A One-Shot Dynamic Optimization Methodology and Application Metrics Estimation Model 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@ece.ufl.edu

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

***Abstract*—Wireless sensor networks (WSNs), consisting of autonomous sensor nodes, have emerged as ubiquitous networks that span diverse application domains (e.g., health care, logistics, defense) each with varying application requirements (e.g., lifetime, throughput, reliability). Typically, sensor-based platforms possess tunable parameters (e.g., processor voltage, processor frequency, sensing frequency), which enable platform specialization for particular application requirements. WSN application design can be daunting for application developers, which are oftentimes not trained engineers (e.g., biologists, agriculturists) who wish to utilize the sensor-based systems within their given domain. Dynamic optimizations enable sensor-based platforms to tune parameters in-situ to automatically determine an optimized operating state. However, rapidly changing application behavior and environmental stimuli necessitate a lightweight and highly responsive dynamic optimization methodology. In this paper, we propose a very lightweight dynamic optimization methodology that determines initial tunable parameter settings to give a high-quality operating state in *one-shot* for time-critical and highly constrained applications. We compare our one-shot dynamic optimization methodology with other lightweight dynamic optimization methodologies (i.e., greedy- and simulated annealing-based) to provide insights into the solution quality and resource requirements of our methodology. Results reveal that the one-shot solution is within 8% of the optimal solution on average. To assist dynamic optimizations in determining an operating state, we propose an application metric estimation model to establish a relationship between application metrics (e.g., lifetime, throughput) and sensor-based platform parameters.**

***Keywords*-Wireless sensor networks, dynamic optimization, application metrics estimation**

I. Introduction and Motivation

Wireless sensor networks (WSNs) consist of spatially distributed autonomous sensor nodes that observe a phenomenon (environment, target, etc.). WSNs are becoming ubiquitous because of their proliferation in diverse application domains (e.g., defense, health care, logistics) each with varying application requirements (e.g., lifetime, throughput, reliability) [1]. For example, a security/defense system may have a higher throughput requirement whereas an ambient conditions monitoring application may be more sensitive to lifetime. This diversity makes WSN design challenging with commercial-off-the-shelf (COTS) sensor nodes.

COTS sensor nodes are mass-produced to optimize cost and are not specialized for any particular application. Furthermore, WSN application developers oftentimes are not trained engineers, but rather biologists, teachers, or agriculturists who wish to utilize the sensor-based systems within their given domain. Fortunately, many COTS sensor nodes possess tunable parameters (e.g., processor voltage and frequency, sensing frequency) whose values can be *tuned* for a specific application. Faced with an overwhelming number of tunable parameter choices, WSN design can be a daunting task for non-experts and necessitates an automated parameter tuning process for assistance.

Parameter optimization is the process of assigning appropriate (optimal or near-optimal) values to tunable parameters either statically or dynamically to meet application requirements. *Static optimizations* assign parameter values at deployment and these values remain fixed during the sensor node's lifetime. Accurate prediction/simulation of environmental stimuli is challenging and applications with changing environmental stimuli do not benefit from static optimizations. Alternatively, *dynamic optimizations* assign parameter values during runtime and reassign/change these values in accordance with changing environmental stimuli, thus enabling close adherence to application requirements.

There exists much research in the area of dynamic optimizations [2][3][4][5][6], but most previous work targets the memory (cache) or processor in computer systems. Little work exists on WSN dynamic optimization, which presents additional challenges because of a unique design space, energy constraints, and operating environment. The dynamic profiling and optimization (DPOP) project aspires to alleviate the complexities associated with sensor-based system design using dynamic profiling methods capable of observing application-level behavior and dynamic optimization to tune the underlying platform accordingly [7]. The DPOP project has evaluated dynamic profiling methods for observing application-level behavior by gathering profiling statistics, but dynamic optimization methods still need exploration. In this paper, we explore a fine-grained design space for sensor-based platforms with many tunable parameters to more closely meet application requirements (Gordon-Ross et al. [8] showed that finer-grained design spaces provide interesting design alternatives and result in increased benefits in the cache subsystem). The exploration of a fine-grained design space coupled with limited battery reserves and rapidly changing application requirements and environmental stimuli necessitates a lightweight and highly responsive dynamic optimization methodology.

Our main contributions in this paper are:

- We propose a lightweight dynamic optimization methodology that determines appropriate initial tunable parameter values to give a good quality operating state (tunable parameter value settings) in *one-shot* with minimal design exploration for highly constrained applications. Results reveal that this one-shot operating state is within 8% of the optimal solution (obtained from exhaustive search) averaged over several different application domains and design spaces.
- We evaluate alternative initial parameter settings to provide a comparison with our one-shot initial parameter settings. Results reveal that the average percentage improvement attained by the one-shot initial parameter settings over alternative initial parameter settings for different application domains and design spaces is 33% on average.
- We analyze memory and execution time requirements of our one-shot dynamic optimization methodology and compare these with other lightweight dynamic optimization methodologies (greedy- and simulated-annealing (SA)-based). Results indicate that our one-shot dynamic optimization methodology requires 204% and 458% less memory on average as compared to the greedy- and SA-based methodologies, respectively. The one-shot solution requires 18% less execution time on average as compared to the greedy- and SA-based methodologies even if these methodologies are restricted to explore only 0.03% of the design space on average.
- To assist dynamic optimizations in determining an operating state, we for the first time, to the best of our knowledge, propose an *application metric estimation model*, which estimates high-level application metrics (lifetime, throughput, and reliability) from sensor-based platform parameters (e.g., processor voltage and frequency, sensing frequency, transceiver transmission power, etc.). Our one-shot dynamic optimization methodology leverages this estimation model when comparing different operating states for optimization purposes. We emphasize that this application metric estimation model can be leveraged by any dynamic optimization methodology and facilitates the WSN design process.

The remainder of this paper is organized as follows. Section II surveys previous work in the area of dynamic optimizations. Section III presents our one-shot dynamic optimization methodology and Section IV describes our application metrics estimation model leveraged by our one-shot dynamic optimization methodology. Section V presents experimental results and Section VI presents conclusions and future research work directions.

II. Related Work

There exists much research in the area of dynamic optimizations [2][3][4][5][6][9][10], however, most previous work focuses on the processor or memory (cache) in computer systems. Whereas previous work can provide valuable insights into WSN dynamic optimizations, these works are not directly applicable due to a WSN's unique design space, energy constraints, and operating environment.

In the area of WSN dynamic profiling and optimizations, Sridharan et al. [11] obtained accurate environmental stimuli by dynamically profiling the WSN's operating environment, but did not propose any methodology to leverage these profiling statistics for optimizations. Shenoy et al. [12] presented profiling methods for dynamically monitoring sensor-based platforms and analyzed the associated network traffic and energy, but did not explore dynamic optimizations. In prior work, Munir et al. [13] proposed a Markov Decision Process (MDP)-based methodology as a first step towards WSN dynamic optimizations, but this method required prohibitively large computational resources for larger design spaces. Ideally, this method required a base station node with more computing resources to carry out the optimal operating state determination process, and these operating states could be communicated to other sensor nodes. The large computational requirements inhibited the methodology's implementation on resource constrained sensor nodes to enable autonomous operating state decisions. Kogekar et al. [14] proposed an approach for dynamic software reconfiguration in WSNs using adaptive software, which used tasks to detect environmental changes (event occurrences) and then adapted the software to the new conditions. Though their work considered software reconfiguration, they did not consider senor node tunable parameters.

