Exploring Self-star Properties in Cognitive Sensor Networking

Pruet Boonma and Junichi Suzuki
Department of Computer Science
University of Massachusetts, Boston
{pruet, jxs}@cs.umb.edu

Abstract

Wireless sensor networks (WSNs) possess inherent tradeoffs among conflicting operational objectives such as data yield, data fidelity and power consumption. In order to address this challenge, this paper proposes a biologically-inspired framework to build cognitive WSN applications, which introspectively understand their conflicting objectives, find optimal tradeoffs with given constraints and autonomously adapt to dynamics of the network. The proposed framework, MONSOON, models an application as a decentralized group of software agents. This is analogous to a bee colony (application) consisting of bees (agents). Agents collect sensor data on individual nodes and carry the data to base stations. They perform this data collection functionality by autonomously sensing their local and surrounding network conditions and adaptively invoking biological behaviors such as pheromone emission, reproduction and migration. Each agent has its own behavior policy, as a gene, which defines how to invoke its behaviors. MONSOON allows agents to evolve their behavior policies via genetic operations such as crossover and mutation. Simulation results show that agents (WSN applications) exhibit the properties of self-configuration, self-optimization and self-healing and adapt to various dynamics of the network (e.g., node/link failures) by satisfying conflicting objectives under given constraints.

1. Introduction

Wireless sensor networks (WSNs) have inherent tradeoffs among conflicting operational objectives such as data yield, data fidelity and power consumption. For example, in data collection applications, hop-by-hop recovery is often applied for packet transmission in order to improve data yield (the quantity of collected data). However, this can degrade data fidelity (the quality of collected data; e.g., data freshness). For improving data fidelity, sensor nodes may transmit data to base stations with the shortest paths; however, data yield can degrade because of traffic congestion and packet losses on the paths.

In order to address this issue, the authors of the paper envision *cognitive* WSN applications that introspectively understand their conflicting objectives, find optimal tradeoffs under given constraints and autonomously adapt to dynamics of the network such as node/link failures. For making this vision a reality, this paper proposes a cognitive sensor networking framework, called MONSOON[1], which allows WSN applications to exhibit the following self-* properties:

- *Self-configuration*: allows WSN applications to automate their own configurations and self-organize into desirable structures and patterns (e.g., routing paths and duty cycles).
- *Self-optimization*: allows WSN applications to constantly seek improvement in their performance by adapting to changing network conditions with minimal human intervention.
- *Self-healing*: allows WSN applications to automatically detect and recover from disruptions in the network (e.g., node and link failures).

As an inspiration for the design strategy of MONSOON, the authors of the paper observe that various biological systems have developed the mechanisms necessary to realize the vision of MONSOON. For example, a bee colony self-organizes to satisfy conflicting objectives simultaneously for maintaining its well-being [8]. Those objectives include maximizing the amount of collected honey, maintaining the temperature in 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. Given this observation, MONSOON applies key biological mechanisms to implement cognitive WSN applications.

Figure 1 shows the architecture of MONSOON. The MONSOON runtime operates atop TinyOS on each node. It consists of two types of software components: *agents* and *middleware platforms*, which are modeled after bees and flowers, respectively. Each 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 on platforms (flowers) atop individual nodes, and carry the data to base stations on a hop-by-hop basis, in turn, to a backend server (the MONSOON server in Figure 1), which is modeled after a nest of bees.

Agents perform this data collection functionality by autonomously sensing their local and surrounding network conditions and adaptively invoking biological behaviors such as pheromone emission, replication, reproduction, migration and death. A middleware platform runs on each node, and hosts one or more agents (Figure 1). It provides a series of runtime services that agents use to perform their functionalities and behaviors.

MONSOON implements a constraint-based evolutionary adaptation mechanism for agents. Each agent has its own

[1] Multiobjective Optimization for a Network of Sensors using an evOlutionary algOrithm with coNstraints

behavior policy, as a *gene*, which defines when to and how to invoke its behaviors. MONSOON allows agents to evolve their behavior policies via genetic operations (mutation and crossover) and simultaneously adapt them to conflicting objectives with associated constraints. A constraint is defined as an upper or lower bound for an objective. For example, a tolerable (lower) bound may be defined for data fidelity. Currently, MONSOON considers six objectives related to data yield, data fidelity and power consumption.

