3

Genetic Algorithm for Mobile Agent Routing in Distributed Sensor Networks*

3.1 Introduction

In the past decade, sensor networks have become an active area of research for computer scientists and network engineers due to their wide usage in both military and civilian applications [1]. The increasing sophistication of multi-sensor systems for state estimation, region surveillance as well as target detection and tracking has generated a great deal of interest in the development of new computational structures and networking paradigms [2].

A distributed sensor network (DSN) consists of intelligent sensors that are geographically dispersed in a region of interest and interconnected via a communication network. The sensed data of different types (such as acoustic, seismic, and infrared, etc.) is preprocessed by sensor nodes and then transmitted over the network to participate in the data integration at processing elements (PE), based on which appropriate inferences can be derived about the environment for certain purposes.

---

* Part of this text was taken from “On Computing the Route of a Mobile Agent for Data Fusion in a Distributed Sensor Network” by Q. Wu, S.S. Iyengar, N.S.V. Rao, J. Barhen, V.K. Vaishnavi, H. Qi, K. Chakrabarty, which is under consideration for publication in the IEEE Transactions on Knowledge and Data Engineering.
The study of information fusion methods have been the research focus since the early stage of DSN development [3–5]. Recent advances in sensor technologies make it possible to deploy a large number of inexpensive and small sensors to “achieve quality through quantity” in very complex scenarios, which necessitates applying new computing techniques such as genetic algorithm (GA) to meet some theoretical and methodological challenges. This chapter provides a general introduction to GA and its application to mobile agent routing problem (MARP) in DSN with a special networking paradigm.

DSNs are typically deployed for remote operations in large unstructured geographical areas, where wireless networks with low bandwidth are usually the only means of communication among the sensors. The communication consumes the limited power available at sensor nodes, and thus power consumption is to be restricted. Furthermore, the massively deployed sensors usually supply huge amount of data of various modalities, which makes it critical to collect only the information that is most desired and to collect it efficiently. Despite the abundance of deployed sensors, not all the information from these sensors need to be collected to ensure the quality of the fused information such as adequate detection energy for target detection or tracking.

Instead of sending all sensor data to the processing element, which performs a one-time data fusion as in a conventional server/client system, the mobile agent-based distributed sensor network (MADSN) proposed in Ref. [6] enables the computation to be spread out onto the participating leaf nodes with the aim of decreasing the consumption of scarce network resources (mostly the bandwidth) and the risk of being spied with hostile intent. In such a network scheme, a mobile agent carrying the executable instructions of data integration is dispatched from the processing element and selectively visits the leaf sensors along a certain path to fuse the data incrementally on a sequential basis.

The path quality of a mobile agent significantly affects the overall performance of MADSN implementation because the communication cost and detection accuracy depend on the order and the number of nodes to be visited. We formulate the MARP as a combinatorial optimization problem with certain constraints and construct an appropriate objective function that reflects the routing requirements. We show the MARP’s NP-hardness by reducing to it a variation of the 3D traveling salesman problem, which rules out any polynomial solutions. Therefore, we propose an approximate solution based on a two-level GA and compare the simulation results with those computed by two other deterministic heuristics, namely Local Closest First (LCF) and Global Closest First (GCF).

The rest of this chapter is organized as follows: In Section 3.2, we introduce a general computing technique based on GA. In Section 3.3, we describe the models for sensor nodes and wireless communication links and then formulate the MARP. The details of the solution using GA are given in Section 3.4, including the design of a two-level encoding scheme, derivation of the objective function, and implementations of genetic operators. Simulation results are presented and discussed in Section 3.5. Concluding remarks are provided in Section 3.6.

### 3.2 Computational Technique Based on Genetic Algorithm

#### 3.2.1 Introduction to Genetic Algorithm

GA is a computational model simulating the process of genetic selection and natural elimination in biologic evolution. Pioneering work in this field was conducted by Holland in 1960s [7,8]. GA was proposed to find global or local optima in the large search space. Comparing to traditional search algorithms in artificial intelligence, GA is able to automatically acquire and accumulate the necessary knowledge about the search space during its search process, and self-adaptively control the entire search process through random optimization techniques. A computational technique based on GA is particularly useful to avoid combinatorial explosion, which is always caused by disregarding the inherent knowledge within the enormous search space. In addition, the GA is characterized by its simplicity, flexibility, robustness, and adaptability to parallel process. It has found many successful applications in various areas solving combinatorial optimization problems and non-linear problems with complicated constraints or non-differentiable objective functions.
3.2.2 A General Method Using Genetic Algorithm

The computation of GA is an iterative process that simulates the process of genetic selection and natural elimination in biologic evolution. In each iteration cycle, good candidate solutions are retained and any unqualified solutions are screened out according to their corresponding fitness values. Genetic operators, such as crossover, mutation, translocation, and inversion, are then performed on those surviving solutions to produce a next generation of new candidate solutions. The above process is carried out repeatedly until certain convergence condition is met. To illustrate the principle of the algorithm, we take the classical knapsack problem as an example [9], which is formulated as follows:

Maximize: \[ \sum_{i=1}^{n} B_i X_i \]  
Constraint: \[ \sum_{i=1}^{n} S_i X_i \leq C \quad X_i \in \{0,1\}, \quad 1 \leq i \leq n \]  

where
- \( S_i \) represents the resource consumption for the \( i \)-th activity
- \( C \) represents the total available resources
- \( B_i \) represents the gained profit from the \( i \)-th activity
- \( X_i \) holds binary values: if \( i \)-th activity is carried out, \( X_i = 1 \); otherwise, \( X_i = 0 \)

The essence of the knapsack problem is to pursue the maximum profit with the constraint of limited total available resources. We now describe the standard steps taken by GA to find a solution to the knapsack problem:

1. Initialization: a set of \( M \) random solutions \( T_k (1 \leq k \leq M) \) are generated, where \( M \) is an appropriately selected initial population size
2. Genetic encoding: a string \( T \) of \( n \) binary bits is used to represent one possible solution. If the \( i \)-th activity is carried out, \( T(i) (1 \leq i \leq n) = 1 \); otherwise \( T(i) (1 \leq i \leq n) = 0 \)
3. Fitness value calculation: the objective function of the knapsack problem can be defined as

\[ J(T_k) = \sum_{i=1}^{n} T_k(i) B_i \]  
Subject to: \[ \sum_{i=1}^{n} T_k(i) S_i \leq C \quad 1 \leq k \leq M \]  

where \( C \) is a bounding constant. We construct a fitness function for the knapsack problem as follows to compute the fitness value for each individual solution:

\[ f(T_k) = J(T_k) + g(T_k), \quad 1 \leq k \leq M \]  

where \( g(T_k) \) is the penalty function when \( T_k \) violates the constraints, which may take the following form:

\[ g(T_k) = \begin{cases} 0 & C \geq \sum_{i=1}^{n} T_k(i) S_i \\ \beta E_m \left( C - \sum_{i=1}^{n} T_k(i) S_i \right) & C < \sum_{i=1}^{n} T_k(i) S_i \end{cases} \]  

where
- \( E_m \) is the maximum value of \( B_i/S_i (1 \leq i \leq n) \)
- \( \beta \) is a proper penalty coefficient
4. Survival probability calculation: the survival probability \( P_k \) for each individual (solution) \( T_k \) can be calculated based on the following fitness proportional model:

\[
P_k = \frac{f(T_k)}{\sum_{j=1}^{M} f(T_j)}
\]

(3.7)

A random selector is then designed to produce the hybridization individuals according to each \( P_k \).

5. New generation production: two hybridization individuals \( T_u \) and \( T_v \) are combined to create two individuals \( T_u' \) and \( T_v' \) of new generation by applying combinatorial rules of selection, crossover, mutation and inversion. This process continues until all \( M \) individual solutions of new generation are produced. There are several genetic operators involved in this procedure.

Crossover is an operation of segment exchange for two solutions. Given two parents (hybridization individuals) solutions with their crossover points represented by “/”:

\[
T_u = 01010/1011100 \\
T_v = 10100/1101010
\]

ea one-point crossover operator produces two children solutions:

\[
T_u' = 01010/1101010 \\
T_v' = 10100/1011100
\]

Inversion is to reverse the order of data in a solution segment. Given one parent solution with the inversion segment enclosed by a pair of “/”:

\[
T_u = 010/01101/1100
\]

an inversion operator produces the following child:

\[
T_u' = 010/01110/1100
\]

The mutation operator chooses one or more gene loci randomly in the individual string and changes their values (e.g., 0–1 reverses) with the preset mutation probability. Given one parent solution as follows:

\[
T_v = 010101011100
\]

two gene loci before mutation

a two-point mutation operator produces the following child:

\[
T_v' = 011101010100
\]

two gene loci after mutation

6. Repeat Step (2) to Step (5) until the predefined convergence condition is met, for example, the maximum generation number is reached or the solution quality is satisfied.
A general description of the above iterative process is given in C language as follows:

```c
main()
{
    int gen_no;
    initialize();
    generate(oldpop);
    for(gen_no=0;gen_no<maxgen;gen_no++)
    {
        evaluate(oldpop);
        newpop=select(oldpop);
        crossover(newpop);
        mutation(newpop);
        inversion(newpop);
        oldpop=newpop;
    }
}
```

The above pseudo code only depicts major steps of a GA. Some auxiliary functions are needed to implement a complete GA for a certain application. During the search process, GA does not require any outside knowledge except fitness values to select qualified solutions. Therefore, the design of fitness function has a significant impact on the overall algorithmic performance.

### 3.2.3 Parameters and Operators in Genetic Algorithm

#### 3.2.3.1 Population Size

Population size is a key parameter in GA. The Schema Theorem [10] establishes that given the population size $M$, the genetic operators are able to produce $M^3$ schemas, which ensures that the number of building blocks is dramatically increased when the search for the optimal solution progresses. Obviously, a GA with a larger population size is more likely to obtain the global optimum because a larger population size produces a wider variety of individuals and therefore the search process has higher probability to avoid being trapped into local optima. On the contrary, a small population size limits the search space and hence the premature convergence may occur under this circumstance, which may greatly impair the performance.

However, a large population size also brings some disadvantages. For instance, the computation complexity increases as a result and some good individuals with high fitness values may be eliminated during the selection operation.

#### 3.2.3.2 Crossover, Mutation and Inversion

Similar to the gene recombination, which plays an essential role during the natural biologic evolution process, the crossover in GA is the most critical operator in the genetic search strategy, which guides the main behavior of the optimization process. There are several commonly used crossover schemes such as one-point crossover, two-point crossover, and multi-point crossover. A good design of any crossover operator must ensure that the desirable gene segments of old individuals be properly inherited by the new individuals of new generation. A high crossover probability may improve GA's capability to explore new solution space, while increase the likelihood of disordering the combination of good gene segments. An inappropriately low crossover probability may cause the search process to be trapped in a dull status and be prone to ceasing.
The main purpose of a mutation operator is to maintain the variety of the population by preventing a single important gene segment from being corrupted. In practice a relatively small mutation probability, such as 0.001, is favorable because GA may tend to be a random search if too frequent mutation operations are conducted.

The inversion is actually a special form of mutation. It is designed to carry out reordering operation and improve the local search ability. Either crossover or mutation is not adequate for search in the local solution space: the search activities of the crossover operator span in the whole feasible solution space while the local search ability of the mutation operator is always suppressed by the genetic selection and natural elimination.

### 3.2.3.3 Encoding and Fitness Function

Since GA is unable to manipulate the parameters in the problem space directly, it is necessary to convert them to individuals made up of genes in the GA domain. This mapping from problem space to algorithm domain is called encoding. Actually, the robustness of GA reduces the reliance of performance on encoding schemes, as long as minimum three encoding criteria, such as completeness, soundness and non-redundancy, are satisfied [11].

In general, the control of the search process in GA does not need any information from outside but fitness values (or objective values). The objective function of a complex system usually has discontinuous or non-differentiable constraints. For a general optimization problem with complicated constraints, the penalty method is often used in the design of a fitness function. For example, an original minimization problem with constraints can be described as follows [10]:

$$
\text{Minimize: } F(x) \quad (3.8)
$$

With constraints: 

$$
b_l(x) \geq 0 \quad l = 1, 2, \ldots, p \quad (3.9)
$$

where 

- $F(x)$ is the objective function 
- $b_l(x)$ are a group of constraint functions

By applying the penalty method, we are able to convert the above problem to a non-constraint problem:

$$
\text{Minimize: } F(x) + \lambda \cdot \sum_{l=1}^{p} \Phi[b_l(x)] \quad (3.10)
$$

where 

- $\lambda$ is the penalty coefficient 
- $\Phi$ is the penalty function, which may take the form of Equation 3.6

### 3.2.3.4 Selection Mechanism

The selection operation, also referred to as reproduction operation, is to select good individuals and eliminate bad individuals from the population according to individual fitness values. A good selection mechanism is able to inherit good individuals directly from last generation or indirectly from the new individuals produced by mating the old individuals. The commonly used selection mechanisms include fitness proportional model, rank-based model, expected value model, and elitist model, etc.