In the area of WSN optimizations, Wang et al. [15] proposed a distributed energy optimization method for target tracking applications. The energy management mechanism consisted of an optimal sensing scheme that leveraged dynamic awakening of sensor nodes. The dynamic awakening scheme awoke the group of sensor nodes located in the target's vicinity for reporting the sensed data. The results verified that dynamic awakening combined with optimal sensor node selection enhanced the WSN energy efficiency. Liu et al. [16] proposed a dynamic node collaboration scheme for mobile target tracking in wireless camera sensor networks (wireless camera sensor networks can provide much more accurate information in target tracking applications as compared to traditional sensor networks). The proposed scheme comprised of two components: a cluster head election scheme during the tracking process and an optimization algorithm to select an optimal subset of camera sensors as the cluster members for cooperative estimation of the target's location. Khanna et al. [17] proposed a reduced-complexity genetic algorithm for secure and dynamic deployment of resource constrained multi-hop WSNs. The genetic algorithm adaptively configured optimal position and security attributes by dynamically monitoring network traffic, packet integrity, and battery usage.

Several papers explored dynamic voltage and frequency scaling (DVFS) for reduced energy consumption in WSNs. Min et al. [18] demonstrated that dynamic processor voltage scaling reduced energy consumption by 60%. Similarly, Yuan et al. [19] 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 better meet application requirements. To the best of our knowledge, our work is the first to explore an extensive sensor node design space.

In prior work, Lysecky et al. [20] proposed SA-based automated application specific tuning of parameterized sensor-based embedded systems and found that automated tuning can better meet application requirements by 40% on average as compared to a static configuration of tunable parameters. Verma [21] studied SA-based and particle swarm optimization (PSO) methods for automated application specific tuning and observed that an SA-based method performed better than PSO because PSO often quickly converged to local minima. Exhaustive search algorithms have been used in literature for performance analysis and comparison with heuristic algorithms. Mannion et al. [22] proposed a PareDown decomposition algorithm for partitioning pre-defined behavioral blocks onto a minimum number of programmable sensor blocks and compared the partitioning algorithm's performance with an exhaustive search algorithm. Meier et al. [23] proposed an exhaustive search based scheme called NoSE (Neighbor Search and link Estimation) for neighbor search, link assessment, and energy consumption minimization.

Even though there exists some work on optimizations in WSNs [15][18][19][24][25][26][27], dynamic optimizations require further research and more in depth considerations. Specifically, a sensor node's constrained energy and storage resources necessitate lightweight dynamic optimization methodologies for sensor node parameter tuning.

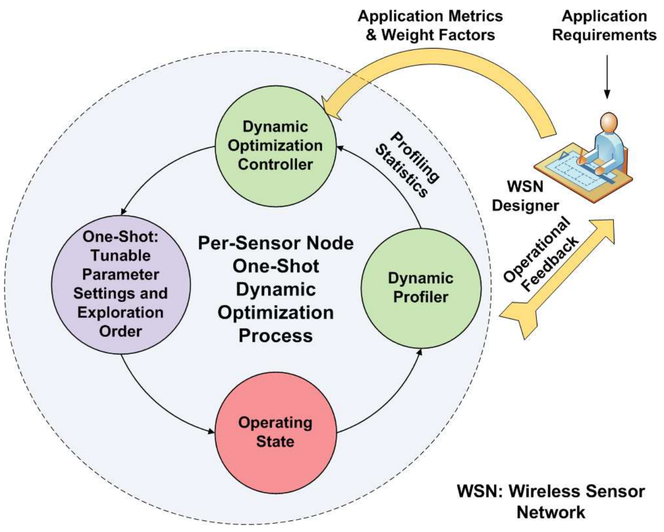
Application Metrics & Weight Factors
Application Requirements
Dynamic Optimization Controller
Profiling Statistics
WSN Designer
One-Shot: Tunable Parameter Settings and Exploration Order
Per-Sensor Node One-Shot Dynamic Optimization Process
Dynamic Profiler
Operational Feedback
Operating State
WSN: Wireless Sensor Network

Figure 1. One-shot dynamic optimization methodology for wireless sensor networks.

III. Dynamic Optimization Methodology

In this section, we give an overview of our one-shot dynamic optimization methodology and the associated algorithm. We also formulate the state space and objective function for our methodology.

A. Overview

Fig. 1 depicts our one-shot 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 signify the relative weightage/importance of application metrics with respect to each other. The dynamic optimization methodology leverages an application metric estimation model to determine application metric values offered by an operating state (we describe this application metric estimation model in Section IV).

Fig. 1 shows the per-node one-shot dynamic optimization process (encompassed by the dashed circle), which is orchestrated by the *dynamic optimization controller*. The dynamic optimization controller invokes the *one-shot step* wherein the sensor node operating state is directly determined by intelligent tunable parameter value settings, and hence the methodology is termed as *one-shot*. The one-shot step also determines an exploration order (ascending or descending) for tunable parameters. This exploration order can be leveraged by an *online optimization algorithm* to provide improvements over the one-shot solution by further design space exploration and is the focus of our future work. This exploration order is critical in reducing the number of states explored by the online optimization algorithm. The sensor node moves directly to the operating state specified by the one-shot step. A *dynamic profiler* records profiling statistics (e.g., battery energy, wireless channel condition) 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 if the current operating state meets the application requirements. If the application requirements are not met, the dynamic optimization controller reinvokes the one-shot dynamic optimization process to determine the new operating state. This feedback process continues to ensure the selection of a good operating state to better meet application requirements in the presence of changing environmental stimuli.

B. State Space

The state space S for our one-shot 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's state space cardinality (the number of tunable values in P_i). S is a set of n-tuples (each n-tuple s represents a sensor node state) formed by taking one tunable parameter value from each tunable parameter. A single n-tuple $s \in S$ is given as:

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

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 treated as *do not care* tuples.

C. Optimization Objection Function

The sensor node dynamic optimization problem can be formulated as an unconstrained optimization problem:

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

where $f(s)$ denotes the objective function characterizing 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. Each state $s \in S$ has an associated objective function value and the optimization goal is to determine a state that gives the maximum (optimal) objective function value $f^{opt}(s)$ ($f^{opt}(s)$ indicates the best possible adherence to the specified application requirements given the design space S). The solution quality for any $s \in S$ can be determined by normalizing the objective function value corresponding to state s with respect to $f^{opt}(s)$. The normalized objective function value corresponding to a state can vary from 0 to 1 where 1 indicates the optimal solution.

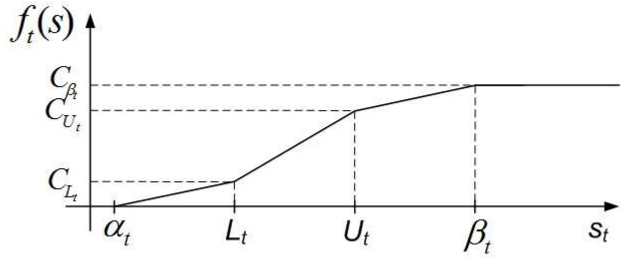

Figure 2. Throughput objective function $f_t(s)$.

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 $f_t(s)$ (Fig. 2) using the piecewise linear function:

$$f_t(s) = \begin{cases} 1, & s_t \geq \beta_t \\ C_{U_t} + \frac{(C_{\beta_t} - C_{U_t})(s_t - U_t)}{(\beta_t - U_t)}, & U_t \leq s_t < \beta_t \\ C_{L_t} + \frac{(C_{U_t} - C_{L_t})(s_t - L_t)}{(U_t - L_t)}, & L_t \leq s_t < U_t \\ C_{L_t} \cdot \frac{(s_t - \alpha_t)}{(L_t - \alpha_t)}, & \alpha_t \leq s_t < L_t \\ 0, & s_t < \alpha_t. \end{cases} \tag{5}$$

where s_t denotes the throughput offered by state s, the constant parameters L_t and U_t denote the *desired* minimum and maximum throughput, respectively, and the constant parameters α_t and β_t denote the *acceptable* minimum and maximum throughput, respectively. The constant parameters C_{L_t}, C_{U_t}, and C_{β_t} in (5) denote the $f_t(s)$ value at L_t, U_t, and β_t, respectively. A piecewise linear objective function captures accurately the desirable and acceptable ranges of a particular application metric. We consider piecewise linear objective functions as a typical example, however, our methodology works well for any other objective function characterization (e.g., linear, non-linear) [13].

