Multi-objective Reinforcement Learning for Energy Harvesting Wireless Sensor Nodes

Shaswot Shresthamali
Keio University
shaswot@acsl.ics.keio.ac.jp

Masaaki Kondo
Keio University
kondo@acsl.ics.keio.ac.jp

Hiroshi Nakamura
The University of Tokyo
nakamura@hal.ipc.i.u-tokyo.ac.jp

***Abstract*—Modern Energy Harvesting Wireless Sensor Nodes (EHWSNs) need to intelligently allocate their limited and unreliable energy budget among multiple tasks to ensure long-term uninterrupted operation. Traditional solutions are ill-equipped to deal with multiple objectives and execute a posteriori tradeoffs. We propose a general Multi-objective Reinforcement Learning (MORL) framework for Energy Neutral Operation (ENO) of EHWSNs. Our proposed framework consists of a novel Multi-objective Markov Decision Process (MOMDP) formulation and two novel MORL algorithms. Using our framework, EHWSNs can learn policies to maximize multiple task-objectives and perform dynamic runtime tradeoffs. The high computation and learning costs, usually associated with powerful MORL algorithms, can be avoided by using our comparatively less resource-intensive MORL algorithms. We evaluate our framework on a general single-task and dual-task EHWSN system model through simulations and show that our MORL algorithms can successfully tradeoff between multiple objectives at runtime.**

I. Introduction

Energy Harvesting Wireless Sensor Nodes (EHWSNs) are becoming increasingly popular as edge devices for the Internet of Things (IoT). They are capable of untethered, autonomous and long-term operation. Modern EHWSNs have sophisticated System-on-Chips (SoCs) that enables them to perform a variety of tasks like sensing, communication and processing [1]. However, these nodes operate with very stringent energy budgets and unpredictable energy availability. It is therefore necessary for them to judiciously budget their energy among multiple tasks to i) ensure energy neutrality for long-term operation and ii) maximize the overall node utility. Energy Neutral Operation (ENO) [2] requires that the average supply and consumption of energy remain balanced for long-term operation. On the other hand, maximizing node utility requires intelligently expending energy for various tasks in order of their user-defined priorities. Thus, optimal operation of a general EHWSN requires i) proportioning energy between different tasks and ii) ensuring that the gross energy consumption does not violate energy neutrality.

Traditionally EHWSNs optimized for a single objective using low-compute analytic methods such as linear programming [2] and control systems [3], [4]. However, these solutions were minimally adaptive and required significant hand-tuning. As EHWSNs became more powerful, Reinforcement Learning (RL)-based methods were proposed as an alternative [5], [6]. In this paradigm, the nodes *learn* the energy management policy on the fly as opposed to having pre-programmed heuristics. During RL, the node (or agent) interacts with its environment and explores various energy management policies via trial-and-error. A reward function gives feedback to the node which it uses to optimize its policies and maximize future rewards. With RL-based policies, the nodes can adapt to their specific working environment and therefore be used in a deploy-and-forget fashion. Moreover, since they require minimal application-specific design bias, they could be massively scaled to a large number of applications with little or no changes to the RL framework. This scalability of RL is an extremely important advantage over analytic methods because it allows a single framework to work for potentially billions of EHWSNs in diverse applications. However, these RL solutions focus on only one aspect of the EHWSN operation such as ensuring ENO [6] or maximizing throughput [7], [8] or the sensing rate [5], [9] and therefore are not suited for working with multiple objectives.

Motivation: Modern EHWSNs require a general, integrated solution that optimizes for multiple task objectives in addition to maintaining ENO. Since all tasks draw energy from the same limited source, tradeoffs need to be made based on user-defined preference or *priority*. Generally, the relative priorities are not known beforehand and can be adjusted only *after* observing the possible tradeoffs. Hence, a multi-objective optimization approach may be more suitable for this kind of problem.

Current multi-objective solutions for EHWSNs have proposed to reduce the multiple objectives to a set of constraints and a single objective [10]. They require very strong application-specific assumptions and design-bias (e.g., non-causal information/ network dependent optimization objectives). This approach is not general and cannot scale well. Traditional single-objective RL (SORL) solutions are also not suitable for optimizing over multiple objectives. SORL is based on learning from a single *scalar* feedback reward signal corresponding to a single objective. They cannot accommodate multiple sources of rewards (i.e., multiple optimization objectives). Some have tried to work around this by projecting the rewards onto a scalar [5], [8], [11], [12]. However, as we will see later, this approach requires strong prior assumptions, produces sub-optimal policies and cannot tradeoff dynamically at runtime. Thus current analytic and RL solutions are insufficient for optimization of modern EHWSNs.

Contributions: In this work, we propose a novel MORL

framework for multi-objective optimization of EHWSNs. We use RL over analytic approaches due to its adaptivity and generality. Our proposed framework consists of a novel Multi-Objective Markov Decision Process (MOMDP) formulation of the EHWSN optimization problem and two MORL algorithms. With our proposed solution, EHWSNs can learn intelligent policies (with reduced learning costs) to i) maximize multiple task objectives, ii) optimize gross device energy consumption for long-term ENO and iii) dynamically tradeoff between objectives during runtime. Specifically, we make the following contributions in this paper:

- We present a simple but general system model for multi-task EHWSNs in Section III that accommodates multiple tasks objectives and their relative priorities. Based on this system model, we specify the MORL problem in the form of a general MOMDP (Section V-A).
- We propose a novel MORL algorithm, *Runtime MORL*, that reuses pre-trained SORL policies for dynamic tradeoffs at runtime (Section V-B).
- We propose another MORL algorithm, *Off-Policy MORL*, that learns energy-neutral policies tabula rasa (within reasonable computing costs) and is also capable of dynamic runtime tradeoffs (Section V-C).

