A Lightweight Dynamic Optimization Methodology for Wireless Sensor Networks

Arslan Munir and Ann Gordon-Ross
Department of Electrical and Computer Engineering
University of Florida, Gainesville, Florida 32611
Email: amunir@ufl.edu, ann@chrec.org

Susan Lysecky and Roman Lysecky
Department of Electrical and Computer Engineering
University of Arizona, Tucson, Arizona 85721
Email: {slysecky, rlysecky}@ece.arizona.edu

***Abstract*—Technological advancements in embedded systems due to Moore's law have lead to the proliferation of wireless sensor networks (WSNs) in different application domains (e.g. defense, health care, surveillance systems) with different application requirements (e.g. lifetime, reliability). Many commercial-off-the-shelf (COTS) sensor nodes can be specialized to meet these requirements using tunable parameters (e.g. voltage, frequency) to specialize the operating state. Since a sensor node's performance depends greatly on environmental stimuli, dynamic optimizations enable sensor nodes to automatically determine their operating state in-situ. However, dynamic optimization methodology development given a large design space and resource constraints (memory and computational) is a very challenging task. In this paper, we propose a lightweight dynamic optimization methodology that intelligently selects initial tunable parameter values to produce a high-quality initial operating state in one-shot for time-critical or highly constrained applications. Further operating state improvements are made using an efficient greedy exploration algorithm, achieving optimal or near-optimal operating states while exploring only 0.04% of the design space on average.**

***Index Terms*— Wireless sensor networks, dynamic optimization, optimization algorithms**

I. Introduction and Motivation

Due to embedded technology and wireless communication advancements, wireless sensor networks (WSNs) have proliferated in many application domains (e.g. defense, health care, surveillance systems) each with varying application requirements (e.g. lifetime, reliability). This diversity makes it difficult for commercial-off-the-shelf (COTS) sensor nodes to meet these application requirements.

Since COTS sensor nodes are mass-produced (to optimize cost), many COTS sensor nodes possess tunable parameters (e.g. processor voltage and frequency, sensing frequency), whose values may be *tuned* for application specialization [1]. However, given the large design space, determining appropriate parameter values (operating state) is challenging. Typically, sensor node vendors assign an initial generic tunable parameter setting, however, no one tunable parameter setting is appropriate for all applications.

Parameter optimization is the process of assigning appropriate (optimal or near-optimal) tunable parameter values, either statically or dynamically to meet application requirements. *Static optimizations* assign parameter values at deployment and these values remain fixed for the lifetime of the sensor node. Challenges include accurately determining values using environmental stimuli prediction/simulation and static methods are not appropriate for applications with varying environmental stimuli. Alternatively, *dynamic optimizations* assign (and re-assign/change) parameter values during runtime enabling the sensor node to adapt to changing environmental stimuli, and thus more accurately meet application requirements.

There exists much research in the area of dynamic optimizations (e.g. [2][3][4][5]), but most previous work targets the processor or memory (cache) in computer systems. There exists little previous work on WSN dynamic optimization, which presents a more challenging endeavor given a unique design space, resource constraints, and platform particulars as well as WSN operating environment. Shenoy et al. [6] dynamically profiled a sensor-based platform to gather profiling statistics, but did not perform optimizations. In prior work, Munir et al. [1] proposed a Markov Decision Process (MDP)-based methodology as a first step towards WSN dynamic optimization, but however this method required a high-performance base station and could not be run on a sensor node autonomously. Kogekar et al. [7] proposed dynamic software reconfiguration to adapt software to new operating conditions, but however, that work did not consider sensor node tunable parameters and application requirements. Some previous works explored greedy, simulated annealing (SA), and particle swarm optimization (PSO)-based WSN optimizations [8], but these previous works did not analyze their algorithms' resources or computational complexity. Finally, some previous works explored WSN dynamic voltage and frequency scaling (DVFS) [9][10]. Although DVFS provides a mean for dynamic optimization, previous work only considered two sensor node tunable parameters (processor voltage and frequency).

WSN dynamic optimization presents additional challenges as sensor nodes have more tunable parameters and a larger design space. The dynamic profiling and optimization (DPOP) project aims to address these challenges and complexities associated with sensor-based system design through the use of automated optimization methods [11]. The DPOP project has gathered dynamic profiling statistics from a sensor-based system, however, the parameter optimization process remains to be completed. In this paper, we investigate an appropriate approach to implementing the parameter optimization given the dynamic profiling data already collected from the platform. We analyze possible ways to achieve dynamic optimization and evaluate algorithms that can provide a good operating state without depleting the battery energy significantly. We explore a large design space with many tunable parameters and values. This expansion provides a finer-grained design space, enabling sensor nodes to more closely meet application requirements (Gordon-Ross et al. [12] showed that finer-grained design spaces contain interesting design alternatives and result in increased benefits in the cache subsystem). However, the large design space exacerbates optimization challenges, taking into consideration a sensor node's constrained memory and computational resources. Additionally, rapidly changing application requirements and environmental stimuli coupled with limited battery reserves necessitates a highly responsive and low overhead methodology.

Our main contributions in this paper are:

- We propose a light-weight dynamic optimization methodology that is able to determine appropriate initial tunable parameter values to give a one-shot solution. This one-shot solution provides a good quality operating state with minimal design exploration for highly constrained applications. Results reveal that this one-shot operating state is within 5.92% of the optimal solution (determined by exhaustive search) averaged over several different application domains and design spaces.
- Our dynamic optimization methodology combines the initial tunable value selection with an intelligent exploration ordering of tunable parameter values and an exploration arrangement of tunable parameters, since some parameters are more critical for an application than others and thus should be explored first [5] (e.g. transmission power parameter may be more critical for a lifetime sensitive application than processor voltage).
- Although greedy-based algorithms and approaches are not novel in general, we propose, for the first time, an online greedy algorithm that leverages intelligent parameter arrangement and explores the design space to improve over the one-shot solution to find the optimal (or near optimal) operating state. Results reveal that our greedy exploration of tunable parameters results in an operating state within 2.5% of the optimal operating state while exploring only 0.04% of the design space.