The $f_l(s)$ and $f_r(s)$ can be defined similar to (5).

We point out that some tunable parameters may affect multiple application metrics (e.g., sending at a lower power might conserve energy but may increase the packet loss ratio). Our dynamic optimization objective function handles such multi-effect parameters appropriately. Since our dynamic optimization objective function is a weighted sum of objective functions of individual application metrics, the overall solution will be an operating state in which the affect of these parameters is balanced out in a way that gives the maximum overall objective function value. An application metric with a higher weight factor will be given precedence in arbitrating between the tunable parameters that affect multiple application metrics.

D. One-Shot Dynamic Optimization Algorithm

In this subsection, we describe the algorithm for our one-shot dynamic optimization methodology. The algorithm determines initial tunable parameter value settings and exploration order (ascending or descending). This exploration order can be used for the exploration of tunable parameters if further improvement over the one-shot solution is desired, and this improvement is the focus of our future work.

```
Input: f(s), N, n, m, $\boldsymbol{P}$
Output: Initial tunable parameter value settings and exploration
        order
for $k \leftarrow 1$ to $m$ do
    for $P_i \leftarrow P_1$ to $P_N$ do
        $f^k_{p_{i_1}} \leftarrow k^{\text{th}}$ metric objective function value when
        parameter setting is $\{P_i = p_{i_1}, P_j = P_{j_0}, \forall\, i \neq j\}$ ;
        $f^k_{p_{i_n}} \leftarrow k^{\text{th}}$ metric objective function value when
        parameter setting is $\{P_i = p_{i_n}, P_j = P_{j_0}, \forall\, i \neq j\}$ ;
        $\delta f^k_{P_i} \leftarrow f^k_{p_{i_n}} - f^k_{p_{i_1}}$ ;
        if $\delta f^k_{P_i} \geq 0$ then
            explore $P_i$ in descending order ;
            $P^k_d[i] \leftarrow$ descending ;
            $P^k_0[i] \leftarrow p^k_{i_n}$ ;
        else
            explore $P_i$ in ascending order ;
            $P^k_d[i] \leftarrow$ ascending ;
            $P^k_0[i] \leftarrow p^k_{i_1}$ ;
        end
    end
end
return $\boldsymbol{P^k_d}$, $\boldsymbol{P^k_0}, \forall\, k \in \{1, \ldots, m\}$
```

Algorithm 1: One-shot dynamic optimization algorithm.

Algorithm 1 describes our one-shot dynamic optimization algorithm to determine initial tunable parameter value settings and exploration order. 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 (we assume for simplicity that tunable parameters have an equal number of tunable values, however, other values can be taken), 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 $f^k_{p_{i_1}}$ and $f^k_{p_{i_n}}$ (the k^{th} application metric objective function values) where the parameter being explored P_i is assigned its first p_{i_1} and last p_{i_n} tunable values, respectively, and the remainder 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 $f^k_{p_{i_1}}$. $\delta f^k_{P_i} \geq 0$ means that p_{i_n} results in an equal or 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). To reduce the number of states explored while considering that an online optimization algorithm (e.g., greedy-based algorithm) will typically stop exploring a tunable parameter if a tunable parameter's value yields a comparatively lower (or equal) 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).

Algorithm 1 determines appropriate initial parameter value settings corresponding to individual application metrics, however, further calculations are required to determine intelligent initial parameter value settings suitable for all the application metrics because the best initial value settings for different application metrics may be different. Since some parameters are more critical to meeting application requirements than other parameters depending on the application metric weight factors, more consideration should be given to the initial parameter value settings corresponding to the application metrics with higher weight factors. For example, sensing frequency is a critical parameter for applications with a high responsiveness weight factor and therefore, initial value settings corresponding to the responsiveness application metric should be given priority. We devise a technique for intelligent initial value settings such that the initial value settings consider the impact of these settings on the overall objective function considering all the application metrics and the application metrics' associated weight factors. Our initial value settings technique is based on the calculations performed in Algorithm 1.

The initial value settings vector $\boldsymbol{P^k_0}$ corresponding to application metric k is given by:

$$\boldsymbol{P^k_0} = \{P^k_{0_1}, P^k_{0_2}, \ldots, P^k_{0_N}\}, \forall\, k \in \{1, 2, \ldots, m\} \qquad (6)$$

where $P^k_{0_i}$ denotes the initial value setting for tunable parameter i, $\forall\, i \in \{1, 2, \ldots, N\}$ corresponding to the k^{th} application metric (as given by Algorithm 1). An intelligent initial value setting vector $\widehat{\boldsymbol{P_0}}$ must consider all application metrics' weight factors with higher importance given to higher weight factors, i.e.,:

$$\widehat{\boldsymbol{P_0}} = \{P^1_{0_1}, \ldots, P^1_{0_{l_1}}, P^2_{0_1}, \ldots, P^2_{0_{l_2}}, P^3_{0_1}, \ldots, P^3_{0_{l_3}}, \ldots, P^m_{0_1}, \ldots, P^m_{0_{l_m}}\} \quad (7)$$

where l_k denotes the number of initial value settings taken from $P^k_0, \forall\ k \in \{1, 2, \ldots, m\}$ such that $\sum_{k=1}^{m} l_k = N$. Our technique allows taking more initial value settings corresponding to application metrics with higher weight factors (since Algorithm 1 gives appropriate initial value settings for each application metric separately), i.e., $l_k \geq l_{k+1} \Leftrightarrow \omega_k \geq \omega_{k+1}, \forall\ k \in \{1, 2, \ldots, m-1\}$. In (7), l_1 initial value settings are taken from vector P^1_0, then l_2 from vector P^2_0, and so on to l_m from vector P^m_0 such that $\{P^k_{0_1}, \ldots, P^k_{0_{l_k}}\} \cap \{P^{k-1}_{0_1}, \ldots, P^{k-1}_{0_{l_{k-1}}}\} = \emptyset, \forall\ k \in \{2, 3, \ldots, m\}$. In other words, we select those initial value settings corresponding to the application metrics with lower weight factors that are not already selected based on the application metrics with higher weight factors (i.e., $\widehat{\boldsymbol{P_0}}$ comprises of disjoint or non-overlapping initial value settings).

In the situation where a weight factor ω_1 is much greater than all of the other weight factors, an intelligent initial value setting $\widetilde{\boldsymbol{P_0}}$ would correspond to the initial value settings based on the application metric with weight factor ω_1, i.e.,:

$$\widetilde{\boldsymbol{P_0}} = \boldsymbol{P^1_0} = \{P^1_{0_1}, P^1_{0_2}, \ldots, P^1_{0_N}\} \Leftrightarrow \omega_1 \gg \omega_q, \forall\ q \in \{2, 3, \ldots, m\} \quad (8)$$

E. Computational Complexity

The computational complexity of our one-shot dynamic optimization methodology is $\mathcal{O}(Nm)$, which is comprised of the intelligent initial parameter value settings for individual application metrics and exploration ordering (Algorithm 1) $\mathcal{O}(Nm)$, and intelligent initial value settings considering all the application metrics $\mathcal{O}(N+m)$, based on the Algorithm 1 calculations (Section III-D). This complexity reveals that our one-shot methodology is lightweight and is thus feasible for sensor nodes with tight resource constraints.

IV. Application Metrics Estimation Model

In this section, we propose an application metric estimation model, which is leveraged by our one-shot dynamic optimization methodology. This estimation model estimates high-level application metrics (lifetime, throughput, reliability) from a sensor node's parameters (e.g., processor voltage and frequency, sensing frequency, transceiver voltage, etc.). For brevity, we describe only the model's key elements. Table I presents a summary of key notations used in our application metrics estimation model.

A. Lifetime Estimation

The *lifetime* of a sensor node is defined as the time duration between the deployment time and the time before which the sensor node fails to perform the assigned task due to sensor node failure, which is normally caused by battery energy depletion. A sensor node typically contains AA alkaline batteries whose energy depletes gradually as the sensor node consumes energy during operation. The critical factors in determining sensor node lifetime are battery energy and energy consumption during operation.

