the above means that we are able to get a better solution or at least get the solution having same value with the parental solutions since we replace the offspring only if it has better quality than the qualities of the currently existing solutions in the population. Therefore, we guarantee that we can create a better solution, however sometimes we get the same solution, after the genetic operations, not a worse solution.

2.2.2.6 Termination Conditions

The process above is repeated until a termination condition is satisfied. There are a number of termination conditions proposed in [21], and [24];

- When a pre-determined number of generations or time has elapsed
- When a satisfactory solution has been achieved
- When no improvement in solution quality has taken place for a pre-determined number of generations
- When a solution is found that satisfies minimum criteria
- Combinations of the above

2.2.3 Edge-Set Representation of Trees in a Genetic Algorithm (*GA*)

As we mentioned in the previous sections, a standard representation of solutions is as an array of bits. However, if we need to use tree-based representation for solutions, it is not possible to use bit-string representation method.

In the GA, the representation of initial candidate solutions in a population is crucial in order to operate the algorithm efficiently. The basic representation method using bit-string was introduced in the previous chapters, and the other methods of representation of the candidate solutions in the GA also have been studied so far. Gunther R. Raidl proposed a method, named 'Edge-Set Representation' of tree [3], and the tree-representation in our GA has been motivated by this technique. Other techniques also have been proposed by many researchers. Piggott and Suraweeera proposed how to represent a solution using a bit string in [17], the most famous coding of trees, named ‘Prufer coding’, for GA was studied by Prufer in [18], and Kim and Gen [19] expended the Prufer coding in a GA.

In this section, we briefly describe the ‘edge-set representation’ of tree which motivated our tree-representation briefly. This representation technique is used to represent *d*-MST, which is the degree-constrained minimum spanning tree. We modified this technique for our trees in GA, which do not have any degree constraint, and we are going to introduce the modified algorithms of ‘edge-set representation’ in Chapter 3.

2.2.3.1 Initialization

In the initialization phase using the edge-set representation, the algorithm of ‘edge-set representation’ creates random *d*-STs in the initial population of a GA. Given node constellation, as shown in Figure 2.9 [3], first we choose an edge from the list of all possible edges to start building a tree with the nodes in random order. When an edge is selected, then we check if the selected edge violates the degree-constraint. If it doesn’t violate the degree constraint, and it also doesn’t create any cycles with currently existing edges in the tree, then the edge is added to the tree. If it does violate the degree constraint, or create any cycles, then we discard the edge and choose another edge in random order from the list of all possible edges.

We now have a feasible population which consists of a number of initial trees built by the above algorithm, and they will yield new offsprings following edge-crossover and insertion-mutation operators.

```
procedure initialize;
Begin
    $E_T \leftarrow \emptyset$;
    for all edges $(i, j) \in E$ in random order do
        if $deg(i) < d$ and $deg(j) < d$ and not (connected($m, n, E_T$)) then
            $E_T \leftarrow E_T \cup \{(m, n)\}$;
        if $|E_T| = |V| - 1$ then
            return $E_T$;
end;
```

Figure 2.9 Creating an initial, random *d*-ST (Pseudo-code from [3])

2.2.3.2 Edge-Crossover

In this representation, the edge-crossover operation considers inheriting, which is a crucial aspect of the genetic algorithm. It inherits as many edges as possible from two parental *d*-STs. The Figure 2.10 [3] shows the pseudo-code for the edge-crossover operation. It starts with parental solutions (we will call these parental solutions parent trees from now on.). These are selected from the initial trees in the population which we created in the previous initialization phase. In the first step, we find the all edges that are in both parent trees and these found edges are going to be included in the new offspring tree. In the second step, all edges either in one parent, E_T^1, or another parent, E_T^2, are selected. As same with the initialization phase, the edges which violate degree-constraint must not be added to the new tree. If a tree can be completed with only these two steps, then it is created. However, unfortunately we may not terminate this procedure due to the unconnected components, which are partial spanning trees and they must be connected together by including edges not contained in the parents. In this case, we set V, a set of all vertices, as disjointed sets U_k as shown in Figure 2.9 [3]. Then the unconnected components are connected to build a final complete *d*-ST by choosing two vertices repeatedly from the set V, and the edge which consists of the previously selected vertices must not violate degree constraint.

2.2.3.3 Edge-Insertion-Mutation

In this step, mutation is operated by inserting an edge as shown algorithm in Figure 2.11. First, a new edge which doesn't violate the degree constraint is selected to be inserted, and the all edges which lie on the path of the previous selected edge to be inserted are also found. Then one edge from the edges lying on the path is selected to be deleted. When the edge is selected to be deleted, the degree constraint also needs to be checked.

```
procedure edge-crossover($E_T^1$, $E_T^2$);
Begin
    $E_T \leftarrow E_T^1 \cap E_T^2$;
    $F \leftarrow (E_T^1 \cup E_T^2) \setminus E_T$;
    for all edges $(i, j) \in F$ in random order do
        if $deg(i) < d$ and $deg(j) < d$ and not (connected($i, j, E_T$)) then
            $E_T \leftarrow E_T \cup \{(i, j)\}$;
        if $|E_T| \neq |V| - 1$ then
            return $E_T$;
    (*determine all unconnected components $U_k$*)
    $U \leftarrow \{U_k\}$ with $\forall i,j \in V, i \neq j$;
        $i \in U_k \wedge$ connected($i, j, E_T$) $\rightarrow j \in U_k$,
        $i \in U_k \wedge$ not connected($i, j, E_T$) $\rightarrow j \notin U_k$,
        $\cup_k U_k = V$;
    (*connect components randomly*)
    for all $U_k \in U \setminus \{U_1\}$ in random order do
        choose $i \in U_1 \mid deg(i) < d$ randomly;
        choose $j \in U_k \mid deg(j) < d$ randomly;
        $E_T \leftarrow E_T \cup \{(i, j)\}$;
        $U_1 \leftarrow U_1 \cup U_k$;
    return $E_T$;
end;
```

Figure 2.10 Edge-Crossover (Pseudo-code from [3])

```
procedure edge-insertion-mutation($E_T$);
begin
    (*choose edge $(i, j)$ for insertion*)
    choose $i \in V$ randomly;
    choose $j \in V \setminus \{i\} \mid deg(i) < d \wedge (i, j) \notin E_T$ randomly;
    (*choose edge $(a, b)$ for deletion *)
    $L \leftarrow \{(k, l) \in E_T \mid (k, l)$ lies on path from $i$ to $j$ $\}$;
    if $deg(i) = d$ then
        $(a, b) \leftarrow (a, b) \in L \mid a = i \vee b = i$ ;
    Else
        choose $(a, b) \in L$ randomly;
    $E_T \leftarrow E_T \cup \{(i, j)\} \setminus \{(a, b)\}$;
    return $E_T$;
end;
```

Figure 2.11 Edge-Insertion-Mutation (Pseudo-code figure from [3])

We modified these tree-representation algorithms for our genetic algorithm to build broadcast tree efficiently. As we mentioned in the beginning of this section, we now introduce our genetic algorithm with the modified edge representation technique in the next chapter.

CHAPTER 3

PROPOSED GENETIC ALGORITHM TO BUILD BROADCAST TREES

In this section we describe our genetic algorithm to build the minimum-energy broadcast trees in the wireless sensor networks.

First, we describe how to create initial broadcast trees, which are also called candidate solutions in a population, and some of these are selected to be parental trees that are evolved to yield offsprings having more efficient energy than their parents.

Second, we present how to evolve the selected parental trees and how to replace the children trees (offsprings) in the population to replace existing members of the population in the case that we have children having more efficient energy.

Third, we discuss the mutation method for our genetic algorithm which needs to be made to prevent the genetic algorithm from falling into a local extreme.

Finally, we describe the environment of our genetic algorithm. We present how many times we iterate the evolving operation and the termination condition that is used in our algorithm.

3.1 Initialization Phase

3.1.1. Creating Initial Trees in a Population

In this phase, we create initial trees which are the candidate solutions in a population. In order to create more feasible initial trees which may yield more efficient offspring trees, we put three different kinds of trees in the population.

The first initial tree is selected by the BIP algorithm [1], and is based on the idea that the genetic algorithm creates offspring trees which have at least the same energy as their parent trees. So this provides a good initial solution.

```
procedure initialize-MST;
begin
    $E_T \leftarrow \emptyset$;
    for all edges $(i, j) \in E$ do
        $E_D \leftarrow$ FindDistanceBetweenNode$(i, j)$;
    for all edges $(m,n) \in E_D$ in ascending order do
        if not (connected$(m, n, E_T)$) then
            $E_T \leftarrow E_T \cup \{(m, n)\}$;
        if $|E_T| = |V| - 1$ then
            return $E_T$;
end;
```

Figure 3.1. Pseudo-Code for “initialize-MST”

The second initial tree is built on the basis of the *d*-ST algorithm [3] which is derived from Kruskal’s MST (Minimum Spanning Tree) algorithm [9]. Figure 3.1 shows the modified pseudo-code for the *d*-ST algorithm. It initializes E_T to the empty set and finds edges in an ascending order of the distances between nodes from the list of all possible edges in the network. After an edge is selected from the list of all possible edges in the network, if it doesn’t create any cycles with any edges which currently exist in the E_T then put the selected edge into the E_T. This procedure is continued until the size of E_T becomes one less then the number of nodes. And we use the union-find data structure [10] to test if the edge makes any cycles with currently existing edges in E_T. In this algorithm, the cost will be the distance between two nodes.