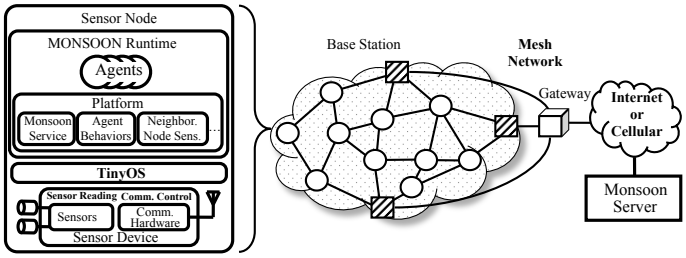


Figure 1. The Architecture of MONSOON

2. The MONSOON Runtime

MONSOON is currently designed to implement data collection applications. An agent is initially deployed with a randomly-generated behavior policy on each node. Each agent collects sensor data on a node periodically (i.e., at each duty cycle) and carry the data toward a base station.

2.1. Agent Behaviors

Each agent implements seven behaviors and performs them in the following sequence at each duty cycle.

Step 1: Energy gain. Each agent collects sensor data and gain *energy*. In MONSOON, the concept of energy does not represent the amount of physical battery in a node. It is a logical concept that impacts agent behaviors. Each agent updates its energy level with a constant energy intake (E_F):

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

$E(t)$ and $E(t-1)$ denote the energy levels in the current and previous duty cycles.

Step 2: Energy expenditure and death. Each agent consumes a constant amount of energy to use computing/networking resources available on a node (e.g., CPU and radio transmitter). It also expends energy to invoke its behaviors. The energy costs to invoke behaviors are constant for all agents. An agent dies due to energy starvation when it cannot balance its 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[2].

[2]If all agents are dying on a node at the same time, a randomly selected agent will survive. At least one agent runs on each node.

Step 3: Replication. Each agent makes a copy of itself in each duty cycle. A replicated (child) agent is placed on the node that its parent resides on, and it inherits the parent's behavior policy (gene). 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 denotes 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 on a hop by hop basis.

Step 4: Swarming. Agents may swarm (or merge) with others at intermediate nodes on their ways to base stations. On each intermediate node, each agent decides whether it migrates to a next-hop node or waits for other agents to arrive at the current node and swarm with them. This decision is made based on the migration probability (p_m). If an agent meets other agents during a waiting period, it merges with them and contains the sensor data they carry. It also uses the behavioral policy of the best one in the aggregating agents in terms of operational objectives. (See Section 3. on how to find the "best" agent.) The swarming behavior is intended to save power consumption by reducing the number of data transmissions. If the size of data an agent carries exceeds the maximum size of a packet, the agent does not consider the swarming behavior.

Step 5: Pheromone sensing and migration. On each intermediate node toward a base station, each agent chooses a migration destination node (next-hop node) by sensing three types of *pheromones* available on the local node: base station, migration and alert pheromones.

Each base station periodically propagates a base station pheromone to individual nodes in the network. Their concentration decays on a hop-by-hop basis. Using base station pheromones, agents can sense where base stations exist approximately, and move toward them by climbing a concentration gradient of base station pheromones.

Agents 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 can 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. The concentration decays by half at each duty cycle. A pheromone disapears when its concentration becomes zero.

Each agent examines Equation 2 to determine which next-hop node it migrates to.

$$WS_j = \sum_{t=1}^{3} w_t \frac{P_{t,j} - P_{t_{min}}}{P_{t_{max}} - P_{t_{min}}} \tag{2}$$

An agent 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.

The weight values in Equation 2 ($w_t, 1 \leq t \leq 3$) 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 a base station by climbing a 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. The trace can be the shortest path to a base station. Conversely, a negative w_2 value allows an agent to go off a migration pheromone trace and follow another path to a base station. 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. If an agent has a negative value for w_3, it avoids moving to a node referenced by an alert pheromone, thereby bypassing failed nodes and links.

Step 6: Pheromone emission. When an agent is migrating to a neighboring node, it emits a migration pheromone on the local node. If the agent's migration fails, it emits an alert pheromone. Each alert pheromone spreads to one-hop away neighboring nodes.

Step 7: Reproduction. Two parent agents may produce a child agent. A child agent is placed on the node that their parents reside on, and it inherits the parents' behavior policies (genes). This behavior is intended to evolve agents. (See Section 3. for more details.)

2.2. Agent Behavior Policy

Each behavior policy consists of two distinctive information: migration probability (p_m) and a set of weight values in Equation 2 ($w_t, 1 \leq t \leq 3$). Migration probability is a non-negative value between zero and one. It is used for each agent to decide whether it performs the migration behavior or swarming behavior. With a higher migration probability, an agent has a higher chance to perform the migration behavior instead of the aggregation behavior.

