# A One-Shot 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*—Wireless sensor networks (WSNs), consisting of autonomous sensor nodes, have emerged as ubiquitous networks which span diverse application domains (e.g., health care, logistics, defense) each with varying application requirements (e.g., lifetime, throughput). Sensor nodes possess tunable parameters (e.g., processor voltage, 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 operating state. However, rapidly changing application behavior and environmental stimuli necessitate a lightweight and highly responsive dynamic optimization methodology. In this paper, we propose One-Shot – a 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. Results reveal that One-Shot solution is within 5.92% 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) and sensor-based platform parameters.**

***Keywords*-Wireless sensor networks; dynamic optimization; application metric 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 due to their proliferation in diverse application domains (e.g., defense, health care, logistics) each with varying application requirements. For example, a security/defense system may have a high throughput requirement whereas an ambient conditions monitoring application may be more sensitive to lifetime. This diversity makes WSN design challenging using commercial-off-the-shelf (COTS) sensor nodes.

COTS sensor nodes are mass-produced to optimize for 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 may be *tuned* for a specific application. Faced with an overwhelming number of tunable parameter choices, WSN design may 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 a sensor node's lifetime. Accurate prediction/simulation of environmental stimuli is challenging and applications with changing environmental stimuli do not benefit from static optimizations. *Dynamic optimizations* assign parameter values during runtime and reassign/change these values in accordance with changing environmental stimuli, thus enabling closer adherence to application requirements.

There exists much research in the area of dynamic optimizations (e.g., [1][2][3][4]), but most previous work targets the memory (cache) or processor in computer systems. Little work exists on WSN dynamic optimization, which presents additional challenges due to a WSN's unique design space, energy constraints, and operating environment. Shenoy et al. [5] 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. [6] proposed a Markov Decision Process (MDP)-based methodology as a first step towards WSN dynamic optimization, but this methodology required excessive computational resources for larger design spaces. Wang et al. [7] proposed an energy efficient optimization method for target tracking applications that consisted of dynamic awakening and an optimal sensing scheme. Khanna et al. [8] proposed a genetic algorithm for secure and dynamic deployment of resource-constrained multi-hop WSNs. Some previous works [9][10] explored WSN dynamic voltage and frequency scaling (DVFS) for dynamic optimization, but DVFS only considered two sensor node tunable parameters (processor voltage and frequency).

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. [11] 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.

We propose One-Shot – 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 the One-Shot operating state is within 5.92% of the optimal solution (obtained from exhaustive search) averaged over several different application domains and design spaces. 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 metrics (lifetime, throughput, and reliability) from sensor-based platform parameters (e.g., processor voltage and frequency, sensing frequency, transceiver transmission power, etc.). The dynamic optimization methodology leverages this estimation model while comparing different operating states for optimization purposes.

## II. Dynamic Optimization Methodology

In this section, we give an overview of One-Shot and associated algorithm. We also formulate the state space and objective function for One-Shot.

### 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 weightage/importance of each application metric with respect to each other. One-Shot leverages an application metric estimation model to determine application metric values offered by an operating state.

Fig. 1 shows the per-node One-Shot process (encompassed by the dashed circle), which is orchestrated by the *dynamic optimization controller*. The dynamic optimization controller invokes *One-Shot* wherein the sensor node operating state is directly determined using an intelligent tunable parameter value setting selection methodology (i.e., in *One-Shot*). One-Shot also determines an exploration order (ascending or descending) for the 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

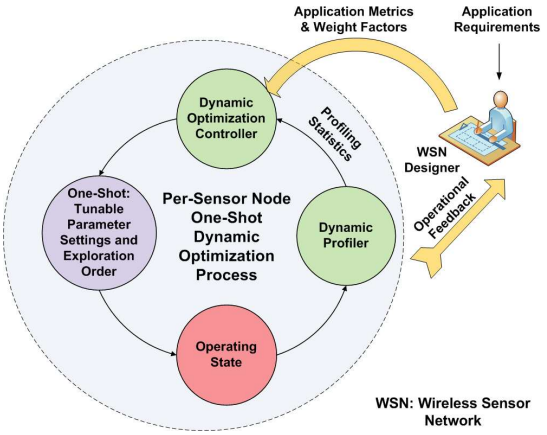


Figure 1. One-Shot dynamic optimization methodology for wireless sensor networks.

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 One-Shot. A *dynamic profiler* records profiling statistics (e.g., remaining 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 One-Shot to determine the new operating state. This feedback process continues to ensure the selection of an appropriate operating state to better meet application requirements.