We give a brief review of related literature in Section II. We explain our experimental setup in Section VI and evaluate our proposed framework extensively through simulations of single-task and dual-task EHWSNs. We discuss the results of our experiments in Section VII before concluding with Section VIII.

II. Related Work

Multi-objective optimization have traditionally focused mostly on network performance rather than node energy-neutrality [10]. Node-centric RL solutions have primarily focused on using scalarized reward functions [8], [11], [12]. Other sophisticated MORL algorithms that overcome the limitations of scalarization and enable a posteriori tradeoffs exist but at extremely high learning and computation costs [13], [14]. Our proposed method cuts down these costs significantly (by at least an order of magnitude). To our knowledge, our work is the first to use "true" MORL for ENO in EHWSNs and enable tradeoffs. Consequently, we have yet to address issues of intermittent computing, task dependency and network-based optimizations. More importantly, there is still work required to implement our MORL solution in very resource-constrained IoT systems. Some possible directions for this would be to use tabular methods [5], [6], linear function approximation [7], [8] or distributed learning [15]; since our framework is theoretically applicable for tabular RL or other less compute-intensive RL. It may also be possible to compress the neural networks to fit within given SoC requirements. There have been encouraging results where researchers have been able to compress and fit a 16 GB ImageNet model in a micro-controller (less than 1 MB) [16] within reasonable limits of accuracy and latency. In [9], [17], the authors implement RL in low-power hardware. With more efficient learning algorithms and power-efficient and powerful processors, it may soon be possible to implement our MORL in small IoT edge devices.

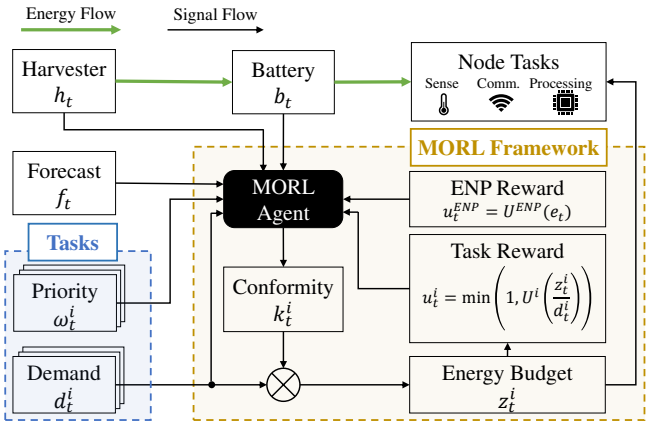

Fig. 1: A general system model for multi-task EHWSNs.

III. System Model

Our system model (Figure 1) assumes a harvest-store-use solar EHWSN that is required to be always on. The EHWSN is equipped with an energy harvester and a finite energy buffer. The node receives user requests to execute different tasks related to sensing, communication and processing. Each request has a corresponding energy *demand* required to meet different performance requirements such as sensing rates [4] and throughputs [5], [7].

The MORL agent works in discrete timesteps. It intelligently allocates the energy among different tasks to generate *task-utilities* and ultimately maximize the *node-utility*. It observes its environment (the system state) at timestep t to decide the conformity of ith task $k_t^i \in [\ 0, 1]$, to its demand $d_t^i \in [\ z_{min}^i, z_{max}^i]$. The energy allocated to task i is $z_t^i = \max(z_{min}^i, d_t^i \times k_t^i)$ which generates a task-utility of $u_t^i = \min(1, U^i(z_t^i/d_t^i))$, where U^i is a monotonically non-decreasing function. The constraint $u_t^i \leq 1$ accounts for the fact that over-provisioning does not lead to increased utility. Traditional solutions usually overlooked the issue of task utilities and opportunistically maximized duty cycles [6], [15]. Our system model is more general and realistic because it maximizes energy consumption *only as requested*.

A. Energy Neutral Operation

At time t, the battery level is $b_t \in [\ 0, b_{max}]$, the harvested energy is $h_t \in [\ 0, h_{max}]$, the total energy demand is $d_t = \sum_{i=1}^{n} d_t^i$ and the energy consumed by the node is $z_t = \sum_{i=1}^{n} z_t^i$ s.t. $z_{min} \leq z_t \leq z_{max}$. The energy dynamics of the system is therefore given by $b_{t+1} = \min(b_t + (h_t - z_t), b_{max})$.

Perfect energy-neutrality is guaranteed if $h_t - z_t = 0 \forall t$. However, this may not always be possible or desirable. The sensor may need to operate during the night ($h_t < z_t$); or, it may not be optimal to greedily use all harvested energy ($h_t = z_t$) because some of that energy could be saved to extract more utility in the future. However, when $\mathbb{E}[z] = \mathbb{E}[h]$, $\mathbb{E}[\cdot]$ being

the expectation operator, the node can be considered energy-neutral. The node is operational at time t if $b_t + h_t \geq z_{min}$. Otherwise, there is a *downtime*. In such a case, the node is non-operational and uses all of the harvested energy to replenish its battery to a user-specified level and then resumes operation.

Ideally we would like an energy management scheme that minimizes downtimes but ensures that all the energy harvested is intelligently used to maximize utility. The energy-neutrality of the node at time t is given by the Energy Neutral Performance (ENP) metric $e_t = \sum_0^t \mathcal{B}(h_t - z_t) = b_0 - b_t$, expressed as the total difference between harvested and consumed energy [5], [6], [15].