### 3.3 Mobile Agent Routing Problem

We now briefly describe the architecture of a MADSN to motivate the later formulation of the optimization problem. A MADSN typically consists of three types of components: processing elements, sensor nodes, and communication network [12]. The various PE and sensors are usually interconnected via a wireless communication network. A group of neighboring sensor nodes that are commanded by the same PE forms a cluster.
3.3.1 Sensor Nodes

A sensor node, also referred to as a leaf node, is the basic functional unit for data collection in a MADSN. A sensor node may have several channels with different sensors connected to each of them. Sensor nodes are always geographically distributed to collect different types of measurements such as acoustic, seismic, and infrared from the environment. The data acquisition is controlled by a sampling subsystem, which provides the acquired data to the main system processor [13]. The signal energy from each channel can be detected individually and processed in the analog front end. A mobile agent migrates among the sensor nodes via the network, integrates local data with a desired resolution sequentially, and carries the final result to the originating PE. The fused data may be used to derive appropriate inferences about the environment for a certain civilian or military application.

We now provide object-oriented descriptions of sensor and PE nodes, which are used in our implementation. The sensor label is a unique ID of a sensor node, which corresponds to its static Internet Protocol (IP) address in the sensor network. We assume that a PE with label “0” remains active during the period of operation of MADSN. Some sensor nodes may be shut down or go to sleep due to intermittent faults or power considerations, and may be brought back up later if necessary. The sensor location is determined by its longitude and latitude obtained from the embedded Global Position System (GPS) module. The abstract sensor class, defined in C++ language, is listed in Appendix. Both the leaf node and processing element are derived from the abstract sensor class.

The signal energy, which is detected in real time at each local node and broadcast over the whole cluster, is an indicator of how close the node is to a potential target. In target detection and tracking applications, a leaf node with higher signal energy carries more information and should have higher priority of being visited. To simplify computation, we use a quantitative value to represent the level of signal energy detected by a local sensor node. The setup time spent by a PE accounts for loading the mobile agent code and performing other initialization tasks.

3.3.2 Communication Links

Wireless communication links need to be established between neighboring nodes as the mobile agent migrates along a route. The embedded RF modems of a sensor node provide such a networking capability with low power requirement. On the WINS NG 2.0 platform, each node is equipped with two RF modems, both of which support 2.4 GHz frequency-hopped spread spectrum (FHSS) communication [13]. The different clusters select different “network numbers,” which correspond to separate hopping pseudo-noise sequences to avoid interferences. The detailed radio configuration and wireless link establishment is beyond the scope of this chapter. We define an abstract link class with only the parameters we are interested in (see Appendix for details).

It is worthwhile to note that the message transmission time between two sensor nodes depends not only on the physical distance between them, but on the channel bandwidth and the data packet loss rate as well as the size of messages to be transmitted, which includes partially integrated data and mobile agent code itself. In general, the electromagnetic propagation time is almost negligible in short-range wireless communication. Hence the physical distance is not explicitly considered in our model but incorporated as a part of the path loss (PL) representing the signal attenuation. The received signal strength below a certain level due to PL may not be acceptable. The system loss factor is a parameter of the free space propagation model, which is not necessarily related to the physical propagation [14].

3.3.3 Mobile Agent Routing

A mobile agent is dispatched from the processing element and expected to visit a subset of sensors within the cluster to fuse data collected in the coverage area. Generally speaking, the more sensors visited, the higher will be the accuracy achieved using any reasonable data fusion algorithm [15]. It is important to
select an appropriate route so that the required signal energy level can be achieved with a low cost in terms of total energy consumption and PL.

A MADSN with a simple configuration is shown in Figure 3.1 for illustrative purposes. The sensor network contains one PE, labeled as $S_0$, and $N = 10$ leaf nodes, labeled as $S_i$, $i = 1, 2, \ldots, N$, one of which is down. The sensor nodes are spatially distributed in a surveillance region of interest, each of which is responsible for collecting measurements in the environment. The signal energy detected by sensor node $S_i$ is denoted by $e_i$, $i = 1, 2, \ldots, N$. Sensor node $S_i$ takes time $t_{i,\text{acq}}$ for data acquisition and time $t_{i,\text{proc}}$ for data processing. The wireless communication link with physical distance $d_{ij}$ between sensor node $S_i$ and $S_j$ has channel width $W$ bits and operates at frequency $B$ Hz. Some sensor nodes may be down temporarily due to intermittent failures, as sensor $S_9$ in Figure 3.1.

The routing objective is to find a path for a mobile agent that satisfies the desired detection accuracy, while minimizing the energy consumption and PL. The energy consumption depends on the processor operational power and computation time, and the PL is directly related to the physical length of the selected path. We define these quantities in the next section.

### 3.3.4 Objective Function

The objective function for the MARP is based on three aspects of a routing path: energy consumption, PL, and detected signal energy.

1. **Energy consumption**: The energy consumption at a sensor node is determined by the processing speed and the computation time. If an energy-driven Real-Time Operating System (RTOS) is installed on the sensor node, the processor speed can be dynamically scaled depending on workload and task deadlines [16]. For wireless message transmissions, the energy consumption depends on the sensor's transmission power and message transmission time. We assume that the message includes the mobile agent code of size $M$ bits and measured data of size $D$ bits. For a given desired resolution, a fixed data size $D$ is used to store the partially integrated data at each sensor. The time for the message to be transmitted over a wireless channel of bandwidth $BW$ bps is calculated as

$$t_{\text{msg}} = \frac{1}{B} \left[ \frac{M + D}{W} \right]$$

The energy consumption $EC$ of path $P$, consisting of nodes $P[0], P[1], \ldots, P[H-1]$, is defined as

$$EC(P) = a \cdot (t_{0,\text{setup}} + t_{0,\text{proc}}) \cdot F_0^2 + P_{0,i} \cdot t_{\text{msg}} + \sum_{k=1}^{H-1} \left\{ b \cdot \left[ t_{P[k],\text{acq}} + t_{P[k],\text{proc}} \cdot F_{P[k]}^2 \right] + P_{P[k],i} \cdot t_{\text{msg}} \right\}$$

(3.12)
Genetic Algorithm for Mobile Agent Routing in Distributed Sensor Networks

where \( k \)-th leaf node \( S_{P[k]} \) on path \( P \) has data acquisition time \( t_{P[k],\text{acq}} \), data processing time \( t_{P[k],\text{proc}} \), operational level \( F_{P[k],i} \), transmitting power \( P_{P[k],i} \), \( i = 1, 2, \ldots, N \), and node \( P[0] = 0 \) corresponds to the PE. Coefficients \( a \) and \( b \) are chosen to “normalize” the processor speed to its power level.