II. Dynamic Optimization Methodology

Fig. 1 depicts our dynamic optimization methodology for WSNs. WSN designers evaluate application requirements and capture these requirements as high-level application metrics (e.g. lifetime, throughput, reliability) and associated weight factors. The weight factors characterize application metrics' relative importance, e.g., since some WSN applications may not be power-centric and throughput may be more important than the lifetime, assigning a higher weight factor for throughput than lifetime can capture this relationship. The sensor nodes use application metrics and weight factors to

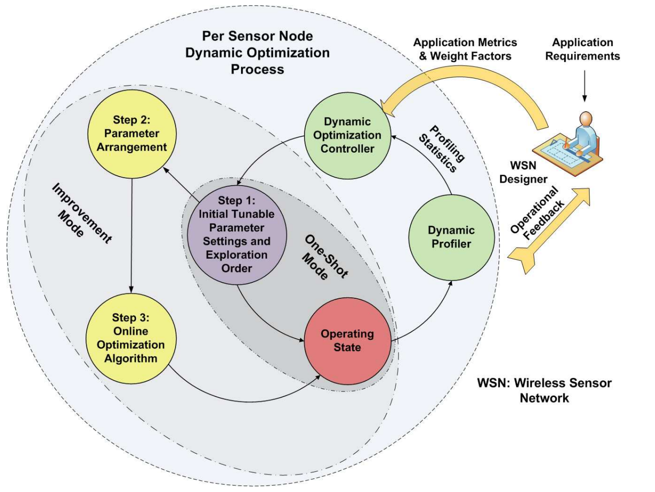
Per Sensor Node Dynamic Optimization Process
Application Metrics & Weight Factors
Application Requirements
Step 2: Parameter Arrangement
Dynamic Optimization Controller
Profiling Statistics
WSN Designer
Improvement Mode
Step 1: Initial Tunable Parameter Settings and Exploration Order
One-Shot Mode
Dynamic Profiler
Operational Feedback
Step 3: Online Optimization Algorithm
Operating State
WSN: Wireless Sensor Network

Fig. 1. Our dynamic optimization methodology for wireless sensor networks.

determine an appropriate operating state (tunable parameter settings) using an application metric estimation model. This model determines high-level application metric values corresponding to the tunable parameter settings. This model details are left for future work.

Fig. 1 shows the per-node dynamic optimization process (encompassed by the dashed circle), which is orchestrated by the *dynamic optimization controller*. The process consists of two operating modes: the *one-shot mode* wherein the sensor node operating state is directly determined and the *improvement mode* wherein the operating state is iteratively improved using an online optimization algorithm. The dynamic optimization process consists of three steps. In the first step, the dynamic optimization controller intelligently determines the initial parameter value settings (operating state) and exploration order (ascending or descending), which is critical in reducing the number of states explored by the improvement mode. In the one-shot operation mode, the dynamic optimization process is complete and the sensor node moves directly to the operating state specified by the initial parameter value settings. In the improvement mode, the second step determines the parameter arrangement based on application metric weight factors (e.g. explore processor voltage then frequency then sensing frequency). The third step invokes an *online optimization algorithm* for parameter exploration to iteratively improve the operating state to more closely meet application requirements. The online optimization algorithm leverages the intelligent initial parameter value settings, exploration order, and parameter arrangement. A *dynamic profiler* records profiling statistics (e.g. processor voltage, wireless channel condition, radio transmission power) given the current operating state and environmental stimuli and passes these profiling statistics to the dynamic optimization controller.

The dynamic optimization controller processes the profiling statistics to determine whether the current operating state meets the application requirements. If the application requirements are not met, the dynamic optimization controller reinvokes the dynamic optimization process to determine the new operating state. This feedback process continues to ensure that the application requirements are best met under changing environmental stimuli. Details of this process are left for future work.

III. Dynamic Optimization Formulation

In this section, we formulate the state space and the objective function for our dynamic optimization.

A. State Space

The state space S for our dynamic optimization methodology given N tunable parameters is defined as:

$$S = P_1 \times P_2 \times \cdots \times P_N \tag{1}$$

where P_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 P_i consists of n values:

$$P_i = \{p_{i_1}, p_{i_2}, p_{i_3}, \ldots, p_{i_n}\} \;\;:\;\; |P_i| = n \tag{2}$$

where $|P_i|$ denotes the tunable parameter P_i state space cardinality (the number of tunable values in P_i). S is a set of n-tuples formed by taking one tunable parameter value from each tunable parameter. A single n-tuple $s \in S$ is given as:

$$\begin{aligned} s \;\;=\;\; & (p_{1_y}, p_{2_y}, \ldots, p_{N_y}) : p_{i_y} \in P_i, \\ & \forall\, i \in \{1, 2, \ldots, N\},\, y \in \{1, 2, \ldots, n\} \end{aligned} \tag{3}$$

Each n-tuple represents a sensor note operating state. We point out that some n-tuples in S may not be feasible (such as invalid combinations of processor voltage and frequency) and can be regarded as *do not care* tuples.

B. Optimization Objection Function

The dynamic optimization problem can be formulated as:

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

where $f(s)$ denotes the objective function which captures application metrics and weight factors. $f_k(s)$ and ω_k in (4) denote the objective function and weight factor for the k^{th} application metric, respectively, given that there are m application metrics.