Table I
Summary of application metrics estimation model notations

Notation	Description
$\mathcal{L}_s$	Lifetime in days
E_b	Battery energy in Joules
E_c	Energy consumption per hour
V_b	Battery voltage in volts
C_b	Battery capacity in mA-h
E_{proc}	Processing energy per hour
E_{com}	Communication energy per hour
E_{sen}	Sensing energy per hour
E^a_{proc}	Processing energy per hour in active mode
E^i_{proc}	Processing energy per hour in idle mode
E^{tx}_{trans}	Transceiver transmission energy per hour
E^{rx}_{trans}	Transceiver receive energy per hour
E^i_{trans}	Transceiver idle energy per hour
N^{tx}_{pkt}	Number of packets transmitted per hour
E^{pkt}_{tx}	Transmission energy per packet
V_t	Transceiver voltage
I_t	Transceiver current
t^{pkt}_{tx}	Time to transmit one packet
I^s_t	Transceiver sleep current
t^i_{tx}	Transceiver idle time per hour
P_s	Packet size in bytes
R_{tx}	Transceiver data rate (in bits/second)
E^m_{sen}	Sensing measurement energy
E^i_{sen}	Sensing idle energy
N_r	Number of sensors on the sensing board
N_s	Number of sensing measurements per second
V_s	Sensing board voltage
I^m_s	Sensing measurement current
t^m_s	Sensing measurement time
I_s	Sensing sleep current
t^i_s	Sensing idle time
R	Aggregate throughput
R_{sen}	Sensing throughput
R_{proc}	Processing throughput
R_{com}	Communication throughput
F_s	Sensing frequency
R^b_{sen}	Sensing resolution bits
F_p	Processor frequency
N^b	Number of instructions to process one bit
t^{pkt}_{tx}	Time to transmit one packet
P^{eff}_s	Effective packet size

The sensor node lifetime in days $\mathcal{L}_s$ can be estimated as:

$$\mathcal{L}_s = \frac{E_b}{E_c \times 24} \quad (9)$$

where E_b denotes the sensor node's battery energy (Joules) and E_c denotes the sensor node's energy consumption per hour. The battery energy in mWh $E_b^{'}$ can be given by:

$$E_b^{'} = V_b \cdot C_b \ \ (mWh) \tag{10}$$

where V_b denotes the battery voltage (Volts) and C_b denotes the battery capacity (typically mA-h). Since 1J = 1 Ws, E_b can be calculated as:

$$E_b = E_b^{'} \times 3600/1000 \ \ (J) \tag{11}$$

We model E_c as the sum of the processing, communication, and sensing energies, i.e.,:

$$E_c = E_{proc} + E_{com} + E_{sen} \ \ (J) \tag{12}$$

where E_{proc}, E_{com}, and E_{sen} denote the processing energy per hour, communication energy per hour, and sensing energy per hour, respectively.

The *processing energy* accounts for the processor energy consumed in processing the sensed data. We assume that the sensor node's processor operates in two modes, active mode and idle mode [28]. We point out that although we consider active and idle modes only, a processor operating in other sleep modes (e.g., power-down, power-save, standby, etc.) can also be incorporated in our model. E_{proc} is given by:

$$E_{proc} = E_{proc}^{a} + E_{proc}^{i} \tag{13}$$

where E_{proc}^{a} and E_{proc}^{i} denote the processor's energy consumption per hour in active mode and idle mode, respectively.

The sensor nodes communicate with each other (e.g., send packets containing the sensed data information) to accomplish the assigned application task and this communication process consumes *communication energy*. The communication energy is the sum of the transmission, receive, and idle energies for a sensor node's transceiver, i.e.,:

$$E_{com} = E_{trans}^{tx} + E_{trans}^{rx} + E_{trans}^{i} \tag{14}$$

where E_{trans}^{tx}, E_{trans}^{rx}, and E_{trans}^{i} denote the transceiver's transmission energy per hour, receive energy per hour, and idle energy per hour, respectively. E_{trans}^{tx} is given by:

$$E_{trans}^{tx} = N_{pkt}^{tx} \cdot E_{tx}^{pkt} \tag{15}$$

where N_{pkt}^{tx} denotes the number of packets transmitted per hour and E_{tx}^{pkt} denotes the transmission energy per packet. E_{tx}^{pkt} is given as:

$$E_{tx}^{pkt} = V_t \cdot I_t \cdot t_{tx}^{pkt} \tag{16}$$

where V_t denotes the transceiver voltage, I_t denotes the transceiver current, and t_{tx}^{pkt} denotes the time to transmit one packet. t_{tx}^{pkt} is given by:

$$t_{tx}^{pkt} = P_s \times 8/R_{tx} \tag{17}$$

where P_s denotes the packet size in bytes and R_{tx} denotes the transceiver data rate (in bits/second).

E_{trans}^{rx} can be calculated using a similar procedure as E_{trans}^{tx}. E_{trans}^{i} can be calculated as:

$$E_{trans}^{i} = V_t \cdot I_t^{s} \cdot t_{tx}^{i} \tag{18}$$

where V_t denotes the transceiver voltage, I_t^s denotes the transceiver sleep current, and t_{tx}^i denotes the transceiver idle time per hour.

The energy consumed by the sensors during sensing the observed phenomenon accounts for the *sensing energy*. The sensing energy mainly depends upon the sensing (sampling) frequency and the number of sensors attached to the sensor board (e.g., the MTS400 sensor board [29] has Sensirion SHT1x temperature and humidity sensors [30]). The sensors consume energy while taking sensing measurements and switch to an idle mode for energy conservation while not sensing. E_{sen} is given by:

$$E_{sen} = E_{sen}^{m} + E_{sen}^{i} \tag{19}$$

where E_{sen}^{m} denotes the sensing measurement energy per hour and E_{sen}^{i} denotes the sensing idle energy per hour. E_{sen}^{m} can be calculated as:

$$E_{sen}^{m} = N_s \cdot V_s \cdot I_s^{m} \cdot t_s^{m} \times 3600 \tag{20}$$

where N_s denotes the number of sensing measurements per second, V_s denotes the sensing board voltage, I_s^m denotes the sensing measurement current, and t_s^m denotes the sensing measurement time. E_{sen}^{i} is given by:

$$E_{sen}^{i} = V_s \cdot I_s \cdot t_s^{i} \times 3600 \tag{21}$$

where I_s denotes the sensing sleep current and t_s^i denotes the sensing idle time.

B. *Throughput Estimation*

In the context of dynamic optimizations, *throughput* can be interpreted as the sensor node's sensing, processing, and transmission rate to observe a phenomenon. Three processes contribute to the sensor node's throughput (i.e., sensing, processing, and communication). The throughput interpretation may vary depending upon the WSN application design as sensing, processing, and communication throughputs can have different relative importance for different applications. The aggregate throughput R (typically measured in bits/second) can be considered as a weighted sum of constituent throughputs:

$$R = \omega_s R_{sen} + \omega_p R_{proc} + \omega_c R_{com} \ : \ \omega_s + \omega_p + \omega_c = 1 \tag{22}$$

where R_{sen}, R_{proc}, and R_{com} denote the sensing, processing, and communication throughputs, respectively. ω_s, ω_p, and ω_c denote the weight factors for the sensing, processing, and communication throughputs, respectively.

The sensing throughput is the throughput due to sensing activity and measures the sensing bits sampled per second. R_{sen} is given by:

$$R_{sen} = F_s \cdot R^b_{sen} \tag{23}$$

where F_s and R^b_{sen} denote the sensing frequency and sensing resolution bits, respectively.

The processing throughput is the throughput due to the processor's processing of sensed measurements and measures the bits processed per second. R_{proc} is given by:

$$R_{proc} = F_p / N^b \tag{24}$$

where F_p and N^b denote the processor frequency and the number of processor instructions to process one bit, respectively.

