Chapter 1
Autonomic and Coevolutionary Sensor Networking with BiSNET/e

Pruet Boonma and Junichi Suzuki

Abstract Wireless sensor networks (WSNs) applications are often required to balance the tradeoffs among conflicting operational objectives (e.g., latency and power consumption) and operate at an optimal tradeoff. This chapter proposes and evaluates a biologically-inspired architecture, called BiSNET/e, which allows WSN applications to overcome this issue. BiSNET/e is designed to support three major types of WSN applications: data collection, event detection and hybrid applications. Each application is implemented as a decentralized group of software agents, which is analogous to a bee colony (application) consisting of bees (agents). Agents collect sensor data or detect an event (a significant change in sensor reading) on individual nodes, and carry sensor data to base stations. They perform these data collection and event detection functionalities by sensing their surrounding network conditions and adaptively invoking behaviors such as pheromone emission, reproduction, migration, swarming and death. Each agent has its own behavior policy, as a set of genes, which defines how to invoke its behaviors. BiSNET/e allows agents to evolve their behavior policies (genes) across generations and autonomously adapt their performance to given objectives. Simulation results demonstrate that, in all three types of applications, agents evolve to find optimal tradeoffs among conflicting objectives and adapt to dynamic network conditions such as traffic fluctuations and node failures/additions. Simulation results also illustrate that, in hybrid applications, data collection agents and event detection agents coevolve to augment their adaptability and performance .

Department of Computer Science
University of Massachusetts, Boston
e-mail: {pruet,jxs}@cs.umb.edu

1.1 Introduction

Autonomous adaptability is a key challenge in wireless sensor networks (WSNs) [1, 2, 6, 28, 33]. With minimal intervention to/from human operators, WSN applications are required to adapt their operations to dynamic changes in network conditions such as traffic fluctuations and node failures/additions. A critical issue in this challenge is that WSN applications have inherent tradeoffs among conflicting operational objectives [22]. For example, success rate of data transmission from individual nodes to base stations is an important objective because higher success rate ensures that base stations receive more sensor data for operators to better understand the current situation in an observation area and make better informed decisions. At the same time, latency of data transmission from individual nodes to base stations is another important objective. Lower latency ensures that base stations can collect sensor data for operators to understand the situation of an observation area more quickly and make more timely decisions. Success rate and latency conflict with each other. For improving success rate, hop-by-hop recovery is often applied; however, this can degrade latency. For improving latency, nodes may transmit data to base stations with the shortest paths; however, success rate can degrade because of traffic congestion on the paths.

In order to address this issue, the authors of the chapter envision autonomic WSN applications that understand their operational objectives, sense dynamic network conditions and act autonomously to satisfy conflicting objectives simultaneously. As inspiration for this vision, the authors observe that various biological systems have developed the mechanisms to overcome the above adaptability issue. For example, each bee colony autonomously satisfies conflicting objectives to maintain its well-being [34]. Those objectives include maximizing the amount of collected honey, maintaining temperature inside a nest and minimizing the number of dead drones. If bees focus only on foraging, they fail to ventilate their nest and remove dead drones. Based on this observation, the proposed application architecture, called BiSNET/e (Biologically-inspired architecture for Sensor NETworks, evolutionary edition), applies key biological mechanisms to design adaptive WSN applications.

Figure 1.1 shows the BiSNET/e runtime architecture. The BiSNET/e runtime operates atop TinyOS on each node. It consists of two software components: *agents* and *middleware platforms*, which are modeled after bees and flowers, respectively. Each WSN application is designed as a decentralized group of agents. This is analogous to a bee colony (application) consisting of bees (agents). Agents collect sensor data and/or detect an event (a significant change in sensor reading) on platforms (flowers) atop individual nodes. Then, they carry sensor data to base stations, in turn, to the MONSOON server (Figure 1.1). The server is modeled after a nest of bees. Agents perform these data collection and event detection functionalities by autonomously sensing their surrounding network conditions and adaptively performing biological behaviors such as pheromone emission, reproduction, migration, swarming and death. A middleware platform runs on each node, and hosts an arbitrary number of agents (Figure 1.1). It provides a series of runtime services that agents use to perform their functionalities and behaviors.

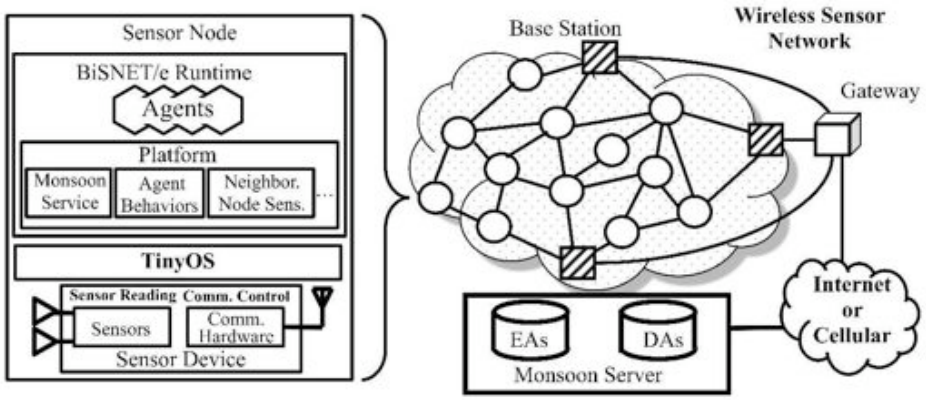

Fig. 1.1 BiSNET/e Runtime Architecture

This chapter describes a key component in BiSNET/e, called MONSOON[1], which is an evolutionary adaptation framework for agents. Each agent has its own behavior policy, as a set of *genes*, which defines when to and how to invoke its behaviors. MONSOON allows agents to evolve their behavior policies via genetic operations (mutation and crossover) across generations and simultaneously adapt their performance to given objectives. Currently, MONSOON considers four objectives: success rate, latency, power consumption and the degree of data aggregation.

The evolution process in MONSOON frees application developers from anticipating all possible network conditions and tuning their agents' behavior policies to the conditions at design time. Instead, agents are designed to autonomously evolve and tune their behavior policies at runtime. This can significantly simplify the implementation and maintenance of agents (i.e., WSN applications).

BiSNET/e supports three major types of WSN applications: *data collection*, *event detection* and *hybrid* applications. Hybrid applications perform both data collection and event detection in order to fulfill complex sensing requirements such as target tracking [25], contour/edge detection [11] and spatiotemporal event detection/monitoring [39]. Different types of applications are implemented with different types of agents. Data collection and event detection applications use *data collection agents* (DAs) and *event detection agents* (EAs), respectively. Hybrid applications use both DAs and EAs. DAs and EAs are designed as different biological species. In hybrid applications, the two types of agents are intended to coevolve and adapt their behavior policies in a symbiotic manner. EAs help DAs improve their behavior policies, and vice versa.

This chapter is organized as follows. Section 1.2 overviews the structure and behaviors of agents in BiSNET/e. Section 1.3 describes the evolution and coevolution processes in MONSOON. Section 1.4 evaluates MONSOON with a series of simulation results. Simulation results demonstrate that, in all three types of applications,

[1] Multiobjective Optimization for Network of Sensors using a cO-evOlutionary mechaNism

agents are robust and adaptive against various dynamic network conditions such as traffic fluctuations, node failures/additions and base station failures. Agents successfully evolve their behavior policies to find optimal tradeoffs among conflicting objectives. Simulation results also illustrate that, in hybrid applications, DAs and EAs coevolve to augment their adaptability and performance with each other. Sections 1.5 and 1.6 conclude with some discussion on related work and future work.

1.2 BiSNET/e Agents

At the beginning of a WSN's operation, one DA and one EA are deployed on each node. They have randomly-generated behavior policies. A DA collects sensor data on each node periodically (i.e., at each duty cycle) and carry the data to a base station on a hop-by-hop basis. An EA collects sensor data on each node periodically, and if it detects an event—a significant change in its sensor reading, carries the data to a base station on a hop-by-hop basis. If an event is not detected, the EA discards the data. (It is not transmitted to a base station.)