For our dynamic optimization methodology, we consider three application metrics (m = 3), lifetime, throughput, and reliability, whose objective functions are denoted by $f_l(s)$, $f_t(s)$, and $f_r(s)$, respectively. We define $f_r(s)$ (Fig. 2) using the piecewise linear function:

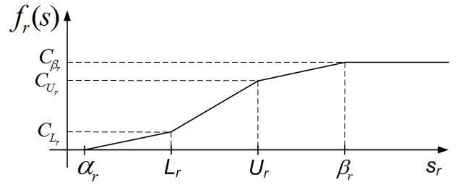

Fig. 2. Reliability objective function $f_r(s)$.

$$f_r(s) = \begin{cases} 1, & s_r \geq \beta_r \\ C_{U_r} + \frac{(C_{\beta_r} - C_{U_r})(s_r - U_r)}{(\beta_r - U_r)}, & U_r \leq s_r < \beta_r \\ C_{L_r} + \frac{(C_{U_r} - C_{L_r})(s_r - L_r)}{(U_r - L_r)}, & L_r \leq s_r < U_r \\ C_{L_r} \cdot \frac{(s_r - \alpha_r)}{(L_r - \alpha_r)}, & \alpha_r \leq s_r < L_r \\ 0, & s_r < \alpha_r. \end{cases} \tag{5}$$

where s_r denotes the reliability offered by state s, the constant parameters L_r and U_r denote the *desired* minimum and maximum reliability, and the constant parameters α_r and β_r denote the *acceptable* minimum and maximum reliability. The piecewise linear objective function provides WSN designers with a flexible application requirement specification, as it allows both desirable and acceptable ranges. The objective function reward gradient (slope) would be greater in the desired range than the acceptable range, however, there would be no reward for operating outside the acceptable range. The constant parameters C_{L_r}, C_{U_r}, and C_{β_r} in (5) denote the $f_r(s)$ value at L_r, U_r, and β_r, respectively.

The $f_l(s)$ and $f_t(s)$ can be defined using increasing piecewise linear functions similar to (5) as higher values of lifetime and throughput (like reliability) are typically desirable and correspond to higher objective function values. Although we define our objective functions using piecewise linear objective functions, our dynamic optimization methodology works well for any other characterization of objective functions (e.g., linear, non-linear).

IV. Algorithms for Dynamic Optimization Methodology

In this section, we describe the three steps, associated algorithms, and operating modes for our dynamic optimization methodology (Section 1). Step one determines initial tunable parameter values and exploration order (ascending or descending). In *one-shot mode* (Fig. 1), these initial tunable parameter value settings result in a high-quality operating state in one-shot (no additional design space exploration) for applications with tight constraints (e.g. limited exploration time due to a rapidly changing environment). For applications with more flexible constraints, the *improvement mode* encompasses steps two and three and iteratively improves the one-shot solution. Step two determines the tunable parameter exploration arrangement based on the application metric weight factors (i.e. some parameters are more critical for an application than others and should be explored first). Step three leverages the outcomes of steps one and two and explores the design space using an online optimization algorithm. For this optimization algorithm, we propose a lightweight greedy algorithm for design space exploration, however, step three can be generalized to any online algorithm.

A. Initial Parameter Value Settings and Exploration Order

```
Input: $f(s)$, $N$, $n$, $m$, $\boldsymbol{P}$
Output: Initial tunable parameter value settings and exploration
        order
1  for $k \leftarrow 1$ to $m$ do
2      for $P_i \leftarrow P_1$ to $P_N$ do
3          $f^k_{p_{i_1}}$ ← k-metric objective function value when parameter
           setting is $\{P_i = p_{i_1}, P_j = P_{j_0}, \forall\, i \neq j\}$ ;
4          $f^k_{p_{i_n}}$ ← k-metric objective function value when parameter
           setting is $\{P_i = p_{i_n}, P_j = P_{j_0}, \forall\, i \neq j\}$ ;
5          $\delta f^k_{P_i} \leftarrow f^k_{p_{i_n}} - f^k_{p_{i_1}}$ ;
6          if $\delta f^k_{P_i} > 0$ then
7              explore $P_i$ in descending order ;
8              $P^k_d[i] \leftarrow$ descending ;
9              $P^k_0[i] \leftarrow p^k_{i_n}$ ;
10         else
11             explore $P_i$ in ascending order ;
12             $P^k_d[i] \leftarrow$ ascending ;
13             $P^k_0[i] \leftarrow p^k_{i_1}$ ;
14         end
15     end
16 end
return $\boldsymbol{P^k_d}$, $\boldsymbol{P^k_0}, \forall\, k \in \{1, \ldots, m\}$
```

Algorithm 1: Initial tunable parameter value settings and exploration order algorithm.

Algorithm 1 describes our technique to determine initial tunable parameter value settings and exploration order (first step of our dynamic optimization methodology). The algorithm takes as input the objective function $f(s)$, the number of tunable parameters N, the number of values for each tunable parameter n, the number of application metrics m, and $\boldsymbol{P}$ where $\boldsymbol{P}$ represents a vector containing the tunable parameters, $\boldsymbol{P} = \{P_1, P_2, \ldots, P_N\}$. For each application metric k, the algorithm calculates vectors $\boldsymbol{P^k_0}$ and $\boldsymbol{P^k_d}$ (where d denotes the exploration direction (ascending or descending)), which store the initial value settings and exploration order, respectively, for the tunable parameters. The algorithm determines the k^{th} application metric objective function values $f^k_{p_{i_1}}$ and $f^k_{p_{i_n}}$ where the parameter being explored P_i is assigned its first p_{i_1} and last p_{i_n} tunable values, respectively, and rest of the tunable parameters $P_j, \forall\, j \neq i$ are assigned initial values (lines 3 - 4). $\delta f^k_{P_i}$ stores the difference between $f^k_{p_{i_n}}$ and $f^k_{p_{i_1}}$. For $\delta f^k_{P_i} > 0$, p_{i_n} results in a greater objective function value as compared to p_{i_1} for parameter P_i (i.e. the objective function value decreases as the parameter value decreases). Therefore, to reduce the number of states explored while considering that the greedy algorithm (Section IV-C) will stop exploring a tunable parameter if a tunable parameter's value yields a comparatively lower objective function value, P_i's exploration order must be descending (lines 6 - 8). The algorithm assigns p_{i_n} as the initial value of P_i for the k^{th} application metric (line 9). If $\delta f^k_{P_i} < 0$, the algorithm assigns the exploration order as ascending for P_i and p_{i_1} as the initial value setting of P_i (lines 11 - 13). This $\delta f^k_{P_i}$ calculation procedure is repeated for all m application metrics and all N tunable parameters (lines 1 - 16).