The communication throughput R_{com} results from the transfer of data packets over the wireless channel and is given by:

$$R_{com} = P^{eff}_s \times 8 / t^{pkt}_{tx} \tag{25}$$

where t^{pkt}_{tx} denotes the time to transmit one packet and P^{eff}_s denotes the effective packet size excluding the packet header overhead.

C. Reliability Estimation

The reliability metric measures the number of packets transferred reliably (i.e., error free packet transmission) over the wireless channel. Accurate reliability estimation is challenging because of dynamic changes in the involved factors, such as network topology, number of neighboring sensor nodes, wireless channel fading, sensor network traffic, etc. The two main factors that affect reliability are the transceiver transmission power P_{tx} and receiver sensitivity. For example, the AT86RF230 transceiver [31] has a receiver sensitivity of -101 dBm with a corresponding packet error rate (PER) $\leq$ 1% for an additive white Gaussian noise (AWGN) channel with a physical service data unit (PSDU) equal to 20 bytes. Reliability can be estimated using Friis free space transmission equation [32] for different P_{tx} values, distance between transmitting and receiving sensor nodes, and fading models (e.g., shadowing fading model). Reliability values can be assigned corresponding to P_{tx} values such that the higher P_{tx} values give higher reliability, however, more accurate reliability estimation requires profiling statistics for the number of packets transmitted and the number of packets received.

V. Experimental Results

In this section, we describe our experimental setup and results for our one-shot dynamic optimization methodology.

Table II
Crossbow IRIS mote platform hardware specifications.

Notation	Description	Value
V_b	Battery voltage	3.6 V
C_b	Battery capacity	2000 mA-h
N_b	Processing instructions per bit	5
R^b_{sen}	Sensing resolution bits	24
V_t	Transceiver voltage	3 V
R_{tx}	Transceiver data rate	250 kbps
I^{rx}_t	Transceiver receive current	15.5 mA
I^s_t	Transceiver sleep current	20 nA
V_s	Sensing board voltage	3 V
I^m_s	Sensing measurement current	550 μA
t^m_s	Sensing measurement time	55 ms
I_s	Sensing sleep current	0.3 μA

A. Experimental Setup

We base our experimental setup on the Crossbow IRIS mote platform [33], which has a battery capacity of 2000 mA-h with two AA alkaline batteries. The IRIS mote platform integrates an Atmel ATmega1281 microcontroller [28], an Atmel AT-86RF230 low power 2.4 GHz transceiver [31], an MTS400 sensor board [29] with Sensirion SHT1x temperature and humidity sensors [30]. Table II shows the sensor node hardware specific values, corresponding to the IRIS mote platform, which are used by the application metrics estimation model [28][30][31][33].

In our experimental setup, we consider a WSN topology where each sensor node has two neighbors, although our topology can be extended for any number of neighboring sensor nodes. The number of neighboring sensor nodes in a topology determines the number of packets received by a sensor node, which affects the expended communication energy. This expended communication energy affects the lifetime of the sensor nodes in the WSN. Our work assumes that the medium access control (MAC) layer handles collisions and packet loss. The packet loss due to any reason (e.g., low transmission power, collision, etc.) is taken into account at a high level by our reliability application metric. The accurate determination of the packet loss requires gathering of profiling statistics, which is the focus of our future work.

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 evaluate 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) [28], F_s = {1, 2, 3} (samples per second) [30], P_s = {41, 56, 64} (bytes), P_{ti} = {60, 300, 600} (seconds), and P_{tx} = {-17, -3, 1} (dBm) [31]. 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) [28], F_s = {0.2, 0.5, 1, 2, 3, 4} (samples per second) [30], 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) [31]. 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). Our consideration of two different design space cardinalities ($|S| = 729$ and $|S| = 31,104$) is important because this consideration helps in investigating the impact of the design space cardinality on dynamic optimization methodologies.

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 the application metrics (Section III-C). We specify the objective function parameters as a multiple of base units for lifetime, throughput, and reliability, however, our application metrics estimation model and one-shot dynamic optimization methodology works equally well for any set of application requirements, weight factors, and assumption of base units. We assume that one lifetime unit is 5 days, one throughput unit is 20 kbps, and one reliability unit is 0.05 (reliability measures error-free packet transmissions on a scale from 0 to 1). Table III depicts the application requirements for the application domains in terms of objective function parameter values and Table IV depicts the associated weight factors used in our experiments. Weight factors for a given application domain depend upon specific application requirements. For example, a security/defense application that requires prolonged operation requires a higher weight factor for the lifetime application metric as compared to the other application metrics, whereas a different security/defense application that requires gathering high resolution images requires a higher weight factor for the throughput application metric as compared to the other application metrics.

Since the objective function values corresponding to different states depends upon the estimation of high-level metrics, we present an example throughput calculation to explain this estimation process using our application metrics estimation model (Section IV) and the IRIS mote platform hardware specifications (Table II). We consider a state $s_y = (V_{p_y}, F_{p_y}, F_{s_y}, P_{s_y}, P_{ti_y}, P_{tx_y}) = (2.7, 4, 1, 41, 60, -17)$ for our example. (17) gives $t_{tx}^{pkt} = 41 \times 8/(250 \times 10^3) =$ 1.312 ms. (23), (24), and (25) give $R_{sen} = 1 \times 24 = 24$ bps, $R_{proc} = 4 \times 10^6/5 = 800$ kbps, and $R_{com} = 21 \times 8/(1.312 \times 10^{-3}) = 128.049$ kbps, respectively ($P_s^{eff} = 41 - 21 = 20$ where we assume $P_h = 21$ bytes). (22) gives $R = (0.4)(24) + (0.4)(800 \times 10^3) + (0.2)(128.049 \times 10^3) =$ 345.62 kbps where we assume ω_s, ω_p, and ω_c equal to 0.4, 0.4, and 0.2, respectively.

In order to evaluate our one-shot dynamic optimization solution quality, we compare the solution from the one-shot initial parameter settings $\widehat{\boldsymbol{P_0}}$ with the solutions obtained from the following four potential initial parameter value settings (although any feasible n-tuple $s \in S$ can be taken as the initial parameter settings):

- $\mathcal{I}_1$ assigns the first parameter value for each tunable parameter, i.e., $\mathcal{I}_1 = p_{i_1}, \ \forall \, i \in \{1, 2, \ldots, N\}$.
- $\mathcal{I}_2$ assigns the last parameter value for each tunable parameter, i.e., $\mathcal{I}_2 = p_{i_n}, \ \forall \, i \in \{1, 2, \ldots, N\}$.
- $\mathcal{I}_3$ assigns the middle parameter value for each tunable parameter, i.e., $\mathcal{I}_3 = \lfloor p_{i_n}/2 \rfloor, \ \forall \, i \in \{1, 2, \ldots, N\}$.
- $\mathcal{I}_4$ assigns a random value for each tunable parameter, i.e., $\mathcal{I}_4 = p_{i_q} : q = \texttt{rand()} \% n, \ \forall \, i \in \{1, 2, \ldots, N\}$ where `rand()` denotes a function to generate a random/pseudo-random integer and % denotes the modulus operator.

Although we analyzed our methodology for the IRIS motes platform, three application domains, two design spaces, and four potential initial parameter value settings, our one-shot dynamic optimization methodology and application metrics estimation model are equally applicable to any platform, application domain, and design space.

Table III
DESIRABLE MINIMUM L, DESIRABLE MAXIMUM U, ACCEPTABLE MINIMUM α, AND ACCEPTABLE MAXIMUM β OBJECTIVE FUNCTION PARAMETER VALUES FOR A SECURITY/DEFENSE (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	**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

Table IV
WEIGHT FACTORS FOR DIFFERENT APPLICATION DOMAINS FOR $|S| = 729$ AND $|S| = 31,104$.