2.3. Middleware Platforms

Each middleware platform provides a set of runtime services for the agents running on the local host (Figure 1). For example, they implement agent behaviors as reusable services, maintain a set of neighboring nodes within the local node's communication range and manage the pheromones emitted on the local node. Also, each platform is responsible of controlling the local node's duty cycle by turning it on and off based on its sleep period. See [1] for full discussion on the design and implementation of platforms.

3. Evolution Process in MONSOON

In the evolution process in MONSOON, *elite selection* and *genetic operations* are performed in the MONSOON server (see Figure 1) and each node, respectively.

The elite selection process evaluates the agents that arrive at base stations, based on given operational 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 (gene) from its parents via crossover, and mutation may occur on the inherited behavior policy.

Reproduction is intended to evolve agents so that the agents that fit better to the environment become more abundant in the network. 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.

3.1. Operational Objectives

Each agent considers six conflicting objectives related to data yield, data fidelity and power consumption: latency, cost, success rate, the degree of data aggregation, sleep period and data accuracy. Success rate and the degree of data aggregation are related to data yield. Latency and data accuracy are related to date fidelity. Cost and sleep period are related to power consumption. MONSOON strives to minimize latency and cost and maximize success rate, the degree of data aggregation, sleep period and data accuracy.

(1) Latency represents the time required for an agent to travel to a base station from a node where the agent is replicated. It (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 replicated.

$$L = \frac{t}{d} \tag{3}$$

(2) Cost represents the amount of power consumption required for an agent to travel to a base station. It (C) is measured with d, each node's radio communication range (r), and the total number of node-to-node data transmissions required for an agent to arrive at a base station (n_{tran}).

$$C = \frac{n_{tran}}{r/d} \tag{4}$$

The total number of data transmissions counts successful and unsuccessful migrations of an agent as well as the transmissions of its migration and alert pheromones.

(3) Success rate (S) is measured as the ratio of the number of sensor data carried to base stations (n_{arrive}) to the total number of nodes in the network (N).

$$S = \frac{n_{arrive}}{N} \tag{5}$$

(4) Degree of data aggregation is measured as the number of sensor data in an agent. It is more than two in a swarming agent.

(5) Sleep period is the period for which a node is turned off between two duty cycles.

(6) Data accuracy (A) is measured based on the mean squared error between collected sensor data (s) and estimated sensor data ($\hat{s}$).

$$A = \frac{1}{\frac{1}{n_{arrive}} \sum_{j=1}^{n_{arrive}} (s_j - \hat{s}_j)^2} \tag{6}$$

$\hat{s}$ is calculated with a data prediction model using Autoregressive Integrated Moving Average (ARIMA). Higher data accuracy indicates either higher quality in estimated data or less noises in collected data.

3.2. Elite Selection

Figure 2 shows how the MONSOON server periodically performs elite selection. The first step is to measure six objective values of each agent that arrives at base stations. If an agent violates given constraints in at least one of six objectives, it is eliminated. Each of remaining agents is evaluated whether it is *dominated* by another one. It is considered to be dominated if another outperforms it in all six objectives.

```
Empty the archive
while true
do {
    Empty the population pool.
    Collect the agents that have arrived at base stations.
    Add collected agents to the population pool.
    Move agents from the archive to the population pool.
    Empty the archive
    for each agent of in the population pool
        do {
            Obtain the agent's objective values.
            if One or more objective values violate
             given constraints.
              then Remove the agent from the population pool.
    for each agent 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, mutation rate and sleep period
     to individual nodes in the network.
    Sleep for the sleep period.
}
```

Figure 2. Elite Selection in MONSOON

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

Figure 3 shows an example hypercube that shows three objectives (success rate, cost and latency). It is divided to two ranges for each objective; eight cubes exist in total. In this example, six (A to F) non-dominated agents are plotted. Three agents (B, C, and D) are plotted in the 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 are in different cubes.

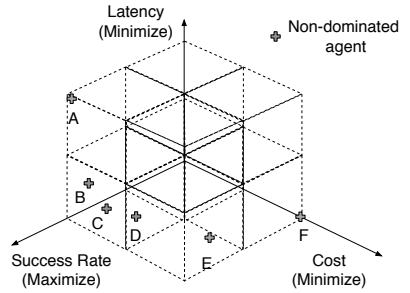


Figure 3. An Example Elite Selection

As Figure 2 shows, the MONSOON server performs elitism based on a (μ, λ) evolution strategy. Non-dominated agents in the current duty cycle will compete with (or will be evaluated together with) a set of agents that arrive at base stations in the next duty cycle.