B. Parameter Arrangement

Depending on the application metric weight factors, some parameters are more critical to meeting application requirements than other parameters. For example, sensing frequency is a critical parameter for applications with a high responsiveness weight factor and therefore, sensing frequency should be explored first. In this subsection, we devise a technique for parameter arrangement such that parameters are explored in an order characterized by the parameter's impact on application metrics based on relative weight factors. Our parameter arrangement technique is based on calculations performed in Algorithm 1. We define:

$$\boldsymbol{\nabla f_P} = \{\nabla f^1_P, \nabla f^2_P, \ldots, \nabla f^m_P\} \tag{6}$$

where ∇f_P is a vector containing $\nabla f^k_P, \forall\, k \in \{1, 2, \ldots, m\}$ arranged in descending order by their respective values and is given as:

$$\begin{aligned} \boldsymbol{\nabla f^k_P} &= \{\delta f^k_{P_1}, \delta f^k_{P_2}, \ldots, \delta f^k_{P_N}\} \\ &: \; |\delta f^k_{P_i}| \geq |\delta f^k_{P_{i+1}}|, \forall\, i \in \{1, 2, \ldots, N-1\} \end{aligned} \tag{7}$$

The tunable parameter arrangement vector $\boldsymbol{P^k}$ corresponding to $\boldsymbol{\nabla f^k_P}$ (one-to-one correspondence) is given by:

$$\boldsymbol{P^k} = \{P^k_1, P^k_2, \ldots, P^k_N\}, \forall\, k \in \{1, 2, \ldots, m\} \tag{8}$$

An intelligent parameter arrangement $\widehat{\boldsymbol{P}}$ must consider all application metrics' weight factors with higher importance given to higher weight factors, i.e.:

$$\begin{aligned} \widehat{\boldsymbol{P}} = \{&P^1_1, \ldots, P^1_{l_1}, P^2_1, \ldots, P^2_{l_2}, \\ &P^3_1, \ldots, P^3_{l_3}, \ldots, P^m_1, \ldots, P^m_{l_m}\} \end{aligned} \tag{9}$$

where l_k denotes the number of tunable parameters taken from $P^k, \forall\, k \in \{1, 2, \ldots, m\}$ such that $\sum_{k=1}^m l_k = N$. Our technique allows taking more tunable parameters from parameter arrangement vectors corresponding to higher weight factor application metrics, i.e. $l_k \geq l_{k+1}, \forall\, k \in \{1, 2, \ldots, m-1\}$. In (9), l_1 tunable parameters are taken from vector P^1, then l_2 from vector P^2, and so on to l_m from vector P^m such that $\{P^k_1, \ldots, P^k_{l_k}\} \cap \{P^{k-1}_1, \ldots, P^{k-1}_{l_{k-1}}\} = \emptyset, \forall\, k \in \{2, 3, \ldots, m\}$. In other words, we select those tunable parameters from parameter arrangement vectors corresponding to lower weight factors which are not already selected from parameter arrangement vectors corresponding to higher weight factors (i.e. $\widehat{\boldsymbol{P}}$ comprises of disjoint or non-overlapping tunable parameters corresponding to each application metric).

In the situation where weight factor ω_1 is much greater than all other weight factors, an intelligent parameter arrangement $\widetilde{\boldsymbol{P}}$ would correspond to the parameter arrangement for the application metric with weight factor ω_1, i.e.:

$$\begin{aligned} \widetilde{\boldsymbol{P}} = \boldsymbol{P^1} &= \{P^1_1, P^1_2, \ldots, P^1_N\} \\ &\Leftrightarrow \; \omega_1 \gg \omega_q, \forall\, q \in \{2, 3, \ldots, m\} \end{aligned} \tag{10}$$

The initial parameter value vector $\widehat{\boldsymbol{P}}_{\boldsymbol{0}}$ and exploration order (ascending or descending) vector $\widehat{\boldsymbol{P}}_{\boldsymbol{d}}$ corresponding to $\widehat{\boldsymbol{P}}$ (9) can be determined from $\widehat{\boldsymbol{P}}$ (9), $\boldsymbol{P}_{\boldsymbol{d}}^{\boldsymbol{k}}$, and $\boldsymbol{P}_{\boldsymbol{0}}^{\boldsymbol{k}}, \forall\, k \in \{1, \ldots, m\}$ (Algorithm 1) by looking at the tunable parameter from $\widehat{\boldsymbol{P}}$ and finding the tunable parameter's initial value from $\boldsymbol{P}_{\boldsymbol{0}}^{\boldsymbol{k}}$ and exploration order from $\boldsymbol{P}_{\boldsymbol{d}}^{\boldsymbol{k}}$.