–	$\boldsymbol{\|S\| = 729}$ **&** $\boldsymbol{\|S\| = 31,104}$		
Application Domain	$\boldsymbol{\omega_l}$	$\boldsymbol{\omega_t}$	$\boldsymbol{\omega_r}$
Security/Defense System	0.25	0.35	0.4
Health Care	0.25	0.35	0.4
Ambient Conditions Monitoring	0.4	0.5	0.1

B. Results

We implemented our one-shot dynamic optimization methodology in C++. To evaluate the effectiveness of our one-shot solution, we compare the one-shot solution's results with four alternative initial parameter arrangements (Section V-A). We normalize the objective function values corresponding to the operating states attained by our dynamic optimization methodology with respect to the optimal solution obtained using an exhaustive search. We compare the relative complexity of our one-shot dynamic optimization methodology with two other dynamic optimization methodologies, which leverage greedy- and SA-based algorithms for design space exploration [34]. Although for brevity we present results for only a subset of the initial parameter value settings, application domains, and design spaces, we observed that results for extensive application domains, design spaces, and initial parameter settings revealed similar trends.

1) Percentage Improvements over other Initial Parameter Settings: Table V depicts the percentage improvements attained by the one-shot parameter settings $\widehat{\boldsymbol{P_0}}$ over other parameter settings for different application domains and weight factors (Table IV). We point out that different weight factors could result in different percentage improvements, however, we observed similar trends for other weight factors. Table V shows that the one-shot initial parameter settings can result in as high as a 155% improvement as compared to other initial value settings. We observe that some arbitrary settings may give a comparable or even a better solution for a particular application domain, application metric weight factors, and design space cardinality, but that arbitrary setting would not scale to other application domains, application metric weight factors, and design space cardinalities. For example, $\mathcal{I}_3$ obtains a 12% better quality solution than $\widehat{\boldsymbol{P_0}}$ for the ambient conditions monitoring application for $|S| = 31,104$, but yields a 10% lower quality solution for the security/defense and health care applications for $|S| = 31,104$, and a 57%, 31%, and 20% lower quality solution than $\widehat{\boldsymbol{P_0}}$ for the security/defense, health care, and ambient conditions monitoring applications, respectively, for $|S| = 729$. The percentage improvement attained by $\widehat{\boldsymbol{P_0}}$ over all application domains and design spaces is 33% on average. Our one-shot methodology is the first approach (to the best of our knowledge) to leverage intelligent initial tunable parameter value settings for sensor nodes to provide a good quality operating state, as arbitrary initial parameter value settings typically result in a poor operating state. Results reveal that on average $\widehat{\boldsymbol{P_0}}$ gives a solution within 8% of the optimal solution.

2) Comparison with Greedy- and SA-based Dynamic Optimization Methodologies: In order to investigate the effectiveness of our one-shot methodology, we compare the one-shot solution's quality (indicated by the attained objective function value) with two other dynamic optimization methodologies, which leverage an SA-based and a greedy-based (denoted by GD$^{\text{asc}}$ where asc stands for ascending order of parameter exploration) exploration of the design space. We assign initial parameter value settings for the greedy- and SA-based methodologies as $\mathcal{I}_1$ and $\mathcal{I}_4$, respectively. Note that, for brevity, we present results for $\mathcal{I}_1$ and $\mathcal{I}_4$, however, other initial parameter settings such as $\mathcal{I}_2$ and $\mathcal{I}_3$ would yield similar trends when combined with greedy-based and SA-based design space exploration.

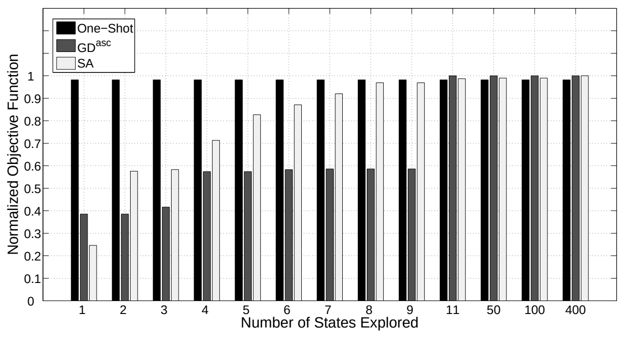

Figure 3. Objective function value normalized to the optimal solution for a varying number of states explored for the one-shot, greedy, and SA algorithms for a security/defense system where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, $|S| = 729$.

Fig. 3 shows the objective function value normalized to the optimal solution versus the number of states explored for the one-shot, GD$^{\text{asc}}$, and SA algorithms for a security/defense system for $|S| = 729$. The one-shot solution is within 1.8% of the optimal solution. The figure shows that GD$^{\text{asc}}$ and SA explore 11 states (1.51% of the design space) and 10 states (1.37% of the design space), respectively, to attain an equivalent or better quality solution than the one-shot solution. Although, greedy- and SA-based methodologies explore few states to reach a comparable solution as that of our one-shot methodology, the one-shot methodology is suitable when design space exploration is not an option due to an extremely large design space and/or extremely stringent computational, memory, and timing constraints. These results indicate that other arbitrary initial value settings (e.g., $\mathcal{I}_1$, $\mathcal{I}_4$, etc.) do not provide a good quality operating state and necessitate design space exploration by online algorithms (e.g., greedy) to provide a good quality operating state. We point out that if the greedy- and SA-based methodologies leverage our one-shot initial tunable parameter value settings $\mathcal{I}$, further improvements over the one-shot solution can produce a very good quality (optimal or near-optimal) operating state [34].

Fig. 4 shows the objective function value normalized to the optimal solution versus the number of states explored for a security/defense system for $|S| = 31,104$. The one-shot solution is within 8.6% of the optimal solution. The figure

Table V
PERCENTAGE IMPROVEMENTS ATTAINED BY $\widehat{\boldsymbol{P_0}}$ OVER OTHER INITIAL PARAMETER SETTINGS FOR $|S| = 729$ AND $|S| = 31,104$.

–	$\boldsymbol{\lvert S\rvert = 729}$				$\boldsymbol{\lvert S\rvert = 31,104}$			
Application Domain	$\mathcal{I}_1$	$\mathcal{I}_2$	$\mathcal{I}_3$	$\mathcal{I}_4$	$\mathcal{I}_1$	$\mathcal{I}_2$	$\mathcal{I}_3$	$\mathcal{I}_4$
Security/Defense System	155%	10%	57%	29%	148%	0.3%	10%	92%
Health Care	78%	7%	31%	11%	73%	0.3%	10%	45%
Ambient Conditions Monitoring	52%	6%	20%	7%	15%	-7%	-12%	18%

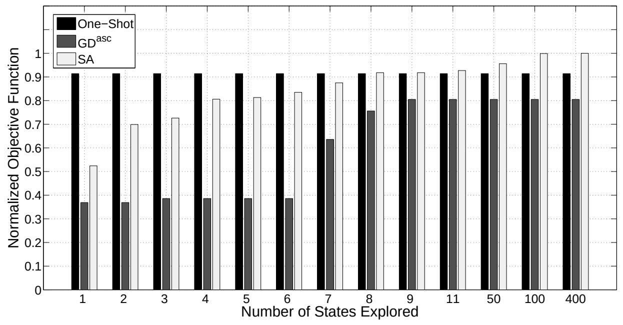

Figure 4. Objective function value normalized to the optimal solution for a varying number of states explored for the one-shot, greedy, and SA algorithms for a security/defense system where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, $|S| = 31,104$.

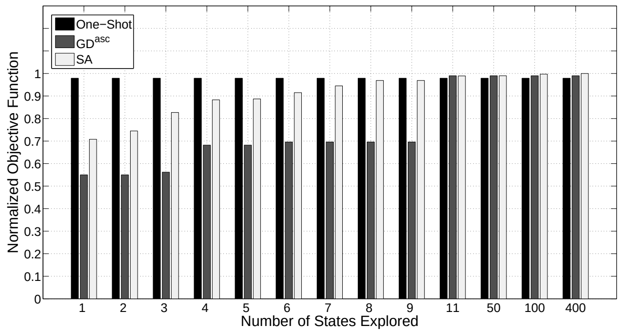

Figure 5. Objective function value normalized to the optimal solution for a varying number of states explored for the one-shot, greedy, and SA algorithms for a health care application where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, $|S| = 729$.

shows that GD^{asc} converges to a lower quality solution than the one-shot solution after exploring 9 states (0.029% of the design space) and SA explores 8 states (0.026% of the design space) to yield a better quality solution than the one-shot solution. These results reveal that the greedy exploration of parameters may not necessarily attain a better quality solution than our one-shot solution.