The last trees of the population are randomly created. Figure 3.2 shows the procedure to create random trees and they are initialized by selecting edges randomly from the list of all possible edges in the network. As shown in the figure, we first initialize E_T to be the empty set and then find any edges in random order from the list of all possible edges and put the selected edges into E_T if they don't make any cycles with the currently existing edges. We also continue this procedure until the size of E_T becomes one less than the number of nodes.

```
procedure initialize-RandomTree;
begin
    $E_T \leftarrow \emptyset$;
    for all edges $(i, j) \in E$ in random order do
        if not (connected($i, j, E_T$)) then
            $E_T \leftarrow E_T \cup \{(i, j)\}$;
        if $|E_T| = |V| - 1$ then
            return $E_T$;
end;
```

Figure 3.2 Pseudo-Code for "initialize-RandomTree"

Figure 3.3 shows all initial candidate trees we just created in the population. The 10 node–network size is 400m x 400m and the initial radio range is 200m. We will find new and better trees with these initial trees doing following procedures.

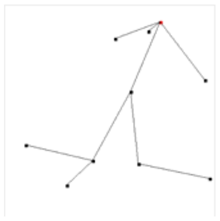
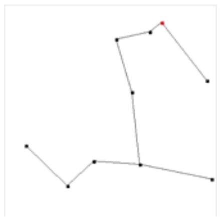
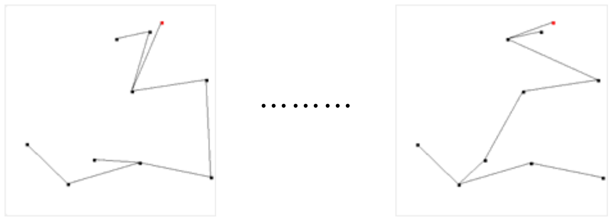
BIP	MST	Random Trees
		

Figure 3.3 Population (Initial Trees)

3.1.2. Parent Selection

Before we generate new offspring trees from the initial candidate trees, we need to select trees which will be parental trees to create a new offspring tree. Figure 3.4 shows how we select parental trees from the population. We first sort the initial trees by consumed energies of each tree, and get the top 20% of the trees from the list of all the sorted initial trees. We choose the tree having smallest energy consumed be one of the parental trees, and randomly select one to be another one of the parental trees from the list of top 20%. We do this procedure to prevent the same tree from being selected again and again. The selected parents are ready to be crossovered.

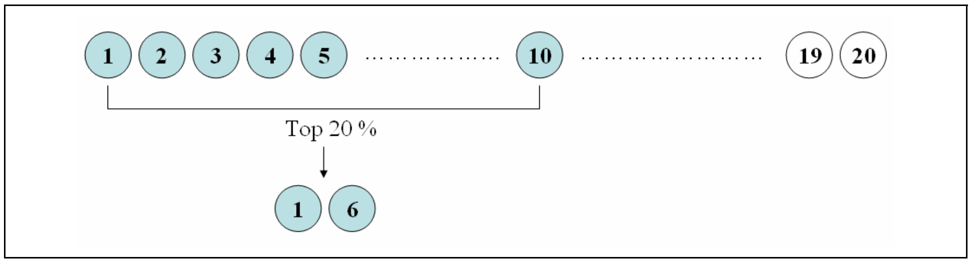


Figure 3.4 Parent Selection

Father(Tree 1) : The tree having the smallest energy

Mother(Tree 6) : The tree selected randomly from the list of top 20%

3.2 Crossover Phase

In the crossover phase, we evolve the parental trees we selected from the list of initial candidate trees. In this operation, the most important thing is not only to find better solution from the initial trees but also to maintain the inheritance by letting the new offspring have the crucial property of its parents.

3.2.1. Modified-Edge-Crossover

Figure 3.5 shows an example of the modified-edge-crossover, and Figure 3.6 shows the pseudo-code of modified-edge-crossover algorithm derived from edge-crossover algorithm [3]. First, we initialize edge set, E_{T}, to have all the edges which are in both of the parental trees (the intersection of the parents $E_{\mathrm{T}}^{1}$and E_{T}^{2}), and the edge set, F, to have all the edges which are in either one of the parental trees or another one of the parental trees, but not in both (the difference between ($E_{\mathrm{T}}^{1} \cup E_{\mathrm{T}}^{2}$) and E_{T}). The edges will be used to create a new offspring tree, and we do this procedure for inheritance so that the new offspring can have the properies of its parents. Next, we choose one vertex in random order from the set of all vertices, V, and find all edges which are in both of the radio range of the selected vertex (node) and F, and put the found edges into the set, E_{V}. For all edges E_{V}, we select edges in random order. If the selected edges do not create any cycles with the currently existing edges in E_{T}, we put the edges into the final edge set, E_{T}. The reason why we found all possible edges in radio range of the selected vertex is that we would like to exploit the *wireless broadcast advantage* (*wireless multicast advantage*) [1], [8] which permits all nodes within communication range to receive a transmission without additional expenditure of transmitter power. Therefore, we may reduce some energy even though it is small. We do this procedure until all the vertices are checked and finally return E_{T}. Since we do not give the algorithm degree constraint like the original edge-crossover algorithm [3], when the above procedure returns E_{T}, we will not get an incomplete tree. We continue this procedure until we complete a new offspring tree.

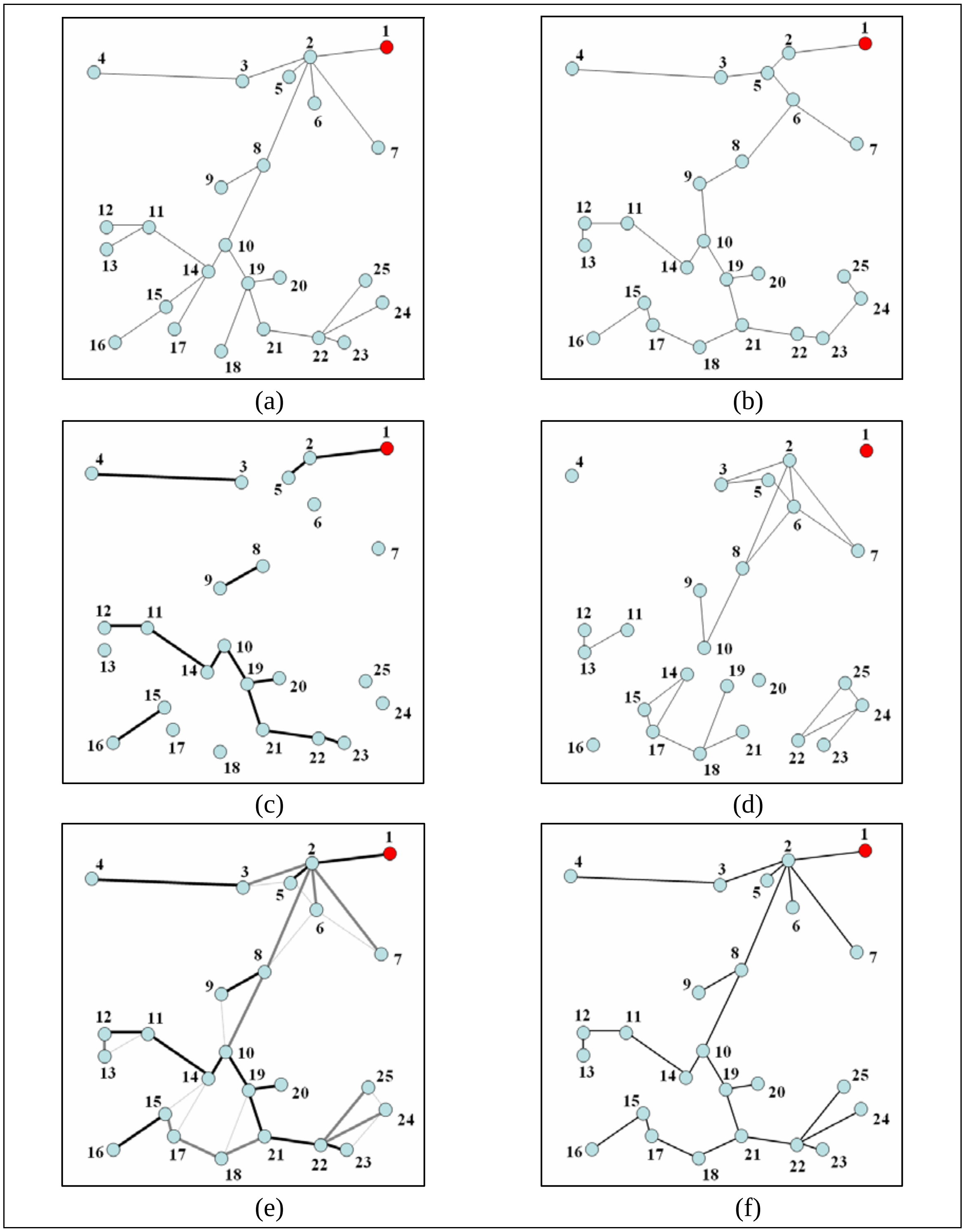


Figure 3.5 An example of ‘modified-edge-crossover’

(a) E_T^1 , (b) E_T^2 , (c) $E_T \leftarrow E_T^1 \cap E_T^2$, (d) $F \leftarrow (E_T^1 \cup E_T^2) \backslash E_T$, (e) E_T , (f) E_T

```
procedure modified-edge-crossover($E_T^1$, $E_T^2$);
begin
    $E_T \leftarrow E_T^1 \cap E_T^2$;
    $F \leftarrow (E_T^1 \cup E_T^2) \backslash E_T$;
    for all vertices $V_k \in V_T$ in random order do
        $E_V \leftarrow$ FindEdges ($k$);
        for all edges $(i, j) \in E_V$ in random order do
            if not (connected($i, j, E_T$)) then
                $E_T \leftarrow E_T \cup \{(i, j)\}$;
    return $E_T$;
end;
```

Figure 3.6 Pseudo-code for “modified-edge-crossover"

3.2.2. Replacement

After we get the new offspring tree, we need to decide whether the offspring should replace a node in the population or not. In the population, we first find the broadcast tree which has the largest energy consumed. If the new offspring has smaller energy consumed, then it replaces the tree having the biggest energy as a member of the population.