Agents are decentralized in a WSN. There are no centralized entities to control and coordinate agents. Decentralization allows agents to be scalable and survivable by avoiding a single point of performance bottlenecks and failures [3, 24].

1.2.1 Agent Structure and Behaviors

Each agent consists of *attributes*, *body* and *behaviors*. *Attributes* carry descriptive information on an agent. They include agent type (DA or EA), behavior policy (genes), sensor data to be reported to a base station, the data's time stamp, and the ID of a node where the data is collected.

Body implements the functionalities of an agent: collecting, processing, discarding and processing sensor data.

Behaviors implement actions inherent to all agents. Inspired by biological entities such as bees, agents sense their surrounding network conditions and behave according to the sensed conditions without any intervention from/to other agents, platforms, base stations and human operators. This chapter focuses on the following seven behaviors.

1. **Food gathering and consumption:** Biological entities strive to seek food for living. For example, bees gather nectar to produce honey. Similarly, in BiSNET/e, each agent periodically reads sensor data (as nectar) to gain *energy* (as honey)[2] and expends a constant amount of energy for living.

[2] In BiSNET/e, the concept of energy does not represent the amount of physical battery in a node. It is a logical concept to affect agent behaviors.

2. **Pheromone emission:** Agents may emit different types of pheromones: *migration* and *alert pheromones*. They emit migration pheromones on their local nodes when they migrate to neighboring nodes. Each migration pheromone references the destination node an agent has migrated to. Agents also emit alert pheromones when they fail migrations within a timeout period. Migration failures may occur because of node failures due to depleted battery and physical damages as well as link failures due to interference and congestion. Each alert pheromone references the node that an agent could not migrate to. Each of migration and alert pheromones has its own concentration, which decays by half at every duty cycle. A pheromone disapears when its concentration becomes zero.
3. **Replication:** EAs may make a copy of themselves in response to the abundance of stored energy, while DAs make a copy of themselves at each duty cycle. A replicated (child) agent is placed on the node that its parent resides on, and it inherits the parent's agent type and behavior policy (a set of genes). Replicated agents are intended to move toward base stations to report collected sensor data.
4. **Migration:** Agents may move from one node to another. Migration is used to transmit agents (sensor data) toward base stations. On an intermediate node, each agent chooses the next-hop node by sensing three types of available pheromones: base station, migration and alert pheromones.
   Each base station periodically propagates base station pheromones to individual nodes. Their concentration decays on a hop-by-hop basis. Using base station pheromones, agents can sense where base stations exist approximately, and they can move toward the base stations by climbing a concentration gradient of base station pheromones[3].
   An agent may move to a base station by following a migration pheromone trace on which many other agents have traveled. The trace can be the shortest path to a base station. Conversely, an agent may go off a migration pheromone trace and follows another path to a base station when the concentration of migration pheromones is too high on the trace (i.e., when too many agents have followed the trace). This avoids separating the network into islands. The network can be separated with the migration paths that too many agents follow, because the nodes on the paths run out of their battery earlier than the others[4].
   An agent may also avoid moving to a node referenced by an alert pheromone. This allows agents to reach base stations by bypassing failed nodes/links.
5. **Swarming:** Agents may swarm (or merge) with others at the nodes on their ways to base stations. With this behavior, multiple agents become a single agent. (A DA can merge with both DAs and EAs, and an EA can merge with both EAs and DAs.) The resulting agent (swarm) aggregates sensor data contained in other agents, and uses the behavioral policy of the best agent in the swarm in terms of given operational objectives.

---

[3] Base station pheromones are designed after the Nasonov gland pheromone, which guides bees to move toward their nest [14].

[4] Data transmission imposes the highest power consumption among all the operations that each node performs [26].

In order to increase the chances of swarming, at each intermediate node toward a base station, an agent may wait for other agents. If an agent(s) arrives at the node during a waiting period, the waiting agent merges with the arriving agent(s). The swarming behavior saves power consumption of nodes because in-node data aggregation requires much less power consumption than data transmission does [26].

6. **Reproduction:** Once agents arrive at the MONSOON server (Figure 1.1), they are evaluated according to their four objectives. Then, MONSOON selects best-performing (or elite) agents, and propagates them to individual nodes. An agent running on each node performs reproduction with one of the propagated agents. A reproduced agent inherits a behavior policy (gene) from its parents via crossover, and mutation may occur on the inherited behavior policy. Reproduced agents trigger a generation change by taking over existing agents running on individual nodes.
   Reproduction is intended to evolve agents so that the agents that fit better to the environment become more abundant. It retains the agents whose fitness to the current network conditions is high (i.e., the agents that have effective behavior policies, such as moving toward a base station in a short latency), and eliminates the agents whose fitness is low (i.e., the agents that have ineffective behavior policies, such as consuming too much power to reach a base station). Through successive generations, effective behavior policies become abundant in agent population while ineffective ones become dormant or extinct. This allows agents to adapt to dynamic network conditions.
7. **Death:** Agents periodically consume energy for living and expend energy to invoke their behaviors. The energy costs to invoke behaviors are constant for all agents. Agents die due to lack of energy when they cannot balance energy gain and expenditure. The death behavior is intended to eliminate the agents that have ineffective behavior policies. For example, an agent would die before arriving at a base station if it follows a too long migration path. When an agent dies, the local platform removes the agent and releases all resources allocated to it.

1.2.2 Behavior Sequence for DAs

Figures 1.2 shows the sequence of behaviors that each DA performs on a node at each duty cycle. A DA reads sensor data with the underlying sensor device and gains a constant amount of energy. Given the energy intake (E_F), each agent updates its energy level as follows.

$$E(t) = E(t-1) + E_F \tag{1.1}$$

$E(t)$ and $E(t-1)$ denote a DA's energy level at the current and previous duty cycle. t is incremented by one at each duty cycle.

for each *duty cycle*
do {
Read sensor data and gain energy (E_F).
Update energy level ($E(t)$).
if $E(t) <$ *the death threshold* (T_D)
then Invoke the death behavior.
Invoke the replication behavior to make a child agent.
Give the half of the current energy level to a replicated (child) agent.
for each *migrating agent*
do {
if not *waiting*
then {
Determine the destination node of migration.
Emit a migration pheromone on the local node.
Migrate to a neighboring node.
if *Migration fails*
then {
Emit an alert pheromone on the local node.
Propagate it to neighboring nodes.
}
}
}
}

Fig. 1.2 Sequence of DA Behaviors

If a DA's energy level ($E(t)$) goes below the death threshold (T_D), the DA dies due to starvation[5].

A DA replicates itself at each duty cycle. A replicating (parent) agent splits its energy units to halves ($\frac{E(t)-E_R}{2}$), gives a half to its child agent, and keeps the other half. E_R is the energy cost for an agent to perform the replication behavior. A child agent contains the sensor data that its parent collected, and carries it to a base station.

Each replicated DA migrates toward a base station on a hop by hop basis. On each intermediate node, it decides whether it migrates to a next-hop node or wait for other agents to swarm (or merge) with them. This decision is made based on a migration probability (p_m). If the agent decides to migrate, it examines Equation 1.2 to determine which next-hop node it migrates to.

$$WS_j = \sum_{t=1}^{3} w_t \frac{P_{t,j} - P_{t_{min}}}{P_{t_{max}} - P_{t_{min}}} \tag{1.2}$$

A DA calculates this weighted sum (WS_j) for each neighboring node j, and moves to a node that generates the highest weighted sum. t denotes pheromone type; P_{1j}, P_{2j} and P_{3j} represent the concentrations of base station, migration and alert pheromones on the node j. $P_{t_{max}}$ and $P_{t_{min}}$ denote the maximum and minimum concentrations of P_t among all neighboring nodes.

When a DA is migrating to a neighboring node, it emits a migration pheromone on the local node. If the DA's migration fails, it emits an alert pheromone, and it spreads to one-hop away neighboring nodes.