Fig. 5 shows the objective function value normalized to the optimal solution versus the number of states explored for a health care application for $|S| = 729$. The one-shot solution is within 2.1% of the optimal solution. The figure shows that GD^{asc} converges to an almost equal quality solution as compared to the one-shot solution after exploring 11 states (1.5% of the design space) and SA explores 10 states (1.4% of the design space) to yield an almost equal quality solution as compared to the one-shot solution. These results indicate that further exploration of the design space is required to find an equivalent quality solution as compared to one-shot if the intelligent initial value settings leveraged by one-shot are not used.

Fig. 6 shows the objective function value normalized to the optimal solution versus the number of states explored for a health care application for $|S| = 31,104$. The one-shot solution is within 1.6% of the optimal solution. The figure shows that GD^{asc} converges to a lower quality solution than the one-shot solution after exploring 9 states (0.029% of the design space) and SA explores 6 states (0.019% of the design space) to yield a better quality solution than the one-shot solution. These results confirm that the greedy exploration of the parameters may not necessarily attain a better quality solution than our one-shot solution.

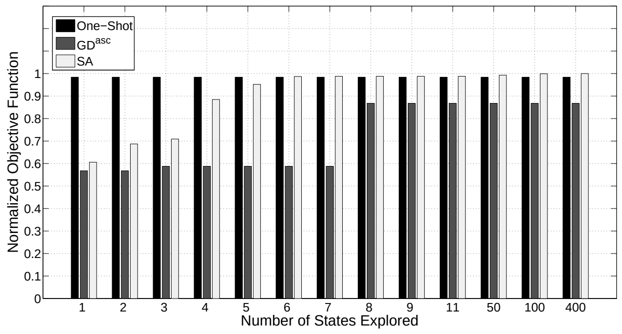

Figure 6. Objective function value normalized to the optimal solution for a varying number of states explored for the one-shot, greedy, and SA algorithms for a health care application where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, $|S| = 31,104$.

Fig. 7 shows the objective function value normalized to the optimal solution versus the number of states explored for an ambient conditions monitoring application for $|S| = 729$. The one-shot solution is within 7.7% of the optimal solution. The figure shows that GD^{asc} and SA converge to an equivalent or better quality solution than the one-shot solution after exploring 4 states (0.549% of the design space)

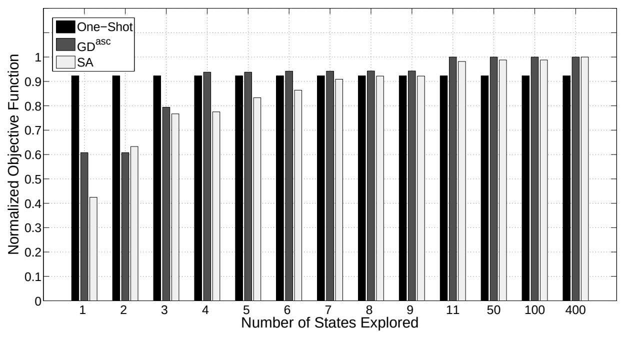

Figure 7. Objective function value normalized to the optimal solution for a varying number of states explored for the one-shot, greedy, and SA algorithms for an ambient conditions monitoring application where $\omega_l = 0.4$, $\omega_t = 0.5$, $\omega_r = 0.1$, $|S| = 729$.

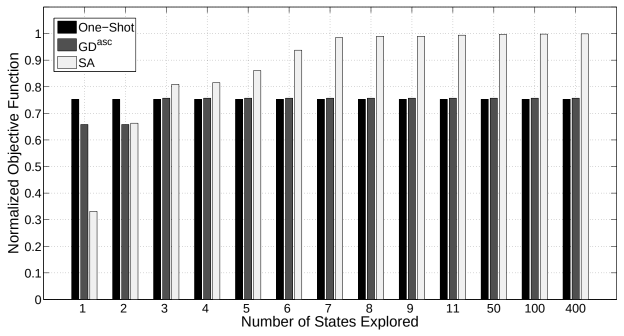

Figure 8. Objective function value normalized to the optimal solution for a varying number of states explored for the one-shot, greedy, and SA algorithms for an ambient conditions monitoring application where $\omega_l = 0.4$, $\omega_t = 0.5$, $\omega_r = 0.1$, $|S| = 31,104$.

and 10 states (1.37% of the design space), respectively. These results again confirm that the greedy- and SA-based explorations can provide improved results over the one-shot solution, but require additional state exploration.

Fig. 8 shows the objective function value normalized to the optimal solution versus the number of states explored for an ambient conditions monitoring application for $|S| = 31,104$. The one-shot solution is within 24.7% of the optimal solution. The figure shows that both GD^{asc} and SA converge to an equivalent or better quality solution than the one-shot solution after exploring 3 states (0.01% of the design space). These results indicate that both greedy- and SA-based methods can give good quality solutions after exploring a very small percentage of the design space and both greedy- and SA-based methods enable lightweight dynamic optimizations [34]. The results also indicate that the one-shot solution provides a good quality solution when further design space exploration is not possible due to resource constraints.

3) Computational Complexity: To verify that our one-shot dynamic optimization methodology (Section III) is lightweight, we compared the data memory requirements and execution time of our one-shot dynamic optimization methodology with the greedy- and SA-based dynamic optimization methodologies.

The data memory analysis revealed that our one-shot methodology requires only 150, 188, 248, and 416 bytes for (number of tunable parameters N, number of application metrics m) equal to (3, 2), (3, 3), (6, 3), and (6, 6), respectively. The greedy-based methodology requires 458, 528, 574, 870, and 886 bytes, whereas the SA-based methodology requires 514, 582, 624, 920, and 936 bytes of storage for design space cardinalities of 8, 81, 729, 31,104, and 46,656, respectively. The data memory analysis shows that the SA-based methodology has comparatively larger memory requirements than the greedy-based methodology. Our analysis reveals that the data memory requirements for our one-shot methodology increases linearly as the number of tunable parameters and the number of application metrics increases. The data memory requirements for the greedy- and SA-based methodologies increase linearly as the number of tunable parameters and tunable values (and thus the design space) increases. The data memory analysis verifies that although the one-shot, greedy- and SA-based methodologies have low data memory requirements (on the order of hundreds of bytes), the one-shot solution requires 204% and 458% less memory on average as compared to the greedy- and SA-based methodologies, respectively.

We measured the execution time for our one-shot and the greedy- and SA-based methodologies 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 [35] using the Linux/Unix `time` command [36]. We scaled the execution time results to the Atmel ATmega1281 microcontroller [28] running at 8 MHz. Although microcontrollers have different instruction set architectures and scaling does not provide 100% accuracy for the microcontroller runtime, scaling enables relative comparisons and provides reasonable runtime estimates. Results showed that one-shot required 1.66 ms both for $|S| = 729$ and $|S| = 31,104$. GD^{asc} 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^{asc}) for $|S| = 729$ and $|S| = 31,104$, respectively. The execution time analysis revealed that our dynamic optimization methodologies required execution times on the order of milliseconds, and the one-shot solution required 18% less execution time on average as compared to greedy- and SA-based methodologies. The one-shot solution required 66% and 73% less execution time for the SA-based methodology when $|S| = 729$ and $|S| = 31,104$. These results indicate that the design space cardinality affects the execution time linearly for greedy- and SA-based methodologies whereas the one-shot solution's execution time is affected negligibly by the design space cardinality and hence our one-shot methodology's advantage increases as the design space cardinality increases. We verified our execution time analysis using the `clock()` function [37], which revealed similar trends.