2. Path loss: The power received by sensor \( S_j \) has the following relation with the power transmitted by sensor \( S_i \) according to the Friis free space propagation model [14]:

\[
P_{j,r}(d_{ij}) = P_{j,t} \cdot \frac{G_{i,t}G_{j,r}\lambda^2}{(4\pi)^2 d_{ij}^2\beta},
\]

where

\( G_{i,t} \) is the gain of sensor \( S_i \) as a transmitter
\( G_{j,r} \) is the gain of sensor \( S_j \) as a receiver

Wavelength \( \lambda \) is the ratio of speed of light \( c \) and carrier frequency \( f \), and \( \beta \) is the system loss factor. The physical distance \( d_{ij} \) between \( S_i \) and \( S_j \) is computed from their spatial locations. PL represents the signal attenuation as a positive quantity measured in dB, and is defined as the difference (in dB) between the effective transmitted power and the received power:

\[
PL(d_{ij}) = 10\log_{10} \left( \frac{P_{j,t}}{P_{j,r}} \right) = 10\log_{10} \left( \frac{(4\pi)^2\beta}{G_{i,t}G_{j,r}\lambda^2} \cdot d_{ij}^2 \right)
\]

Therefore, the total PL along path \( P \) can be calculated as

\[
PL(P) = \sum_{k=0}^{H-1} \left[ 10\log_{10} \left( \frac{(4\pi)^2\beta}{G_{P[k],t}G_{P[(k+1)\text{mod}H],r}\lambda^2} \cdot d_{P[k],P[(k+1)\text{mod}H]}^2 \right) \right]
\]

3. Signal energy: An active sensor detects a certain amount of energy emitted by the potential target, which may or may not be used by a mobile agent for data integration. A mobile agent always tries to accumulate as much signal energy as possible for accurate decision in target classification or tracking application. The sum of the detected signal energy \( SE \) along path \( P \) is defined as

\[
SE(P) = \sum_{k=1}^{H-1} s_{P[k]}
\]

where \( s_{P[k]} \) is the signal energy detected by the \( k \)-th sensor node on path \( P \).

By combining the above three aspects of a routing path, we consider an objective function as follows:

\[
O(P) = SE(P) \left( \frac{1}{EC(P)} + \frac{1}{PL(P)} \right)
\]

wherein three terms \( SE(P) \), \( EC(P) \), and \( PL(P) \) are first normalized to appropriately reflect the contribution by various loss terms. This objective function prefers paths with higher signal energies by penalizing those with high path losses and energy consumption. A path providing high signal energy at the expense of a considerable amount of energy consumption and PL may not be preferable. Alternative objective functions may be used as long as they correctly reflect the tradeoff between detected signal energy, energy consumption, and PL.
To facilitate the GA algorithm in Section 3.4, we define a fitness function based on the objective function as follows:

$$f(P) = O(P) + g$$  \hspace{1cm} (3.18)

where $g$ is the penalty function for overrunning the constraint defined by

$$g = \begin{cases} 
0, & SE(P) \geq E \\
\delta \cdot (SE(P) - E) / E, & SE(P) < E 
\end{cases}$$  \hspace{1cm} (3.19)

where

- $E$ is the desired detection accuracy or signal energy level
- $\delta$ is a properly selected penalty coefficient

### 3.3.5 NP-Hardness of Mobile Agent Routing Problem

The MARP is to compute a path $P$ in a MADSN such that $O(P) > \tau$ and $k$-hop mobile agent routing problem (k-MARP) additionally requires that the path $P$ have exactly $k$ edges. We now show the latter to be NP-hard by reducing the 3D Maximum Traveling Salesman Problem (MTSP) to it, which is an indication of the intractability of MARP. We first present the definition of MTSP. Given a completely connected graph $G = (V, E)$, and a nonnegative real number $\alpha$, does there exist a closed-loop path $P$, with nodes $P[0], P[1], \ldots, P[n - 1], P[n] = P[0]$, such that $\sum_{i=0}^{n-1} l_i(P[i], P[(i + 1) \mod n]) \geq \alpha$? Here, each vertex corresponds to a point in three-dimensional Euclidean space $\mathbb{R}^3$. The starting point $v_{P[0]}$ and ending point $v_{P[n]}$ in the space refer to the same vertex in the graph.

The quantity $l_i(P[i], P[i + 1]) = \sqrt{(x_{P[i]} - x_{P[i + 1]})^2 + (y_{P[i]} - y_{P[i + 1]})^2 + (z_{P[i]} - z_{P[i + 1]})^2}$ is the Euclidean distance between two adjacent vertices $P[i]$ and $P[i + 1]$ on path $P$. The MTSP under Euclidean distances in $\mathbb{R}^d$ for any fixed $d \geq 3$ is proved to be NP-hard in Ref. [17]. The conventional traveling salesmen problem requires that path length be minimized and the cities are defined for dimension $d = 2$. On the contrary, MTSP requires maximization of path length and is known to be intractable in three or higher dimensions. Note that k-MARP requires maximization of $O(P)$ but is defined for $d = 2$, which makes a direct reduction from MTSP non-trivial.