The *ENP-utility* is u_t^{ENP}, given by a function $U^{ENP}(e_t)$ which reflects the energy-neutrality of the node. The node-utility is a weighted sum of the individual task-utilities w.r.t. their priorities. For an EHWSN with n-tasks, the relative priorities between its $n+1$ objectives is expressed by $\omega_t = (\omega_t^1, \omega_t^2, ..., \omega_t^{n+1})$ where $\omega_t^i \in [0, 1]$ is the priority/preference for the ith objective. $\omega_t^{n+1} = (1 - \sum_{i=1}^{n} \omega_t^i)$ is the implied priority for the energy-neutrality of the node. With a slight abuse of notation, for single-task EHWSNs, the priorities of task maximization and ENO are represented by ω_t and $(1-\omega_t)$. The agent's ultimate objective is to maximize the overall node-utility $w_t = \sum_{i=1}^{n+1} u_t^i \omega_t^i$.

IV. Theoretical Background

A. Single-objective RL

In standard SORL, at each timestep, an agent observes its environment state $s_t \in \mathcal{S}$ and executes an action $a_t \in \mathcal{A}$ according to some policy $\pi(a|s)$. Consequently, it receives a scalar reward r_t that reflects the optimality of the action w.r.t. the optimization objective and transitions to the next state s'. The rewards are given by the reward function $\mathcal{R}(s, a, s')$ and are discounted by a factor of $\gamma \in [0, 1]$. This process iterates until the agent reaches a terminal state (end of an episode) and the process restarts. Using the rewards and its past experiences, the agent learns increasingly better policies that maximize its cumulative reward or *return*.

The Q-value of a state-action pair $Q^\pi(s, a)$, w.r.t. policy π, gives the *expected* return when executing action a from state s as defined in Equation (1a) for an episode of length T. It can be can be computed recursively by using the Bellman equation in Equation (1b).

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1}) | s_0 = s, a_0 = a \right] \quad (1a)$$

$$Q^\pi(s, a) = \mathbb{E}_\pi [R(s, a, s') + \gamma Q(s', \pi(s'))] \quad (1b)$$

Thus, RL basically involves i) learning the Q-values of all state-actions pairs (predictive knowledge) and ii) learning a policy to choose an action that maximizes the Q-value for every state (procedural knowledge). We approximate the Q-values with a neural function approximator $Q_\theta(s, a)$ parameterized by θ (the *critic*). The policy is output by a function $\pi_\phi(s)$ with parameters ϕ (the *actor*). In this work we use Deep Deterministic Policy Gradient (DDPG) [18], to learn the actor and critic functions. We choose DDPG because it accommodates continuous states and actions and is simple to implement (compared to other similar RL algorithms).

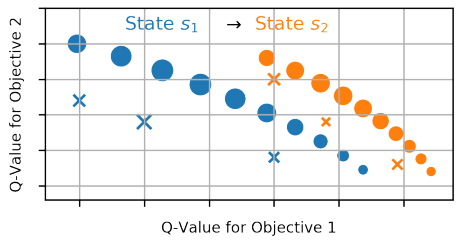

Fig. 2: The Pareto-front changes when the agent transitions form state s_1 to s_2. Crosses indicate Pareto-dominated actions. The marker size represents the utility of the action.

B. Multi-objective RL

In a general multi-objective problem, an optimizing variable is mapped by *objective functions* into a multi-dimensional objective space. Pareto-optimal points correspond to those mappings where the value of one objective cannot be increased without a decrease in the value of at least one another objective. The overall utility of the variable is a linear combination of its mapped objective values according to the user-defined weights (non-linear utility functions are outside the scope of this work).

In our MORL framework, the optimizing variable is the agent's action $\pi(s) = a$. The Q-functions $Q^\pi(s, a)$ (and not the instantaneous reward) are the objective functions because they represent the long-term effect of the actions. However, $Q^\pi(s, a)$ is dependent on the state s, which changes at every timestep. This is shown in Figure 2 where the same set of sampled actions map to different points in the objective space when the state changes from s_1 (blue) to s_2 (orange). It is due to this non-static objective space that traditional elitist multi-objective optimization methods such as Monte Carlo (MC) rollouts or Evolutionary Algorithms (EA) are infeasible. These methods require multiple rollouts and generations which consume a lot of time and resources.

In general, MORL solutions need to learn the Q-functions and a policy that can infer the optimal action from Pareto-frontier. A general Multi-Objective Markov Decision Process (MOMDP) for MORL with m objectives is defined by $(\mathcal{S}, \mathcal{A}, \mathbb{P}, \boldsymbol{R}, \gamma, \Omega)$. $\mathcal{S}$ is the continuous state space, $\mathcal{A}$ is the continuous action space and $\mathbb{P}(s'|s, a)$ defines the transition probability from state s to s' as a result of action a. We choose continuous states and actions for generality (as compared to MDPs with discrete state-actions spaces in previous works [5], [6]). The reward function is now a vector function, $\boldsymbol{R}(s, a) = [R_1(s, a), R_2(s, a), ..., R_m(s, a)]$ and the discount factors for each objective are given by $\gamma = [\gamma_1, \gamma_2, ..., \gamma_m]$. Ω determines the space of preferences among the m objectives. The MORL policy μ is now associated with an m-dimensional vector Q-value function $\boldsymbol{Q}^\mu = [Q_1, Q_2, ..., Q_m]$ whose components correspond to the different objective functions. μ extracts the

Pareto-optimal action w.r.t. the user-preference $\omega \in \Omega$. In the following section, we develop a suitable MOMDP for EHWSNs and present solutions to learn the vector Q-functions and extract the optimal actions.