[5] If all agents are dying on a node at the same time, a randomly selected agent for each type (i.e., EA and DA) will survive. At least one agent of each type runs on each node.

1.2.3 Behavior Sequence for EAs

Figures 1.3 shows the sequence of behaviors that each EA performs on a node at each duty cycle. When an EA reads sensor data (as nectar) with the underlying sensor device and gains energy (as honey), its current energy level ($E(t)$) is updated with Equation 1.3.

$$E(t) = E(t-1) + S \cdot M \tag{1.3}$$

S denotes the absolute difference between the current and previous sensor data. M is metabolic rate, which is a constant between 0 and 1.

Each EA replicates itself if its energy level exceeds the replication threshold: $T_R(t)$. The replication threshold is continuously adjusted as an EWMA (Exponentially Weighted Moving Average) of energy level:

$$T_R(t) = (1-\alpha)T_R(t-1) + \alpha E(t) \tag{1.4}$$

$T_R(t)$ and $T_R(t-1)$ denote the replication thresholds at the current and previous duty cycle, respectively. EWMA is used to smooth out short-term minor oscillations in the data series of E. It places more emphasis on the long-term transition trend of E; only significant changes in E have the effects to change T_R. The α value is a constant to control the responsiveness of EWMA against the changes of E.

A parent EA splits its energy units to halves, gives a half to its child agent, and keeps the other half. The parent EA keeps replicating itself until its energy level becomes less than its T_R. Each child agent contains the sensor data that its parent collected, and carries it to a base station.

As DAs do, each migrating EA decides whether it performs the migration behavior or the swarming behavior using its migration probability (p_m). It performs the migration behavior with Equation 1.2, followed by the pheromone emission behavior, in the same way as DAs do.

1.2.4 Agent Behavior Policy

EAs and DAs have the same structure of behavior policies (genes). Each behavior policy consists of two distinctive information: migration probability (p_m) and a set of weight values in Equation 1.2 ($w_t, 1 \leq t \leq 3$). Migration probability is a non-negative value between zero and one. With higher migration probability, an agent has a higher chance to perform the migration behavior instead of the swarming behavior. With a lower migration probability, an agent has a higher chance to perform the swarming behavior. Weight values govern how agents perform the migration behavior. For example, if an agent has zero for w_2 and w_3, the agent ignores migration and alert pheromones, and moves toward the base stations by climbing the concentration gradient of base station pheromones. If an agent has a positive value for w_2, it follows a migration pheromone trace on which many other agents have traveled. A

for each *duty cycle*
do {
- Read sensor data and gain energy (E_F).
- Update energy level ($E(t)$).
- **if** $E(t) <$ *the death threshold* (T_D)
  - **then** Invoke the death behavior.
- **while** $E(t) >$ *the replication threshold* ($T_R(t)$)
  - **do** {
    - Invoke the replication behavior to make a child agent.
    - Give the half of the current energy level to the child agent.
  }
- **for each** *migrating agent*
  - **do** {
    - **if not** *waiting*
      - **then** {
        - Determine the destination node of migration.
        - Emit a migration pheromone on the local node.
        - Migrate to a neighboring node.
        - **if** *Migration fails*
          - **then** {
            - Emit an alert pheromone on the local node.
            - Propagate it to neighboring nodes.
          }
      }
  }

Fig. 1.3 Sequence of EA Behaviors

negative w_2 value allows an agent to go off a migration pheromone trace and follow another path toward a base station. If an agent has a negative value for w_3, it moves to a base station by bypassing failed nodes/links.

1.3 MONSOON

In order to drive agent evolution and coevolution, MONSOON performs *elite selection* and *genetic operations*. The elite selection process evaluates each type of agents (DAs and EAs) that arrive at base stations, based on given objectives, and chooses the best (or elite) ones. Elite agents are propagated to individual nodes in the network. Through genetic operations (crossover and mutation), an agent running on each node performs the reproduction behavior with one of elite agents. A reproduced agent inherits a behavior policy (a set of genes) from its parents via crossover, and mutation may occur on the inherited behavior policy. Reproduced agents trigger a generation change by taking over parent agents. Elite selection is performed in the MONSOON server (Figure 1.1), and genetic operations are performed in each node.

Reproduction is intended to evolve agents so that the agents that fit better to the current network conditions become more abundant. It retains the agents that have effective behavior policies, such as moving toward a base station in a short latency, and eliminates the agents that have ineffective behavior policies, such as consuming too much power to reach a base station. Through successive generations, effective behavior policies become abundant in agent population while ineffective ones become dormant or extinct. This allows agents to adapt to dynamic network conditions.

1.3.1 Operational Objectives

Each agent (DA or EA) considers four conflicting objectives: *latency*, *cost*, *success rate* and *data yield*. MONSOON strives to minimize latency and cost and maximize success rate and data yield.

1. **Latency** represents the time required for an agent (DA or EA) to travel to a base station from a node where the agent is born (replicated). As depicted below, latency (L) is measured as a ratio of this agent travel time (t) to the physical distance (d) between a base station and a node where the agent is born. The MONSOON server knows the location of each node with a certain localization mechanism.

$$L = \frac{t}{d} \tag{1.5}$$

2. **Cost** represents power consumption required for an agent (DA or EA) to travel to a base station from a node where the agent is born. Cost (C) is measured with the total number of node-to-node data transmissions required for an agent to arrive at a base station (n_{tran}), each node's radio transmission range (r_{tran}), and physical distance (d).

$$C = \frac{n_{tran}}{d/r_{tran}} \tag{1.6}$$

   The total number of data transmissions include successful and unsuccessful (failed) agent migrations as well as the transmissions of migration or alert pheromones.
3. **Success rate** is measured differently for DAs and EAs. For DAs, it is measured as follows.

$$S_{DA} = \frac{n_{arrive}}{N} \tag{1.7}$$

   n_{arrive} indicates the number of agents that arrive at base stations, and N indicates the total number of nodes in the network.
   For EAs, success rate is measured as follows.

$$S_{EA} = \frac{m_{success}}{m_{total}} \tag{1.8}$$

   $m_{success}$ indicates the number of successful migrations that an EA performs until it arrives at a base station. m_{total} indicates the total number of migration attempts that an EA makes. This includes the number of successful migrations (i.e., $m_{success}$) and the number of failed migrations.
4. **Data yield** is measured as the number of sensor data that an agent (DA or EA) aggregates and carries to a base station. Its initial value is one; however, it increases as the an agent swarms with other agents.

1.3.2 Elite Selection

Figure 1.4 shows how elite selection occurs at the MONSOON server in each duty cycle. The MONSOON server performs the same selection process for EAs and DAs separately. The first step is to measure four objective values (i.e., latency, cost, success rate and data yield) of each agent that reaches the MONSOON server via base stations. Then, each agent is evaluated whether it is *dominated* by another one. MONSOON determines that agent A dominates agent B iif:

- A's objective values are better than, or equal to, B's in all objectives, and
- A's objective values are better than B's in at least one objective.

```
Empty the archive
for each duty cycle
do { Empty the population pool.
     Collect agents from the network.
     Add collected agents to the population pool.
     Move agents from the archive to the population pool.
     Empty the archive
     for each agent of the ones in the population pool
     do { if not dominated by all other agents in
          the population pool
          then Add the agent to the archive.
     Select elite agents from the archive.
     Propagate elite agents to the network.
```

Fig. 1.4 Elite Selection in MONSOON

In the next step, a subset of non-dominated agents are selected as elite agents. This is performed with a four dimensional hypercube space whose axes represent four objectives. Each axis of the hypercube space is divided so that the space contains small cubes. Non-nominated agents are plotted in this hypercube space based on their objective values. If multiple non-dominated agents are plotted in a cube, one of them is randomly selected as an elite agent. If no non-dominated agents are plotted in a cube, no elite agent is selected from the cube. This elite selection is designed to maintain the diversity of elite agents. Diversity of agents can improve their adaptability to unanticipated network conditions.