Given an instance of MTSP, we generate an instance of k-MARP as follows. We create a graph for the k-MARP with $k = n$ using only $x$ and $y$ coordinates of vertices of MTSP (without loss of generality we assume that all coordinate values are distinct). We consider the objective $O(P) = (SE(P) / PL(P))$ by ignoring the energy consumption component. Recall that the PL is given by

$$PL(P) = \sum_{i=0}^{k-1} 10 \cdot \log \left( A \cdot d_{P[i], P[(i + 1) \mod k]}^2 \right)$$  \hspace{1cm} (3.20)

Let $e_{P[i]}$ represent the edge between vertices $P[i]$ and $P[i + 1]$ on path $P$. We define $s_{P[i]} = s(e_{P[i]}) = l_i(P[i], P[i + 1]) + \tau \cdot (10 \cdot \log (A \cdot d_{P[i], P[(i + 1) \mod k]}^2)) - \frac{\alpha}{k}$. A solution to the k-MARP is a path $P$ with $k$ hops such that $O(P) = \left( \sum_{i=0}^{k-1} s_{P[i]} / \sum_{i=0}^{k-1} d(P[i], P[(i + 1) \mod k]) \right) \geq \tau$, $\tau$ is a given nonnegative real number. After reorganizing, the objective function can be rewritten as

$$\sum_{i=0}^{k-1} (s(e_{P[i]}) - \tau \cdot d(e_{P[i]})) = \sum_{i=0}^{k-1} \left( l_i(P[i], P[(i + 1) \mod k]) - \frac{\alpha}{k} \right) \geq 0$$  \hspace{1cm} (3.21)

which guarantees the condition necessary for a solution to the corresponding MTSP.
On the other hand, if there exists a solution to the MTSP, that is, a closed-loop path $P$ consisting of $n$ edges such that $\sum_{i=0}^{n-1} l_i(P[i], P[(i+1) \mod n]) \geq \alpha$, this path can be used to solve the corresponding n-MARP such that $O(P) \geq \tau$. Note that above reduction from MTSP to n-MARP is polynomial-time computable, and hence NP-hardness of the latter follows from that of the former.

The restriction of n-MARP is studied in Ref. [6], where two heuristics LCF and GCF are proposed. In the next section we propose a GA based method for MARP and show that it outperforms LCF and GCF.

### 3.4 Genetic Algorithm for Mobile Agent Routing Problem

#### 3.4.1 Two-Level Genetic Encoding

We design a two-level encoding scheme to adapt the GA to the MARP in MADSN. The first level is a numerical encoding of the sensor (ID) label sequence $L$ in the order of sensor nodes being visited by mobile agent. For the MADSN shown in Figure 3.1, the sensor label sequence $L$ has the following contents:

\[
0 \ 1 \ 2 \ 3 \ 7 \ 5 \ 6 \ 8 \ 4 \ 10 \ 9
\]

The first element is always set to be “0” for reason that a mobile agent starts from the PE $S_0$. The mobile agent returns to $S_0$ from the last visited sensor node, which is not necessarily the last element of the label sequence if there are any inactive sensor nodes in the network. This sequence consists of a complete set of sensor labels because it takes part in the production of a new generation of solutions through the genetic operations. It is desired to inherit as much information as possible in the new generation from the old one. For example in Figure 3.1, although node 3, 6, 8, and 9 are not visited in the given solution (the second level sequence is designed to do so), they or some of them may likely make up a segment of a better solution than the current one in the new generation.

The second level is a binary encoding of the visit status sequence $V$ in the same visiting order. For the MADSN in Figure 3.1, the visit status sequence $V$ contains the following binary codes:

\[
1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0
\]

where
- “1” indicates “visited”
- “0” indicates “unvisited”

The first bit corresponds to the PE and is always set to be “1” because the PE is the starting point of the itinerary. If a sensor is inactive, its corresponding bit remains “0” until it is reactivated and visited.

Masking the first level of numerical sensor label sequence $L$ with the second level of binary visit status sequence $V$ yields a candidate path $P$ for mobile agent. In the above example, the path $P$ is obtained as

\[
0 \ 1 \ 2 \ 3 \ 7 \ 5 \ 6 \ 8 \ 4 \ 10 \ 9
\]

These two levels of sequences are arranged in the same visiting order for the purpose of convenient manipulations of visited/unvisited and active/inactive statuses in the implementation of GA. The number of hops $H$ in a path $P$ can be easily calculated from the second level of binary sequence as follows:

\[
H = \sum_{i=0}^{N} V[i], \begin{cases} 
V[i] = 1, & \text{sensor } S_i \text{ is active and visited} \\
V[i] = 0, & \text{sensor } S_i \text{ is inactive or unvisited}
\end{cases}
\]

#### 3.4.2 Implementations of Genetic Operators

We now describe the genetic operators. These operators are similar to those used in the conventional solution to Traveling Salesman Problem. However, we adapt the details to the current routing problem.
3.4.2.1 Selection Operator

As discussed above, the purpose of the selection operation is to select good individuals and at the same time eliminate bad individuals from the population based on the evaluation of individual fitness. In our implementation, each pair of individuals is selected randomly from the old generation to perform the crossover, mutation, and inversion operations. The fitness is computed for every newly generated child for evaluation. To maintain the same population sizes for each generation, the fitness of every newly generated child is compared with the minimum fitness of the whole population. If it is bigger than the minimum fitness value, then this child is added to the population and the individual with the minimum fitness is removed; otherwise, the new child is discarded.

3.4.2.2 Crossover Operator

We design a two-point crossover operator in our implementation for both levels of sequences. These two crossover points are selected randomly. Given two parents as follows:

Parent 1: First level sequence: 0-2-7-3-/-5-1-6-/-4-9-8 Second level sequence: 1-0-1-1-/-1-0-0-/-1-1-1
Parent 2: First level sequence: 0-3-5-2-/-9-6-4-/-1-7-8 Second level sequence: 1-0-0-0-/-1-0-1-/-1-0-1

where “/” represents the crossover points, the crossover operator produces two children:

Child 1: First level sequence: 0-9-6-4-2-7-3-5-1-8 Second level sequence: 1-0-1-1-/-1-0-1-/-1-1-1
Child 2: First level sequence: 0-5-1-6-3-2-9-4-7-8 Second level sequence: 1-0-0-0-/-1-0-0-/-1-0-1

For the first level of label sequence, the crossover portion (between the two crossover points) of one individual is copied and inserted at the front of the other individual (immediately after label 0). All the duplicate genes in the resulting individual are knocked out to guarantee that each node appears exactly once in that individual. For the second level of visit status binary sequence, the crossover portions are simply exchanged between two individuals.