### B. State Space

The state space $S$ for One-Shot 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 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:

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

where $f(s)$ denotes the objective function characterizing application metrics and weight factors. $f_k(s)$ and $\omega_k$ in (4) denote the objective function and weight factor for the $k^{\text{th}}$ 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 $f^{opt}(s)$ where $f^{opt}(s)$ indicates the best possible adherence to the application requirements given the design space $S$.

For our dynamic optimization methodology, we consider three application metrics ($m$ = 3), which are lifetime, throughput, and reliability, each with piecewise linear objective functions. A piecewise linear objective function captures the desirable and acceptable ranges of a particular application metric. For example, for a particular application, a lifetime metric may have an acceptable minimum value of 40 days and reliability may be a more important metric than the lifetime. The objective function delineates this inter-metric relative importance and attainable application metric values. Even though we consider piecewise linear objective functions, our methodology works well for any other objective function characterization (e.g., linear, non-linear).

### D. One-Shot Dynamic Optimization Algorithm

In this subsection, we describe associated algorithm for One-Shot. The algorithm determines initial tunable parameter value settings and exploration order (ascending or descending).

Algorithm 1 describes One-Shot's 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 $n$, 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 $\boldsymbol{P_0^k}$ and $\boldsymbol{P_d^k}$ (where $d$ denotes the exploration direction (ascending or descending)), which store the initial value settings and exploration order, respectively, for the

**Input**: *f(s), N, n, m,* $\boldsymbol{P}$
**Output**: Initial tunable parameter value settings and exploration order

```
1  for k ← 1 to m do
2      for P_i ← 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}, ∀ i ≠ 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}, ∀ i ≠ j} ;
5          δf^k_{P_i} ← f^k_{p_{i_n}} − f^k_{p_{i_1}} ;
6          if δf^k_{P_i} ≥ 0 then
7              explore P_i in descending order ;
8              P^k_d[i] ← descending ;
9              P^k_0[i] ← p^k_{i_n} ;
10         else
11             explore P_i in ascending order ;
12             P^k_d[i] ← ascending ;
13             P^k_0[i] ← p^k_{i_1} ;
14         end
15     end
16 end
   return P^k_d, P^k_0, ∀ k ∈ {1, ..., m}
```

**Algorithm 1**: One-shot dynamic optimization algorithm.

tunable parameters. The algorithm determines $f^k_{p_{i_1}}$ and $f^k_{p_{i_n}}$ (the $k^{\text{th}}$ 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 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 $f^k_{p_{i_1}}$. $\delta f^k_{P_i} \geq 0$ means that $p_{i_n}$ results in a greater (or equal when $\delta f^k_{P_i} = 0$) 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 $k^{\text{th}}$ 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).

## III. Application Metric Estimation Model

In this section, we propose an application metric estimation model leveraged by One-Shot. This estimation model estimates high-level application metrics (lifetime, throughput, reliability) from sensor node parameters (e.g., processor voltage and frequency, transceiver voltage, etc.). For brevity, we describe only the estimation model's key elements.

### A. Lifetime Estimation

*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. The sensor failure due to battery energy depletion is normally taken into account for lifetime estimation. The 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.

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

$$\mathcal{L}_s = \frac{E_b}{E_c \times 24} \tag{5}$$

where $E_b$ denotes the sensor node's battery energy (Joules) and $E_c$ denotes the sensor node's energy consumption per hour.

We model $E_c$ as the sum of processing energy, communication energy, and sensing energy:

$$E_c = E_{proc} + E_{com} + E_{sen} \ \ (J) \tag{6}$$

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

The *processing energy* accounts for the energy consumed in processing the sensed data by the sensor node's processor. We assume that the sensor node's processor operates in two modes: active mode and idle mode [12]. We point out that although we only consider active and idle modes, a processor operating in other sleep modes apart from idle mode (e.g., power-down, power-save, standby, etc.) can also be incorporated in our model. $E_{proc}$ is given by:

$$E_{proc} = E^a_{proc} + E^i_{proc} \tag{7}$$

where $E^a_{proc}$ and $E^i_{proc}$ 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), which consumes *communication energy*. The communication energy is the sum of transmission, receive, and idle energy for a sensor node's transceiver:

$$E_{com} = E^{tx}_{trans} + E^{rx}_{trans} + E^i_{trans} \tag{8}$$

where $E^{tx}_{trans}$, $E^{rx}_{trans}$, and $E^i_{trans}$ denote the transceiver's transmission energy per hour, receive energy per hour, and idle energy per hour, respectively.