Figure 1.5 shows an example hypercube space. For simplicity, it shows only three of four objectives (i.e., cost, latency and data yield). Each axis is divided into two ranges; therefore, eight cubes exist in total. In this example, six non-dominated agents (A to F) are plotted in the hypercube space. Three agents (B, C, and D) are plotted in a lower left cube, while the other three agents (A, E, and F) are plotted in three different cubes. From the lower left cube, only one agent is randomly selected as an elite agent. A, E, and F are selected as elite agents because they exist in different cubes.

In addition to select elite agents, the MONSOON server adjusts the mutation rate of agents based on performance improvement of non-dominated agents. The smaller

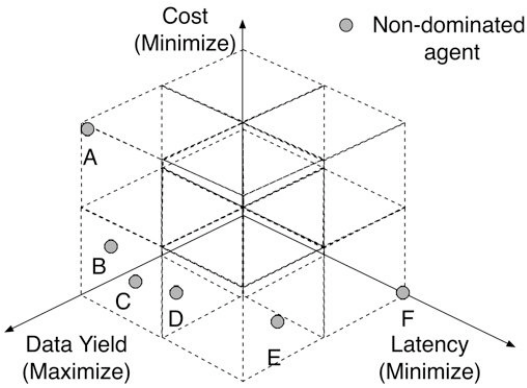

Fig. 1.5 An Example Elite Selection

improvement they make in objective values, the higher mutation rage the MONSOON server assigns to agents, thereby accelerating agent evolution/coevolution.

The performance of non-dominated agents is measured as a set of performance representative points in different objectives. Equation 1.9 shows how to obtain a performance representative point ($\bar{o}_i$) in each objective i.

$$\bar{o}_i = \frac{\sum_{a \in A} o_i(a)}{|A|} \tag{1.9}$$

A denotes the set of non-dominated agents. $o_i(a)$ denotes the objective value that agent a yields in objective i. o_i is a value that a performance representative point is projected on objective i. It is normalized between 0 and 1.

The improvement of performance is measured as the Euclidean distance (d) between the performance representative points at the current and previous duty cycles:

$$d = \sqrt{\frac{\sum_{i \in O} (\bar{o}_i(t) - \bar{o}_i(t-1))^2}{|O|}} \tag{1.10}$$

O denotes the set of all objectives. $\bar{o}_i(t)$ and $\bar{o}_i(t-1)$ denote the performance representative points projected on objective i in the current and previous duty cycles, respectively.

Mutation rate (m) is adjusted with Equation 1.11 where k is a constant and less than one.

$$m = k(1-d) \tag{1.11}$$

1.3.3 Genetic Operations

Once elite DAs and EAs are selected, the MONSOON server propagates them and adjusted mutation rate to each node in the network. They are propagated with a base station pheromone.

Upon receiving a base station pheromone, an agent running on each node performs the reproduction behavior with a certain reproduction rate through genetic operations (crossover and mutation). It selects one of propagated elite agents, as a mating partner, which has the most similar behavior policy (genes). This similarity is measured with the Euclidean distance between the values of behavior policies. If two or more elite agents have the same similarity to the local agent, one of them is randomly selected. During reproduction, a child agent performs one-point half-and-half crossover; it randomly inherits the half of its gene from its parent agent and the other half from the parent's mating partner.

DAs can mate with elite EAs, and EAs can mate with elite DAs. This cross-mating allows DAs and EAs to coevolve their behavior policies; DAs can improve EAs' genes, and vice versa. This is particularly important when no events occur in a WSN. In this case, EAs have no chance to evolve their genes because they do not migrate toward the MONSOON server. Through cross-mating with DAs, EAs can reproduce offspring and coevolve their genes even if no events occur.

Mutation occurs on a child agent's gene with a certain mutation rate. Mutation randomly changes gene values within a predefined value range. As discussed in Section 1.3.2, the MONSOON server periodically adjusts mutation rate. After reproduction, a child agent takes over the local parent agent as the next generation agent.

1.4 Simulation Results

This section shows a set of simulation results to evaluate BiSNET/e and MONSOON. Sections 1.4.1, 1.4.2 and 1.4.3 discuss the simulation results obtained with a data collection application, event detection application and hybrid application. Each application is used to monitor an oil spill at the sea. The spill is simulated as 100 barrels (approximately 3,100 gallons) of crude oil spreads at the middle of the Dorchester Bay of Massachusetts. Simulation data of this spill is generated with an oil spill trajectory model implemented in the General NOAA Oil Modeling Environment [5].

A simulated WSN consists of 100 nodes uniformly deployed in an observation area of 300x300 square meters. An oil spill starts at the middle of this observation area. Each node's communication range is 30 meters and equips a surface roughness sensor to detect spilled oil. A base station is deployed on the observation area's northwestern corner. The base station links the MONSOON server via emulated serial port connection. All software components in the BiSNET/e runtime are implemented in nesC, and the MONSOON server is implemented in Java.