C. Online Optimization Algorithm

```
Input: f(s), N, n, $\boldsymbol{P}$, $\widehat{\boldsymbol{P}}_0$, $\widehat{\boldsymbol{P}}_d$
Output: Sensor node state that maximizes $f(s)$ and the
        corresponding $f(s)$ value
1  $\kappa \leftarrow$ initial tunable parameter value settings from $\widehat{\boldsymbol{P}}_0$ ;
2  $f_{best} \leftarrow$ solution from initial parameter settings $\kappa$ ;
3  for $\widehat{P}_i \leftarrow \widehat{P}_1$ to $\widehat{P}_N$ do
4      explore $\widehat{P}_i$ in ascending/descending order suggested by $\widehat{\boldsymbol{P}}_d$ ;
5      foreach $\widehat{P}_i = \{\widehat{p}_{i_1}, \widehat{p}_{i_2}, \ldots, \widehat{p}_{i_n}\}$ do
6          $f_{temp} \leftarrow$ current state $\zeta$ solution ;
7          if $f_{temp} > f_{best}$ then
8              $f_{best} \leftarrow f_{temp}$ ;
9              $\xi \leftarrow \zeta$ ;
10         else
11             break ;
12         end
13     end
14 end
   return $\xi$, $f_{best}$
```

Algorithm 2: Online optimization algorithm for tunable parameters exploration.

The third step of our dynamic optimization process uses a greedy lightweight online optimization algorithm for tunable parameters exploration in an effort to determine a better operating state than the one obtained from step one (Section IV-A). Algorithm 2 depicts our online optimization algorithm, which leverages the initial parameter value settings (Section IV-A), parameter value exploration order (Section IV-A), and parameter arrangement (Section IV-B). The algorithm takes as input the objective function $f(s)$, the number of tunable parameters N, the number of values for each tunable parameter n, the tunable parameters' vector $\boldsymbol{P}$, the tunable parameters' initial value vector $\widehat{\boldsymbol{P}}_{\boldsymbol{0}}$, and the tunable parameter's exploration order (ascending or descending) vector $\widehat{\boldsymbol{P}}_{\boldsymbol{d}}$. The algorithm initializes state κ from $\widehat{\boldsymbol{P}}_{\boldsymbol{0}}$ (line 1) and f_{best} with κ's objective function value (line 2). The algorithm explores each parameter in $\widehat{P}_i$ (9) in ascending or descending order as given by $\widehat{\boldsymbol{P}}_{\boldsymbol{d}}$ (lines 3 - 4). For each tunable parameter $\widehat{P}_i$ (line 5), the algorithm assigns f_{temp} the objective function value from the current state ζ (line 6). If $f_{temp} > f_{best}$ (increase in objection function value), f_{temp} is assigned to f_{best} and the state ζ is assigned to state ξ (lines 7 - 9). If $f_{temp} \leq f_{best}$, the algorithm stops exploring the current parameter $\widehat{P}_i$ and starts exploring the next tunable parameter (lines 10 - 12). The algorithm returns the best found objective function value f_{best} and the state ξ corresponding to f_{best}.

D. Computational Complexity

The computational complexity (running time and storage) for our dynamic optimization methodology is $\mathcal{O}(Nm \log N + Nn)$, which is comprised of the intelligent initial parameter value settings and exploration ordering (Algorithm 1) $\mathcal{O}(Nm)$, parameter arrangement $\mathcal{O}(Nm \log N)$ (sorting ∇f_P^k (7) contributes the $N \log N$ factor) (Section IV-B), and the online optimization algorithm for parameter exploration (Algorithm 2) $\mathcal{O}(Nn)$. Assuming that the number of tunable parameters N is larger than the number of parameter's tunable values n, the computational complexity of our methodology can be given as $\mathcal{O}(Nm \log N)$. This complexity reveals that our proposed methodology is lightweight and is thus feasible for implementation on sensor nodes with tight resource constraints.

V. Experimental Results

A. Experimental Setup

Our experimental setup is based on the Crossbow IRIS mote platform [13] with a battery capacity of 2000 mA-h using two AA alkaline batteries. The IRIS mote platform integrates an Atmel ATmega1281 microcontroller [14], an MTS400 sensor board [15] with Sensirion SHT1x temperature and humidity sensors [16], and an Atmel AT-86RF230 low power 2.4 GHz transceiver [17].

We analyze six tunable parameters: processor voltage V_p, processor frequency F_p, sensing frequency F_s, packet size P_s, packet transmission interval P_{ti}, and transceiver transmission power P_{tx}. In order to explore the fidelity of our methodology 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 tunable parameters for $|S| = 729$ are V_p = {2.7, 3.3, 4} (volts), F_p = {4, 6, 8} (MHz) [14], F_s = {1, 2, 3} (samples per second) [16], P_s = {41, 56, 64} (bytes), P_{ti} = {60, 300, 600} (seconds), and P_{tx} = {-17, -3, 1} (dBm) [17]. 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) [14], F_s = {0.2, 0.5, 1, 2, 3, 4} (samples per second) [16], 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) [17]. 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).

In order to evaluate the robustness of our methodology across different applications with varying application metric weight factors, we model three sample application domains (a security/defense system, a health care application, and an ambient conditions monitoring application) and 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 (Section III-B). Our selected objective function parameter values and application metric weight factors represent typical application requirements [18]. Although, we analyzed our methodology for the IRIS motes platform, three application domains, and two design spaces, our algorithms are equally applicable to any platform, application domain, and design space.

B. Results

For comparison purposes, we implemented a simulated annealing (SA)-based algorithm, our greedy online

TABLE I
GREEDY ALGORITHMS WITH DIFFERENT PARAMETER ARRANGEMENTS AND EXPLORATION ORDERS

Notation	**Description**
GD	Our greedy algorithm with parameter exploration order $\widehat{\boldsymbol{P}}_{\boldsymbol{d}}$ and arrangement $\widehat{\boldsymbol{P}}$
GD$^{\text{ascA}}$	Explores parameter values in ascending order with arrangement $\mathcal{A} = \{V_p, F_p, F_s, P_s, P_{ti}, P_{tx}\}$
GD$^{\text{ascB}}$	Explores parameter values in ascending order with arrangement $\mathcal{B} = \{P_{tx}, P_{ti}, P_s, F_s, F_p, V_p\}$
GD$^{\text{ascC}}$	Explores parameter values in ascending order with arrangement $\mathcal{C} = \{F_s, P_{ti}, P_{tx}, V_p, F_p, P_s\}$
GD$^{\text{desD}}$	Explores parameter values in descending order with arrangement $\mathcal{D} = \{V_p, F_p, F_s, P_s, P_{ti}, P_{tx}\}$
GD$^{\text{desE}}$	Explores parameter values in descending order with arrangement $\mathcal{E} = \{P_{tx}, P_{ti}, P_s, F_s, F_p, V_p\}$
GD$^{\text{desF}}$	Explores parameter values in descending order with arrangement $\mathcal{F} = \{P_s, F_p, V_p, P_{tx}, P_{ti}, F_s\}$

