Online Algorithms for Wireless Sensor Networks Dynamic Optimization

Arslan Munir and Ann Gordon-Ross
Department of Electrical and Computer Engineering
University of Florida, Gainesville, Florida
Email: amunir@ufl.edu, ann@ece.ufl.edu

Susan Lysecky and Roman Lysecky
Department of Electrical and Computer Engineering
University of Arizona, Tucson, Arizona
Email: {slysecky, rlysecky}@ece.arizona.edu

***Abstract*—Technological advancements in wireless communications and embedded systems have led to the proliferation of wireless sensor network (WSN) applications, each with varying application requirements (i.e., lifetime, throughput, reliability, etc.). Sensor node tunable parameters enable WSN designers to specialize/tune a sensor node to meet application requirements, but however, parameter tuning is a challenging process that requires designer expertise to consider sensor node complexities and changing environmental stimuli. In this paper, we develop lightweight, online optimization algorithms for sensor node parameter tuning, which enables dynamic optimizations to meet application requirements and adapt to changing environmental stimuli. Results reveal that our online optimizations quickly converge to a near optimal solution using minimal computational and storage resources, and are thus amenable for implementation on resource and energy-constrained sensor nodes.**

***Index Terms*—Wireless sensor networks; dynamic optimization; lightweight; low-power;**

I. Introduction and Motivation

A wireless sensor network (WSN) typically consists of a set of spatially distributed sensor nodes that wirelessly communicate with each other to collectively accomplish an application specific task. Due to technological advancements in wireless communications and embedded systems, there exists a plethora of WSN applications, including security/defense systems, industrial automation, health care, and logistics.

Given the wide range of WSN applications, an application designer is left with the challenging task of designing a WSN while taking into consideration *application requirements* (lifetime, throughput, reliability, etc.). Moreover, these application requirements are affected by environmental stimuli (e.g., poor wireless channel conditions may necessitate increased transmission power) and can change over time as operational situations evolve (e.g., unexpected winds fuel a dying forest fire). Since commercial off-the shelf (COTS) sensor nodes have limited resources (i.e., battery lifetime, processing power, etc.), delicate design and tradeoff considerations are necessary to meet often competing application requirements (e.g., high processing requirements with long lifetime requirements).

In order to meet a wide range of application requirements, COTS sensor nodes are generically designed, but however, *tunable parameters* (e.g., processor voltage, processor frequency, sensing frequency, radio transmission power, packet size, etc.) enable the sensor node to *tune* operation to meet application requirements. Nevertheless, application designers are left with the task of *parameter tuning* during WSN design time. Parameter tuning is the process of assigning appropriate values for sensor node tunable parameters in order to meet application requirements. Parameter tuning involves several challenges such as optimal parameter value selection given large design spaces, consideration for competing application requirements and tunable parameters, difficulties in creating accurate simulation environments, slow simulation times, etc. In addition, design time static determination of these parameters leaves the sensor node with little or no flexibility to adapt to the actual operating environment. Furthermore, many application designers are non-experts (e.g., agriculturist, biologists, etc.) and lack sufficient expertise for parameter tuning. Therefore, autonomous parameter tuning methodologies may alleviate many of these design challenges.

Dynamic optimizations enable autonomous sensor node parameter tuning using special hardware/software algorithms to determine parameter values in situ according to application requirements and changing environmental stimuli. Dynamic optimizations require minimal application designer effort and enable application designers to specify only high-level application requirements without knowledge of parameter specifics. Nevertheless, dynamic optimizations rely on fast and lightweight online optimization algorithms for in situ parameter tuning.

The dynamic profiling and optimization project aspires at alleviating the complexities associated with sensor-based system design through the use of dynamic profiling methods capable of observing application-level behavior and dynamic optimization to tune the underlying platform accordingly [1]. The dynamic profiling and optimization project has evaluated dynamic profiling methods for observing application-level behavior by gathering profiling statistics, but dynamic optimization methods still need exploration. In our previous work [2], we proposed a Markov Decision Process (MDP)-based methodology to prescribe optimal sensor node operation to meet application requirements and adapt to changing environmental stimuli. However, the MDP-based policy was not autonomous because the methodology required the application designer to orchestrate MDP-based

policy reevaluation whenever application requirements and environmental stimuli changed. In addition, since policy reevaluation was computationally and memory expensive, this process was done offline on a powerful desktop machine.

To enable in situ autonomous WSN dynamic optimizations, we propose an online WSN optimization methodology which extends static design time parameter tuning [3]. Our methodology is advantageous over static design time parameter tuning because our methodology enables the sensor node to automatically adapt to actual changing environmental stimuli, resulting in closer adherence to application requirements. Furthermore, our methodology is more amenable to non-expert application designers and requires no application designer effort after initial WSN deployment. Lightweight (low computational and memory resources) online algorithms are crucial for sensor nodes considering limited processing, storage, and energy resources of sensor nodes. Our online lightweight optimization algorithms impart fast design space exploration to yield an optimal or near optimal parameter value selection.

II. Related Work

There exists much research in the area of dynamic optimizations [4][5][6][7], but however, most previous work focuses on the processor or memory (cache) in computer systems. Whereas these endeavors can provide valuable insights into WSN dynamic optimizations, they are not directly applicable to WSNs due to different design spaces, platform particulars, and a sensor node's tight design constraints.