3.3 Mutation Phase

We use the remaining trees except the trees which are already used for parent selection to operate mutation phase. Figure 3.7 shows the modified-edge-insertion-mutation algorithm which is derived from the edge-insertion-mutation algorithm in [3]. First, a new edge is selected to be inserted, and all the edges which lie on path of the selected edge to be inserted are also found. Then one edge of the edges we just found is selected randomly to be deleted.

```
procedure modified-edge-insertion-mutation($E_T$);
begin
    (*choose edge ($i$, $j$) for insertion*)
    choose $i \in V$ randomly;
    choose $j \in V \setminus \{i\} \wedge (i, j) \notin E_T$ randomly;
    (*choose edge ($a$, $b$) for deletion *)
    $L \leftarrow \{(k, l) \in E_T \mid (k, l)$ lies on path from $i$ to $j$ $\}$;
    choose $(a, b) \in L$ randomly;
    $E_T \leftarrow E_T \cup \{(i, j)\} \setminus \{(a, b)\}$;
    return $E_T$;
end;
```

Figure 3.7 Pseudo-code for “modified-edge-insertion-mutation”

CHAPTER 4

IMPLEMENTATION AND EVALUATION

We programmed and implemented our genetic algorithm in order to construct the minimum-energy broadcast trees in several different networks using Java. In this section, we show the implementation result of our genetic algorithm and evaluate our technique and also compare the result with the traditional broadcast algorithm, BIP [1] which was briefly introduced in Chapter 2.

4.1 Implementation Environment

We considered source-initiated and circuit-switched networks, and generated 10 different networks whose implementation results are averaged to be evaluated. The specified number of nodes, typically 25, 50, and 75 nodes, are randomly deployed and they are fixed in the 400m x 400m networks. The initial radio range of each node is 200m, and a node which resides on the top right hand side of the network was chosen to be the source node.

4.2 Evaluation

We implemented our genetic algorithm with three different population sizes, 10, 20, and 30 with 10,000 iterations. We compared the energy consumptions of the broadcast trees which were constructed under our genetic algorithm with the tree under BIP algorithm in section 4.2.1.

We also evaluate our genetic algorithm itself according to the number of iterations in section 4.3.2. We show how the genetic algorithm improves itself at each iteration with several population sizes. Since we can not guarantee how many iterations and how many population sizes must be good for a genetic algorithm, we just show the energy consumed and improved in each of population size and iteration.

4.2.1 Comparison of Energy Consumptions of Trees built under BIP and GA

Table 4.1 shows the result of implementation of the BIP and GA. In the network with 25 nodes, BIP has the largest energy consumption and the GA with 20 initial trees has the smallest energy consumption. In the network with 50 nodes, the BIP also has the largest energy consumption and the GA with 30 initial trees has the smallest energy consumption, and in the network with 75 nodes, the GA with 10 initial trees has the smallest energy consumption.

Table 4.1 Comparison of Energy Consumption between BIP and Genetic Algorithm

No. of nodes	Energy Consumed (Average of 10 networks)			
	BIP	Genetic Algorithm (Population Size)		
		(10)	(20)	(30)
25	10,600	10,600	8,780	10,600
50	34,900	27,065	27,420	25,550
75	86,400	63,715	68,605	65,285

Table 4.2 shows the energy improvements of GAs over the BIP algorithm. In the network with 25 nodes, the GA with 10 initial trees and another GA with 30 initial trees improve 0% over the BIP algorithm. Only the GA with 20 initial trees improves 17.17%

over the BIP in the network. In the network with 50 nodes, the GA with 30 initial trees shows the best improvement over the BIP algorithm (26.79%), the GA with 10 initial trees has improvement of 22.45% over the BIP, and the smallest improvement of 21.43% was achieved by the GA with 20 initial trees. In the network with 75 nodes, the GA with 10 initial trees has the largest improvement (26.26%) over the BIP, the GA with 30 initial trees improves in 24.44% over the BIP, and finally the GA with 20 initial trees has the smallest improvement of 20.60% over the BIP algorithm.

Table 4.2 Energy Improvements of the Genetic Algorithm (%) over BIP

No. of nodes	Energy Improvements over BIP (%)			
	(Population Size)			Average
	(10)	(20)	(30)	
25	0 %	17.17 %	0 %	5.723 %
50	22.45 %	21.43 %	26.79 %	23.558 %
75	26.26 %	20.60 %	26.44 %	23.764 %

The averages of energy improvements of the GAs over the BIP are also shown in Table 2. On average, the GA has improvement of 5.72%, 23.56% and 23.76% in the 25-node, 50-node, and 75-node network, respectively. Figure 4.1 also shows the data in Table 4.1 as a graph. And Figure 4.2 show the percentage improvements of our genetic algorithm over the BIP algorithm.

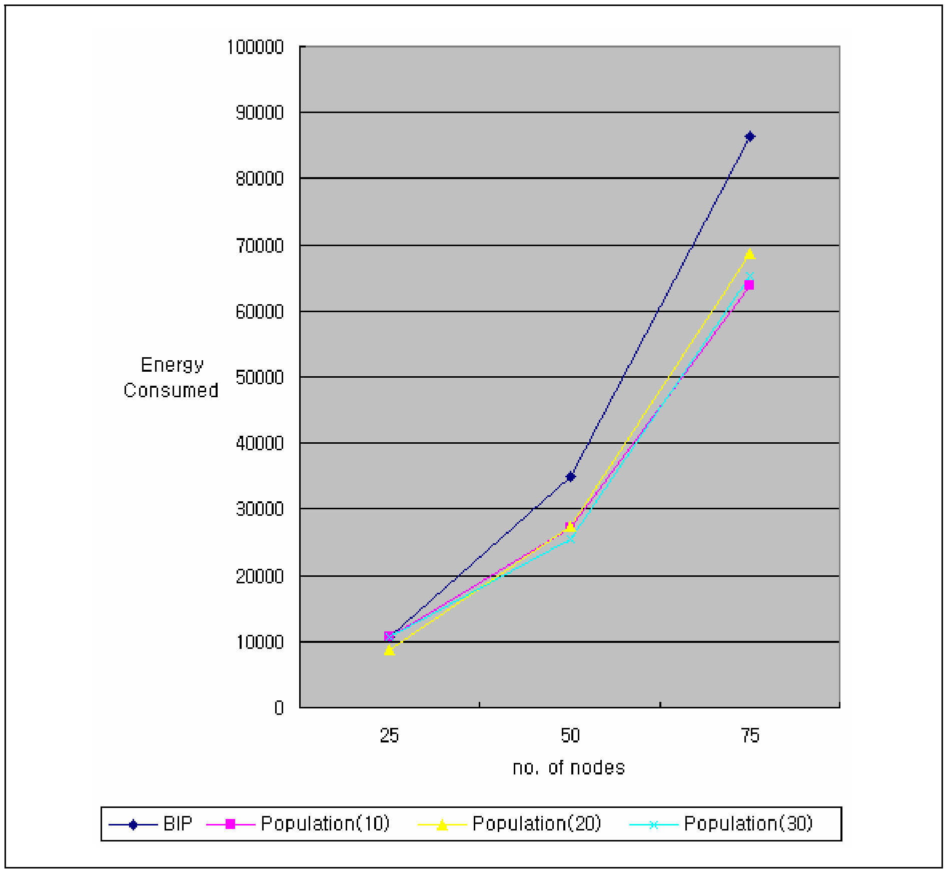


Figure 4.1 Energy Consumption (BIP vs. GA)

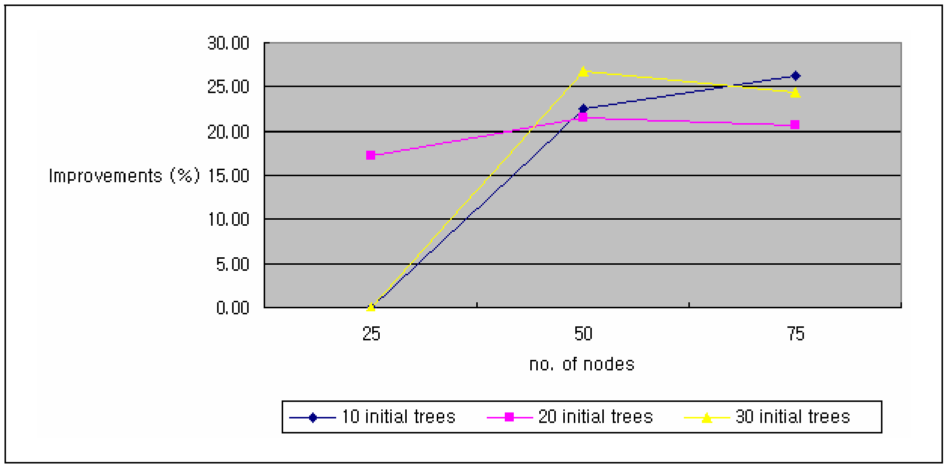


Figure 4.2 Percentage Improvements (%) of GA over BIP

4.2.2 Analysis of the Genetic Algorithm used

Since the genetic algorithm can be implemented with the different size of population and different number of iterations, we evaluate the result of our genetic operation itself.