The energy consumption due to sensing the observed phenomenon is termed as *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 [13] has Sensirion SHT1x temperature and humidity sensors [14]). The sensors consume energy while taking sensing measurements and switch to the idle mode for energy conservation while not sensing. $E_{sen}$ is given by:

$$E_{sen} = E^m_{sen} + E^i_{sen} \tag{9}$$

where $E^m_{sen}$ denotes the sensing measurement energy per hour and $E^i_{sen}$ denotes the sensing idle energy per hour.

### B. Throughput Estimation

In the context of dynamic optimizations, *throughput* can be interpreted relative to the state (tunable parameter value settings) that deliver the maximum quality (rate) sensing process, processing, and transmission to observe a phenomenon while minimizing the cost (energy consumption). Three processes contribute to the throughput for sensor nodes: sensing, processing, and communication. The throughput interpretation may vary depending upon the WSN application design as these 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:

$$R = \omega_s R_{sen} + \omega_p R_{proc} + \omega_c R_{com} \ : \ \omega_s + \omega_p + \omega_c = 1 \tag{10}$$

where $R_{sen}$, $R_{proc}$, and $R_{com}$ denote the sensing throughput, processing throughput, and communication throughput, respectively. $\omega_s$, $\omega_p$, and $\omega_c$ denote the weight factors for sensing, processing, and communication throughput, respectively.

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

$$R_{sen} = F_s \cdot R^b_{sen} \tag{11}$$

where $F_s$ and $R^b_{sen}$ denote 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:

$$R_{proc} = F_p / N^b \tag{12}$$

where $F_p$ and $N^b$ denote 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:

$$R_{com} = P^{eff}_s \times 8 / t^{pkt}_{tx} \tag{13}$$

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 the various factors involved change dynamically, such as network topology, number of neighboring sensor nodes, wireless channel fading, sensor network traffic, packet size, etc. The two main factors that affect reliability are transceiver transmission power $P_{tx}$ and receiver sensitivity. For example, the AT86RF230 transceiver [15] has a receiver sensitivity of -101 dBm with corresponding packet error rate (PER) $\leq$ 1% for additive white Gaussian noise (AWGN) channel with physical service data unit (PSDU) equal to 20 bytes. Reliability can be estimated using Friis free space transmission equation [16] 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.

## IV. Experimental Results

### A. Experimental Setup

We base our experimental setup on the Crossbow IRIS mote platform [17], which has a battery capacity of 2000 mA-h with two AA alkaline batteries. The IRIS mote platform integrates an Atmel ATmega1281 microcontroller [12], an Atmel AT-86RF230 low power 2.4 GHz transceiver [15], an MTS400 sensor board [13] with Sensirion SHT1x temperature and humidity sensors [14].

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 $|S| = 31,104$ ($|S| = 729$ is a subset of $|S| = 31,104$). 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) [12], $F_s$ = {0.2, 0.5, 1, 2, 3, 4} (samples per second) [14], $P_s$ = {32, 41, 56, 64, 100, 127} (bytes), $P_{ti}$ = {10, 30, 60, 300, 600, 1200} (seconds), and $P_{tx}$ = {-17, -3, 1, 3} (dBm) [15]. All state space tuples are feasible for $|S| = 729$, whereas $|S| = 31,104$ contains 7,779 infeasible state space tuples (e.g., all $V_p$ and $F_p$ pairs are not feasible).

We model three application domains (a security/defense system (S/D), a health care application (HC), and an ambient conditions monitoring application (AC)) to evaluate the robustness of One-Shot across different applications. We assign objective function parameter values such as minimum and maximum values of application metrics and their associated weight factors considering typical application requirements [18].

In order to evaluate One-Shot's solution quality, we compare the solution from One-Shot's initial parameter settings $\mathcal{I}$ 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., $\mathcal{I}_4 = p_{i_q} : q = \texttt{rand()} \,\%\, n,\ \forall\ i \in \{1, 2, \ldots, N\}$).

### B. Results

We implemented One-Shot in C++. We compare our results with four different initial parameter arrangements (Section IV-A) and normalize the objective function value corresponding to the operating state attained by One-Shot with respect to the optimal solution obtained using an exhaustive search. We compare the relative complexity of One-Shot with two other dynamic optimization methodologies.