In the area of WSN dynamic profiling and optimizations, Sridharan et al. [8] obtained accurate environmental stimuli by dynamically profiling the WSN's operating environment, but however, did not propose any methodology to leverage these profiling statistics for optimizations. In our previous work [2], we proposed an automated Markov Decision Process (MDP)-based methodology to prescribe optimal sensor node operation to meet application requirements and adapt to changing environmental stimuli. Kogekar et al. [9] proposed an approach for dynamic software reconfiguration in WSNs using dynamically adaptive software, which used tasks to detect environmental changes (event occurrences) and adapt the software to the new conditions. Their work did not consider sensor node tunable parameters.

Several papers explored dynamic voltage and frequency scaling (DVFS) for reduced energy consumption in WSNs. Min et al. [10] demonstrated that dynamic processor voltage scaling reduced energy consumption by 60%. Similarly, Yuan et al. [11] studied a DVFS system that used additional transmitted data packet information to select appropriate processor voltage and frequency values. Although DVFS provides a mechanism for dynamic optimizations, considering additional sensor node tunable parameters increases the design space and the sensor node's ability to meet application requirements. To the best of our knowledge, our work is the first to explore an extensive sensor node design space.

Some previous works in WSN optimizations explore greedy and simulated annealing (SA)-based methods, but these previous works did not analyze execution time and memory requirements. Huber et al. [12] maximized the amount of data gathered using a distributed greedy scheduling algorithm that aimed at determining an optimal sensing schedule, which consisted of a time sequence of scheduled sensor node measurements. In prior work, Lysecky et al. [3] proposed an SA-based automated application specific tuning of parameterized sensor-based embedded systems and found that automated tuning can improve WSN operation by 40% on average. Verma [13] studied SA and particle swarm optimization (PSO) methods for automated application specific tuning and observed that SA performed better than PSO because PSO often quickly converged to local minima.

Although previous works in WSN optimizations explore greedy and SA-based methods, these previous works did not analyze execution time and memory requirements. Furthermore, the previous works did not investigate greedy and SA algorithms as online algorithms for dynamic optimizations. Prior work [3][13] considered a limited design space with a few sensor node tunable parameters. To address the deficiencies in previous work, we analyze greedy and SA algorithms as online algorithms for performing dynamic optimizations considering a large design space containing many tunable parameters and values. This fine-grained design space enables sensor nodes to more closely meet application requirements, but exacerbates optimization challenges considering a sensor node's constrained memory and computational resources.

III. Dynamic Optimization Methodology

In this section, we give an overview of our dynamic optimization methodology. We also formulate the state space, objective function, and online lightweight optimization algorithms/heuristics for our dynamic optimization methodology.

A. Methodology Overview

Fig. 1 depicts our dynamic optimization methodology. The application designer specifies application requirements using high-level application metrics (e.g., lifetime, throughput, reliability), associated minimum and maximum desired/acceptable values, and associated *weight factors* that specify the importance of each high-level metric with respect to each other.

The shaded box in Fig. 1 depicts the overall operational flow, orchestrated by the *dynamic optimization controller*, at each sensor node. The dynamic optimization controller receives application requirements and invokes the *dynamic optimization module*. The dynamic optimization module determines the sensor node's operating *state* (tunable parameter value settings) using an online optimization algorithm. The sensor node will operate in that state until a state change is necessary. State changes occur to react to

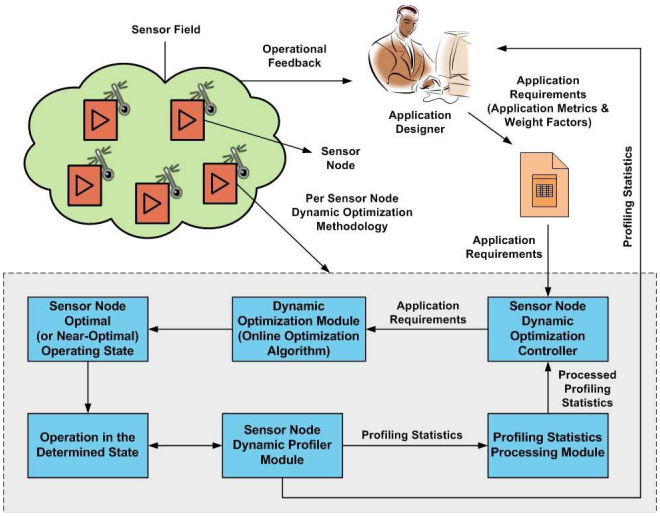

Fig. 1. Dynamic optimization methodology for WSNs.

changing environmental stimuli using the *dynamic profiler module* and *profiling statistics processing module*.

The dynamic profiler module records profiling statistics (e.g., wireless channel condition, number of dropped packets, packet size, radio transmission power, etc.) and the profiling statistics processing module performs any necessary data processing. The dynamic optimization controller evaluates the processed profiling statistics to determine if the current operating state meets the application requirements. If the application requirements are not met, the dynamic optimization controller reinvokes the dynamic optimization module to determine a new operating state. This feedback process continues to ensure the selection of an appropriate operating state to best meet the application requirements. Currently, our online algorithms do not directly consider these profiling statistics, but that incorporation is the focus of our future work.