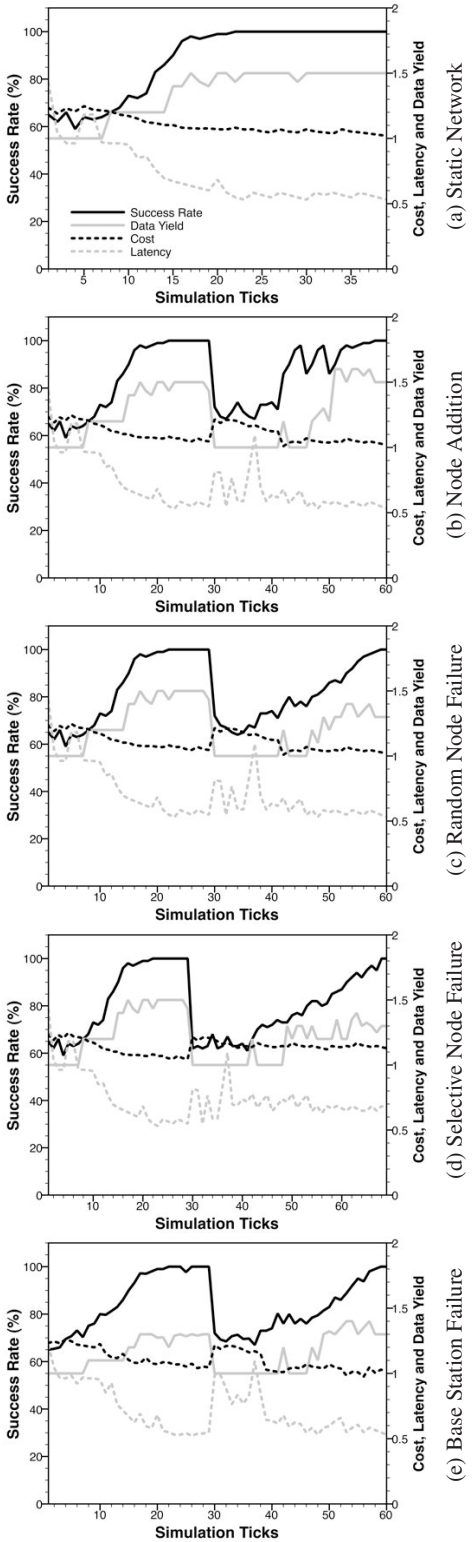

Fig.6. Objective Values of DAs without EAs

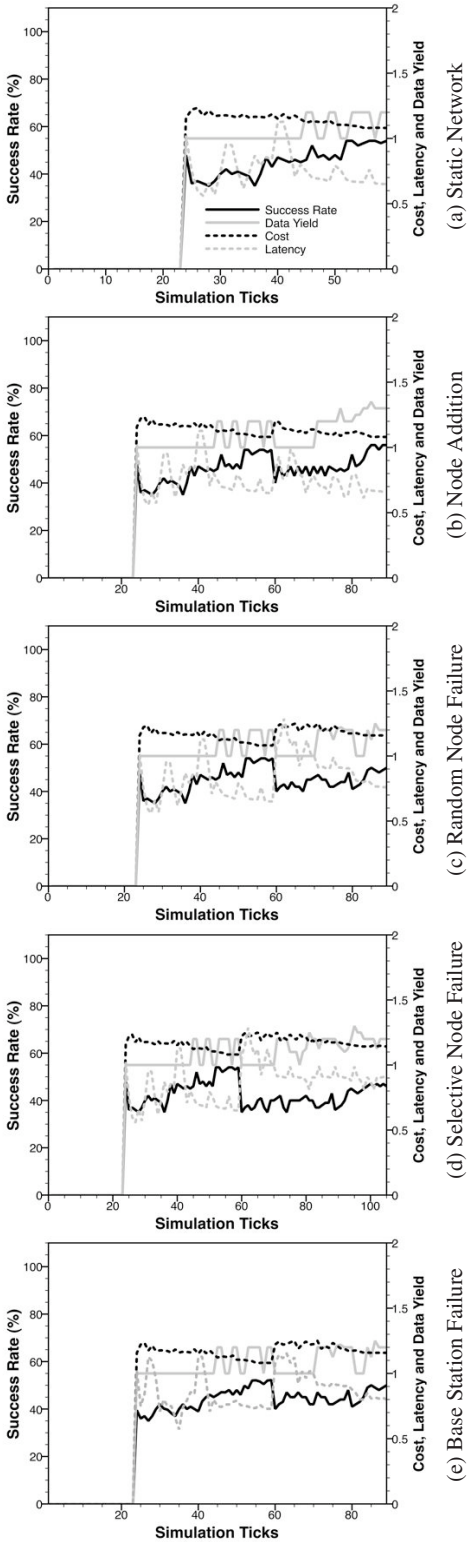

Fig.7. Objective Values of EAs without DAs

Simulation time is counted with ticks. Each tick represents five minutes. For genetic operations in MONSOON, reproduction probability and the maximum mutation rate are configured as 0.75 and 0.2, respectively.

1.4.1 Data Collection Application

A data collection application is implemented with DAs that perform the sequence of behaviors shown in Figure 1.2. No EAs are used in this application. The duty cycle corresponds to a simulation tick (five minutes).

Figure 6 (a) shows the average objective values produced by DAs at each simulation tick. Each objective value gradually improves and converges at the 22nd tick. This simulation result shows that MONSOON allows DAs to simultaneously satisfy conflicting objectives by evolving their behavior policies.

Figure 6 (b) shows how the performance of DAs changes against a dynamic node addition. 25 nodes are added at random locations at the 30th tick. Upon this change in the network environment, objective values degrade dramatically because DAs have randomly-generated behavior policies on the new nodes. Those DAs cannot migrate efficiently toward the base station. Also, enough pheromones are not available on new nodes; DAs cannot make proper migration decisions when they move to the new nodes. However, DAs gradually improve their performance again, and objective values converge again at the 56th tick. Interestingly, after 50th tick, average data yield is greater than that before 30th tick. Because there are more DAs from the additional nodes, so DAs have higher chance to swarm. MONSOON allows DAs to autonomously recover application performance despite dynamic node addition by evolving their behavior policies.

Figure 6 (c) shows how the performance of DAs changes against dynamic node failures. 25 nodes randomly fail at the 30th tick. Objective values degrade because some DAs try to migrate to failed nodes referenced by migration pheromones. This increases the number of unsuccessful agent migrations. However, DAs gradually improve their performance again, and objective values converge again at the 56th tick. MONSOON allows DAs to autonomously recover application performance despite dynamic node failures by evolving their behavior policies.

Figure 6 (d) shows how the performance of DAs changes when nodes selectively fail in a specific area. At the 30th tick, 20 nodes fail in the middle of WSN observation area. Hence, a WSN has a hole in its middle area. Compared with Figure 6 (c), it takes longer time for DAs to recover their performance. Objective values converge at 66th tick again. The converged cost and latency are worse than the ones at the 30th tick because DAs have to detour a hole (i.e., a set of failed nodes) and take longer migration paths to the base station. This simulation results shows that MONSOON allows DAs to survive selective node failures through evolution.

Figure 6 (e) shows how the performance of DAs changes against base station failures. In this simulation scenario, two base stations are deployed at the northwestern and southeastern corners of WSN observation area. At the 30th tick, a base

station at the southeastern corner fails. Objective values degrade because some DAs try to migrate toward the failed base station referenced by base station pheromones. This increases the number of unsuccessful agent migrations. However, DAs gradually improve their performance again, and objective values converge again at the 56th tick. MONSOON allows DAs to autonomously evolve and recover application performance despite dynamic base station failures.

1.4.2 Event Detection Application

An event detection application is implemented with EAs that perform the sequence of behaviors shown in Figure 1.3. No DAs are used in this application. This simulation study simulates an oil spill, which occurs in the middle of WSN observation area at the 24th tick and radially spreads over time.

Figure 7 (a) shows the average objective values at each simulation tick. Upon an event detection, objective values are low because EAs use random behavior policies at first. However, each objective value gradually improves and converges at the 52nd tick. This simulation result shows that MONSOON allows EAs to simultaneously satisfy conflicting objectives by evolving their behavior policies.

Figure 7 (b) shows how the performance of EAs changes against a dynamic node addition. 25 nodes are added at random locations at the 60th tick. Upon this environmental change, objective values degrade slightly because EAs have randomly-generated behavior policies on the new nodes. Those EAs cannot migrate efficiently toward the base station. However, EAs gradually improve their performance immediately, and objective values converge again at the 85th tick. MONSOON allows EAs to autonomously recover application performance despite dynamic node addition by evolving their behavior policies.

Figure 7 (c) shows how the performance of EAs changes against dynamic node failures. 25 nodes randomly fail at the 60th tick. Objective values degrade slightly because some EAs try to migrate to failed nodes referenced by migration pheromones. This increases the number of unsuccessful agent migrations. However, EAs gradually improve their performance again, and objective values converge again at the 85th tick. MONSOON allows EAs to autonomously recover application performance despite dynamic node failures by evolving their behavior policies.

Figure 7 (d) shows the result of a simulation when 20 sensor nodes are selected in selective fashion, i.e. create a hole in the middle of network, to be deactivated at the 60th tick. Compared with the result in Figure 7 (c), MONSOON takes longer time to improve the performance of the WSN. The success rate converges at about the 100th tick to approximately 48%. The cost and latency also show the similar trend. Particularly, after the 52nd tick, the average value of cost and latency are higher than the values just before the 20th tick because agents have to detour in a longer path to avoid the hole in the middle of the network.