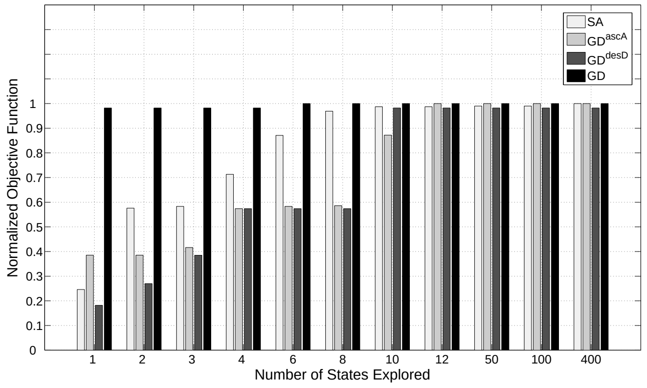

Fig. 3. Objective function value normalized to the optimal solution for a varying number of states explored for SA and the greedy algorithms for a security/defense system where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, $|S| = 729$.

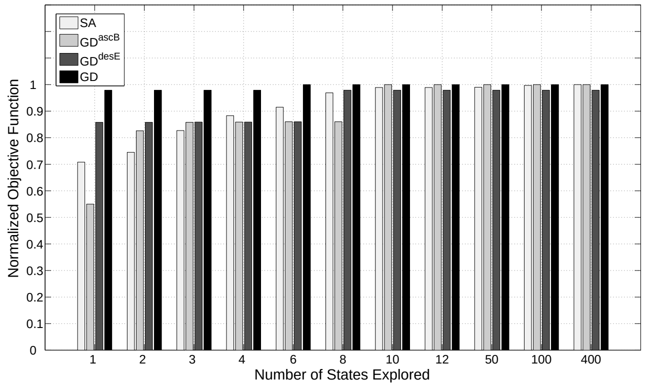

Fig. 4. Objective function value normalized to the optimal solution for a varying number of states explored for SA and greedy algorithms for a health care application where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, $|S| = 729$.

optimization algorithm (GD) (which leverages intelligent initial parameter value selection, exploration ordering, and parameter arrangement), and several other greedy online algorithm variations (Table I) in C/C++. First, we evaluate our methodology and algorithms with a desktop implementation to reduce the analysis time and study the feasibility of our algorithms for a dynamic environment. We compared our results with SA to provide relative comparisons of greedy algorithms with another heuristic algorithm. We compared GD results with different greedy algorithm variations (Table I) to provide an insight into how initial parameter value settings, exploration ordering, and parameter arrangement affect the final operating state quality. We normalized the objective function value (corresponding to the operating state) attained by the algorithms with respect to the optimal solution obtained using an exhaustive search. We analyzed the relative complexity of the algorithms by measuring the execution time and data memory requirements. Note that for brevity, our presented results are a subset of the greedy algorithms listed in (Table I), the application domains, and the design spaces. However, we evaluated all the greedy algorithms and application domains and the subset presented in this section is representative of all greedy algorithms and application domain trends and characteristics.

Fig. 3 shows the objective function value normalized to the optimal solution versus number of states explored for a security/defense system for $|S| = 729$. GD$^{\text{ascA}}$, GD$^{\text{desD}}$, and GD converge to a steady state solution after exploring 11, 10, and 8 states, respectively. These convergence results show that GD converged to the optimal solution slightly faster than other greedy algorithms, exploring only 1.1% of the design space. The order in which greedy algorithms explore tunable parameters (e.g. V_p, then F_p, and so on, etc.) affect the number of iterations for convergence similarly to how the order of variables in a binary decision diagram (BDD) impacts the size of the tree and processing speed [19]. GD$^{\text{ascA}}$ converges to a better solution than GD$^{\text{desD}}$ showing that ascending parameter values exploration is better for a security/defense system with the given weight factors. In addition, the SA algorithm outperforms all greedy algorithms and converges to the optimal solution for $|S| = 729$ after exploring 400 states (54.9% of the design space). Fig. 3 also verifies the ability of our methodology to determine a good quality (near-optimal) solution in one-shot, as GD achieves only a 1.83% percentage improvement over the initial state after exploring 8 states.

Fig. 4 shows the objective function value normalized to the optimal solution versus number of states explored for a health care application for $|S| = 729$. Fig. 4 shows similar trends as seen in Fig. 3 for convergence rates on all algorithms, GD's one-shot solution quality, and GD$^{\text{ascB}}$'s superior solution as compared to GD$^{\text{desE}}$.

Fig. 5 shows the objective function value normalized to the optimal solution versus number of states explored for an ambient conditions monitoring application for $|S| = 31,104$. GD converges to the optimal solution after exploring 13 states (0.04% of design space), with a 16.69% improvement over the one-shot solution. GD$^{\text{ascC}}$ and GD$^{\text{desF}}$ converge to the solution after exploring 9 and 7 states, respectively. These

TABLE II
PERCENTAGE IMPROVEMENT ATTAINED BY GD AFTER ONE STATE EXPLORATION WHEN $|S| = 729$.