B. State Space

The state space S for our dynamic optimization methodology is defined as:

$$S = S_1 \times S_2 \times \cdots \times S_N \tag{1}$$

where N denotes the number of tunable parameters, S_i denotes the state space for tunable parameter i, $\forall\ i \in \{1, 2, \ldots, N\}$, and $\times$ denotes the Cartesian product. Each tunable parameter's state space S_i consists of n values:

$$S_i = \{s_{i_1}, s_{i_2}, s_{i_3}, \ldots, s_{i_n}\} \quad : \quad |S_i| = n \tag{2}$$

where $|S_i|$ denotes the tunable parameter i's state space cardinality (the number of tunable values in S_i). S is a set of n-tuples where each n-tuple represents a sensor node state s. Note that some n-tuples in S may not be feasible (e.g., all processor voltage and frequency pairs are not feasible) and can be regarded as *do not care* tuples.

C. Objective Function

The sensor node dynamic optimization problem can be formulated as:

$$\begin{aligned} \max \quad & f(s) \\ s.t. \quad & s \in S \end{aligned} \tag{3}$$

where $f(s)$ represents the objective function and captures application requirements and can be given as:

$$\begin{aligned} f(s) \quad &= \quad \sum_{k=1}^{m} \omega_k f_k(s) \\ s.t. \quad & s \in S \\ & \omega_k \geq 0, \quad k = 1, 2, \ldots, m. \\ & \omega_k \leq 1, \quad k = 1, 2, \ldots, m. \\ & \sum_{k=1}^{m} \omega_k = 1, \end{aligned} \tag{4}$$

where $f_k(s)$ and ω_k denote the objective function and weight factor for the k^{th} application metric, respectively, given that there are m application metrics. Our objective function characterization considers lifetime, throughput, and reliability, i.e., $m = 3$ (additional application metrics can be included) and is given as:

$$f(s) = \omega_l f_l(s) + \omega_t f_t(s) + \omega_r f_r(s) \tag{5}$$

where $f_l(s)$, $f_t(s)$, and $f_r(s)$ denote the lifetime, throughput, and reliability objective functions, respectively, and ω_l, ω_t, and ω_r denote the weight factors for lifetime, throughput, and reliability, respectively.

We consider piecewise linear objective functions for lifetime, throughput, and reliability [3][13]. We define the lifetime objective function (Fig. 2) in (5) as:

$$f_l(s) = \begin{cases} 1, & s_l \geq \beta_l \\ C_{U_l} + \frac{(C_{\beta_l} - C_{U_l})(s_l - U_l)}{(\beta_l - U_l)}, & U_l \leq s_l < \beta_l \\ C_{L_l} + \frac{(C_{U_l} - C_{L_l})(s_l - L_l)}{(U_l - L_l)}, & L_l \leq s_l < U_l \\ C_{L_l} \cdot \frac{(s_l - \alpha_l)}{(L_l - \alpha_l)}, & \alpha_l \leq s_l < L_l \\ 0, & s_l < \alpha_l. \end{cases} \tag{6}$$

where s_l denotes the lifetime offered by state s, the constant parameters L_l and U_l denote the desired minimum and maximum lifetime, and the constant parameters α_l and β_l denote the acceptable minimum and maximum lifetime. Using both desirable and acceptable values enable the application designer to specify the reward (gain) for operating in either a desired range or an acceptable range, where the reward gradient (slope) in the desired range would be greater than the reward gradient in the acceptable range, but however, there would be no reward for operating outside of the acceptable range. The constant parameters C_{L_l}, C_{U_l}, and C_{β_l} in (6) denote the lifetime objective function value at L_l, U_l, and β_l, respectively. The throughput and reliability objective functions can be defined similar to (6).

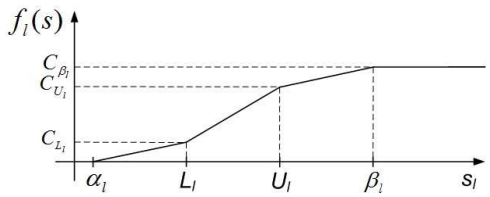

Fig. 2. Lifetime objective function $f_l(s)$.

D. Online Optimization Algorithms

In this subsection, we present our online optimization algorithms/heuristics for dynamic optimizations. We focus on two main online optimization algorithms, a greedy and an SA-based.

```
   Input: f(s), N, n
   Output: Sensor node state that maximizes f(s) and the
           corresponding f(s) value
1  μ ← initial tunable parameter values ;
2  objBestSol ← solution from state μ ;
3  foreach Sensor Node Tunable Parameter do
4      for i ← 1 to n do
5          objSolTemp ← current state β solution ;
6          if objSolTemp > objBestSol then
7              objBestSol ← objSolTemp ;
8              μ ← β ;
9          else
10             break ;
11         end
12     end
13     select the next tunable parameter ;
14 end
15 return μ, objBestSol
```

Algorithm 1: Greedy algorithm for sensor node dynamic optimization.

1) Greedy Algorithm: Algorithm 1 depicts our greedy algorithm, which takes as input the objective function $f(s)$ (5), the number of sensor node tunable parameters N, and each tunable parameter's design space cardinality n (the algorithm assumes the same state space cardinality for all tunable parameters for notational simplicity). The algorithm sets the initial state μ with initial tunable parameter values (line 1) and the best solution objective function value $objBestSol$ to the value obtained from the initial state μ (line 2). The algorithm explores each parameter in turn, starting from the last parameter (with state space S_N in (1)), while holding all other parameters fixed according to μ. For each parameter values (explored in ascending order) denoted as current state β, the algorithm computes the objective function value $objSolTemp$ (lines 4 and 5). If the current state results in an improvement in the objective function value (line 6), $objSolTemp$ and μ are updated to the new best state (lines 6 - 8). This process continues until there is no objective function value improvement ($objSolTemp < objBestSol$), at which point that parameter value is fixed (lines 9 - 11) and the next parameter is explored (e.g., S_{N-1} is explored after S_N (1)). After exploring all parameters (lines 3 - 14), the algorithm returns the best state μ and μ's objective function value $objBestSol$ (line 15).