V. Proposed MORL Framework

A. MOMDP Formulation

Multiple Reward Functions: Modern EHWSNs requires a reward function that accommodates multiple objectives. Previous SORL methods project these objectives into a scalar that introduces noise and obfuscates the reward incentives. The most significant drawback is that tradeoffs cannot be made at runtime because it requires a priori knowledge of the weights [11]. Another disadvantage is that scalarization involves complicated reward shaping [5], [12] and introduces significant design bias.

To avoid these limitations, our framework learns from multiple separate independent reward signals. Each of these reward signals represent one and only one optimization objective without any complicated reward shaping. We define the rewards for each task by their task-utilities and for energy-neutrality by the ENP-utility.

Robust State-Action Space: The state at time t is defined as a tuple $s_t = (\tau, b_t, \bar{b_t}, h_t, f_t, d_t)$. h_t is the harvested energy and $d_t = (d_t^1, d_t^2, ..., d_t^n)$ is a tuple of the different task requests. We also include additional temporal information, τ which represents the time of the day, $\bar{b_t} = \sum_{k=0}^{T} b_{t-k}/T$ which is a moving average of battery values over a horizon T, and $f_t \in [0, 1]$ which is a rough prediction of future energy supply similar to [6].This addition of temporal information makes learning process more robust and stable than previous works [5], [8] which didn't include sufficient temporal information in their state definitions.

We define the action as the conformity of the node to different task demands in our formulation. For an n-task EHWSN, the action is $k_t = (k_t^1, k_t^2, ..., k_t^n)$ where k_t^i is the conformity of the node to the request d_t^i. This definition ensures actions are *safe* in that it never over-provisions energy and thus prevents unwanted battery depletion. It minimizes the effects of catastrophic policies even if the learning fails unlike actions definitions in previous methods [5], [6], [8].

B. Runtime MORL

We now describe our first algorithm, *Runtime MORL* (Algorithm 1). An n-task EHWSN has an n-dimensional actions space i.e., $|k| = n$. It has to optimize over $n + 1$ objectives where the $n + 1$th objective is w.r.t. energy-neutrality. Algorithm 1 uses n pre-trained greedy actor-critic networks (π_i, Q_i) to output each component of the action space. π_i outputs the greedy conformity g_i for the ith task. An additional pre-trained greedy actor-critic (π_{ENP}, Q_{ENP}) outputs the *total* conformity of the node g_{ENP} w.r.t the *total* demand $d = \sum_{i=1}^{n} d^i$. We can construct a plane in the action space given by $g_{ENP} = \sum_{i=1}^{n} k_i$ that represents all actions that are greedily energy-neutral. We then determine the convex hull that contains this plane and all the greedy actions. The main intuition behind Algorithm 1 is that as we move from one greedy action to another along this convex hull, we are actually traversing along the Pareto-front in the objective space making tradeoffs. This is valid in our case because task-utility functions are monotonically non-decreasing function of conformity (or action).

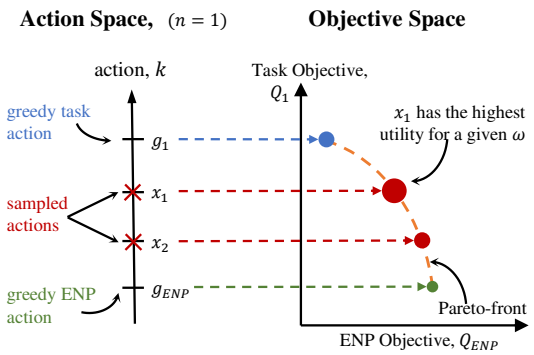

Fig. 3: As we move from one greedy action to another in the action space, we traverse along the Pareto-optimal frontier in the objective space. *Runtime MORL* samples actions (red crosses) from in between the greedy actions, g_1 and g_{ENP}.

1) Minimizing Computation Costs: To find the optimal action k^* we *sample* the locally approximate Pareto-optimal from the convex hull. For a given user-preference $\omega = (\omega_1, \omega_2, ..., \omega_{ENP})$, the algorithm samples j actions $x_1, x_2, ..., x_j$. The Q-values for all actions are evaluated by linear combination of outputs of the critics Q_i and Q_{ENP}, weighted by the task preferences ω_i and ω_{ENP}. The locally optimal action is the one with the maximum value. This way, a change in ω at *runtime* only requires inferring the critics and a recomputation of the action-values, which is not a very costly operation. Overall, this algorithm requires much less computation than having to infer the entire Pareto-optimal front and perform gradient descent to find the optimal action. Another practical advantage of this sampling approach is that we can always find an action at least as good as the greedy actions without any assumption about the concavity of the Pareto-frontier. This is significant because some MORL methods do not perform well for concave Pareto-frontiers.

Figure 3 illustrates Algorithm 1 for a single-task scenario ($n = 1$). Action g_1 greedily maximizes the task-utility by increasing conformity k, whereas g_{ENP} greedily maintains energy-neutrality by decreasing k. Here, the convex hull is simply a straight line bounded by g_1 and g_{ENP}. Algorithm 1 samples two points on the line indicated by the red crosses. The Q-values for all actions (greedy and sampled) are mapped to the objective space using the outputs of the critics (Q_1, Q_{ENP}) and ω to find the optimal action. The action values are represented by the size of the circles in the figure.