In addition to select elite agents, the MONSOON server adjusts the mutation rate of agents and the sleep period of nodes. Mutation rate is adjusted based on how fast non-dominated agents improve their performance (objective values). The smaller improvement they make, the higher mutation rate the MONSOON server assigns to agents, thereby accelerating agent evolution.

The performance of non-dominated agents is measured as a set of performance representative points in different objectives. Equation 7 shows how to obtain a performance representative point ($\bar{o}_i$) in each objective i.

$$\bar{o}_i = \frac{\sum_{a \in A_{nd}} o_i(a)}{|A_{nd}|} \tag{7}$$

A_{nd} denotes the set of non-dominated agents, and $o_i(a)$ denotes the objective value that a non-dominated agent a yields in an objective i.

The improvement of performance is measured as the normalized Euclidean distance (d) between the current and the Utopian point ($\bar{o}_i(t)$ and $\bar{o}_{U_i}$)) in each objective.

$$d = \sqrt{\frac{\sum_{i \in O} (\bar{o}_i(t) - \bar{o}_{U_i})^2}{|O|}} \quad (8)$$

O is the set of all objectives. $\bar{o}_{U_i}$ exists around the ideal value in objective i; it is a slightly smaller or larger point from the possible minimum or maximum objective value.

Mutation rate (m) is adjusted with Equation 9 where k is the maximum mutation rate. c is a constant (≥ 1).

$$m = k \frac{\ln(d) + c}{c} \quad (9)$$

Sleep period is adjusted in a stepwise manner in between a predefined minimum and maximum values. If data accuracy (A) is lower than a given constraint, sleep period is decreased by one minute; otherwise, increased by one minute.

The MONSOON server propagates adjusted mutation rate and sleep period as well as elite agents to individual nodes in the network. This propagation is performed with a base station pheromone. When a node receives an adjusted sleep period, it changes its current sleep period accordingly.

3.3. Genetic Operations

Upon receiving a base station pheromone, an agent running on each node performs the reproduction behavior with a certain reproduction rate. It selects one of propagated elite agents, as a mating partner, which has the most similar gene (behavior policy). Gene similarity is measured with the Euclidean distance between the values of two genes. 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.

Mutation occurs on a child agent's gene, with a certain mutation rate, by randomly changing gene values within a predefined value range. As described in the previous section, mutation rate is periodically adjusted by the MONSOON server and propagated to individual nodes. After reproduction, a child agent takes over the local parent as the next generation agent.

4. Simulation Results

This section shows simulation results to evaluate MONSOON in terms of self-configuration, self-optimiztion and self-healing. The MONSOON server is implemented in Java, and the MONSOON runtime is implemented in nesC. All simulations were run with the TOSSIM simulator [6].

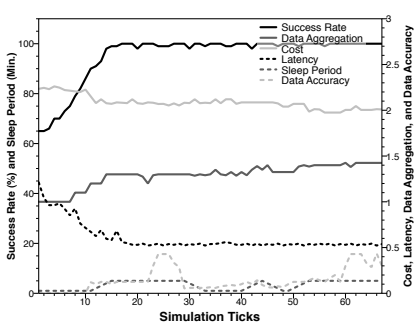
The network consists of 100 nodes deployed uniformly in a 300x300 meters observation area. Each node's communication range is 30 meters. A base station is deployed on the northwestern corner of the observation area. The base station connects to the MONSOON server via emulated serial port connection. The initial sleep period is five minutes, and its minimum and maximum is 1 and 10 minutes, respectively. For genetic operations, the reproduction rate is 0.9, and maximum mutation rate (k in Equation 9) is 0.25. c in Equation 9 is set to 5. The constraint (lower bound) of data accuracy is 0.1. To measure data accuracy, a data prediction model is configured as ARIMA(4, 1, 1); using four prior data, one random term and first-order differential. Each sensor data is affected by Gaussian noises.

4.1. Results with a Single Constraint

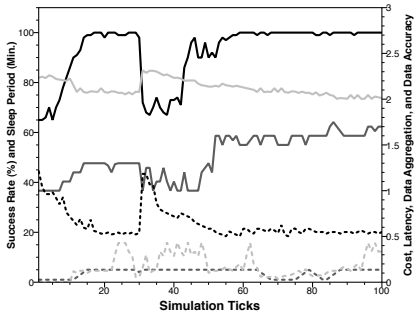
Figure 4 shows the average objective values that agents yield with a constraint for data accuracy. Each simulation tick represents a duty cycle. Figure 4 (a) shows a result in the static network, in which node/link failures never occur. Each objective value improves and converges around the 20th tick. This result shows that, through evolution, agents self-configure their behavior policies and self-optimize their performance against conflicting objectives. Please note that agents satisfy a data accuracy constraint after the 10th tick.