In the greedy algorithm, the exploration order of the tunable parameters and parameter values (i.e., ascending or descending) can be governed by high-level metric weight factors to produce higher quality results and/or explore fewer states. For example, parameters with the greatest effect on a high-level metric with a high weight factor could be explored first. Currently, our greedy algorithm explores tunable parameter values in ascending order considering a generic WSN application assuming all weight factors to be equal (i.e., $\omega_l = \omega_t = \omega_r$ in (5)). However, our future work will explore specializing parameter exploration with respect to high-level metrics and weight factors.

```
   Input: f(s), N, n, T_0, α, c_0, t_0
   Output: Sensor node state that maximizes f(s) and the
           corresponding f(s) value
1  c, t, q ← 0 ;
2  μ ← rand()%N ;
3  T_q ← T_0 ;
4  objSolInit ← solution from state μ ;
5  objSolTemp ← objSolInit ;
6  objBestSol ← objSolInit ;
7  q ← q + 1 ;
8  while t < t_0 do
9      while c < c_0 do
10         if rand() > RAND_MAX/2 then
11             β ← |(μ + rand()%N)|%N ;
12         else
13             β ← |(μ − rand()%N)|%N ;
14         end
15         objSolNew ← new state β solution ;
16         if objSolNew > bestSol then
17             objBestSol ← objSolNew ;
18             ζ ← β ;
19         end
20         if objSolNew > objSolTemp then
21             P ← 1 ;
22         else
23             P ← exp((objSolNew − objSolTemp)/T_q) ;
24         end
25         rP ← rand()/RAND_MAX;
26         if P > rP then
27             objSolTemp ← objSolNew ;
28             ζ ← β ;
29         end
30         q ← q + 1 ;
31         c ← c + 1 ;
32     end
33     T_q ← α · T_q ;
34     t ← t + 1 ;
35     c ← 0 ;
36 end
37 return ζ, objBestSol
```

Algorithm 2: Simulated annealing algorithm for sensor node dynamic optimization.

2) Simulated Annealing Algorithm: Algorithm 2 depicts our SA algorithm, which takes as input the objective function $f(s)$ (5), the number of sensor node tunable parameters N, each tunable parameter's design space cardinality n, the SA initial temperature T_0, the cooling schedule scale factor α (which determines the annealing schedule), the number of trials c_0 performed at each temperature t_i, and the total number of temperature reductions t_0. The algorithm performs the following initializations (lines 1 – 6): the number of trials c at a given temperature t_i, the number of temperature reductions t,

and the number of states explored q are initialized to zero (line 1); the initial values for all tunable parameters μ where $|\mu| = N$ are set pseudo-randomly (line 2) [14]; the current annealing temperature T_q is initialized to T_0 (line 3); the initial state μ's objective function value is assigned to the variable $objSolInit$ (line 4); the current state objective function value $objSolTemp$ and the best state objective function value $objBestSol$ are initialized to $objSolInit$ (lines 5 and 6).

The algorithm begins the state exploration by first incrementing the number of states explored q from 0 to 1 (line 7). For each *trial* (lines 10 – 31), the algorithm explores new neighboring states β where $|\beta| = N$ pseudo-randomly (lines 10 – 14) in search of a better solution and calculates the resulting objective function value $objSolNew$ (line 15). If the new state β offers a higher objective function value as compared to the previous $objBestSol$, the new state becomes the best solution (lines 16 – 19), otherwise the algorithm determines the *acceptance probability* P of the new state being selected as the current state using the Metropolis-Hastings random walk algorithm (lines 20 – 29) [15]. At high temperatures, the Metropolis-Hastings algorithm accepts all moves (random walk) while at low temperatures, the Metropolis-Hastings algorithm performs stochastic hill-climbing (the acceptance probability depends on the difference between the objective function and the annealing temperature). At the end of each trial, the annealing temperature is decreased exponentially (line 33) and the process continues until $t \rightarrow t_0$ (lines 8 – 36). After all trials have completed, the algorithm returns the current best state ζ and the resulting objective function value $objBestSol$ (line 37).

The selection of the SA algorithm's parameters is critical in determining a good quality solution for sensor node parameter tuning. Specifically, the selection of the T_0 value is important because an inappropriate T_0 value may yield lower quality solutions. We propose to set T_0 equal to the maximal objective function difference between any two neighboring states (i.e., $T_0 = \max|\triangle f(s)|$ where $\triangle f(s)$ denotes the objective function difference between any two neighboring states). This proposition is an extension of T_0 selection based on the maximal energy difference between neighboring states [15][16]. However, it is not possible to estimate $\max|\triangle f(s)|$ between two neighboring states because SA explores the design space pseudo-randomly. We propose an approximation $T_0 \approx |\max|f(s)| - \min|f(s)||$ where $\min|f(s)|$ and $\max|f(s)|$ denote the minimum and maximum objective function values in the design space, respectively. The exhaustive search algorithm can be used to find $\min|f(s)|$ and $\max|f(s)|$ by minimizing and maximizing the objective function, respectively.