C. MORL with Off-Policy Corrections

We now consider the case when pre-trained greedy actors and critics are not available and have to be trained from scratch. With our second algorithm, *Off-policy MORL* (Algorithm 2), the actors learn greedy policies, and critics learn

Algorithm 1: Runtime MORL

Input : $\omega \in \Omega$, priorities for $n+1$ objectives
$s \in \mathcal{S}$, State
(Q_i, π_i), Greedy actor-critics, $i = 1, 2, .., n$
(Q_{ENP}, π_{ENP}), Energy-neutral actor-critic
Output: Action k^* that maximizes utility, $|k^*| = n$

$actionList = [\,]$ `// Proposed actions list`
for $i = 1, 2, ...n, ENP$ **do**
  $g_i = \pi_i(s)$ `// Greedy Actions`
  $actionList.append(g_i)$
end for
Sample j actions $x_1, x_2, ...x_j$ from the convex hull of actions in $actionList$ and append to $actionList$
$W = \{\,\}$
for x *in* $actionList$ **do**
  $W[x] = \sum_{i=1}^{n+1} Q_i(s, x)\omega^i$
end for
return $k^* = \underset{x}{\operatorname{argmax}}\, W[x]$

Algorithm 2: Off-policy MORL

Input : γ_i, discount factor for ith objective, $i = 1, 2, ...n+1$
Initialize: Randomly initialize actor-critic pairs (Q_i, π_i) and empty replay buffer D

for $episode = 1, L$ **do**
  for $t = 1, T$ **do** `// T = episode length`
    Observe the preferences $\omega_t \in \Omega$
    Observe state s_t
    Select action a_t using Algorithm 1
    Execute action a_t, observe the reward vector $\boldsymbol{r_t} = [r_t^1, r_t^2, ..., r_t^m]$ and the next state s_{t+1}
    Store $(s_t, a_t, \boldsymbol{r}_t, s_{t+1}, \omega_t)$ in D
    Sample minibatch $\mathbb{B}$ of N transitions from D
    for $i = 1, n+1$ **do**
      Update Q_i using r_i and γ_i in Equation (2),
      Update π_i
    end for
  end for
end for

their Q-values from the experience samples generated by the agent's interaction with the environment. The agent executes actions that lie in between the greedy actions depending on the preference ω using the method in Algorithm 1. This means that the critics have to learn the Q-values Q_i for a greedy policy π_i from transitions generated by a different policy, say μ.

To account for this, we note that the probability of state transitions from s to s' are different for π_i and μ. Thus, we apply an off-policy correction by multiplying the expected Q-values of the next state by their ratio of their transition probabilities, $\rho_i(s, s') = \frac{\mathbb{P}(s'|s,\pi_i)}{\mathbb{P}(s'|s,\mu)}$. For our case, we approximate $\rho_i(s, s') \approx \omega_i$. Although theoretical guarantees for this are beyond the scope of this work, this approximation makes sense. When ω_i is closer to unity, it implies that μ is more dominated by π_i so the off-policy correction does not change the expectation much. On the other hand, when ω_i is very low, μ is much further away from π_i and thus the expected return is reduced proportionately.

$$\begin{aligned} Q_i(s,a) &= \mathbb{E}_{\pi_i}[r^i + \gamma_i Q_i(s', \pi_i(s'))] \\ &= \mathbb{E}_{\mu}[r^i] + \gamma_i \rho_i(s,s') \mathbb{E}_{\mu}[Q_i(s', \pi_i(s'))] \\ &\approx \mathbb{E}_{\mu}[r^i + \gamma_i \omega^i Q_i(s', \pi_i(s'))] \end{aligned} \tag{2}$$

VI. Evaluation Setup

A. Simulation Environment

We evaluate our framework by simulating solar EHWSN systems based on the solar radiation data from 1995 to 2018, logged hourly by outdoor rooftop pyranometers [19]. We therefore set the time interval for one timestep to be one hour. We use the first ten years (1995-2004) for training the RL agents and the remaining years to test their performance. We use an episodic MDP - with one episode lasting one day (or 24 timesteps). An episode abruptly terminates with zero reward when a downtime occurs. The parameters of the evaluated EHWSN system, normalized to b_{max}, are shown in Table I. These parameters roughly correspond to a realistic EHWSN [20] with a current rating of $100\,\mathrm{mA}$ and a $2000\,\mathrm{mA\,h}$ battery. The node goes into recovery mode when the battery capacity drops below 10%. The requests are randomly generated by the request function such that $\mathbb{E}[h_t] \approx \mathbb{E}[d_t]$. This ensures that ENO is actually feasible. A rough estimate of the weather prediction f_t, is given by adding some random noise to the rolling average of h_t over the next ten days.

TABLE I: Simulation Parameters

Parameter	maximum value	minimum value
Battery, b_t	b_{max}	$b_{min} = 10\%$ of b_{max}
Harvester, h_t	$h_{max} = 5\%$ of b_{max}	$h_{min} = 0$
EHWSN, z_t	$z_{max} = 5\%$ of b_{max}	$z_{min} = 0.5\%$ of b_{max}
Requests, d_t	$z_{max} = 5\%$ of b_{max}	$z_{min} = 0.5\%$ of b_{max}

B. Utilities and Reward Functions

We run our evaluations on single-task (bi-objective) and dual-task (tri-objective) EHWSNs. The single-task node maximizes its sensing utility by increasing the conformity to the user demand while maintaining long-term energy-neutrality. Typically, the utility of sensed data increases linearly with energy consumption [2]–[4], [6]. Thus, we define the rewards for sensing rate maximization by the *sense-utility* u_t^{sense}, which grows linearly with conformity (energy usage) according the *linear-utility* function in Figure 4 (left).