Application Domain	**GD^{ascA}**	**GD^{ascB}**	**GD^{ascC}**	**GD^{desD}**	**GD^{desE}**	**GD^{desF}**
Security/Defense System	155.06%	155.06%	155.06%	439.56%	22.9%	9.2%
Health Care	78%	78%	78%	211.78%	14.1%	6.64%
Ambient Conditions Monitoring	51.81%	51.81%	51.81%	142.89%	39.85%	6.09%

TABLE III
PERCENTAGE IMPROVEMENT ATTAINED BY GD AFTER ONE STATE EXPLORATION WHEN $|S| = 31,104$.

Application Domain	**GD^{ascA}**	**GD^{ascB}**	**GD^{ascC}**	**GD^{desD}**	**GD^{desE}**	**GD^{desF}**
Security/Defense System	147.97%	147.97%	147.97%	435.09%	13.66%	0.33%
Health Care	73.42%	73.42%	73.42%	218.77%	11.47%	0.3%
Ambient Conditions Monitoring	0%	0%	0%	146.97%	-8.24%	75.61%

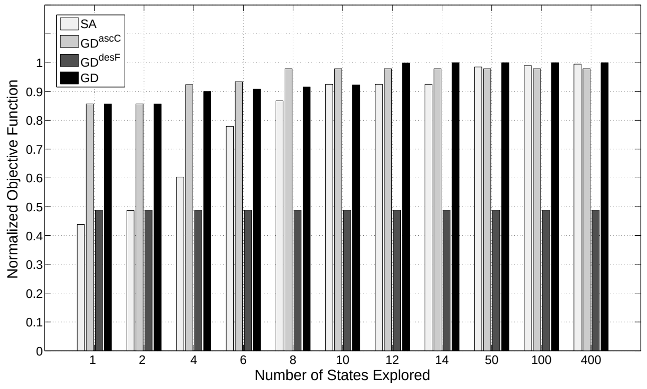

Fig. 5. Objective function value normalized to the optimal solution for a varying number of states explored for SA and greedy algorithms for an ambient conditions monitoring application where $\omega_l = 0.6$, $\omega_t = 0.25$, $\omega_r = 0.15$, $|S| = 31,104$.

convergence and percentage improvement results show that GD may explore more states than other greedy algorithms if state exploration provides a noticeable improvement over the one-shot solution. The figure also shows that SA converges to a near-optimal solution after exploring 400 states (1.29% of design space). These convergence results show that even though the design space cardinality increases by 43x, both heuristic algorithms (greedy and SA) still explore only a small percentage of the design space and result in high-quality solutions.

In order to prove the effectiveness of our intelligent initial parameter value selection technique (Section IV-A), we calculated the percentage improvements in the normalized objective function value obtained by the intelligent initial parameter value settings over other arbitrary initial value settings used in other greedy algorithms for $|S| = 729$ (Table II) and $|S| = 31,104$ (Table III). Results reveal that our one-shot operating state using only intelligent initial parameter settings is within 5.92% of the optimal averaged over several different application domains and design spaces. Table II and Table III verify that our intelligent initial parameter settings technique (which gives a near-optimal solution) provides substantial percentage improvements over other arbitrary initial settings for different application domains and design space cardinalities. We point out that some arbitrary initial parameter settings may attain a slightly higher normalized objective function value for a particular application and design space cardinality (e.g. initial parameter settings for GD^{desE} for ambient conditions monitoring application when $|S| = 31,104$), but in general, the arbitrary selection would not scale to other applications and design space cardinalities.

We performed data memory analysis for each step of our dynamic optimization methodology (Section 1). Step one (one-shot solution) requires only 150, 188, 248, and 416 bytes whereas step two requires 94, 140, 200, and 494 bytes for (number of tunable parameters N, number of application metrics m) equal to (3, 2), (3, 3), (6, 3), and (6, 6), respectively. For step three, we compared data memory requirements for GD with SA for different design space cardinalities. We observed that GD requires 458, 528, 574, 870, and 886 bytes, whereas SA requires 514, 582, 624, 920, and 936 bytes of storage for design space cardinalities of 8, 81, 729, 31104, 46656, respectively. The data memory analysis shows that SA has comparatively larger memory requirements than the greedy algorithm. Our analysis reveals that the data memory requirements for all three steps of our dynamic optimization methodology increases linearly as the number of tunable parameters, tunable values, and application metrics (and thus the design space) increases. Furthermore, the data memory analysis verifies that although our dynamic optimization methodology (all three steps) has low data memory requirements, the one-shot solution (from step one) requires 361.36% less memory on average.

We measured the execution time for all three steps of our dynamic optimization methodology averaged over 10,000 runs (to smooth any discrepancies in execution time due to operating system overheads) on an Intel Xeon CPU running at 2.66 GHz [20] using the Linux/Unix `time` command [21]. We scaled these execution times to the Atmel ATmega1281 microcontroller [14] running at 8 MHz. Even though scaling does not provide 100% accuracy for the microcontroller runtime because of different instruction set architectures, scaling provides reasonable runtime estimates and enables relative comparisons. Results showed that step one and step two required 1.66 ms and 0.332 ms, respectively, both for

$|S| = 729$ and $|S| = 31,104$. For step three, we compared GD with SA. GD explored 10 states and required 0.887 ms and 1.33 ms on average to converge to the solution for $|S| = 729$ and $|S| = 31,104$, respectively. SA took 2.76 ms and 2.88 ms to explore the first 10 states (to provide a fair comparison with GD) for $|S| = 729$ and $|S| = 31,104$, respectively. The other greedy algorithms required comparatively more time than GD because they required more states exploration to converge than GD, but however, all greedy algorithms required less execution time than SA. To verify that our dynamic optimization methodology is lightweight, we compared the execution time results for our dynamic optimization methodology (including all three steps) with the exhaustive search. The exhaustive search required 29.526 ms and 2.765 seconds for $|S| = 729$ and $|S| = 31,104$, respectively. Compared with the exhaustive search, our dynamic optimization methodology required 10.26x and 832.33x less execution time for $|S| = 729$ and $|S| = 31,104$, respectively. The execution time analysis reveals that our dynamic optimization methodology (including all three steps) requires execution time on the order of milliseconds, and the one-shot solution requires 138.32% less execution time on average as compared to all three steps of the dynamic optimization methodology. Execution time savings attained by the one-shot solution as compared to the three steps of our dynamic optimization methodology are 73.43% and 186.26% for GD and SA, respectively, when $|S| = 729$, and are 100.12% and 138.32% for GD and SA, respectively, when $|S| = 31,104$. These results indicate that the design space cardinality affects the execution time linearly and our dynamic optimization methodology's advantage increases as design space cardinality increases.