IV. Experimental Results

In this section, we describe our experimental setup and experimental results for greedy and simulated annealing (SA) algorithms for different application domains. These results evaluate the greedy and SA algorithms in terms of solution quality and the percentage of state space explored. We also present data memory, execution time, and energy results to provide insights into the complexity and energy requirements of our online algorithms.

A. Experimental Setup

Our experiments are based on the Crossbow IRIS motes [17] that operate using two AA alkaline batteries with a battery capacity of 2000 mA-h. This platform integrates an Atmel ATmega1281 microcontroller [18], an Atmel AT-86RF230 low power 2.4 GHz transceiver [19], and a MTS400 sensor board [20] with Sensirion SHT1x humidity and temperature sensors [21]. In order to investigate the fidelity of our online algorithms across small and large design spaces, we consider two design space cardinalities (number of states in the design space) $|S| = 729$ and $|S| = 31,104$. The state space $|S| = 729$ results from six tunable parameters with three tunable values each: processor voltage V_p = {2.7, 3.3, 4} (volts), processor frequency F_p = {4, 6, 8} (MHz) [18], sensing frequency F_s = {1, 2, 3} (samples per second) [21], packet transmission interval P_{ti} = {60, 300, 600} (seconds), packet size P_s = {41, 56, 64} (bytes), and transceiver transmission power P_{tx} = {-17, -3, 1} (dBm) [19]. The tunable parameters for $|S| = 31,104$ are V_p = {1.8, 2.7, 3.3, 4, 4.5, 5} (volts), F_p = {2, 4, 6, 8, 12, 16} (MHz) [18], F_s = {0.2, 0.5, 1, 2, 3, 4} (samples per second) [21], P_s = {32, 41, 56, 64, 100, 127} (bytes), P_{ti} = {10, 30, 60, 300, 600, 1200} (seconds), and P_{tx} = {-17, -3, 1, 3} (dBm) [19]. All state space tuples are feasible for $|S| = 729$, whereas $|S| = 31,104$ contains 7,779 infeasible state space tuples (e.g., all V_p and F_p pairs are not feasible).

We analyze three sample application domains: a security/defense system, a health care application, and an ambient conditions monitoring application. To model each application domain, we assign application specific values for the desirable minimum L, desirable maximum U, acceptable minimum α, and acceptable maximum β objective function parameter values for application metrics and associated weight factors. We specify the objective function parameters as a multiple of a base unit where one lifetime unit is equal to 5 days, one throughput unit is equal to 20 kbps, and one reliability unit is equal to 0.05 (percentage of error-free packet transmissions). We assign application metric values for an application considering the application's typical requirements [2]. For example, a health care application with a sensor implanted into a patient to monitor physiological data (e.g., heart rate, glucose level, etc.) may have a longer lifetime requirement because frequent battery replacement may be difficult. Table I depicts the application requirements in terms of objective function parameter values for the three application domains.

The lifetime, throughput, and reliability objective function values corresponding to the desirable minimum and maximum parameter values are 0.1 and 0.9, respectively and the objective function values corresponding to the acceptable minimum and maximum parameter values are 0 and 1, respectively.

TABLE I

DESIRABLE MINIMUM L, DESIRABLE MAXIMUM U, ACCEPTABLE MINIMUM α, AND ACCEPTABLE MAXIMUM β OBJECTIVE FUNCTION PARAMETER VALUES FOR A SECURITY/DEFENSE SYSTEM, HEALTH CARE, AND AN AMBIENT CONDITIONS MONITORING APPLICATION. ONE LIFETIME UNIT = 5 DAYS, ONE THROUGHPUT UNIT = 20 KBPS, ONE RELIABILITY UNIT = 0.05.

Notation	Security/Defense	Health Care	Ambient Monitoring
L_l	8 units	12 units	6 units
U_l	30 units	32 units	40 units
α_l	1 units	2 units	3 units
β_l	36 units	40 units	60 units
L_t	20 units	19 units	15 unit
U_t	34 units	36 units	29 units
α_t	0.5 units	0.4 units	0.05 units
β_t	45 units	47 units	35 units
L_r	14 units	12 units	11 units
U_r	19.8 units	17 units	16 units
α_r	10 units	8 units	6 units
β_r	20 units	20 units	20 units

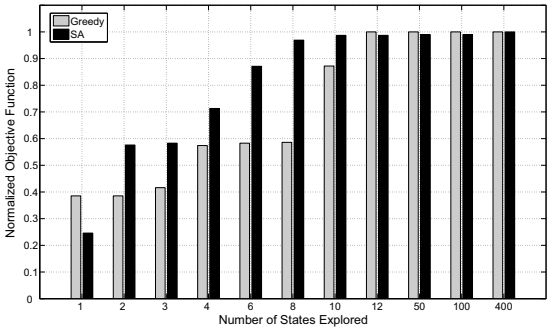

Fig. 3. Objective function value normalized to the optimal solution for a varying number of states explored for the greedy and simulated annealing algorithms for a security/defense system where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, $|S| = 729$.

For brevity, we selected a single sample WSN platform configuration and three application domains, but we point out that our dynamic optimization methodology and online optimization algorithms are equally applicable to any WSN platform and application.