The ENP-utility u_t^{ENP} corresponds to the rewards that represent the energy-neutrality of the node given by

$$u_t^{ENP} = \begin{cases} 1, & \text{if } \bar{b_t} \geq b_{th} \\ \frac{\bar{b_t} - b_{min}}{b_{th} - b_{min}}, & \text{otherwise} \end{cases} \tag{3}$$

This reward function is illustrated in Figure 4 (right). $\bar{b_t}$ is the the moving average battery level over ten days and b_{th} is a

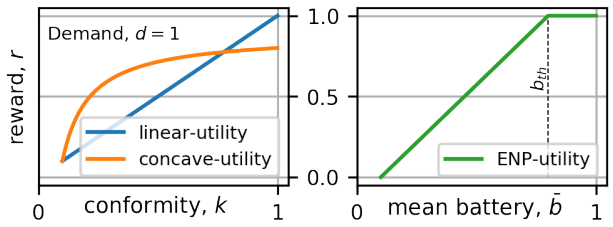

Fig. 4: Linear and concave reward functions quantify the utilities for different tasks such as sensing, communication and processing. The ENP-utility function indicates the long-term energy-neutrality of the node.

user-defined battery threshold. This is similar to the reward functions in [5], [6], [12], [15] except that it has minimal reward shaping and uses $\bar{b}_t$ instead of b_t. b_{th} is fixed at 80% of b_{max} for our case but this may be adjusted as necessary. We use the average battery level here because it reflects the long-term temporal nature of ENO. For single-task EHWSN, ω reflects the relative preference between maximizing the sense-utility and ENP-utility. $\omega = 1$ maximizes sense-utility and $\omega = 0$ emphasizes energy-neutrality.

For dual-task EHWSNs, we consider an EHWSN system that needs to sense and transmit its data. In addition to remaining energy-neutral and maximizing its sense-utility, it also has to maximize its throughput. Here, the conformity $k = z/d$ represents the SNR and the node's throughput P is given by Shannon's capacity formula $P = log(1 + k)$. The task-utility and reward is therefore $u_t^{tx} = P$. Higher throughput requires higher SNR but has diminishing returns. This relationship is represented by the concave-utility reward function in Figure 4. For sake of example, this reward function assumes Binary Phase Shift Keying (BPSK) modulation with additive white Gaussian noise and Rayleigh fading. At every instant, the dual-task node has to decide how much energy it must allocate to transmission and sensing tasks so as to maximize its throughput (or tx-utility) u_t^{tx} and sense-utility u_t^{sense}, while ensuring long-term ENO (maximizing u_t^{ENP}). The preference between the objectives is given by a tuple $\omega = (\omega_{sense}, \omega_{tx}, \omega_{ENP})$ s.t. $\omega_{ENP} = 1 - (\omega_{sense} + \omega_{tx})$.

C. Metrics

We compare the *task-utilities* between different solutions by their annual average. The *energy-neutrality* of different methods is compared on the basis of the total number of times the node goes into recovery mode (downtimes). This is a more direct representation of the actual energy-neutrality than the ENP-utility and facilitates fair comparisons. An intelligent policy finds the right balance between maximizing task-utilities and minimizing downtimes. Since the training period is the same for all RL agents, the *learning costs* are given by the number of downtimes that happen during the training period, lesser being better. Ideally, we would like the node to learn an optimal energy management policy with as few downtimes as possible.

TABLE II: Multi-objective ENO agents

Method	Name	Rewards/Values
Ours	*morl_runtime* (2-task)	$\omega Q^{\pi_{sense}} + (1-\omega)Q^{\pi_{enp}}$
Ours	*morl_multi* (3-task)	$\omega_{sense}Q^{\pi_{sense}} + \omega_{tx}Q^{\pi_{tx}} + \omega_{ENP}Q^{\pi_{enp}}$
Scalar Product [5], [8]	*mul_scalar*	$u^{sense} \times u^{ENP}$
Scalar Sum [11]	*add_scalar*	$\omega u^{sense} + (1-\omega)u^{ENP}$

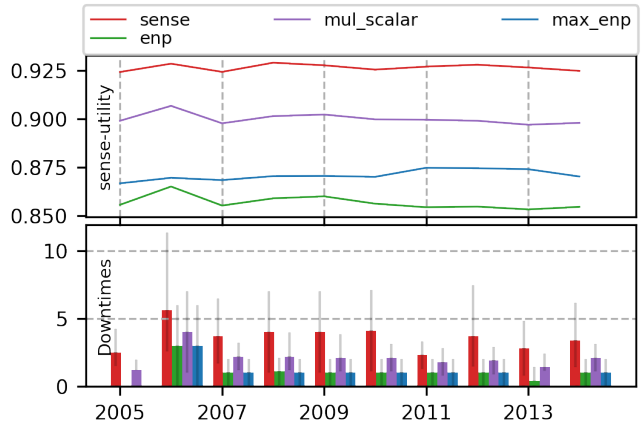

Fig. 5: Agent *enp* (ours) has similar energy-neutrality (downtimes) as the most energy-neutral policy *max_enp*. *sense* (ours) sacrifices some of its energy-neutrality to increase its utility.

VII. Experimental Results

Each experiment is executed ten times with different seeds. The average over all seeds are used for comparison. The interquartile range (IQR) is indicated by error bars (not shown in some figures for clarity). A summary of the different MORL agents we evaluate is given in Table II.

A. Baselines

In Figure 5, we compare between different RL policies against a baseline heuristic policy called *max_enp* (blue) w.r.t. their sense-utilities (top) and downtimes (bottom). *max_enp* is the most energy-neutral policy with the minimum possible downtimes. *max_enp* increases node conformity k monotonically with rising battery levels in a non-linear fashion. The hyperparameters for this function were empirically determined using greedy search and non-causal information. No policy can extract higher utility than *max_enp* without increasing the number of downtimes.