As shown in Figure 4.3, we are not able to guarantee how much number of iterations is needed to get the near optimal solution. We are just able to know that we can eventually reduce the energy consumption if we use more iterations.

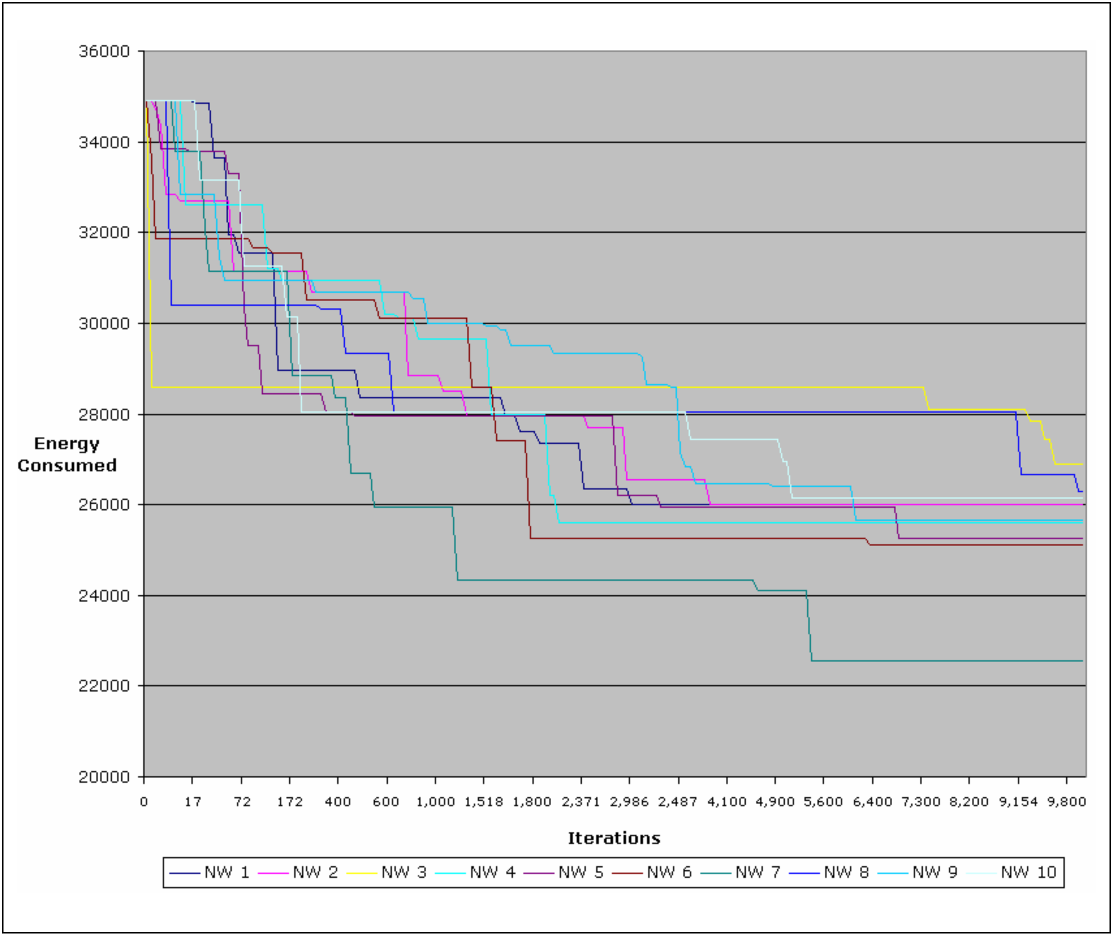


Figure 4.3 Energy Consumption at each iteration
(50-node network and Population size 30)

Let us look up the networks which were analyzed in Figure 4.3 in more detail. As shown in the graph, we are also able to know that even though we used less than 400 iterations, we were able to get more efficient trees than the initial best trees. And after 3,000 iterations, the rates of energy gained according to the iteration are gradually decreased.

CHAPTER 5

COMPARISONS WITH OTHER WORKS

In the previous chapter, we compared our implementation result with the traditional BIP algorithm and we were able to get the evaluation result that our genetic algorithm is more efficient to build broadcast trees in wireless sensor networks than the BIP algorithm.

In this chapter, we compare our genetic algorithm used to build broadcast trees with other related works which have been studied to solve the MPB problem so far.

5.1 Related Work

First of all, we introduce two kinds of operations, sweep operation [1] and *r*-shrink operation [15], which can be used to improve algorithms. These operations are used in some algorithms in related work. The algorithms are an ant colony system (ACS) algorithm [5], cluster-merge algorithm [14] and simulated annealing algorithm [6] which is used to solve the MPB problem.

We compare our genetic algorithm with the BIP and the other algorithms together. We also apply the sweep operation and the r-shrink operation to the BIP and the SA, and the r-shrink operation to the CM algorithm.

5.1.1 Sweep Operation: *Removing Unnecessary Transmissions*

This operation called “sweep” operation was introduced in [1] to improve the BIP algorithm by eliminating unnecessary transmissions. We apply the sweep operation to the final tree generated by a minimum broadcast algorithm. In this operation, we examine the relay nodes, but we ignore the leaf nodes because they do not transmit and hence they do not effect on the power consumption.

Figure 5.1 shows an example of the sweep operation. We examine the relay nodes, node 2, 5, 8 and 9 which are in ascending ID order. The non-leaf node 2 can also reach node 3 without further expenditure of power. Therefore the transmission by node 1 can reduce its power by reaching only node 2. We continue the examination through the non-leaf nodes, and we finally get the tree in Figure 2.2 (b).

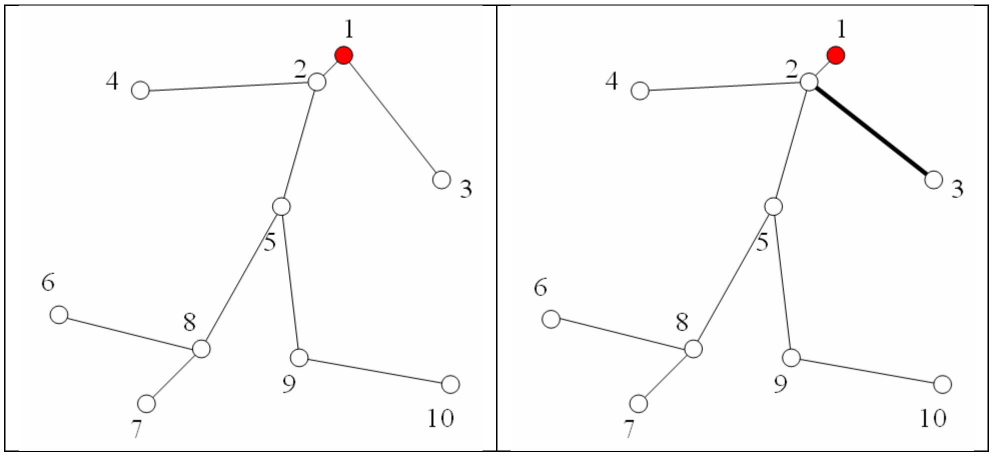


Figure 5.1 An Example of “Sweep” operation

(a) Original Tree, (b) After “Sweep” Operation

5.1.2 r-shrink Operation: r = 1

r-shrink procedure which is a heuristic for improving sub-optimal MPB trees in wireless networks is introduced in [15]. Given a tree which is generated by the BIP algorithm or the other MPB algorithms, the transmission radii of transmitting nodes in the tree are shrunk sequentially. The purpose of the procedure is to reduce the transmission power of the node whose radio range is shrunk.

Figure 5.2 shows the r-shrink operation of the node *j*. For *r* = 1, the farthest node from node *j* which the *r*-shrink operation is to be applied to will be changed to another node *k* which is the farthest node from *j* except for the original farthest node *i*. It means that there will be the reduction of the transmission power level of node *j* by 1 *notch*. Similarly, for *r* = 2, the transmission power of the node *j* which is decided to be applied to will be reduced by 2 *notches*. It means the farthest node will be changed to the node *l*, and in the radio range of the node *j* there were two further nodes than the node *l*.