Figures 4 (b), (c), (d) and (e) show how agents perform against dynamic changes in the network. Upon each change that occurs at the 30th tick, objective values drops. In Figures 4 (b), when 25 nodes are added at random locations, objective values degrade because agents initially have random behavior policies on new nodes. Those agents cannot migrate efficiently to the base station. Also, enough pheromones are not available on new nodes when they are deployed; agents cannot make proper migration decisions on those nodes. In Figures 4 (c), randomly-selected 25 nodes fail. As a result, objective values drop because some agents try to migrate to failed nodes. In Figure 4 (d), objective values degrade when 20 nodes fail at the center of the observation area. (This means the network has a hole at its center.) In Figure 4 (e), two base stations are initially deployed at the northwestern and southeastern corners of the observation area. Then, a base station fails at the southeastern corner. Consequently, objective values drop because some agents try to migrate to the failed base station.

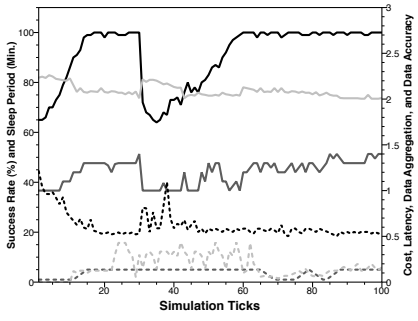
Once objective values drop due to a dynamic change in the network, agents gradually improve and converge their objective values again. Objective values are mostly same before and after each dynamic change. In Figures 4 (b), agents yield a higher degree of data aggregation after a dynamic node addition because there are more agents migrating in the network and there are higher chances for them to aggregate. Figures 4 (b), (c), (d) and (e) show that MONSOON allows agents to posses a self-healing property; agents autonomously detect and recover from dynamic changes in the network. Despite those changes, agents self-configure their behavior policies and self-optimize their performance by evolving their behavior policies. Also, agents always satisfy a data accuracy constraint after the 10th tick.



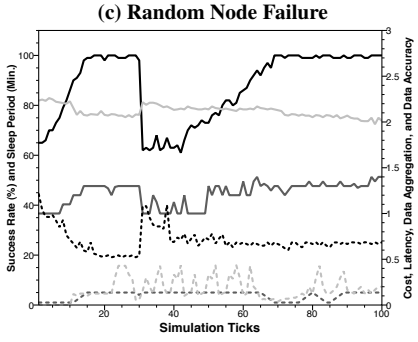
(a) Static Network



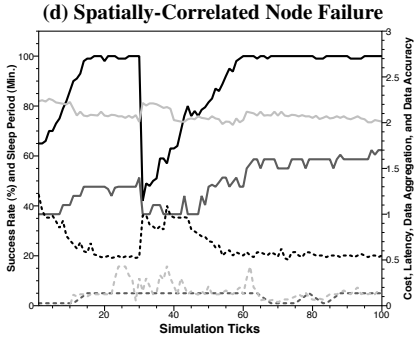
(b) Node Addition



(c) Random Node Failure

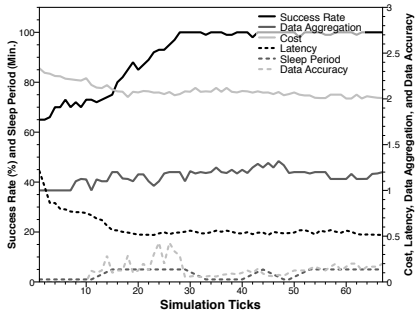


(d) Spatially-Correlated Node Failure

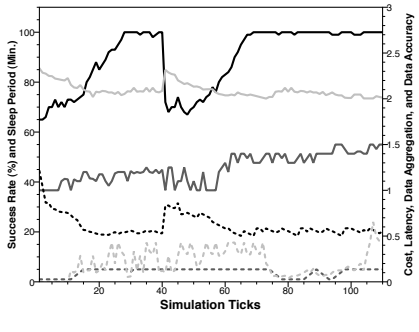


(e) Base Station Failure

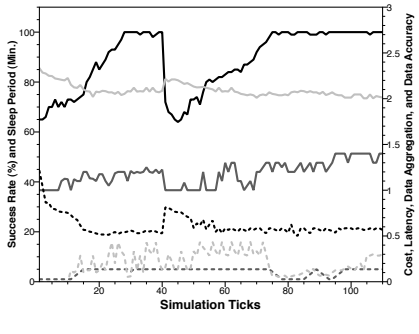
Figure 4. Performance with a Data Accuracy Constraint