3.4.2.3 Mutation Operator

We implement a two-point mutation operator that randomly selects two points and exchanges the values of these two points in both strings. As an example, consider the following parent individual:

First level sequence: 0-2-9-3-7-1-4-5-8-6
Second level sequence: 1-0-1-1-0-1-0-0-1

↑↑
two selected gene loci

The mutation operator produces the following child:

First level sequence: 0-2-9-3-8-1-4-5-7-6
Second level sequence: 1-0-1-1-1-0-0-0-1

3.4.2.4 Inversion Operator

We implement the inversion operator as follows. At a time, two inversion points are selected randomly to determine the inversion portion of the individual. The inversion operation is executed by reversing the order of the inversion portion of the original individual. Given one parent as follows:

First level sequence: 0-5-7-/-1-2-8-9-/-6-3-4
Second level sequence: 1-0-1-/-1-0-1-/-0-0-1

where the inversion portions are enclosed by two “/” signs, the inversion operator produces the following child:

First level sequence: 0-5-7-/-9-8-2-1-/-6-3-4
Second level sequence: 1-0-1-/-1-0-1-/-0-0-1
3.4.3 Parameter Selection for Genetic Algorithm

We usually select a high probability value above 0.9 for a genetic operator like crossover, which controls the main direction of evolution process. A low probability value below 0.1 is appropriate for genetic operators like mutation or inversion to reduce the risk of destroying the good gene segments in later generations. Observed from experimental data, small variation of these probabilities does not have significant impact on the performance of GA. With respect to the maximum generation number, we select different values for different test examples in order to ensure that the optimization process approaches a steady state eventually. The difference of the best fitness values between two adjacent generations may be used as an alternative convergent indicator. In this case, the GA does not have to wait for a long time to reach the pre-specified maximum generation number if the optimization process converges quickly. Its disadvantage is that the program may prematurely terminate if the optimization process does not converge quickly.

3.5 Simulation Results and Algorithm Analysis

3.5.1 Simulation Results

We compare the search results of the GA with those computed by LCF and GCF. In most of cases, LCF is able to deliver satisfying route for mobile agent, hence it is a comparable algorithm with GA. GCF may find a path with less number of hops than LCF, but it usually has significantly longer path length resulting in unacceptable PL. A series of experimental networks of different sensor node sizes and distribution patterns are created to conduct the optimal routing. The spatial locations of all the nodes are randomly selected. The LCF and GCF algorithm pick up the center node as the starting point in each network. About 1%–10% of the sensors are shut down uniformly over the surveillance region. All sensor parameters of data acquisition and wireless channel in the MADSN use the real-life data of the field demo listed in Table 3.1.

In order to make a visual comparison, the search results computed by GA, LCF, and GCF for the first relatively small sensor network are shown in Figures 3.2 through 3.4, respectively. This sensor network consists of 200 nodes, 8 of which are in the sleep state. A quantified amount of signal energy associated with each active sensor ranging from 0 to 64 is displayed under the corresponding sensor node. The minimum acceptable amount of signal energy detected by an individual sensor node is 5, and inactive nodes do not detect any signal energy. There is one potential target located in the region. The sensor nodes in the vicinity of the targets detect higher signal energy than other nodes. The total detected signal energy is 1467 units, and the acceptable signal level for correct inference is set to be 1200 units.

<table>
<thead>
<tr>
<th>TABLE 3.1 Parameters of the MADSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor Node Processor Type</td>
</tr>
<tr>
<td>Sensor node processor speed</td>
</tr>
<tr>
<td>Mobile agent sizes</td>
</tr>
<tr>
<td>Ave. data sizes</td>
</tr>
<tr>
<td>Carrier frequency band</td>
</tr>
<tr>
<td>Transmitting power</td>
</tr>
<tr>
<td>Transmitter gain</td>
</tr>
<tr>
<td>Receiving power</td>
</tr>
<tr>
<td>Receiver gain</td>
</tr>
<tr>
<td>Channel operation frequency</td>
</tr>
<tr>
<td>Channel width</td>
</tr>
<tr>
<td>Data sampling rate</td>
</tr>
<tr>
<td>Sample data format</td>
</tr>
</tbody>
</table>
FIGURE 3.2 Visualization of the search result computed by GA for an MADSN with 200 nodes.

FIGURE 3.3 Visualization of the search result computed by LCF for an MADSN with 200 nodes.
The maximum generation number 200 is specified for GA as the convergent indicator, which informs the program when to stop searching process.

It has been observed that the optimization process of GA moves forward rapidly in the beginning, and becomes slow and stable in the later stages of computation, especially after the generation number reaches 100.

Table 3.2 shows that GA uses 140 hops to achieve the acceptable signal level, while LCF uses 168 hops and GCF uses 147 hops, respectively. The path losses of GA, LCF and GCF are 22,919 units, 26,864 units, and 28,748 units, respectively. The migration of mobile agent along the path computed by GA consumes 407 units of energy, while LCF consumes 489 units of energy, and GCF consumes 428 units of energy. The simulation results of other sensor networks with larger node sizes and randomized distribution patterns are also tabulated in Table 3.2.

In Table 3.2, the total detected signal energy represents the maximum energy detected by all active sensors deployed in the region under surveillance. The acceptable signal energy level is a given value desired for a specific application.

Figures 3.5 through 3.8 illustrate the corresponding curves of node sizes vs. number of hops, node sizes vs. PL, node sizes vs. energy consumption, and node sizes vs. objective values, respectively. From Table 3.2 and Figures 3.5 through 3.7, it has been seen that in most of cases GA is able to find a satisfying path with less number of hops, less energy consumption, and less PL than LCF and GCF algorithm. Figure 3.8 clearly shows that the GA has a superior overall performance over two other heuristics in terms of the objective function defined in this implementation.