VI. Conclusions and Future Work

In this paper, we proposed a dynamic optimization methodology for WSNs, which provided a high-quality solution in just one-shot using an intelligent initial tunable parameter value settings for highly constrained applications. Additionally, we proposed an online greedy optimization algorithm that leveraged intelligent design space exploration techniques to iteratively improve on the one-shot solution for less constrained applications. Compared with SA and different greedy algorithm variations, results showed that one-shot technique yielded improvements as high as 439.56% over other arbitrary initial parameter settings. Results indicated that our greedy algorithm converged to the optimal (or near-optimal) solution after exploring only 1.1% and 0.04% of the design space whereas SA explored 54.9% and 1.29% of the design space for $|S| = 729$ and $|S| = 31,104$, respectively. Data memory and execution time analysis revealed that our one-shot solution (step one) required 361.36% and 138.32% less data memory and execution time, respectively, when compared to using all the three steps of our dynamic optimization methodology. Results revealed that our dynamic optimization methodology (including all three steps) required 10.26x and 832.33x less execution time as compared to the exhaustive search for $|S| = 729$ and $|S| = 31,104$, respectively. Computational complexity analysis confirmed that our methodology is lightweight and thus feasible for sensor nodes with limited resources.

Future work includes the incorporation of profiling statistics into our dynamic optimization methodology to provide feedback with respect to changing environmental stimuli. In addition, we plan to further verify our dynamic optimization methodology by implementation on a hardware sensor node platform.

Acknowledgments

This work was supported by the National Science Foundation (NSF) (CNS-0834080) and 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 NSERC.

References

[1] A. Munir and A. Gordon-Ross, "An MDP-based Application Oriented Optimal Policy for Wireless Sensor Networks," in *Proc. ACM CODES+ISSS'09*, Grenoble, France, October 2009, pp. 183–192.

[2] 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.

[3] S. Patel and S. Lumetta, "rePLay: A Hardware Framework for Dynamic Optimization," *IEEE Trans. on Computers*, vol. 50, no. 6, pp. 590–608, June 2001.

[4] C.-Y. Seong and B. Widrow, "Neural Dynamic Optimization for Control Systems," *IEEE Trans. on Systems, Man, and Cybernatics*, vol. 31, no. 4, pp. 482–489, August 2001.

[5] C. Zhang, F. Vahid, and R. Lysecky, "A Self-Tuning Cache Architecture for Embedded Systems," *ACM Trans. on Embedded Computing Systems*, vol. 3, no. 2, pp. 407–425, May 2004.

[6] A. Shenoy, J. Hiner, S. Lysecky, R. Lysecky, and A. Gordon-Ross, "Evaluation of Dynamic Profiling Methodologies for Optimization of Sensor Networks," *IEEE Embedded Systems Letters*, vol. 2, no. 1, pp. 10–13, March 2010.

[7] S. Kogekar et al., "Constraint-Guided Dynamic Reconfiguration in Sensor Networks," in *Proc. ACM IPSN'04*, Berkeley, California, April 2004, pp. 379–387.

[8] R. Verma, "Automated application specific sensor network node tuning for non-expert application developers," Master's thesis, ECE Dept., Univ. of Arizona, 2008.

[9] R. Min, T. Furrer, and A. Chandrakasan, "Dynamic Voltage Scaling Techniques for Distributed Microsensor Networks," in *Proc. IEEE WVLSI'00*, Orlando, Florida, April 2000, pp. 43–46.

[10] L. Yuan and G. Qu, "Design Space Exploration for Energy-Efficient Secure Sensor Network," in *Proc. IEEE ASAP'02*, San Jose, California, July 2002, pp. 88–97.

[11] (2010, May) Dynamic Profiling and Optimization (DPOP) for Sensor Networks. [Online]. Available: http://www.ece.arizona.edu/~dpop/

[12] A. Gordon-Ross, F. Vahid, and N. Dutt, "Fast Configurable-Cache Tuning With a Unified Second-Level Cache," *IEEE Trans. on VLSI Systems*, vol. 17, no. 1, pp. 80–91, January 2009.

[13] Crossbow. (2010, May) Crossbow IRIS Datasheet. [Online]. Available: http://www.xbow.com/

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

[15] Crossbow. (2010, May) MTS/MDA Sensor Board Users Manual. [Online]. Available: http://www.xbow.com/

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

[17] Atmel. (2010, May) ATMEL AT86RF230 Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE and ISM Applications. [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf

[18] I. Akyildiz et al., "Wireless Sensor Networks: A Survey," *Elsevier Computer Networks*, vol. 38, no. 4, pp. 393–422, March 2002.

[19] M. Fujita, Y. Matsunaga, and T. Kakuda, "On Variable Ordering of Binary Decision Diagrams for the Application of Multi-level Logic Synthesis," in *Proc. ACM EuroDAC'91*, Amsterdam, Netherlands, February 1991, pp. 50–54.

[20] (2010, May) Intel Xeon Processor E5430. [Online]. Available: http://processorfinder.intel.com/details.aspx?sSpec=SLANU

[21] (2010, May) Linux Man Pages. [Online]. Available: http://linux.die.net/man/