*1) Percentage Improvements over other Initial Parameter Settings:* Table I depicts the percentage improvements attained by One-Shot parameter settings $\mathcal{I}$ over other parameter settings for different application domains. We observe that some arbitrary settings may give a comparable 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, $\mathcal{I}_1$ achieves the same solution quality as of $\mathcal{I}$ for AC, but yields 73.27% and 147.87% lower quality solutions than $\mathcal{I}$ for HC and S/D, respectively, for $|S| = 31,104$. Furthermore, $\mathcal{I}_1$ yields a 51.85% lower quality solution than $\mathcal{I}$ for AC when $|S| = 729$. The average percentage improvement attained by $\mathcal{I}$ over all application domains and design spaces is 44.79%. In summary, results reveal that on average $\mathcal{I}$ gives a solution within 5.92% of the optimal solution obtained from exhaustive search.

*2) Comparison with Greedy- and SA-based Dynamic Optimization Methodologies:* In order to investigate the effectiveness of One-Shot, we compare the One-Shot solution's quality (indicated by the attained objective function value) with two other dynamic optimization methodologies, which leverage SA-based and greedy-based (denoted by $GD^{asc}$ where asc stands for ascending order of parameter exploration) design space exploration. We assign initial parameter value settings for greedy and SA-based methodologies as $\mathcal{I}_1$ and $\mathcal{I}_4$, respectively. For brevity we

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

| – | $\|S\| = 729$ | | | | $\|S\| = 31,104$ | | | |
|---|---|---|---|---|---|---|---|---|
| **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 (S/D) | 154.9% | 10.25% | 56.62% | 29.18% | 147.87% | 0.318% | 9.72% | 91.88% |
| Health Care (HC) | 77.6% | 6.66% | 30.73% | 10.86% | 73.27% | 0.267% | 9.62% | 45.17% |
| Ambient Conditions Monitoring (AC) | 51.85% | 6.17% | 20.39% | 6.85% | 0% | 75.5% | 50.97% | 108.31% |

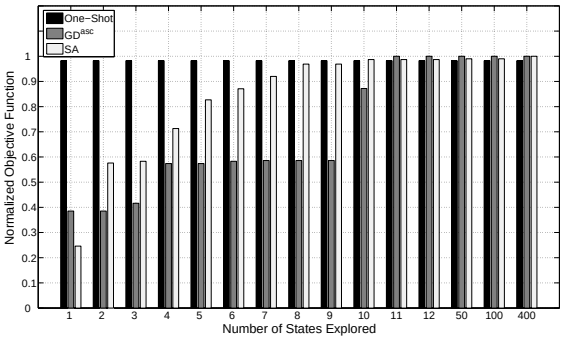


Figure 2. Objective function value normalized to the optimal solution for S/D where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, $|S| = 729$.

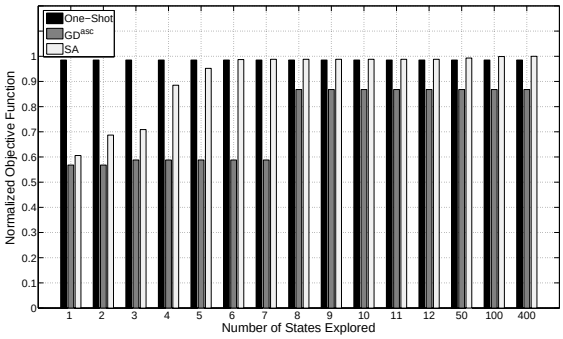


Figure 3. Objective function value normalized to the optimal solution for HC where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, $|S| = 31,104$.

present results for $\mathcal{I}_1$ and $\mathcal{I}_4$, but results for $\mathcal{I}_2$ and $\mathcal{I}_3$ revealed similar trends.

Fig. 2 shows the objective function value normalized to the optimal solution versus the number of states explored for One-Shot, GD$^{\text{asc}}$, and SA algorithms for S/D for $|S| = 729$. One-Shot's 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 explore few states to reach a comparable solution as that of One-Shot, One-Shot 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 reveal that other arbitrary initial value settings do not provide a good quality operating state and necessitate additional design space exploration to provide a good quality operating state.

Fig. 3 shows the objective function value normalized to the optimal solution versus number of states explored for HC for $|S| = 31,104$. One-shot's solution is within 1.5% of the optimal solution. The figure shows that GD$^{\text{asc}}$ converges to a lower quality solution than One-Shot's solution after exploring 8 states (0.026% of the design space) and SA explores 6 states (0.019% of the design space) to yield a better quality solution than One-shot's solution. These results reveal that the greedy exploration of parameters may not necessarily attain a better quality solution than One-Shot.