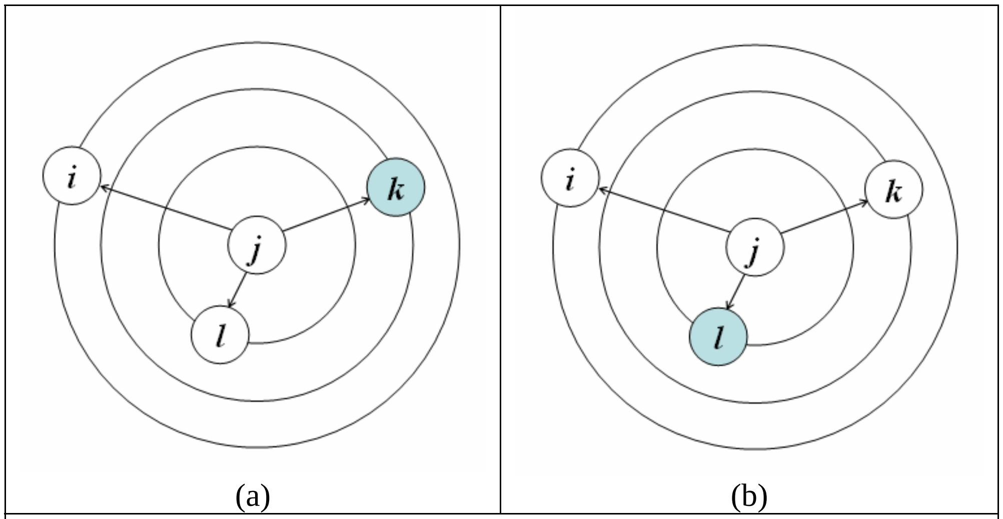


Figure 5.2 An Example of *r*-shrink operation of node *j*

(a) 1-shrink : node *i* (the original furthest node of node *j*)
node *k* (the new furthest node of node *j*)
(b) 2-shrink : node *i* (the original furthest node of node *j*)
node *l* (the new furthest node of node *j*)

As shown in Figure 5.2 (a), if we apply 1-shrink operation to the node j, then the node *i* will be disconnected from the rest of the tree. Then we determine whether the temporarily disconnected child *i* retains its existing parent *j* or is assigned a new parent from the set of its *foster parents* by using the *incremental* costs and *decremental* costs which were used in the BIP algorithm that we explained in Chapter 2. The foster parents of a node are the set of the non-descendants, excluding its current parent.

5.1.3 The Other Algorithms Compared with the Genetic Algorithm

5.1.3.1 Ant Colony System (ACS) Algorithm

The ant colony system (ACS) algorithm was first proposed in [25] and the ACS procedure for solving MPB problem is introduced in [5]. The ACS algorithm is based on a swarm based optimization procedure.

Swarm intelligence approach gives rise to intelligent behavior through complex interaction which is from independent swarm members, and the interaction is from instincts. Finally, it accomplishes complex forms of the behavior and fulfills a number of optimizations.

5.1.3.2 Cluster – merge (CM) Algorithm

The cluster – merge (CM) algorithm is used to solve the MPB problem in two phases, *cluster phase* and *merge phase,* and is introduced in [14].

In the *cluster phase,* it varies the BIP algorithm. The tree will have the clusters which represent the partial connected subtrees. If *n* nodes are divided into *m* clusters, then we have at most *m* – 1 links to connect the clusterheads. It means the algorithm is able to reduce much complexity, and also make the algorithm suitable for use in large scale dense networks. In the *merge phase,* it uses positive reinforcement search procedure adopted in swarm intelligence algorithms.

5.1.3.3 Simulated Annealing (SA) Algorithm

Montemanni *et el*. [6] introduces a simulated annealing approach to solve the MPB problem. The simulated annealing is also related with global optimization like the genetic algorithm which we used in this paper, and it travels over the search space by testing random mutations on an individual solution. The simulated annealing also accepts the new solution having increased fitness values as the genetic algorithm; however it accepts lowered fitness values probabilistically based on the difference in the fitness.

5.2 Evaluation

First of all, Table 5.1 shows the energy consumptions of trees generated by all algorithms explained so far. In case of the GA, we evaluate the result with 20 initial trees and 10,000 iterations. As shown in the table, in the 25-node network, SA with sweep algorithm has the smallest energy consumption, and the GA is the fourth algorithm in terms of small energy consumption. In the 50-node network, the GA has the most efficient energy consumption, and the SA with sweep algorithm has the second best energy efficiency. In the 75-network, the GA also has the best result among the algorithms, and also the SA with sweep algorithm has the secondary energy efficiency. In the 100-node network, the GA could not produce a result because it ran out of memory space; however, the SA with sweep algorithm was able to get the best efficiency for energy consumption. Figure 5.3 shows the data in the table as a graph. As shown in Figure 5.3, in 50-network and 75-network, the energy-line of the GA is below all the other lines of each algorithm.

Table 5.1 Energy Consumptions of Trees generated by different algorithms

No. of nodes	BIP	BIP + Sweep	BIP + 1-Shrink	ACS	CM + 1-Shrink	SA	SA + Sweep	GA
25	10,600	10,328	9,571	8,686	8,703	8,491	8,465	8,780
50	34,900	34,240	31,940	30,038	29,606	28,918	28,859	27,420
75	86,400	84,465	79,272		73,397	73,103	72,360	68,605
100	119,700	117,222	108,867		101,853	102,571	101,338	

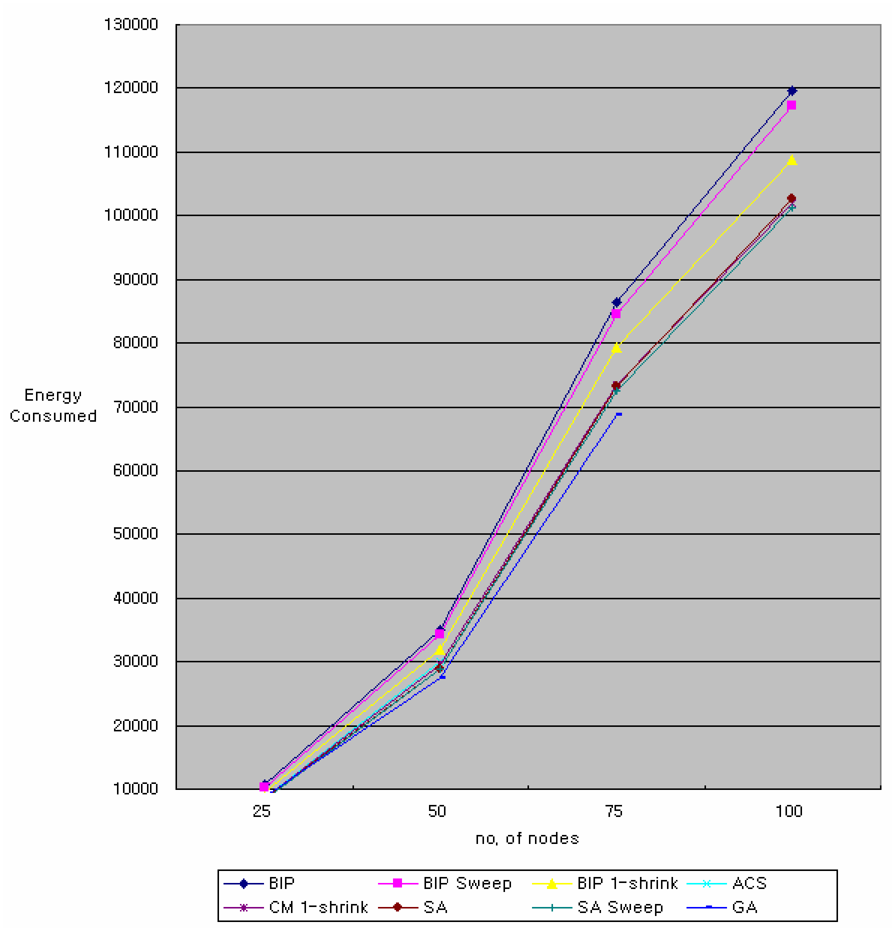


Figure 5.3 Energy Consumptions of Trees generated by each algorithm

Table 5.2 Energy Improvements of the different Algorithms (%) over BIP

No. of nodes	BIP + Sweep	BIP + 1-Shrink	ACS	CM + 1-Shrink	SA	SA + Sweep	GA
25	2.57 %	9.71 %	18.06 %	17.90 %	19.90 %	20.14 %	17.17 %
50	1.89 %	8.48 %	13.93 %	15.17 %	17.14 %	17.31 %	21.43 %
75	2.24 %	8.25 %		15.05 %	15.39 %	16.25 %	20.60 %
100	2.07 %	9.05 %		14.91 %	14.31 %	15.34 %	