The current GA program was implemented in C++ using MFC with GUI and per-generation result display. The code takes a few seconds to run the first 100 generations for a network of hundreds of nodes. GA runs much faster and its executable code size decreases significantly when GUI is not implemented. In such cases, the GA run time may not be a serious problem for semi-dynamic routing, which will be discussed in the next subsection.
### TABLE 3.2 Comparisons of Search Results of GA, LCF, and GCF for Networks of Different Node Sizes and Distribution Patterns

<table>
<thead>
<tr>
<th>Case #</th>
<th>Node Sizes</th>
<th># of Dead Sensors</th>
<th># of Potential Targets</th>
<th>Total Detected Signal Energy</th>
<th>Acceptable Signal Energy Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>8</td>
<td>1</td>
<td>1,467</td>
<td>1,200</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
<td>4</td>
<td>2</td>
<td>2,985</td>
<td>2,750</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>5</td>
<td>2</td>
<td>4,000</td>
<td>3,680</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>8</td>
<td>3</td>
<td>4,080</td>
<td>3,710</td>
</tr>
<tr>
<td>5</td>
<td>600</td>
<td>9</td>
<td>4</td>
<td>4,980</td>
<td>4,800</td>
</tr>
<tr>
<td>6</td>
<td>700</td>
<td>10</td>
<td>5</td>
<td>5,340</td>
<td>5,190</td>
</tr>
<tr>
<td>7</td>
<td>800</td>
<td>11</td>
<td>4</td>
<td>6,200</td>
<td>5,950</td>
</tr>
<tr>
<td>8</td>
<td>900</td>
<td>13</td>
<td>3</td>
<td>7,000</td>
<td>6,380</td>
</tr>
<tr>
<td>9</td>
<td>1000</td>
<td>14</td>
<td>5</td>
<td>7,525</td>
<td>7,000</td>
</tr>
<tr>
<td>10</td>
<td>1100</td>
<td>13</td>
<td>4</td>
<td>8,515</td>
<td>7,990</td>
</tr>
<tr>
<td>11</td>
<td>1200</td>
<td>18</td>
<td>5</td>
<td>10,050</td>
<td>9,425</td>
</tr>
<tr>
<td>12</td>
<td>1300</td>
<td>16</td>
<td>4</td>
<td>11,410</td>
<td>9,600</td>
</tr>
<tr>
<td>13</td>
<td>1400</td>
<td>15</td>
<td>4</td>
<td>11,500</td>
<td>9,800</td>
</tr>
<tr>
<td>14</td>
<td>1500</td>
<td>19</td>
<td>5</td>
<td>12,380</td>
<td>11,000</td>
</tr>
<tr>
<td>15</td>
<td>1600</td>
<td>25</td>
<td>6</td>
<td>13,505</td>
<td>12,210</td>
</tr>
</tbody>
</table>

#### GA

<table>
<thead>
<tr>
<th>Case #</th>
<th>No. Hops</th>
<th>Path Loss</th>
<th>Energy Consumpt</th>
<th>Achieved Signal Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140</td>
<td>22,919</td>
<td>407</td>
<td>1,202</td>
</tr>
<tr>
<td>2</td>
<td>216</td>
<td>34,154</td>
<td>670</td>
<td>2,785</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>47,100</td>
<td>875</td>
<td>3,691</td>
</tr>
<tr>
<td>4</td>
<td>424</td>
<td>64,302</td>
<td>1237</td>
<td>3,716</td>
</tr>
<tr>
<td>5</td>
<td>483</td>
<td>76,205</td>
<td>1389</td>
<td>4,815</td>
</tr>
<tr>
<td>6</td>
<td>544</td>
<td>88,344</td>
<td>1552</td>
<td>5,195</td>
</tr>
</tbody>
</table>

#### LCF

<table>
<thead>
<tr>
<th>Case #</th>
<th>No. Hops</th>
<th>Path Loss</th>
<th>Energy Consumpt</th>
<th>Achieved Signal Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>168</td>
<td>26,864</td>
<td>489</td>
<td>1,215</td>
</tr>
<tr>
<td>2</td>
<td>273</td>
<td>42,027</td>
<td>796</td>
<td>2,757</td>
</tr>
<tr>
<td>3</td>
<td>379</td>
<td>57,842</td>
<td>1106</td>
<td>3,684</td>
</tr>
<tr>
<td>4</td>
<td>460</td>
<td>68,645</td>
<td>1342</td>
<td>3,721</td>
</tr>
<tr>
<td>5</td>
<td>561</td>
<td>82,627</td>
<td>1637</td>
<td>4,807</td>
</tr>
<tr>
<td>6</td>
<td>663</td>
<td>96,370</td>
<td>1935</td>
<td>5,197</td>
</tr>
</tbody>
</table>

#### GCF

<table>
<thead>
<tr>
<th>Case #</th>
<th>No. Hops</th>
<th>Path Loss</th>
<th>Energy Consumpt</th>
<th>Achieved Signal Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>147</td>
<td>28,748</td>
<td>428</td>
<td>1,204</td>
</tr>
<tr>
<td>2</td>
<td>228</td>
<td>43,191</td>
<td>664</td>
<td>2,763</td>
</tr>
<tr>
<td>3</td>
<td>291</td>
<td>55,897</td>
<td>849</td>
<td>3,688</td>
</tr>
<tr>
<td>4</td>
<td>424</td>
<td>82,817</td>
<td>1237</td>
<td>3,714</td>
</tr>
<tr>
<td>5</td>
<td>490</td>
<td>95,417</td>
<td>1430</td>
<td>4,819</td>
</tr>
<tr>
<td>6</td>
<td>566</td>
<td>110,972</td>
<td>1652</td>
<td>5,194</td>
</tr>
</tbody>
</table>
Genetic Algorithm for Mobile Agent Routing in Distributed Sensor Networks

<table>
<thead>
<tr>
<th>Case #</th>
<th>GCF Objective Value</th>
<th>LCF Objective Value</th>
<th>OCF Objective Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.017345</td>
<td>2.854905</td>
<td>2.854905</td>
</tr>
<tr>
<td>2</td>
<td>3.086556</td>
<td>2.885079</td>
<td>2.939742</td>
</tr>
<tr>
<td>3</td>
<td>3.111365</td>
<td>2.709512</td>
<td>2.879914</td>
</tr>
<tr>
<td>4</td>
<td>3.093068</td>
<td>2.745518</td>
<td>2.914104</td>
</tr>
<tr>
<td>5</td>
<td>3.260774</td>
<td>2.783227</td>
<td>2.844873</td>
</tr>
</tbody>
</table>

**Objective Function:**

- **Case #**
- **GCF Objective Value** $O(P) = SE(P)(1/E(P) + 1/L(P))$
- **LCF Objective Value** $O(P) = SE(P)(1/E(P) + 1/L(P))$
- **OCF Objective Value** $O(P) = SE(P)(1/E(P) + 1/L(P))$
3.5.2 Algorithm Comparison and Discussion

GCF algorithm is relatively simple and fast but suffers from poor performance in terms of PL. GCF algorithm essentially utilizes sorting to compute the path. Its computational complexity is $O(N \log N)$ if using a comparison based sorting algorithm. LCF algorithm has the computation complexity of $O(N^2)$ if the closest neighbor node is obtained by simple comparison in each step. The analysis for
the computation complexity of the GA is more complicated. After making some simplifications on its implementation, the computation complexity for the GA is $O(NMG)$, where $N$ is the number of nodes in the network, $M$ is the initial population size, and $G$ is the maximum generation number used to indicate the end of the computation.