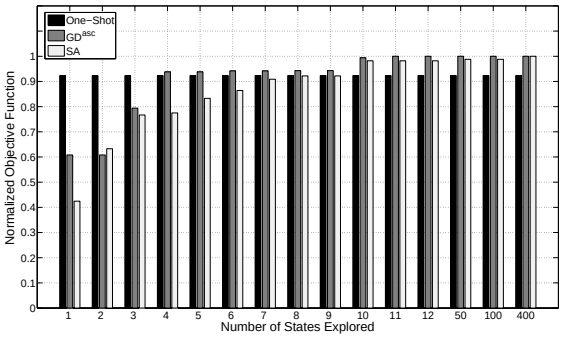


Figure 4. Objective function value normalized to the optimal solution for AC where $\omega_l = 0.4$, $\omega_t = 0.5$, $\omega_r = 0.1$, $|S| = 729$.

Fig. 4 shows the objective function value normalized to the optimal solution versus number of states explored for AC for $|S| = 729$. One-Shot solution is within 7.7% of the optimal solution. The figure shows that GD$^{\text{asc}}$ and SA converge to an equivalent or better quality solution than One-Shot solution after exploring 4 states (0.549% of the design space) and 10 states (1.37% of the design space), respectively. These results show that greedy and SA can provide improved results over One-Shot, but require additional state exploration.

*3) Computational Complexity:* To verify that One-Shot (Section II) is lightweight, we compared the data memory requirements and execution time of One-Shot with greedy- and SA-based dynamic optimization methodologies.

The data memory analysis revealed that One-Shot 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. Greedy requires 458, 528, 574, 870, and 886 bytes, whereas SA requires 514, 582, 624, 920, and 936 bytes of storage for $|S| = 8, 81, 729, 31104, 46656$, respectively. The data memory analysis shows that SA has comparatively larger memory requirements than greedy. Our analysis reveals that the data memory requirements for One-Shot increases

linearly as the number of tunable parameters and the number of application metrics increases. The data memory requirements for greedy and SA increases linearly as the number of tunable parameters and tunable values (and thus the design space) increases. The data memory analysis verifies that although One-Shot, greedy, and SA have low data memory requirements (on the order of hundreds of bytes), One-Shot requires 203.94% and 457.94% less memory on average as compared to greedy and SA, respectively.

We measured the execution time for One-Shot, greedy, and SA 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 [19] using the Linux/Unix `time` command [20]. We scaled the execution time to the Atmel ATmega1281 microcontroller [12] running at 8 MHz. Although microcontrollers have different instruction set architectures and scaling does not provide 100% accuracy, scaling enables relative comparisons and provides reasonable runtime estimates. Results showed that One-Shot required 1.66 ms both for $|S| = 729$ and $|S| = 31,104$. Greedy 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 greedy) 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 One-Shot required 18.325% less execution time on average as compared to greedy and SA. One-Shot required 66.26% and 73.49% less execution time than SA when $|S| = 729$ and $|S| = 31,104$, respectively. These results indicate that the design space cardinality affects the execution time linearly for greedy and SA whereas One-Shot's execution time is affected negligibly by the design space cardinality and hence One-Shot's advantage increases as the design space cardinality increases.

## V. Conclusions and Future Work

In this paper, we proposed One-Shot – a dynamic optimization methodology for highly-constrained WSNs that provides a high-quality operation state using intelligent initial tunable parameter value settings. We also proposed an application metric estimation model to estimate high-level metrics from sensor node parameters. This estimation model was leveraged by One-Shot and provided a prototype model for application metric estimation. To evaluate the effectiveness of initial parameter settings, we compared One-Shot's solution quality with four other typical initial parameter settings. Results revealed that the percentage improvement attained by One-Shot over other initial parameter settings was as high as 154.9% and within 5.92% of the optimal solution. Computational complexity analysis revealed that One-Shot used 203.94% and 457.94% less memory and required 18.325% less execution time on average as compared to greedy- and SA-based methodologies. Execution time and data memory analysis confirmed that One-Shot is lightweight and suitable for time-critical or highly constrained applications.

Future work includes incorporating profiling statistics into One-Shot to provide feedback with respect to changing environmental stimuli.

## Acknowledgments

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

## References

[1] 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, Sep. 2006.

[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, Dec. 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. 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] 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, Mar. 2010.

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

[7] X. Wang and et al., "Distributed Energy Optimization for Target Tracking in Wireless Sensor Networks," *IEEE Trans. on Mobile Computing*, vol. 9, no. 1, pp. 73–86, Jan. 2009.

[8] R. Khanna, H. Liu, and H.-H. Chen, "Dynamic Optimization of Secure Mobile Sensor Networks: A Genetic Algorithm," in *Proc. IEEE ICC'07*, June 2007.

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

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

[11] 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, Jan. 2009.

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

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

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

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

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

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

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

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

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