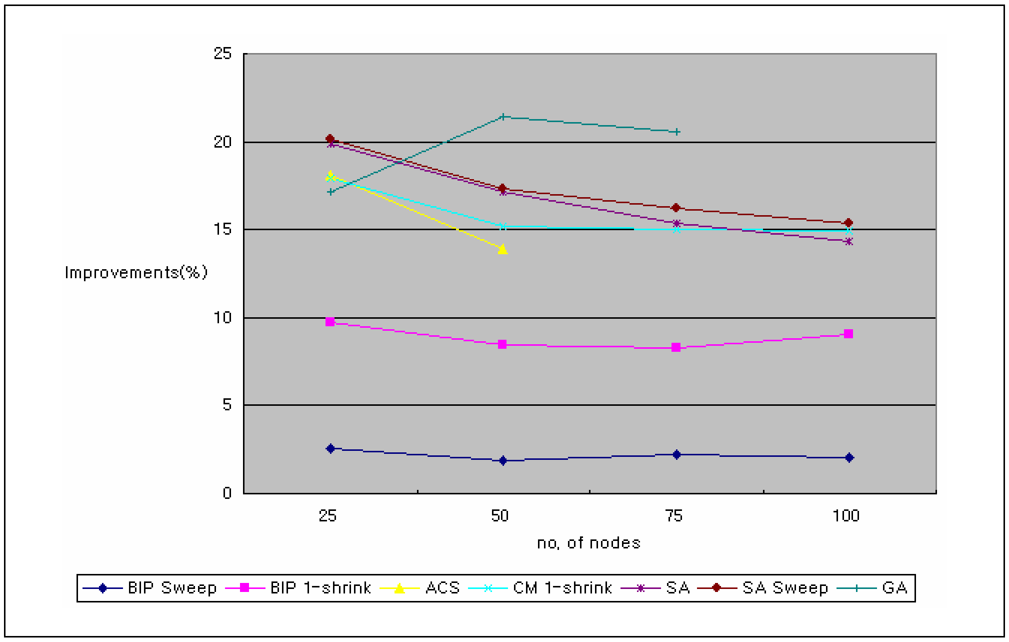


Figure 5.4 Percentage Improvements (%) of different algorithms over BIP

Table 5.2 shows the energy improvements of all the algorithms over the BIP algorithm. In 25-node network, the SA with sweep algorithm has improvement of 20.14% over the BIP. It shows the largest improvements among the algorithms, and our algorithm, GA, has improvement of 17.17% over the BIP. In 50-node network, our GA shows the largest improvement, 21.43%, over the BIP, and the SA with sweep algorithm has the second best improvement, 17.31%, over the BIP. In the 75-node network, the GA has the largest improvement again with 20.60%, and the SA with sweep algorithm also has the second best energy improvement, 16.25%, over the BIP. And in the 100-node network, as we mentioned, the GA ran out of memory, and the SA with sweep algorithm shows the largest improvement with 15.34% over the BIP algorithm. Figure 5.4 also shows the data in Table 4.4 as a graph. It shows the percentage improvements of the various algorithms over the BIP algorithm.

Table 5.3 Energy Improvements of GA (%) over different algorithms

No. of nodes	BIP	BIP + Sweep	BIP + 1-Shrink	ACS	CM + 1-Shrink	SA	SA + Sweep
25	17.17 %	14.98 %	8.26 %	0 %	0 %	0 %	0 %
50	21.43 %	19.92 %	14.15 %	8.72 %	7.38 %	5.18 %	4.99 %
75	20.60 %	18.78 %	13.46 %		6.53 %	6.15 %	5.19 %

We now consider the improvements of our GA over the various algorithms introduced above. Table 5.3 shows the energy improvements of our GA over different algorithms. In the 25-node network, the GA made 17.17% improvement over the BIP, 14.98% improvement over the BIP with sweep algorithm, and no improvement over ACS, CM with 1-Shrink, SA and SA with sweep algorithm. In the 50-node network, the GA achieved 21.43% improvement over the BIP, and achieved the smallest improvement over the SA with sweep algorithm with 4.99%. In the 75-node network, the GA achieved the largest improvement with 20.60% over the BIP and the smallest improvement with 5.19% over the SA with sweep algorithm. Figure 5.5 show the data in Table 5.3 as a graph, which shows that the GA has the largest improvements over the BIP, and the smallest improvements over the SA with sweep algorithm.

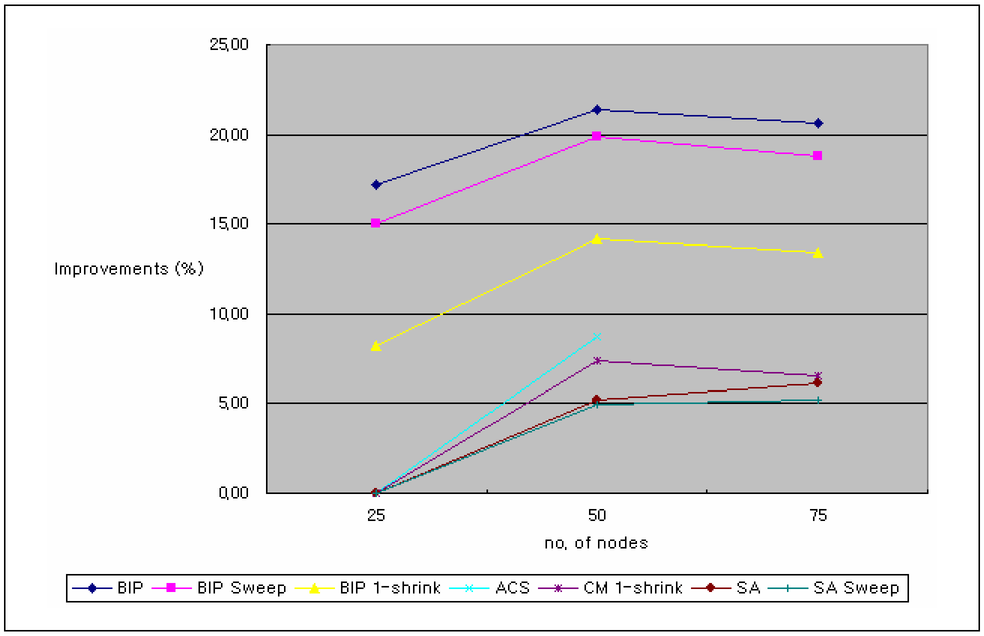


Figure 5.5 Energy Improvements of GA (%) over different algorithms

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, we represented a technique to build the minimum-energy broadcast trees using a genetic algorithm. The technique provides an efficient way to broadcast data to deployed sensors in wireless sensor networks reducing the energy consumption. The genetic algorithm we used here creates an initial population of trees using random generation, as well as two trees based on the BIP algorithm [1] and the MST algorithm [9] to create a search space wherein the near optimal solution to build the broadcast trees is found. We implemented the operations of a genetic algorithm, which are initialization, replacement, crossover, and mutation, to find more efficient broadcast trees.

We have evaluated and compared the trees resulting from the genetic algorithm with other trees resulting from techniques. The experimental result shows that the genetic algorithm improves up to 20.60% over the traditional algorithm and up to 5.16% over the latest proposed broadcast algorithm, SA.

We have identified several opportunities for future research. Since the proposed technique using the genetic algorithm is designed for broadcasting problem, as our future work, we plan to modify the method to apply to constructing multicasting tree. And we also try to advance our genetic algorithm to be available for the larger sizes of networks.

APPENDIX A

ENERGY CONSUMPTIONS OF THE NETWORKS AT EACH OF ITERATION

*** Energy consumption at each iteraton (50-node & Population Size(30))**

Iteration	Network #									
	1	2	3	4	5	6	7	8	9	10
0	**34900**	**34900**	**34900**	**34900**	**34900**	**34900**	**34900**	**34900**	**34900**	**34900**
1	34900	34900	**28600**	34900	34900	**33600**	34900	34900	34900	34900
2	34900	**34750**	28600	34900	34900	**31850**	34900	34900	34900	34900
4	34900	**34350**	28600	34900	**33850**	31850	34900	34900	34900	34900
6	34900	**32850**	28600	34900	33850	31850	34900	34900	34900	34900
8	34900	32850	28600	34900	33850	31850	34900	**30400**	34900	34900
9	34900	32850	28600	34900	33850	31850	**33800**	30400	34900	34900
12	34900	**32700**	28600	34900	33850	31850	33800	30400	**32850**	34900
13	34900	32700	28600	**32600**	33850	31850	33800	30400	32850	34900
14	34900	32700	28600	32600	**33800**	31850	33800	30400	32850	34900
17	**34850**	32700	28600	32600	33800	31850	33800	30400	32850	34900
19	34850	32700	28600	32600	33800	31850	33800	30400	32850	**33150**
22	34850	32700	28600	32600	33800	31850	**32100**	30400	32850	33150
24	34850	32700	28600	32600	33800	31850	**31150**	30400	32850	33150
25	**33650**	32700	28600	32600	33800	31850	31150	30400	32850	33150
27	33650	32700	28600	32600	33800	31850	31150	30400	**31450**	33150
30	33650	32700	28600	32600	33800	31850	31150	30400	**30950**	33150
42	**31950**	32700	28600	32600	**33300**	31850	31150	30400	30950	33150
50	31950	**31150**	28600	32600	33300	31850	31150	30400	30950	33150
64	**31550**	31150	28600	32600	33300	31850	31150	30400	30950	33150
72	31550	31150	28600	32600	**30350**	31850	31150	30400	30950	**31250**
74	31550	31150	28600	32600	**29500**	31850	31150	30400	30950	31250
76	31550	31150	28600	32600	29500	**31650**	31150	30400	30950	31250
100	31550	31150	28600	32600	29500	31650	31150	30400	30950	31250
140	31550	31150	28600	32600	**28450**	31650	31150	30400	30950	31250
149	31550	31150	28600	**31200**	28450	31650	31150	30400	30950	31250
156	31550	31150	28600	31200	28450	**31550**	31150	30400	30950	31250
161	**28950**	31150	28600	31200	28450	31550	31150	30400	30950	31250
165	28950	31150	28600	**30950**	28450	31550	31150	30400	30950	31250
171	28950	31150	28600	30950	28450	31550	31150	30400	30950	**30150**
172	28950	31150	28600	30950	28450	31550	**28850**	30400	30950	30150
200	28950	31150	28600	30950	28450	31550	28850	30400	30950	30150
228	28950	31150	28600	30950	28450	31550	28850	30400	30950	**28050**