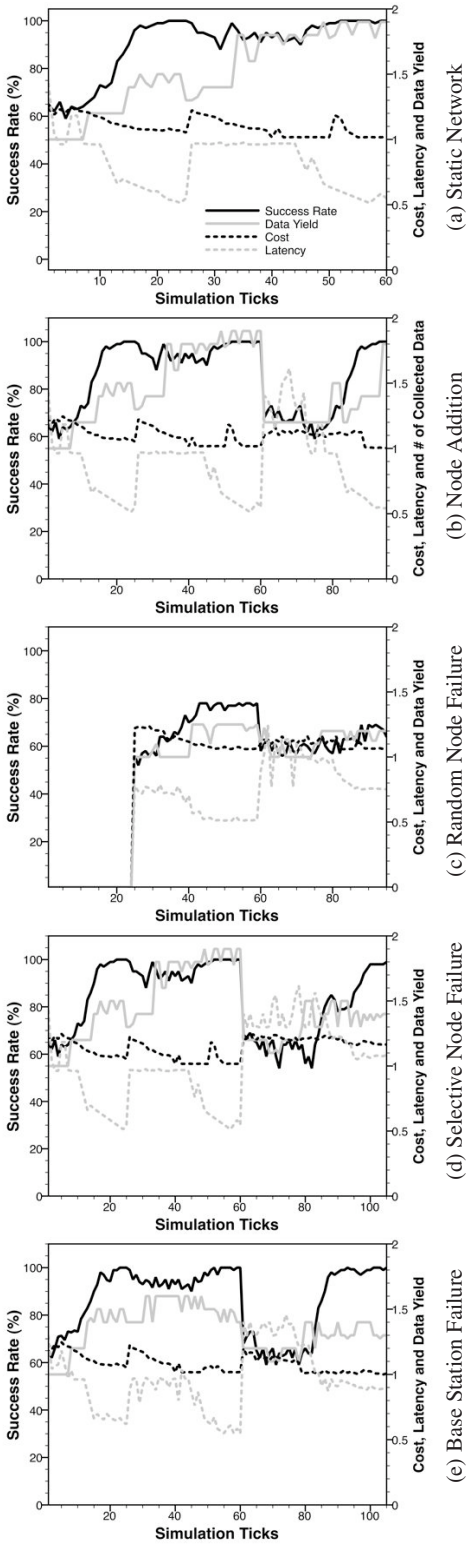

Fig.8. Objective Values of DAs with EAs

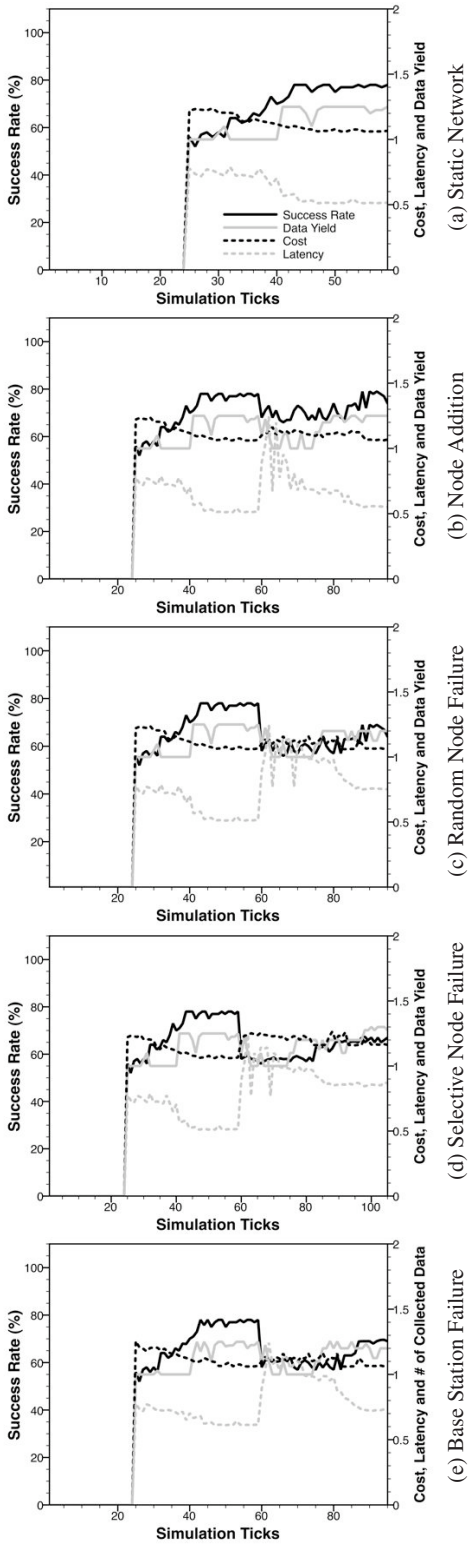

Fig.9. Objective Values of EAs with DAs

The simulation results shows that MONSOON allows WSN to survives a selective sensor nodes failure by adjusting the operational parameters of WSN to be suitable to the changes in network condition.

Figure 7 (e) shows the result of a simulation which initially has two base stations deployed at the northwestern and southeastern corner of the observation area. Then, at the 60th tick, the base station at the southeastern corner is deactivated. In this figure, at the 61st tick, the success rate drops to about 40% from around 50%. However, the success rate is improved successively and reach the same level as before the base station is deactivated at the 85th tick. Cost and latency show the same trend. MOSOON allows WSN to survives a base station failure by autonomously directing all agents to the remaining base station.

1.4.3 Hybrid Application

This section represents simulation results from a sensor network with two applications deployed simultaneously. Figure 8 shows the average objective values from collected DAs, i.e. for data collection application, in each simulation ticks. On the other hand, Figure 9 shows the average objective values from collected EAs, i.e. for event collection application, in each simulation ticks.

In Figure 9 (a), at the 24th simulation tick, oil spill happens and EAs start detecting and moving to the base station. The impact of EAs on DAs can be observed from Figure 8 (a) with the drop in success rate and the increase of cost and latency around 24th tick. However, within thirty simulation ticks, MONOON allows DAs to adapt to the EAs and retain their performance. The simulation results shows that MONSOON allows a WSN application to adapt to the other application such that they can co-exist tranquilly in a same sensor network.

Figure 9 (b), (c), (d) and (e) show the similar scenario as in Figure 7 (b), (c), (d) and (e), respectively. The simulation result in the former set of the figures also show the similar trend as in the later set of the figures; therefore, MONSOON allows a WSN application to adapt to network changes, i.e. partial node failure or the base station failure, even when it has to work simultaneously with another application on the same network.

Figure 9 (a) portraits the same scenario as in Figure 7 (a). In Figure 9 (a), sensor network hosts two applications, data collection and event detection. However, the objective values of event detection application, i.e. EAs, in Figure 9 (a) are improved faster than in Figure 7 (a). For example, the latency is reduced to lower than 0.5 at around the 44th tick in Figure 9 (a) but it takes about the 58th tick in Figure 7 (a) to reduce to about 0.6. Thanks to cross-mating (see section 1.3.3) , MONSOON allows event detection application, i.e., EAs, to improve its objective values by using information from the other application. Figure 9 (b), (c), (d) and (e) also show the similar results.

1.4.4 Adaptive Mutation

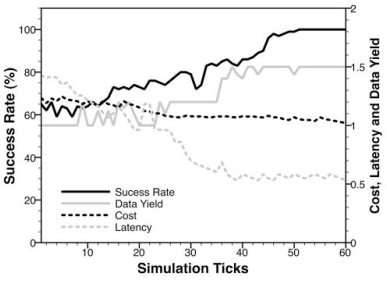

Fig. 1.10 Objective Values of DAs without EAs

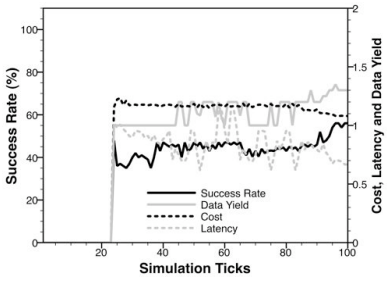

Fig. 1.11 Objective Values of EAs without DAs

In the current implementation of BiSNET/e, mutation rate of EAs and DAs is adaptively adjusted by MONSOON server. Figure 1.10 and 1.11 show simulation result from the same simulation setup as in Figure 6 (a) and 7 (a) respectively; however, in Figure 1.10 and 1.11, the BiSNET/e does not use adaptive mutation, a fix mutation rate of 0.05 is used instead. It is clear that, without adaptive mutation, MONSOON has to take about two times longer to archive the same optimized objective values. The simulation results shows that adaptive mutation in BiSNET/e allow MONSOON to quickly adjust the WSN applications to suit to environment condition.

1.4.5 Power Consumption

Figure 1.12 shows the impact of MONSOON and BiSNET/e on power consumption, and compare between hybrid application and individual applications. The figure represents the power consumption on each simulation tick for the sensor network with node addition scenario, e.g. as in Figure 6 (b). In this figure, individual applications represents summation of the power consumption of data collection and event detection application when they are implemented separately, i.e. the summation of power consumption from sensor network in Figures 6 (b) and 7 (b) in each simulation tick. On the other hand, hybrid application represents the power consumption of a sensor network which implements both data collection and event detection on the same application, i.e. from Figure 8 (b). In this figure, MONSOON and BiSNET/e can reduce the power consumption of WSN by optimizing the agent's behavior policy. Moreover, by implementing hybrid application on a same framework, the power consumption can be further reduced which can be seen when compare the power consumption of hybrid application and individual applications.