As is the case for GCF algorithm, the performance of LCF algorithm also depends significantly on the network structure [6]. In some bad cases, it may end up with unacceptable results. Comparatively, the network structure has much less influence on the performance of the GA due to its random search technique.

Unlike the LCF and GCF algorithms, it is not necessary to specify a starting node for the implementation of GA. Actually, any active sensor node can be designated as a starting node in the GA, while the performance of LCF and GCF algorithms are crucially dependent on the location of its starting node. In addition, no matter in what order the nodes are visited, GA always comes up with a closed route for the mobile agent to come back to its starting node.

Mobile agent routing algorithm can be classified as dynamic and static routing according to the place where routing decisions are taken. A dynamic method determines the route locally on the fly at each hop of the migration of a mobile agent among sensor nodes. A static method uses centralized routing, which computes the route at PE node in advance of mobile agent migration. For different sensor network applications, either dynamic or static method can be applied. For example, it might be sufficient to use a static routing for target classification, but target tracking may require a dynamic routing due to its real-time constraint.

LCF algorithm is suitable for carrying out dynamic routing because each step of its computation depends only on the location of the current node, while the GCF is in favor of static routing, whose computation can be carried out offline based on the global network structure. Both LCF and GCF are deterministic routing, which always supply the same path between a source/destination pair in a given network. The GA collects information about the network status from all sensor nodes so that it is able to conduct adaptive routing. However, broadcasting the detected signal energy produces extra communication overhead.

Since it is desired to keep the mobile agent code as compact as possible, the GA may be used to implement a semi-dynamic routing. In this routing scheme, the routing code does not go with the mobile. If the network system is notified of some events (e.g., some nodes are shut down or activated, or do not have enough remaining energy to transmit the signal along the previously designated link), which causes the previously computed route to be invalid, the routing code is rerun based on the updated system information, and the new route is transmitted to the mobile agent for its further migration.

This semi-dynamic routing scheme is supported by the robustness of sensor network. According to the experience from the field demo, sensor nodes usually function well once brought up, and the network may remain stable continuously for 1–2 h of sessions.
3.6 Conclusions

We presented a mobile agent-based paradigm for data fusion in DSNs. By utilizing a simplified analytical model for the DSN, we formulated a route computation problem for the mobile agent in terms of maximizing the received signal strength while keeping PL and energy consumption low. This route computation problem turned out to be NP-hard thereby making it highly unlikely to develop a polynomial-time algorithm to compute an optimal route. Hence we proposed a GA to solve this problem by employing a two-level genetic encoding and suitable genetic operators. Simulation results are presented for comparison between our GA, and existing LCF, and GCF heuristics. Various aspects of the proposed algorithm such as computational complexity, impact of network structure and starting node, dynamic and static routing, are discussed.

Future research work is to be focused on exploring more complex routing models with more general objective function. For example, in the current model we assume that the sensor locations are fixed once they are manually deployed, which is the case in the field demo. However, in a real world sensor network, sensors could be airborne or installed on vehicles or robots. The mobility of sensors brings new challenges to the design of dynamic routing algorithm for the mobile agent. Besides, instead of using the simple free space propagation model to compute PL, more complex empirical propagation models may be studied and applied in the construction of objective function.

3.A Appendix

Class definitions of abstract sensor, leaf node, processing element, and wireless link used in the algorithm implementations are listed as follows:

```cpp
class CSensor:
{
    unsigned int m_sensorLabel; // a unique sensor ID: 0 for PE, else for leaf nodes
    bool m_sensorStatus; // TRUE: active; FALSE: inactive
    double m_processorSpeed;
    double m_locationLongitude;
    double m_locationLatitude;
    double m_dataProcessingTime;
    double m_transmittedPower;
    double m_transmitterGain;
    double m_receivedPower;
    double m_receiverGain;
}
```

```cpp
class CLeafNode: public CSensor
{
    bool m_visited;
    double m_dataAcquisitionTime;
    double m_detectedSignalEnergy;
    double m_dataSamplingRate;
    double m_sampleDataFormat;
}
```

```cpp
class CProcessingElement: public CSensor
{
    double m_setupTime;
}
```

3.A.1 Definition of Sensor Class

```cpp
class CLeafNode: public CSensor
{
    bool m_visited;
    double m_dataAcquisitionTime;
    double m_detectedSignalEnergy;
    double m_dataSamplingRate;
    double m_sampleDataFormat;
}
```

```cpp
class CProcessingElement: public CSensor
{
    double m_setupTime;
}
3.A.2 Leaf Node Class Derived from CSensor and Processing Element Class Derived from CSensor

class CLink:
{
    CSensor* m_pSensorTransmitter;
    CSensor* m_pSensorReceiver;
    double m_linkDistance;
    double m_bandWidth;
    double m_channelWidth;
    double m_operateFrequency;
    double m_carrierFrequency;
    double m_linkPropagationTime;
    double m_msgTransmissionTime;
    double m_linkPowerLoss;
    double m_systemLossFactor;
}

3.A.3 Definition of Wireless Link Class

Most of the attributes defined in these classes are self-explanatory. Leaf node and processing element classes are in turn derived from the abstract sensor class CSensor. In the definition of leaf node class, the attribute `m_dataSamplingRate` represents the frequency at which the signal data is sampled. The amount of memory space used to store the sampled data is determined by the attribute `m_sampleDataFormat`. In the definition of wireless link class, `m_systemLossFactor` is a parameter of the free space propagation model, which is not necessarily related to the physical propagation.

References

17. A. Barvinok, S.P. Fekete, D.S. Johnson et al., The geometric maximum traveling salesman problem, combined journal version of previous paper and Fekete’s 1999 SODA paper on the maximum TSP, including a faster algorithm for arbitrary polyhedral metrics. Submitted to *JACM*, 2002.