Iteration	Network #									
	1	2	3	4	5	6	7	8	9	10
239	28950	31150	28600	30950	28450	**30500**	28850	30400	30950	28050
245	28950	**30700**	28600	30950	28450	30500	28850	30400	30950	28050
264	28950	30700	28600	30950	28450	30500	28850	30400	**30700**	28050
272	28950	30700	28600	30950	28450	30500	28850	**30300**	30700	28050
288	28950	30700	28600	30950	**28050**	30500	28850	30300	30700	28050
300	28950	30700	28600	30950	28050	30500	28850	30300	30700	28050
305	28950	30700	28600	30950	28050	30500	**28350**	30300	30700	28050
400	28950	30700	28600	30950	28050	30500	28350	30300	30700	28050
426	28950	30700	28600	30950	28050	30500	28350	**29350**	30700	28050
427	28950	30700	28600	30950	28050	30500	**26700**	29350	30700	28050
434	28950	30700	28600	30950	**27950**	30500	26700	29350	30700	28050
460	**28350**	30700	28600	30950	27950	30500	26700	29350	30700	28050
469	28350	30700	28600	30950	27950	30500	26700	29350	30700	28050
500	28350	30700	28600	30950	27950	30500	26700	29350	30700	28050
561	28350	30700	28600	30950	27950	30500	**25950**	29350	30700	28050
580	28350	30700	28600	30950	27950	**30100**	25950	29350	30700	28050
599	28350	30700	28600	**30200**	27950	30100	25950	29350	30700	28050
600	28350	30700	28600	30200	27950	30100	25950	29350	30700	28050
619	28350	30700	28600	30200	27950	30100	25950	**28050**	30700	28050
671	28350	30700	28600	**30100**	27950	30100	25950	28050	30700	28050
700	28350	30700	28600	30100	27950	30100	25950	28050	30700	28050
722	28350	**28850**	28600	30100	27950	30100	25950	28050	30700	28050
736	28350	28850	28600	30100	27950	30100	25950	28050	**30550**	28050
768	28350	28850	28600	**29650**	27950	30100	25950	28050	30550	28050
800	28350	28850	28600	29650	27950	30100	25950	28050	30550	28050
829	28350	28850	28600	29650	27950	30100	25950	28050	**30000**	28050
900	28350	28850	28600	29650	27950	30100	25950	28050	30000	28050
1,000	28350	28850	28600	29650	27950	30100	25950	28050	30000	28050
1,067	28350	**28500**	28600	29650	27950	30100	25950	28050	30000	28050
1,100	28350	28500	28600	29650	27950	30100	25950	28050	30000	28050
1,200	28350	28500	28600	29650	27950	30100	25950	28050	30000	28050
1,249	28350	28500	28600	29650	27950	30100	**24350**	28050	30000	28050
1,300	28350	28500	28600	29650	27950	30100	24350	28050	30000	28050
1,357	28350	**27950**	28600	29650	27950	30100	24350	28050	30000	28050
1,377	28350	27950	28600	29650	27950	**28600**	24350	28050	30000	28050

Iteration	Network #									
	1	2	3	4	5	6	7	8	9	10
1,400	28350	27950	28600	29650	27950	28600	24350	28050	30000	28050
1,500	28350	27950	28600	29650	27950	28600	24350	28050	30000	28050
1,518	28350	27950	28600	29650	27950	28600	24350	28050	**29950**	28050
1,521	28350	27950	28600	**28000**	27950	28600	24350	28050	29950	28050
1,540	28350	27950	28600	28000	27950	**27400**	24350	28050	29950	28050
1,571	28350	27950	28600	28000	27950	27400	24350	28050	**29850**	28050
1,577	**27950**	27950	28600	28000	27950	27400	24350	28050	29850	28050
1,592	27950	27950	28600	28000	27950	27400	24350	28050	**29500**	28050
1,600	27950	27950	28600	28000	27950	27400	24350	28050	29500	28050
1,626	**27600**	27950	28600	28000	27950	27400	24350	28050	29500	28050
1,700	27600	27950	28600	28000	27950	27400	24350	28050	29500	28050
1,744	27600	27950	28600	28000	27950	**25250**	24350	28050	29500	28050
1,800	27600	27950	28600	28000	27950	25250	24350	28050	29500	28050
1,871	**27350**	27950	28600	28000	27950	25250	24350	28050	29500	28050
1,900	27350	27950	28600	28000	27950	25250	24350	28050	29500	28050
1,921	27350	27950	28600	**26200**	27950	25250	24350	28050	29500	28050
1,940	27350	27950	28600	26200	27950	25250	24350	28050	**29350**	28050
1,964	27350	27950	28600	**25600**	27950	25250	24350	28050	29350	28050
2,000	27350	27950	28600	25600	27950	25250	24350	28050	29350	28050
2,100	27350	27950	28600	25600	27950	25250	24350	28050	29350	28050
2,200	27350	27950	28600	25600	27950	25250	24350	28050	29350	28050
2,300	27350	27950	28600	25600	27950	25250	24350	28050	29350	28050
2,371	**26350**	27950	28600	25600	27950	25250	24350	28050	29350	28050
2,378	26350	**27700**	28600	25600	27950	25250	24350	28050	29350	28050
2,400	26350	27700	28600	25600	27950	25250	24350	28050	29350	28050
2,500	26350	27700	28600	25600	27950	25250	24350	28050	29350	28050
2,600	26350	27700	28600	25600	27950	25250	24350	28050	29350	28050
2,700	26350	27700	28600	25600	27950	25250	24350	28050	29350	28050
2,800	26350	27700	28600	25600	27950	25250	24350	28050	29350	28050
2,839	26350	27700	28600	25600	**26200**	25250	24350	28050	29350	28050
2,900	26350	27700	28600	25600	26200	25250	24350	28050	29350	28050
2,908	26350	**26550**	28600	25600	26200	25250	24350	28050	29350	28050
2,986	**26000**	26550	28600	25600	26200	25250	24350	28050	29350	28050
3,000	26000	26550	28600	25600	26200	25250	24350	28050	29350	28050
3,007	26000	26550	28600	25600	26200	25250	24350	28050	**29250**	28050

Iteration	Network #									
	1	2	3	4	5	6	7	8	9	10
3,020	26000	26550	28600	25600	26200	25250	24350	28050	**28650**	28050
3,100	26000	26550	28600	25600	26200	25250	24350	28050	28650	28050
3,200	26000	26550	28600	25600	26200	25250	24350	28050	28650	28050
3,245	26000	26550	28600	25600	**25950**	25250	24350	28050	28650	28050
3,300	26000	26550	28600	25600	25950	25250	24350	28050	28650	28050
3,326	26000	26550	28600	25600	25950	25250	24350	28050	**28600**	28050
3,400	26000	26550	28600	25600	25950	25250	24350	28050	28600	28050
2,487	26000	26550	28600	25600	25950	25250	24350	28050	**27150**	28050
3,500	26000	26550	28600	25600	25950	25250	24350	28050	**26850**	28050
3,574	26000	26550	28600	25600	25950	25250	24350	28050	26850	**27450**
3,579	26000	26550	28600	25600	25950	25250	24350	28050	**26450**	27450
3,600	26000	26550	28600	25600	25950	25250	24350	28050	26450	27450
3,700	26000	26550	28600	25600	25950	25250	24350	28050	26450	27450
3,798	26000	**26000**	28600	25600	25950	25250	24350	28050	26450	27450
3,800	26000	26000	28600	25600	25950	25250	24350	28050	26450	27450
3,900	26000	26000	28600	25600	25950	25250	24350	28050	26450	27450
4,000	26000	26000	28600	25600	25950	25250	24350	28050	26450	27450
4,100	26000	26000	28600	25600	25950	25250	24350	28050	26450	27450
4,200	26000	26000	28600	25600	25950	25250	24350	28050	26450	27450
4,300	26000	26000	28600	25600	25950	25250	24350	28050	26450	27450
4,400	26000	26000	28600	25600	25950	25250	24350	28050	26450	27450
4,500	26000	26000	28600	25600	25950	25250	24350	28050	26450	27450
4,600	26000	26000	28600	25600	25950	25250	24350	28050	26450	27450
4,667	26000	26000	28600	25600	25950	25250	**24100**	28050	26450	27450
4,700	26000	26000	28600	25600	25950	25250	24100	28050	26450	27450
4,800	26000	26000	28600	25600	25950	25250	24100	28050	26450	27450
4,849	26000	26000	28600	25600	25950	25250	24100	28050	**26400**	27450
4,900	26000	26000	28600	25600	25950	25250	24100	28050	26400	27450
4,933	26000	26000	28600	25600	25950	25250	24100	28050	26400	**26950**
5,000	26000	26000	28600	25600	25950	25250	24100	28050	26400	26950
5,004	26000	26000	28600	25600	25950	25250	24100	28050	26400	26150
5,100	26000	26000	28600	25600	25950	25250	24100	28050	26400	26150
5,200	26000	26000	28600	25600	25950	25250	24100	28050	26400	26150
5,300	26000	26000	28600	25600	25950	25250	24100	28050	26400	26150
5,332	26000	26000	28600	25600	25950	25250	**22550**	28050	26400	26150