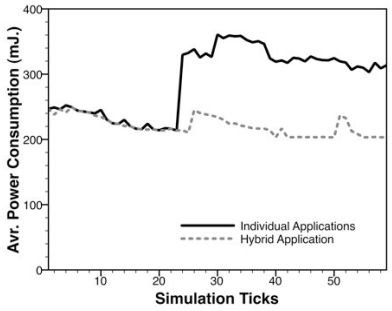

Fig. 1.12 Average Power Consumption

1.4.6 Memory Footprint

Table 1.1 shows the memory footprint of the BiSNET/e runtime in a MICA2 mote, and compares it with the footprint of Blink (an example program in TinyOS), which periodically turns on and off an LED, and Agilla, which is a mobile agent platform for WSNs [13]. The BiSNET/e runtime is lightweight in its footprint thanks to the simplicity of the biologically-inspired mechanisms in BiSNET/e. BiSNET/e can even run on a smaller-scale nodes, for example, TelosB, which has 48KB ROM.

Table 1.1 Memory Footprint in a MICA2 Node

	RAM (KB)	**ROM (KB)**
BiSNET	2.8	31.2
Blink	0.04	1.6
Agilla	3.59	41.6

1.5 Related Work

This chapter extends the authors' prior work [7–9]. In [7], the authors proposed a biologically-inspired WSN architecture, called BiSNET. BiSNET does not investigate evolutionary adaptation. Thus, agent behavior policies are manually configured through trial-and-errors and fixed at runtime. Unlike BiSNET, BiSNET/e allows agents to dynamically adapt their behavior policies to unanticipated network conditions. In [8], MONSOON was proposed and studied with data collection applications. This chapter considers event detection applications and hybrid applications as well as data collection applications. Moreover, this chapter evaluates how coevolution between DAs and EAs augments agent adaptability. This is beyond the scope of [8]. Compared with [9], this chapter investigates new operational objective (the

degree of data aggregation) and new mechanisms in MONSOON (e.g., swarming behavior, migration probability and adaptive mutation).

Agilla proposes a programming language to implement mobile agents for WSNs, and provides an interpreter to operate agents on TinyOS [13]. Similarly, BiSNET/e exploits mobile agents (DAs and EAs); however, this chapter does not focus on investigating a new programming language for those agents. While BiSNET/e and Agilla implement a similar set of agent behaviors such as migration and replication, BiSNET/e studies a wider range of agent behaviors. For example, Agilla does not consider energy gain/expenditure, swarming and pheromone emission. Moreover, Agilla does not consider evolutionary and coevolutionary adaptation of agents to seek optimal tradeoffs among conflicting objectives. As shown in Table 1.1, BiSNET/e is implemented more lightweight than Agilla.

Virtual pheromone (VP) is a biologically-inspired node-to-node communication primitive in TinyOS-based WSNs [37]. It has a generic set of properties such as pheromone type, strength, source and payload. Therefore, VP can be used to implement base station, migration and alert pheromones in BiSNET/e. However, VP does not address a research issue that BiSNET/e does: autonomous adaptability of WSN applications.

Quasar is similar to BiSNET/e in that it proposes a data collection protocol that balances the tradeoff between data accuracy and power efficiency [16]. Although BiSNET/e does not focus on data accuracy as its operational objective, it studies extra objectives in data transmission such as success rate and latency. Also, it considers not only data collection applications but also event detection and hybrid applications in dynamic WSNs. (Quasar is considered and evaluated for static WSNs.) Quasar and BiSNET/e employ different optimization/adaptation processes; BiSNET performs a population-based evolutionary algorithm while Quasar employs time series data analysis.

[4] proposes a cost function (or fitness function) that comprises conflicting objectives regarding data transmission cost, power consumption, latency, reliability (the time between node/link failures) and link interference. These objectives are similar to the ones BiSNET/e considers. However, in [4], the total cost (or fitness) is calculated as a weighted sum of objective values. This means that application designers need to manually configure every weight value in a fitness function through trial-and-errors. In BiSNET/e, no manually-configured parameters exist for elite selection because of a domination ranking mechanism. BiSNET/e minimizes the number of manually-configured parameters to minimize configuration costs for application designers. Moreover, BiSNET/e does not require each node to have global network information as [4] does.

Genetic algorithms (GAs) have been investigated in various aspects in WSNs; for example, routing [10, 12, 18, 20, 23], data processing [17], localization [38, 42], node placement [15,43] and object tracking [10]. All of these work use fitness functions, each of which combines multiple objective values as a weighted sum and rank agents/genes in elite selection. As discussed above, it is always non-trivial to manually configure weight values in a fitness function through. In contrast, BiSNET/e

eliminates parameters in elite selection by design. Moreover, [10, 12, 15, 18, 23, 38, 43] do not assume dynamic WSNs, but static WSNs.

Beyond these classical GAs, multiobjective GAs (MOGAs) have also been investigated in WSNs; for example, for routing [30, 31, 35, 40], node placement [19, 21, 27, 29, 32] and duty cycle management [41]. In all of these work except [35], a central server performs an evolutionary optimization process. This can lead to scalability issue as network size increases. In contrast, MONSOON is carefully designed to perform its optimization process in both the MONSOON server and individual nodes. Moreover, all of these work do not assume dynamic WSNs, but static WSNs.

[30,31,40] investigates MOGAs that optimize migration routes for mobile agents to travel from a base station to cluster head nodes and collect sensor data from individual clusters. In BiSNET/e, agents make their migration and other behavior decisions by themselves. MONSOON optimizes their behavior policies, not agents' migration routes.

[35] is similar to BiSNET/e in that both follow the agent designs proposed in BiSNET and exploit MOGAs to adapt agent behavior policies. Unlike [35], BiSNET/e studies coevolution between DAs and EAs as well as their regular evolution (i.e., single-species evolution). [35] considers data collection applications only in static WSNs. Also, BiSNET/e performs adaptive mutation and crossover, which [35] does not consider.

Adaptive mutation was initially proposed in [36], and it has been used in WSNs [10, 23]. In [10, 23, 36], mutation rate is dynamically adjusted based on the current fitness that is a weighted sum of objective values. In MONSOON, mutation rate is adjusted based on the progress of performance improvement by the non-dominated individuals.

1.6 Conclusion

This chapter describes a coevolutionary multiobjective adaptation framework for WSNs, called MONSOON. MONSOON allows WSN applications to simultaneously satisfy conflicting operational objectives by adapting to dynamic network conditions (e.g., network traffic and node/link failures) through evolution. Thanks to a set of simple biologically-inspired mechanisms, the BiSNET/e runtime is implemented lightweight.

Some extensions to MONSOON and BiSNET/e are planed. The extensions include associating a constraint(s) with each operational objective. A constraint is defined as an upper or lower bound for each objective. For example, a tolerable (upper) bound may be defined for the latency objective. Constraints allow agent designers to flexibly specify their specific requirements (or priorities) on objectives. They can also improve evolution speed by dedicating agents to satisfy those constraints.