VI. Conclusions and Future Work

In this paper, we proposed a lightweight dynamic optimization methodology for WSNs, which provided a high-quality solution in just one-shot using intelligent initial tunable parameter value settings for highly constrained applications. To assist dynamic optimization methodologies for operating states' comparisons, we proposed an application metric estimation model to estimate high-level metrics (lifetime, throughput, and reliability) from sensor node's parameters. This estimation model was leveraged by our one-shot dynamic optimization methodology and provided a prototype model for application metric estimation. To evaluate the effectiveness of the initial parameter settings by our one-shot methodology, we compared the one-shot solution quality with four other typical initial parameter settings. Results revealed that the percentage improvement attained by our one-shot solution over other initial parameter settings for different application domains and design spaces was 33% on average and as high as 155%. Results indicated that our one-shot solution was within 8% of the optimal solution obtained from exhaustive search. We compared the computational complexity of our one-shot dynamic optimization methodology with two other dynamic optimization methodologies that leveraged greedy- and simulated annealing (SA)-based exploration of the design space. Results showed that the one-shot solution required 204% and 458% less memory on average as compared to the greedy- and SA-based methodologies, respectively. The one-shot solution required 18% less execution time on average as compared to the greedy- and SA-based methodologies even if these methodologies were restricted to explore only 0.03% of the design space on average. The execution time and data memory analysis confirmed that our one-shot methodology is lightweight and suitable for time-critical or highly constrained applications.

Future work includes the incorporation of profiling statistics into our one-shot dynamic optimization methodology to provide feedback with respect to changing environmental stimuli. We plan to further verify our one-shot dynamic optimization methodology using implementation on a hardware sensor node platform. We also plan to further investigate online optimization algorithms leveraging our one-shot initial value settings for further higher quality solutions for comparatively less constrained applications.

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] A. Munir, A. Gordon-Ross, S. Lysecky, and R. Lysecky, "A One-Shot Dynamic Optimization Methodology for Wireless Sensor Networks," in *Proc. IARIA IEEE International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, Florence, Italy, October 2010.

[2] 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.

[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. 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.

[5] 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.

[6] 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.

[7] (2012, January) Dynamic Profiling and Optimization (DPOP) for Sensor Networks. [Online]. Available: http://www.ece.arizona.edu/~dpop/

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

[9] 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.

[10] 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.

[11] S. Sridharan and S. Lysecky, "A First Step Towards Dynamic Profiling of Sensor-Based Systems," in *Proc. IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'08)*, San Francisco, California, June 2008, pp. 600–602.

[12] 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.

[13] A. Munir and A. Gordon-Ross, "An MDP-based Application Oriented Optimal Policy for Wireless Sensor Networks," in *Proc. ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'09)*, Grenoble, France, October 2009, pp. 183–192.

[14] S. Kogekar, S. Neema, B. Eames, X. Koutsoukos, A. Ledeczi, and M. Maroti, “Constraint-Guided Dynamic Reconfiguration in Sensor Networks,” in *Proc. ACM International Symposium on Information Processing in Sensor Networks (IPSN’04)*, Berkeley, California, April 2004, pp. 379–387.

[15] X. Wang, J. Ma, S. Wang, and D. Bi, “Distributed Energy Optimization for Target Tracking in Wireless Sensor Networks,” *IEEE Trans. on Mobile Computing*, vol. 9, no. 1, pp. 73–86, January 2009.

[16] L. Liu, X. Zhang, and H. Ma, “Dynamic Node Collaboration for Mobile Target Tracking in Wireless Camera Sensor Networks,” in *Proc. IEEE (INFOCOM’09)*, Rio de Janeiro, Brazil, April 2009, pp. 1188–1196.

[17] R. Khanna, H. Liu, and H.-H. Chen, “Dynamic Optimization of Secure Mobile Sensor Networks: A Genetic Algorithm,” in *Proc. IEEE International Conference on Communications (ICC’07)*, Glasgow, Scotland, June 2007, pp. 3413–3418.

[18] R. Min, T. Furrer, and A. Chandrakasan, “Dynamic Voltage Scaling Techniques for Distributed Microsensor Networks,” in *Proc. IEEE Workshop on VLSI (WVLSI’00)*, Orlando, Florida, April 2000, pp. 43–46.

[19] L. Yuan and G. Qu, “Design Space Exploration for Energy-Efficient Secure Sensor Network,” in *Proc. IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP’02)*, San Jose, California, July 2002, pp. 88–97.

[20] S. Lysecky and F. Vahid, “Automated Application-Specific Tuning of Parameterized Sensor-Based Embedded System Building Blocks,” in *Proc. of the International Conference on Ubiquitous Computing (UbiComp)*, Orange County, California, September 2006, pp. 507–524.

[21] R. Verma, “Automated application specific sensor network node tuning for non-expert application developers,” Master’s thesis, Department of Electrical and Computer Engineering, University of Arizona, 2008.

[22] R. Mannion, H. Hsieh, S. Cotterell, and F. Vahid, “System Synthesis for Networks of Programmable Blocks,” in *Proc. IEEE Conference on Design, Automation and Test in Europe (DATE)*, Munich, Germany, March 2005, pp. 888–893.

[23] A. Meier, M. Weise, J. Beutel, and L. Thiele, “NoSE: Efficient Initialization of Wireless Sensor Networks,” in *Proc. ACM Conference on Embedded Networked Sensor Systems (SenSys’08)*, Raleigh, North Carolina, November 2008, pp. 397–398.

[24] D. Ma, J. Wang, M. Somasundaram, and Z. Hu, “Design and Optimization on Dynamic Power System for Self-Powered Integrated Wireless Sensing Nodes,” in *Proc. IEEE International Symposium on Low Power Electronics and Design (ISLPED’05)*, San Diego, California, August 2005, pp. 303–306.

[25] R. Jurdak, P. Baldi, and C. Lopes, “Adaptive Low Power Listening for Wireless Sensor Networks,” *IEEE Trans. on Mobile Computing*, vol. 6, no. 8, pp. 988–1004, August 2007.

[26] M. Hasegawa, T. Kawamura, N. Tran, G. Miyamoto, Y. Murata, H. Harada, and S. Kato, “Decentralized Optimization of Wireless Sensor Network Lifetime based on Neural Network Dynamics,” in *Proc. IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC’08)*, Cannes, France, September 2008, pp. 1–5.

[27] X. Ning and C. Cassandras, “Optimal Dynamic Sleep Time Control in Wireless Sensor Networks,” in *Proc. IEEE Conference on Decision and Control (CDC’08)*, Cancun, Mexico, December 2008, pp. 2332–2337.

[28] Atmel. (2012, January) Atmel atmega1281 microcontroller with 256k bytes in-system programmable flash. [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/2549S.pdf

[29] Crossbow. (2012, January) MTS/MDA sensor board users manual. [Online]. Available: http://www.xbow.com/

[30] Sensirion. (2012, January) Datasheet sht1x (sht10, sht11, sht15) humidity and temperature sensor. [Online]. Available: http://www.sensirion.com/

[31] Atmel. (2012, January) 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

[32] H. Friis, “A Note on a Simple Transmission Formula,” *Proc. IRE*, vol. 34, p. 254, 1946.

[33] Crossbow. (2012, January) Crossbow iris datasheet. [Online]. Available: http://www.xbow.com/

[34] A. Munir, A. Gordon-Ross, S. Lysecky, and R. Lysecky, “A Lightweight Dynamic Optimization Methodology for Wireless Sensor Networks,” in *Proc. IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Niagara Falls, Canada, October 2010, pp. 129–136.

[35] (2012, January) Intel Xeon Processor E5430. [Online]. Available: http://processorfinder.intel.com/details.aspx?sSpec=SLANU

[36] (2012, January) Linux Man Pages. [Online]. Available: http://linux.die.net/man/

[37] (2012, January) C++ reference library. [Online]. Available: http://cplusplus.com/reference/clibrary/ctime/clock/