Iteration	Network #									
	1	2	3	4	5	6	7	8	9	10
5,400	26000	26000	28600	25600	25950	25250	22550	28050	26400	26150
5,500	26000	26000	28600	25600	25950	25250	22550	28050	26400	26150
5,600	26000	26000	28600	25600	25950	25250	22550	28050	26400	26150
5,700	26000	26000	28600	25600	25950	25250	22550	28050	26400	26150
5,800	26000	26000	28600	25600	25950	25250	22550	28050	26400	26150
5,900	26000	26000	28600	25600	25950	25250	22550	28050	26400	26150
6,000	26000	26000	28600	25600	25950	25250	22550	28050	26400	26150
6,100	26000	26000	28600	25600	25950	25250	22550	28050	26400	26150
6,188	26000	26000	28600	25600	25950	25250	22550	28050	**25650**	26150
6,200	26000	26000	28600	25600	25950	25250	22550	28050	25650	26150
6,300	26000	26000	28600	25600	25950	25250	22550	28050	25650	26150
6,303	26000	26000	28600	25600	25950	**25100**	22550	28050	25650	26150
6,400	26000	26000	28600	25600	25950	25100	22550	28050	25650	26150
6,500	26000	26000	28600	25600	25950	25100	22550	28050	25650	26150
6,600	26000	26000	28600	25600	25950	25100	22550	28050	25650	26150
6,700	26000	26000	28600	25600	25950	25100	22550	28050	25650	26150
6,800	26000	26000	28600	25600	25950	25100	22550	28050	25650	26150
6,844	26000	26000	28600	25600	**25250**	25100	22550	28050	25650	26150
6,900	26000	26000	28600	25600	25250	25100	22550	28050	25650	26150
7,000	26000	26000	28600	25600	25250	25100	22550	28050	25650	26150
7,100	26000	26000	28600	25600	25250	25100	22550	28050	25650	26150
7,200	26000	26000	28600	25600	25250	25100	22550	28050	25650	26150
7,300	26000	26000	28600	25600	25250	25100	22550	28050	25650	26150
7,326	26000	26000	**28100**	25600	25250	25100	22550	28050	25650	26150
7,400	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
7,500	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
7,600	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
7,700	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
7,800	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
7,900	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
8,000	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
8,100	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
8,200	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
8,300	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
8,400	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150

Iteration	Network #									
	1	2	3	4	5	6	7	8	9	10
8,500	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
8,600	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
8,700	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
8,800	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
8,900	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
9,000	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
9,100	26000	26000	28100	25600	25250	25100	22550	28050	25650	26150
9,154	26000	26000	28100	25600	25250	25100	22550	**26650**	25650	26150
9,200	26000	26000	28100	25600	25250	25100	22550	26650	25650	26150
9,277	26000	26000	**27850**	25600	25250	25100	22550	26650	25650	26150
9,300	26000	26000	27850	25600	25250	25100	22550	26650	25650	26150
9,400	26000	26000	27850	25600	25250	25100	22550	26650	25650	26150
9,419	26000	26000	**27450**	25600	25250	25100	22550	26650	25650	26150
9,500	26000	26000	27450	25600	25250	25100	22550	26650	25650	26150
9,537	26000	26000	**26900**	25600	25250	25100	22550	26650	25650	26150
9,600	26000	26000	26900	25600	25250	25100	22550	26650	25650	26150
9,700	26000	26000	26900	25600	25250	25100	22550	26650	25650	26150
9,800	26000	26000	26900	25600	25250	25100	22550	26650	25650	26150
9,900	26000	26000	26900	25600	25250	25100	22550	26650	25650	26150
9,904	26000	26000	26900	25600	25250	25100	22550	**26300**	25650	26150
10,000	26000	26000	26900	25600	25250	25100	22550	26300	25650	26150

REFERENCES

[1] J. E. Wieselthier, G. D. Nguyen and A. Ephremides, "On the Construction of energy-efficient broadcast and multicast trees in wireless networks", in Proc. of IEEE Infocom 2000and multicast trees in wireless networks", in Proc. of IEEE Infocom 2000.

[2] M. Cagalj, J-P. Hubaux, and C. Enz, “Minimum-Energy Broadcast in All-Wireless Networks: NP-Completeness and Distribution Issues”, in Proc. of the MOBICOM 2002 Conference, Atlanta, Georgia, USA, September 23 – 26, 2006

[3] G. R. Raidl, “An Efficient Evolutionary algorithm for the Degree-Constrained Minimum Spanning Tree Problem”, in Proc. of IEEE Progress in Evolutionary Computation, 2002.

[4] R. M. Ⅱ, A. Das, M. El-Sharkawi, P.Arabshashi, and A. Gray, “ Minimum Power Broadcast Trees for Wireless Networks: Optimizing using the viability lemma”, in Proc. of the IEEE international Symposium on Circuits and Systems, Scottsdale, AZ, May 26-29, 2002.

[5] A. K. Das, R. J. Marks, M. El-Sharakawi, P. Arabshashi, and A. Gray, “The Minimum Power Broadcast Problem in Wireless Networks: An Ant Colony System Apporach”, in Proc. of the IEEE international Symposium on Circuits and Systems, 2002.

[6] R. Montemanni, L. M. Gambardella, and A.K. Das, “The Minimum Power Broadcast Problem in Wireless Networks: a Simulated Annealing Approach”, in Proc. of IEEE Wireless Communications and Networking Conference, 2005.

[7] S. Banerjee, A. Misra, J. Yeo, and A. Agrawala, “Energy-Efficient Broadcast and Multicast Trees for Reliable Wireless Communication”, in Proc. of IEEE Wireless Communications and Networking (WCNC), March 16 – 20, 2003.

[8] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides, “Energy-Efficient Broadcast and Multicast trees in Wireless Networks”, in Proc. of Mobile Networks and Applications 7, pp. 481 – 492, Kluwer Academic Publishers, 2002.

[9] J. B. Kruskal. “On the shortest spanning subtree of a graph and the traveling salesman problem”, in Proc. of the American Mathematics Society, vol.7(1), 1956.

[10] T. H. Cormen, C. E. Leiseron, R. L. Rivest, and C. Stein, “Introduction to Algorithm: the 2nd edition”, The MIT Press ISBN 0262032937, September 1, 2001.

[11] W. Liang, “Constructing Minimum-Energy Boradcast Trees In Wireless Ad Hoc Networks”, in ACM MOBIHOC, EPFL Lausanne, Switzerland, June 9 – 11, 2002.

[12] I. Stojmenovic, M. Seddigh, and J. Zunic, “Internal nodes based broadcasting in wireless networks”, in Proc. of the HICCS-34 Conference, Island of Maui, HA, January 3-6, 2001.

[13] J. Cargigny, D. Simplot, and I. Stojmenovic, “Localized minimum energy broadcasting in ad-hoc networks”, in Proc. of the IEEE Infocom 2003 Conference, San Francisco, CA, March 30 – April 3, 2003.

[14] A. K. Das, R. J. Marks, El-Sharkawi, P. Arabshahi, and A. Gray, “A Cluster-merge algorithm for solving the minimum power broadcast problem in large scale wireless networks”, in Proc. of the Milcom 2003 Conference, Boston, MA, October 13 – 16, 2003.

[15] A. K. Das, R. J. Marks, M. El-Sharkawi, P. Arabshahi, and A. Gray, “r-Shrink: A heuristic for improving minimum power broadcast trees n wireless networks”, in

Proc. of the IEEE Globecom 2003 Conference, San Francisco, CA, December 1 – 5, 2003.

[16] A. Das, “Minimum Power broadcast and maximization of time-to-first-failure in broadcast wireless networks”, Ph.D. dissertation, University of Washington, May 2003.

[17] P. Piggott and F.Suraweera. “Encoding graphs for genetic algorithm: An investigation using the minimum spanning tree problem”. In X. Yao, ed., Progress in Evolutionary Computation, Springer, Berlin, 1995.

[18] S. Even. “Algorithmic Cominatorics”, the Macmillan Company, New York, 1973.

[19] J. R. Kim and M. Gen. “Genetic Algorithm for solving bicriteria network topology design problem”. In P.J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, A. Zalzala, and W. porto, eds., In Proc. of IEEE Congress on Evolutionary Computation, 1999.

[20] http://cs.felk.cvut.cz/~xobitko/ga/

[21] http://en.wikipedia.org/wiki/Genetic_algorithm

[22] G. R. Raidl and B. A. Julstrom, “Edge Sets: An Effective Evolutionary Coding of Spanning Trees”, in Proc. of IEEE Transactions on Evolutionary Computation, Vol.7, No.3, June 2003.

[23] A. E. Eiben, and J. E. Smith, “Introduction to Evolutionary Computing”, Springer, ISBN 3-540-40184-9, 2003.

[24] http://www.cs.bgu.ac.il/~assafza

[25] Dorigo M. and L. M. Gambardella, “Ant Colonies for the Traveling Salesman Problem”, BioSystems 43, pp. 73 – 81, 1997

[26] R. Prim. Shortest connection networks and some generalizations. Bell System Technical Journal, vol.36, pp. 1389 – 1401, 1957 .

BIOGRAPHICAL INFORMATION

Min Kyung An received her Bachelor degree in Computer Science and Statistics from Cheju National University, Cheju, Korea, in 2004. She began her graduate studies in the Dept. of Computer Science and Engineering at The University of Texas at Arlington in August 2005. Her major areas of interests are Database and Wireless Sensor Networks. She received her master degree in Computer Science and Engineering from The University of Texas at Arlington in August 2007.