References

1. Akkaya, K., Younis, M.: A survey of routing protocols in wireless sensor networks. Elsevier Ad Hoc Networks **3**(3), 325–349 (2005)
2. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: A survey. Elsevier J. of Computer Networks **38**(4), 393–422 (2002)
3. Albert, R., Jeong, H., Barabasi, A.: Error and attack tolerance of complex networks. Nature **406**(6794), 378–382 (2000)
4. Baldi, P., Nardis, L.D., Benedetto, M.G.D.: Modeling and optimization of uwb communication networks through a flexible cost function. IEEE J. on Sel. Areas in Comm. **20**(9), 1733–1744 (2002)
5. Beegle-Krause, C.: General NOAA oil modeling environment (GNOME): A new spill trajectory model. In: Proc. of Int'l Oil Spill Conf. (2001)
6. Blumenthal, J., Handy, M., Golatowski, F., Haase, M., Timmermann, D.: Wireless sensor networks - new challenges in software engineering. In: Proc. of IEEE Emerging Technologies and Factory Automation (2003)
7. Boonma, P., Suzuki, J.: BiSNET: A biologically-inspired middleware architecture for self-managing wireless sensor networks. Elsevier J. of Computer Networks **51** (2007)
8. Boonma, P., Suzuki, J.: Evolutionary constraint-based multiobjective adaptation for self-organizing wireless sensor networks. In: Proc. of ACM/IEEE/Create-Net/ICST Int'l Conf. Bio-Inspired Models of Network, Info. and Comp. Sys. (2007)
9. Boonma, P., Suzuki, J.: Monsoon: A coevolutionary multiobjective adaptation framework for dynamic wireless sensor networks. In: Proc. of IEEE Hawaii Int'l Conf on System Sciences (2008)
10. Buczaka, A.L., Wangb, H.: Optimization of fitness functions with non-ordered parameters by genetic algorithms. In: Proc. of IEEE Congress on Evolutionary Comp. (2001)
11. Chintalapudi, K.K., Govindan, R.: Localized edge detection in sensor fields. Elsevier Ad-hoc Networks **1**, 59–70 (2003)
12. Ferentinos, K.P., Tsiligiridis, T.A.: Adaptive design optimization of wireless sensor networks using genetic algorithms. Elsevier J. of Computer Nets. **51**(4), 1031–1051 (2007)
13. Fok, C.L., Roman, G.C., Lu, C.: Rapid development and flexible deployment of adaptive wireless sensor network applications. In: Proc. of IEEE Int'l Conf. on Distributed Computing Systems (2005)
14. Free, J.B., Williams, I.H.: The role of the nasonov gland pheromone in crop communication by honey bees. Brill Int'l J. of Behavioural Biology **41**(3–4), 314–318 (1972)
15. Guo, H.Y., Zhang, L., Zhang, L.L., Zhou, J.X.: Optimal placement of sensors for structural health monitoring using improved genetic algorithms. IOP Smart Materials and Structures **13**(3), 528–534 (2004)
16. Han, Q., Mehrotra, S., Venkatasubramanian, N.: Energy efficient data collection in distributed sensor environments. In: Proc. of IEEE Int'l Conf. on Distributed Computing Systems (2004)
17. Hauser, J., Purdy, C.: Sensor data processing using genetic algorithms. In: Proc. of IEEE Midwest Symp. on Circuits and Systems (2000)
18. Hussain, S., Matin, A.W.: Hierarchical cluster-based routing in wireless sensor networks. In: Proc. of IEEE/ACM Conf. on Info. Processing in Sensor Nets (2006)
19. Jia, J., Chen, J., Chang, G., Tan, Z.: Energy efficient coverage control in wireless sensor networks based on multi-objective genetic algorithm. Elsevier Computers & Mathematics with Applications **10** (2008)
20. Jin, S., Zhou, M., Wu, A.S.: Sensor network optimization using a genetic algorithm. In: Proc. of IIIS World Multiconf. on Systemics, Cybernetics and Informatics (2003)
21. Jourdan, D.B., de Weck, O.L.: Multi-objective genetic algorithm for the automated planning of a wireless sensor network to monitor a critical facility. In: Proc. of SPIE Defense and Security Symp. (2004)
22. Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. Wiley-Interscience (2007)

23. Khanna, R., Liu, H., Chen, H.: Self-organisation of sensor networks using genetic algorithms. Inderscience Int'l J. of Sensor Networks **1**(3), 241–252 (2006)
24. Leibnitz, K., Wakamiya, N., Murata, M.: Biologically inspired networking. In: Q. Mahmoud (ed.) Cognitive Networks: Towards Self-Aware Networks. Wiley (2007)
25. Li, D., Wong, K.D., Hu, Y., Sayeed, A.M.: Detection, classification, and tracking of targets. IEEE Signal Processing Magazine **19**(2), 17–20 (2002)
26. Mathur, G., Desnoyers, P., Genesan, D., Shenoy, P.: Ultra-low power data storage for sensor networks. In: Proc. of IEEE/ACM Conf. on Info. Processing in Sensor Nets (2006)
27. Molina, G., Alba, E., Talbi, E.G.: Optimal sensor network layout using multi-objective metaheuristics. J. of Universal Computer Science **14**(15), 2549–2565 (2008)
28. Phoha, S., La Porta, T.F., Griffin, C.: Sensor Network Operations. Wiley-IEEE Press (2006)
29. Raich, A.M., Liszkai, T.R.: Multi-objective genetic algorithm methodology for optimizing sensor layouts to enhance structural damage identification. In: Proc. of Int'l Workshop on Structural Health Monitoring (2003)
30. Rajagopalan, R., Mohan, C., Varshney, P., Mehrotra, K.: Multi-objective mobile agent routing in wireless sensor networks. In: Proc. of IEEE Congress on Evolutionary Comp. (2005)
31. Rajagopalan, R., Varshney, P.K., Mehrotra, K.G., Mohan, C.K.: Fault tolerant mobile agent routing in sensor networks: A multi-objective optimization approach. In: Proc. of IEEE Upstate New York Workshop on Communication and Networking (2005)
32. Rajagopalan, R., Varshney, P.K., Mohan, C.K., Mehrotra, K.G.: Sensor placement for energy efficient target detection in wireless sensor networks: A multi-objective optimization approach. In: Proc. of IEEE Annual Conf. on Information Sciences and Systems (2005)
33. Rentala, P., Musunuri, R., Gandham, S., Sexena, U.: Survey on sensor networks. In: Proc. of ACM Int'l Conf. on Mobile Computing and Networking (2001)
34. Seeley, T.: The Wisdom of the Hive. Harvard University Press (2005)
35. Sin, H., Lee, J., Lee, S., Yoo, S., Lee, S., Lee, J., Lee, Y., , Kim, S.: Agent-based framework for energy efficiency in wireless sensor networks. World Academy of Science, Engineering and Technology **35**, 305–309 (2008)
36. Srinivas, M., Patnaik, L.: Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Tran. on Systems, Man and Cybernetics **24**(4), 656–667 (1994)
37. Szumel, L., Owens, J.D.: The virtual pheromone communication primitive. In: Proc. of IEEE Int'l Conf. on Distributed Computing in Sensor Systems (2006)
38. Tam, V., Cheng, K.Y., Lui, K.S.: Using micro-genetic algorithms to improve localization in wireless sensor networks. Academy J. of Comm. **1**(4), 1–10 (2006)
39. Wada, H., Boonma, P., Suzuki, J.: Macroprogramming spatio-temporal event detection and data collection in wireless sensor networks: An implementation and evaluation study. In: Proc. of IEEE Hawaii Int'l Conf on System Sciences (2008)
40. Xuea, F., Sanderson, A., Graves, R.: Multi-objective routing in wireless sensor networks with a differential evolution algorithm. In: Proc. of IEEE Int'l Conf. on Networking, Sensing and Control (2006)
41. Yang, E., Erdogan, A.T., Arslan, T., Barton, N.: Multi-objective evolutionary optimizations of a space-based reconfigurable sensor network under hard constraints. In: Proc. of ECSIS Symp. on Bio-inspired, Learning, and Intelligent Sys. for Security (2007)
42. Zhang, Q., Wang, J., Jin, C., Ye, J., Ma, C., Zhang, W.: Genetic algorithm based wireless sensor network localization. In: Proc. of IEEE Int'l Conf. on Natural Computation (2008)
43. Zhao, J., Wen, Y., Shang, R., Wang, G.: Optimizing sensor node distribution with genetic algorithm in wireless sensor network. In: Proc. of IEEE Int'l Symp. on Neural Nets. (2004)