Figure 5 also shows three SORL agents ($\gamma = 0.997$): *sense* (red), *enp* (green) and *mul_scalar* (purple). *sense* and *enp* maximize sense-utility and ENP-utility respectively using the linear-utility and ENP-utility reward functions (Figure 4). *mul_scalar* uses a scalarized reward given by $r_t = u_t^{sense} \times u_t^{ENP}$ similar to [8], [12].

We observe that *sense* has the highest utility but also correspondingly highest number of downtimes. This is due to its reward function that is designed solely to maximize its utility. Although *sense* extracts higher utility within acceptable limits of energy-neutrality, we cannot say whether it is optimal. A more optimal solution would achieve a higher sense-utility

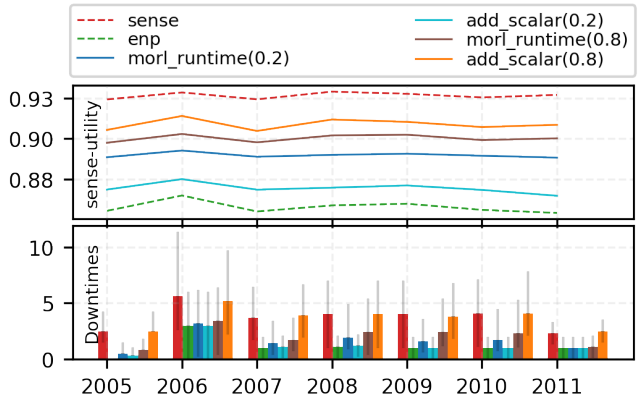

Fig. 6: *morl_runtime*(ω) (Algorithm 1) can tradeoff sense-utility with energy-neutrality (downtimes) at *runtime*.

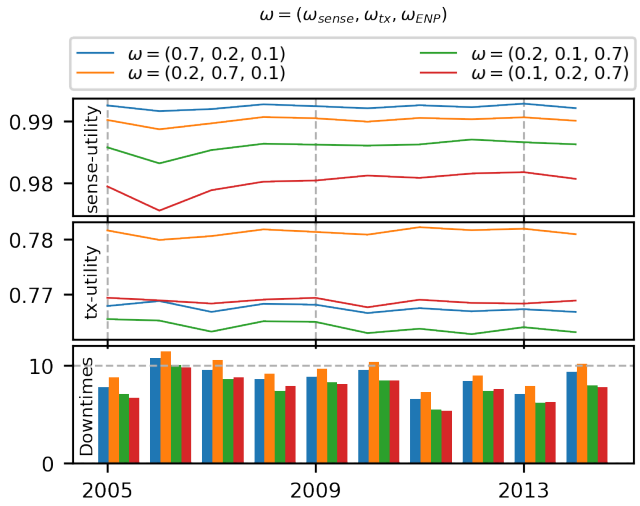

Fig. 7: *morl_multi* agent has lower downtimes when ENO has higher priority for ENO (green and red) than when it is low (blue and orange). Similarly, for the same ω_{ENP}, the agent trades off between sense-utility and tx-utility. (2015-2019 not shown for clarity.)

while maintaining the same number of downtimes as *sense*. It is difficult to define absolute optimal baselines in this case and so we use *sense* as the baseline for comparing sense-utility among RL agents.

On the other hand, *enp* has the lowest utility and lowest number of downtimes on account of its ENO-centric reward function. Although *enp* has lower utility than *max_enp*, the energy-neutrality (downtimes) of *enp* and *max_enp* are similar (green and blue bars). *This means that enp is performing near-optimally w.r.t. the ENO objective.* These observations lead us to conclude that the SORL agents are learning policies that are consistent with their reward functions.

The performance of *mul_scalar* agent lies somewhere in between that of *sense* and *enp*. Due to its scalarized reward function, *mul_scalar* is neither as energy-neutral as *enp* nor as useful as *sense*. This is a limitation of scalarization methods for SORL. It shows that while one can end up with a working policy with scalarization, due to the "mixing" of rewards, it is not fully clear what objective is actually being optimized. Furthermore, it is not at all possible to tradeoff between different objectives with the reward scheme used in *mul_scalar*.

B. Scalarization and Runtime Tradeoffs

We now consider the dual-objective case when the single-task EHWSN agent has to tradeoff between maximizing the sense-utility u_t^{sense}, and the ENP-utility u_t^{ENP}. For a given ω, the agent has to maximize the node-utility, $w_t = \omega u_t^{sense} + (1-\omega)u_t^{ENP}$.

We compare between two RL agents: *morl_runtime*(ω) and *add_scalar*(ω). *morl_runtime*(ω) uses reuses the *sense* and *enp* agents from the previous section and our proposed *Runtime MORL* algorithm (Algorithm 1) to generate tradeoff policies. *add_scalar*(ω) uses scalarized reward functions given by $r_t = \omega u_t^{sense} + (1-\omega)u_t^{ENP}$, similar to [5], [11], [12]. The important distinction between these two methods is that *morl_runtime*(ω) can tradeoff between objectives even if ω is provided only at runtime. In contrast, add_scalar(ω) needs ω to be known before the training process and has to be retrained every time ω changes. Scalarization methods are very limited in their application precisely because of this requirement of prior knowledge of ω.