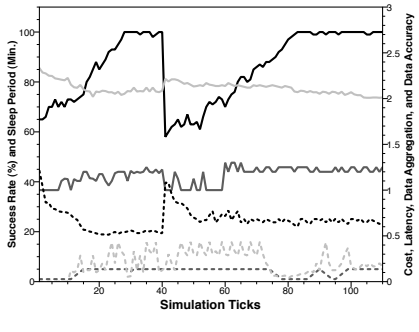
(a) Static Network



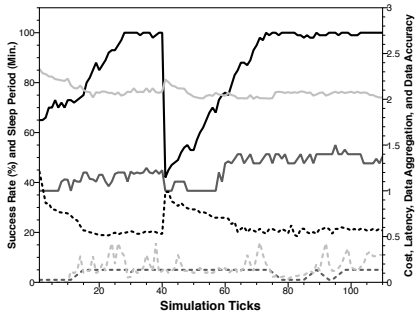
(b) Node Addition



(c) Random Node Failure

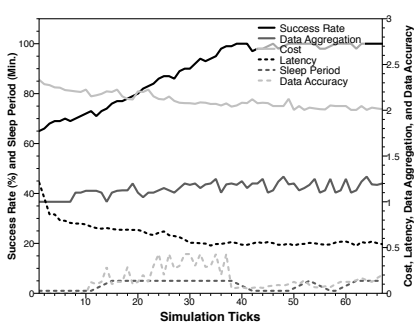


(d) Spatially-Correlated Node Failure

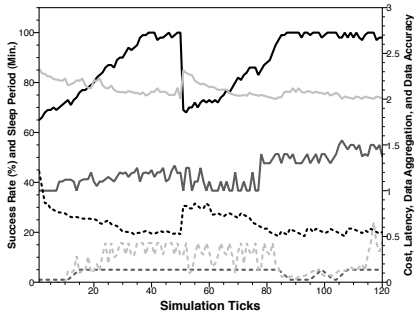


(e) Base Station Failure

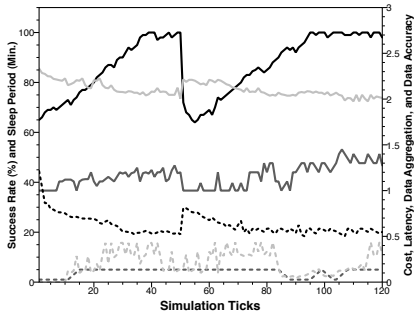
Figure 5. Performance with Data Accuracy, Latency and Cost Constraints



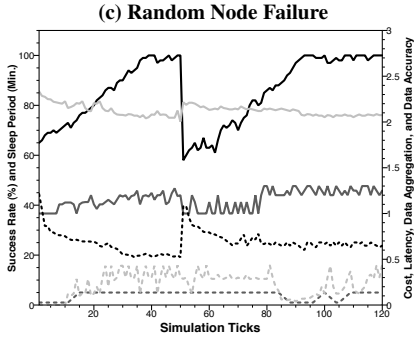
(a) Static Network



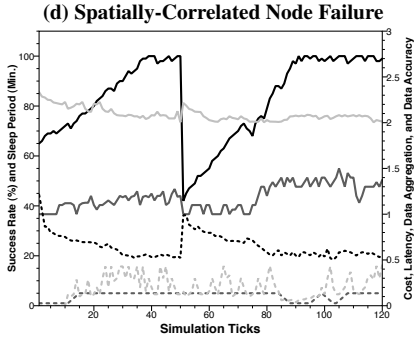
(b) Node Addition



(c) Random Node Failure



(d) Spatially-Correlated Node Failure



(e) Base Station Failure

Figure 6. Performance with Adaptive Mutation Disabled

4.2. Results with Multiple Constraints

Figure 5 shows the average objective values that agents yield with three constraints for data accuracy, latency and cost. Latency and cost are expected to be lower than 0.05 second per 30 meters and 2.05 transmissions per 30 meters, respectively. All other simulation configurations are same as the ones used for Figure 4.