B. Results

We implemented our greedy and SA-based online optimization algorithms in C++ and evaluated the algorithms in terms of the percentage of the design space explored, the quality (objective function value) of each algorithm's determined best state as compared to the optimal state determined using an exhaustive search, and the total execution time. We also performed data memory and energy analysis to analyze scalability for different design space sizes.

Fig. 3 shows the objective function value normalized to the optimal solution for the SA and greedy algorithms versus the number of states explored for a security/defense system for $|S| = 729$ where $\omega_l = 0.25$, $\omega_t = 0.35$, and $\omega_r = 0.4$. The SA parameters are calculated as outlined in Section III-D2 (e.g., $T_0 = |\max|f(s)| - \min|f(s)|| = |0.7737 - 0.1321| = 0.6416$ and $\alpha = 0.8$ [15]). Fig. 3 shows that the greedy and SA algorithms converged to a steady state solution after exploring 11 and 400 states, respectively. These convergence results show that the greedy algorithm converged to the final solution faster than the SA algorithm, exploring only 1.51% of the design space, whereas the SA algorithm explored 54.87% of the design space. The figure reveals that the average growth rate for increasing solution quality was faster in the initial iterations than in the later iterations. Fig. 3 shows an average growth rate of approximately 22.96% and 52.56% for the initial iterations for the greedy and SA algorithms, respectively, and decreased to 12.8% and 0.00322% for the later iterations of the greedy and SA algorithms, respectively. Both the algorithms converged to the optimal solution as was obtained from an exhaustive search of the design space.

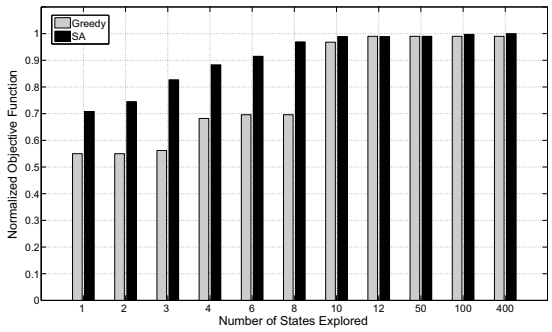

Fig. 4. Objective function value normalized to the optimal solution for a varying number of states explored for the greedy and simulated annealing algorithms for a health care application where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, $|S| = 729$.

Fig. 4 shows the objective function value normalized to the optimal solution for the SA and greedy algorithms versus the number of states explored for a health care application for $|S| = 729$ where $\omega_l = 0.25$, $\omega_t = 0.35$, and $\omega_r = 0.4$. The SA parameters are $T_0 = |0.7472 - 0.2254| = 0.5218$ and $\alpha = 0.8$ [15]. Fig. 4 shows that the greedy and SA algorithms converged to a steady state solution after exploring 11 states (1.51% of the design space) and 400 states (54.87% of the design space), respectively. The SA algorithm converged to the optimal solution after exploring 400 states whereas the greedy algorithm's solution quality after exploring 11 states was within 0.027% of the optimal solution. Fig. 4 shows an average growth rate of approximately 11.76% and 5.22% for the initial iterations for the greedy and SA algorithms, respectively, and decreased to 2.27% and 0.001% for the later iterations of the greedy and SA algorithms, respectively.

Fig. 5 shows the objective function value normalized to the optimal solution for the SA and greedy algorithms versus the number of states explored for an ambient condition monitoring application for $|S| = 31,104$ where $\omega_l = 0.6$, $\omega_t = 0.25$, $\omega_r = 0.15$. The SA parameters are $T_0 = |0.8191 - 0.2163| = 0.6028$ and $\alpha = 0.8$ [15]. Fig. 5 shows that the greedy and SA algorithms converged to a steady state solution after exploring 9 states (0.029% of the design space) and 400 states (1.29% of the design space), respectively (this represents a similar trend as for the security/defense and health care applications). The greedy and SA algorithms' solutions after exploring 9

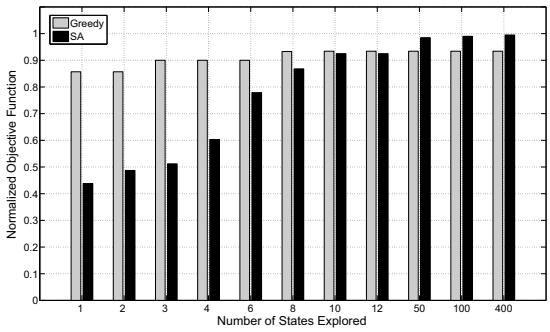

Fig. 5. Objective function value normalized to the optimal solution for a varying number of states explored for the greedy and simulated annealing algorithms for an ambient conditions monitoring application where $\omega_l = 0.6$, $\omega_t = 0.25$, $\omega_r = 0.15$, $|S| = 31,104$.

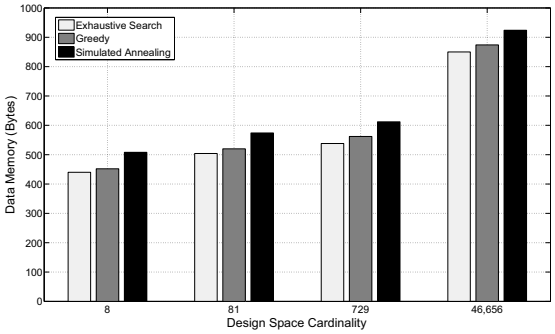

Fig. 6. Data memory requirements for exhaustive search, greedy, and simulated annealing algorithms for design space cardinalities of 8, 81, 729, and 46656.

and 400 states are within 6.6% and 0.5% of the optimal solution, respectively. Fig. 5 shows an average growth rate of approximately 5.02% and 8.16% for the initial iterations for the greedy and SA algorithms, respectively, and 0.11% and 0.0017% for the later iterations of the greedy and SA algorithms, respectively.