In Figure 5 (a), data accuracy, latency and cost improve faster than the other three objectives because agents focus on satisfying given three constraints. In fact, latency and cost improve faster and remain more stable than they do in Figure 4 (a). They are always around or lower than their constraints (0.05 and 2.05) after the 20th tick, and data accuracy is always higher than its constraint (0.1) after the 10th tick. Figure 5 (a) shows that agents simultaneously satisfy conflicting objectives under given constraints by self-optimizing their behavior policies through evolution.

In Figures 5 (b), (c) and (d), agents perform similarly to Figures 4 (b), (c) and (d) in that they exhibit a self-healing property against dynamics of the network. Object values converge again after each failure. Compared with Figures 4 (b), (c) and (d), agents recover their latency and cost performance faster and retain the performance more stable by following given constraints. These results show that agents to evolve and simultaneously satisfy conflicting objectives in dynamic networks.

4.3. Impacts of Adaptive Mutation

In order to evaluate the impacts of adaptive mutation (Equation 9) Figure 6 shows the average objective values that agents yield with adaptive mutation disabled. Their mutation rate is fixed at 0.2. All other simulation configurations are same as the ones used for the Figures 5.

Compared with Figure 5, all objective values improve slower and fluctuate more in Figure 6. For example, in Figure 6 (b), it takes about 35 simulation ticks for success rate to reach 100% while it takes only 25 ticks in Figures 5 (b). Simulation results show that adaptive mutation allows agents to perform more efficiently and stably.

4.4. Mutation Rate, Power Consumption and Self-organization

Figures 7 shows the average amount of power that nodes consume as well as the average mutation rate that agents have. All simulation configurations are same as the ones used for the Figures 5. As Figures 7 shows, mutation rate dynamically adjusted lower by the MONSOON server as agents adapt to network conditions. This allows them to stabilize the fluctuations in their performance, as discussed in the previous section.

Power consumption decreases as agents adapt to network conditions and sleep period increases. When a failure occurs, the degree of agent adaptation drops. Also, the sleep period of nodes decreases because the data prediction model in MONSOON fails to estimate sensor data. As a result, power consumption increases upon a failure. However, power consumption goes down again soon. Figures 7 shows that MONSOON strives to minimize power consumption.

Figures 7 also shows the degree of self-orgnization in agent population. It is measured with the notion of *entropy*. In this paper, entropy indicates how similar performance different agents yield. The lower entropy, the more similar performance they yield. Entropy (E) is given as follows.

$$E = \sum_{i \in S} p_i \log(p_i) \tag{10}$$

i identifies individual cubes in the hypercube space in MONSOON. (See also Section 3 and Figure 3.2.), and S denotes the set of all cubes. p_i denotes the probability in which agents are plotted in the cube i. Figures 7 shows agents gradually decrease their entropy. This means they adapt to network conditions and perform similar with each other. Entropy spikes upon a failure; however, it goes lower again through agent evolution.

4.5. Memory Footprint

The MONSOON runtime is implemented lightweight. Its memory footprint is 3.1 KB in RAM and 32 KB in ROM on a MICA2 mote. (The capacity of a MICA2 mote is 4 KB in RAM and 128 KB in ROM.) It can run on a smaller-scale node, for example, a TelosB mote, which has a 48 KB ROM. The MONSOON runtime is more lightweight than Agilla, which is a mobile agent platform for mesh networks. Agilla consumes 3.59 KB in RAM and 41.6 KB in ROM [5].

5. Related Work

This work is an extension to the authors' prior work [2]. This paper considers three additional objectives and investigates adaptive mutation and constraints in agent evolution, which were not studied in [2].

Genetic algorithms are applied to several aspects in WSNs, such as cluster-based routing [4], localization [9] and node placement [10]. Every work uses a fitness function that aggregates multiple objective values as a weighted sum, and uses the function to rank agents/genes in elite selection. Application designers need to manually configure every weight value in a fitness function through trial and errors. In MONSOON, no manually-configured parameters exist for elite selection because of domination ranking. MONSOON imposes much less configuration cost. Also, [3, 4, 9, 10] do not assume dynamic networks, but static networks.

Evolutionary multiobjective optimization algorithms are used for node placement [7] in WSNs. In each of these work, an optimization process is performed only in a central server. This can lead to a scalability issue as the network size increases. In contrast, MONSOON is carefully de-

signed to perform its adaptation process in both the MONSOON server and individual nodes.

6. Conclusion

This paper proposes and evaluates a cognitive sensor networking framework, called MONSOON, for data collection applications in WSNs. MONSOON allows applications to introspectively understand given objectives and constraints, adapt to dynamic network conditions in a self-configuring, self-optimizing and self-heling manner, and simultaneously satisfy conflicting objectives. MONSOON is implemented lightweight and power efficient.