The results also provide insights into the convergence rates and reveal that even though the design space cardinality increases by 43x (from 729 to 31,104), the greedy and SA algorithms still explore only a small percentage of the design space and result in high-quality solutions. The results indicate that the SA algorithm converges to the optimal (or near optimal) solution slowly, however, the SA algorithm can result in a desired solution quality by controlling the allowable number of states explored. The results reveal that for tightly constrained runtimes, the greedy algorithm can provide better results than the SA algorithm (e.g., when exploration of only 6 states (0.82% of S) or less is allowed), however, the SA algorithm requires longer runtimes to achieve a near optimal solutions (e.g., the greedy algorithm obtained a solution within 8.3% of the optimal solution on average after exploring 1.37% of design space whereas the SA algorithm obtained a solution within 0.237% of the optimal solution on average after exploring 54.87% of design space for $|S| = 729$).

To verify that our algorithms are lightweight, we analyzed the execution time, energy consumption, and data memory requirements. We measured the execution time (averaged over 10,000 runs to smooth any discrepancies due to operating system overheads) for both algorithms on an Intel Xeon CPU running at 2.66 GHz [22] using the Linux/Unix `time` command [23]. We scaled these runtimes to the Atmel ATmega1281 microcontroller [18] running at 8 MHz. Whereas this scaling does not provide exact absolute runtimes for the Atmel processor, the comparison of these values provides valuable insights. For each SA run, we initialized the pseudo-random number generator with a different seed using `srand()` [24]. We observe that the greedy algorithm explores 1 (0.14% of the design space S), 4 (0.55% of S), and 10 (1.37% of S) states in 0.366, 0.732, and 0.964 ms, respectively (the greedy algorithm converged after 10 iterations). The SA algorithm explores 1, 4, 10, 100 (13.72% of S), 421 (57.75% of S), and 729 (100% of S) states in 1.097, 1.197, 1.297, 3.39, 11.34, and 18.19 ms, respectively, for $|S| = 729$. On average, the execution time linearly increases by 0.039 and 0.023 ms per state for the greedy and SA algorithms, respectively. The greedy algorithm requires 34.54% less execution time on average as compared to SA (after exploring 10 states). We measured the greedy and SA algorithms' execution time for $|S| = 31,104$ and observed similar results as for $|S| = 729$ because both the algorithms' execution time depends upon the number of states explored and not on the design space cardinality. The exhaustive search requires 29.526 ms and 2.765 seconds for $|S| = 729$ and $|S| = 31,104$, respectively. Compared with an exhaustive search, the greedy and SA algorithms (after exploring 10 states) requires 30.63x and 22.76x less execution time, respectively, for $|S| = 729$, and requires 2868.26x and 2131.84x less execution time, respectively, for $|S| = 31,104$. We verified our execution time analysis using `clock()` [24] and observed similar trends. These execution time results indicate that our online algorithms' efficacy increases as the design space cardinality increases.

We calculated the energy consumption of our algorithms E_{algo} for an Atmel ATmega1281 microcontroller [18] operating at $V_p = 2.7$ V and $F_p = 8$ MHz as $E_{algo} = V_p \cdot I_p^a \cdot T_{exe}$ where I_p^a and T_{exe} denote the processor's active current and the algorithm's execution time at (V_p, F_p), respectively (we observed similar trends for other processor voltage and frequency settings). Our calculations indicate that the greedy algorithm requires 5.237, 10.475, and 13.795 μJ to explore 1, 4, and 10 states, respectively whereas the SA algorithm requires 15.698, 17.129, 18.56, 48.51, 162.28, and 260.3 μJ for exploring 1, 4, 10, 100, 421, and 729 states, respectively, both for $|S| = 729$ and $|S| = 31,104$. The exhaustive search requires 0.422 and 39.567 mJ for $|S| = 729$ and $|S| = 31,104$, respectively. The SA algorithm requires 34.54% more energy as compared to the greedy algorithm for exploring 10 states whereas both the algorithms are highly energy-efficient as compared to exhaustive search.

Fig. 6 depicts low data memory requirements for both algorithms for design space cardinalities of 8, 81, 729, and 46,656. We observe that the greedy algorithm requires

452, 520, 562, and 874 bytes, whereas the SA algorithm requires 508, 574, 612, and 924 bytes of storage for design space cardinalities of 8, 81, 729, and 46,656, respectively. The data memory analysis shows that the SA algorithm has comparatively larger memory requirements (9.35% on average for analyzed design space cardinalities) than the greedy algorithm. The data memory requirements for both the algorithms increase linearly as the number of tunable parameters and tunable values (and thus the design space) increases. We point out that the data memory requirements for the exhaustive search is comparable to the greedy algorithm because the exhaustive search simply evaluates the objective function value for each state in the design space. However, the exhaustive search yields a high penalty in execution time because of complete design space evaluation. The figure reveals that our algorithms scale well with increased design space cardinality, and thus our proposed algorithms are appropriate for sensor nodes with a large number of tunable parameters and parameter values.

V. Conclusions and Future Work

In this paper, we proposed a dynamic optimization methodology using greedy and simulated annealing online optimization algorithms for wireless sensor networks. Compared to previous work, our methodology considers an extensive sensor node design space, which allows sensor nodes to more closely meet application requirements. Results revealed that our online algorithms are lightweight, requiring little computational, memory, and energy resources and thus are amenable for implementation on sensor nodes with tight resource and energy constraints. Furthermore, our online algorithms can perform in situ parameter tuning to adapt to changing environmental stimuli to meet application requirements.

Future work includes further results verification using larger state spaces containing more sensor node tunable parameters and tunable values. In addition, we will implement our dynamic optimization methodology on a hardware sensor node platform to gather and incorporate profiling statistics into our lightweight optimization algorithms.

Acknowledgments

This work was supported by the National Science Foundation (NSF) (CNS-0834080) and the Natural Sciences and Engineering Research Council of Canada (NSERC). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF and the NSERC.

References

[1] "Dynamic Profiling and Optimization (DPOP) for Sensor Networks," July 2010. [Online]. Available: http://www.ece.arizona.edu/~dpop/

[2] A. Munir and A. Gordon-Ross, "An MDP-based Application Oriented Optimal Policy for Wireless Sensor Networks," in *Proc. of IEEE/ACM CODES+ISSS*, October 2009.

[3] S. Lysecky and F. Vahid, "Automated Application-Specific Tuning of Parameterized Sensor-Based Embedded System Building Blocks," in *Proc. of IEEE UbiComp*, September 2006.

[4] D. Brooks and M. Martonosi, "Value-based Clock Gating and Operation Packing: Dynamic Strategies for Improving Processor Power and Performance," *ACM Trans. on Computer Systems*, vol. 18, no. 2, pp. 89–126, May 2000.

[5] H. Hamed, A. El-Atawy, and A.-S. Ehab, "On Dynamic Optimization of Packet Matching in High-Speed Firewalls," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1817–1830, October 2006.

[6] K. Hazelwood and M. Smith, "Managing Bounded Code Caches in Dynamic Binary Optimization Systems," *ACM Trans. on Architecture and Code Optimization*, vol. 3, no. 3, pp. 263–294, September 2006.

[7] S. Hu, M. Valluri, and L. John, "Effective Management of Multiple Configurable Units using Dynamic Optimization," *ACM Trans. on Architecture and Code Optimization*, vol. 3, no. 4, pp. 477–501, December 2006.

[8] S. Sridharan and S. Lysecky, "A First Step Towards Dynamic Profiling of Sensor-Based Systems," in *Proc. of IEEE SECON*, June 2008.

[9] S. Kogekar, S. Neema, B. Eames, X. Koutsoukos, A. Ledeczi, and M. Maroti, "Constraint-Guided Dynamic Reconfiguration in Sensor Networks," in *Proceedings of the ACM IPSN*, April 2004.

[10] R. Min, T. Furrer, and A. Chandrakasan, "Dynamic Voltage Scaling Techniques for Distributed Microsensor Networks," in *Proc. of IEEE WVLSI*, April 2000.

[11] L. Yuan and G. Qu, "Design Space Exploration for Energy-Efficient Secure Sensor Network," in *Proc. of the IEEE ASAP*, July 2002.

[12] M. Huber, A. Kuwertz, F. Sawo, and U. Hanebeck, "Distributed Greedy Sensor Scheduling for Model-based Reconstruction of Space-Time Continuous Physical Phenomenon," in *Proc. of IEEE FUSION*, July 2009.

[13] R. Verma, "Automated Application Specific Sensor Network Node Tuning for Non-Expert Application Developers," *M.S. Thesis, Department of Electrical and Computer Engineering, University of Arizona*, 2008.

[14] L. Xu and E. Oja, "Improved Simulated Annealing, Boltzmann Machine, and Attributed Graph Matching," in *Proc. of the EURASIP Workshop on Neural Networks*. Springer-Verlag, February 1990, pp. 151–160.

[15] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[16] W. Ben-Ameur, "Computing the Initial Temperature of Simulated Annealing," *Computational Optimization and Applications*, vol. 29, no. 3, pp. 369–385, December 2004.

[17] Crossbow, "Crossbow IRIS Datasheet," September 2010. [Online]. Available: http://www.xbow.com

[18] Atmel, "ATMEL ATmega1281 Microcontroller with 256K Bytes In-System Programmable Flash," September 2010. [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/2549S.pdf

[19] ——, "ATMEL AT86RF230 Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE and ISM Applications," September 2010. [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf

[20] Crossbow, "MTS/MDA Sensor Board Users Manual," September 2010. [Online]. Available: http://www.xbow.com

[21] Sensirion, "Datasheet SHT1x (SHT10, SHT11, SHT15) Humidity and Temperature Sensor," September 2010. [Online]. Available: http://www.sensirion.com

[22] "Intel Xeon Processor E5430," July 2010. [Online]. Available: http://processorfinder.intel.com/details.aspx?sSpec=SLANU

[23] "Linux Man Pages," July 2010. [Online]. Available: http://linux.die.net/man/

[24] "C++ Reference Library," in *cplusplus.com*, September 2010. [Online]. Available: http://cplusplus.com/reference/clibrary/ctime/clock/