References

[1] P. Boonma and J. Suzuki. BiSNET: A biologically-inspired middleware architecture for self-managing wireless sensor networks. *Elsevier J. of Computer Networks*, 51, 2007.

[2] P. Boonma and J. Suzuki. Monsoon: A coevolutionary multi-objective adaptation framework for dynamic wireless sensor networks. In *Proceeding of Hawaii Int'l Conf on System Sciences*, January 2008.

[3] A. L. Buczaka and H. Wangb. Optimization of fitness functions with non-ordered parameters by genetic algorithms. In *Proc. of IEEE Congress on Evolutionary Comp.*, 2001.

[4] K. P. Ferentinos and T. A. Tsiligiridis. Adaptive design optimization of wireless sensor networks using genetic algorithms. *Elsevier J. of Computer Nets.*, 51(4), 2007.

[5] C. L. Fok, G. C. Roman, and C. Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Proc. of 25th IEEE Int'l Conf. on Distributed Computing Systems*, June 2005.

[6] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proc. of Int'l Conf. on Embedded Network Sensor System*, 2003.

[7] R. Rajagopalan, P. K. Varshney, C. K. Mohan, and K. G. Mehrotra. Sensor placement for energy efficient target detection in wireless sensor networks: A multi-objective optimization approach. In *Proc. of Annual Conf. on Information Sciences and Systems*, 2005.

[8] T. Seeley. *The Wisdom of the Hive*. Harvard University Press, 2005.

[9] V. Tam, K. Y. Cheng, and K. S. Lui. Using micro-genetic algorithms to improve localization in wireless sensor networks. *J. of Comm., Academy Publisher*, 1(4), 2006.

[10] J. Zhao, Y. Wen, R. Shang, and G. Wang. Optimizing sensor node distribution with genetic algorithm in wireless sensor network. In *Proc. of Int'l Symp. on Neural Nets.*, 2004.

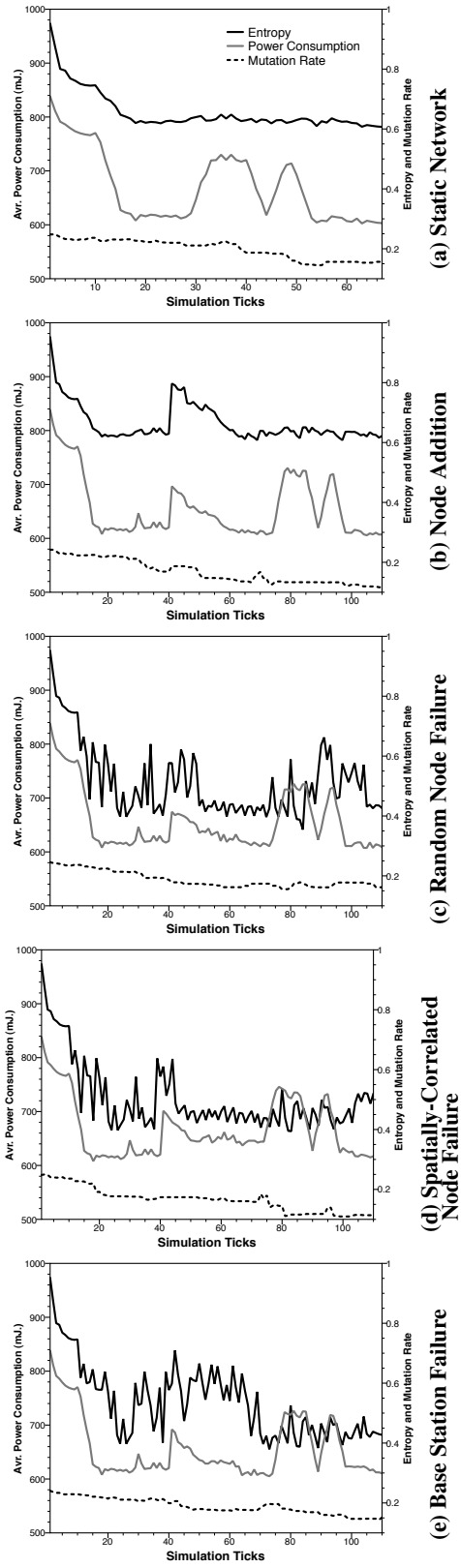


Figure 7. Mutation Rate, Power Consumption and Entropy