Figure 6 shows the test results of *morl_runtime*(ω) and add_scalar(ω) for different values of ω (held constant during testing). We observe that our *morl_runtime*(ω) agent can indeed tradeoff for different ω. High priority ($\omega = 0.8$, brown) increases utility and downtimes and vice versa for low priority ($\omega = 0.2$, blue). Similar tradeoffs are achieved by *add_scalar*(ω). Since we are plotting the annual average over ten different seeds, the differences in the utility do not seem very significant. However, when we consider the cumulative utilities over long periods of operation, the differences are quite substantial.

We note that the effect of tradeoffs is lesser for *morl_runtime* than *add_scalar*. This is because the tradeoff takes place in the Q-value space for *morl_runtime* and in the reward space for *add_scalar*. This is actually preferable because *morl_runtime* is more optimal than *add_scalar*. In Figure 6, *morl_runtime (0.2)* (blue) consistently extracts much higher utility than *add_scalar (0.2)* (cyan) for similar downtimes. In contrast, *add_scalar (0.8)* (orange) has disproportionately higher downtimes for only a slight increase in sense-utility than our *morl_runtime (0.8)* (brown).

C. MORL for three objectives

Finally, we consider the case of the dual-task EHWSN. We use Algorithm 2 to train an agent tabula rasa to optimize and tradeoff between its three objectives (sensing, transmission and ENO). During training, the user priority $\omega = (\omega_{sense}, \omega_{tx}, \omega_{ENP})$ changes randomly at each timestep. At test, we maintain a constant ω for each experiment for comparison purposes.

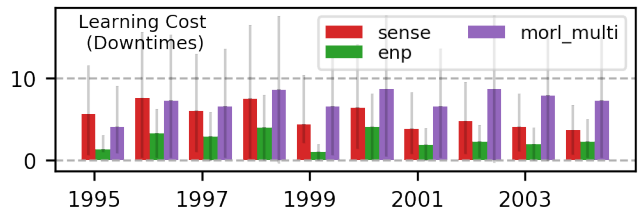

Fig. 8: *morl_multi* has acceptable learning costs (slightly higher than *sense*) even though the learning problem is much harder.

First, we observe how *morl_multi* adapts its energy-neutrality with changing ω (Figure 7. When energy-neutrality has low priority ($\omega = (\omega_{sense}, \omega_{tx}, 0.1)$), the downtimes (bottom figure) are much higher than when $\omega = (\omega_{sense}, \omega_{tx}, 0.7)$ (blue and orange vs green and red). Secondly, we observe how the agent trades off between *sense-utility* and *tx-utility* when $\omega_{ENP} = 0.1$. In the top and middle figures, the blue lines corresponds to a higher sensing priority than transmission ($\omega_{sense} > \omega_{tx}$). As a result, the blue line scores higher on the *sense-utility* (top) and lower on the *tx-utility* (middle) than the orange line ($\omega_{sense} < \omega_{tx}$). This clearly shows the tradeoff behavior. A similar observation can be made for when $\omega_{ENP} = 0.7$ (green and red lines).

We now turn our attention to the increased learning costs of our MORL solutions. Figure 8 shows the learning costs for *sense* and *enp*, which is competitive with previous SORL methods [15]. The other agents, *mul_scalar* and *add_scalar*, also have similar learning costs (not shown). What is interesting is that *morl_multi* also has similar learning costs in spite of having to learn three different greedy actor-critics within the same training period as previous SORL methods. This is primarily due to off-policy corrections in Algorithm 2 and auto-regulation among the greedy agents in Algorithm 1.*Thus, our framework can learn to optimize between three objectives successfully in dual-task EHWSNs, without a drastic increase in learning costs.*

VIII. Conclusion

Modern EHWSNs are complex SoCs that need to optimize multiple objectives to maximize their utility and maintain ENO. Traditional approaches cannot handle multiple objectives and tradeoff between them at runtime. Our proposed MORL framework can learn policies and perform tradeoffs within feasible learning costs. Simulations on single-task and dual-task EHWSNs show that our framework results in near-optimal policies that can dynamically tradeoff between objectives at runtime. By compromising between compute-intensive MORL methods and elitist MC/EA methods, our proposed MORL algorithms are a feasible solution for resource-constrained EHWSN.

Acknowledgment

This work was partially supported by JST CREST Grant Number JPMJCR20F2 and JSPS KAKENHI Grant Number 18J20946.

References

[1] D. Ma, G. Lan, M. Hassan, W. Hu, and S. K. Das, "Sensing, computing, and communications for energy harvesting iots: A survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1222–1250, 2019.

[2] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 4, p. 32, 2007.

[3] C. M. Vigorito, D. Ganesan, and A. G. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks," in *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. IEEE, 2007, pp. 21–30.

[4] K. Geissdoerfer, R. Jurdak, B. Kusy, and M. Zimmerling, "Getting more out of energy-harvesting systems: energy management under time-varying utility with preact," in *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*. IEEE, 2019, pp. 109–120.

[5] R. C. Hsu, C.-T. Liu, and H.-L. Wang, "A reinforcement learning-based tod provisioning dynamic power management for sustainable operation of energy harvesting wireless sensor node," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2, pp. 181–191, 2014.

[6] S. Shresthamali, M. Kondo, and H. Nakamura, "Adaptive power management in solar energy harvesting sensor node using reinforcement learning," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 181, 2017.

[7] A. Ortiz, H. Al-Shatri, X. Li, T. Weber, and A. Klein, "Reinforcement learning for energy harvesting point-to-point communications," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[8] F. A. Aoudia, M. Gautier, and O. Berder, "Rlman: an energy manager based on reinforcement learning for energy harvesting wireless sensor networks," *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 2, pp. 408–417, 2018.

[9] F. Fraternali, B. Balaji, Y. Agarwal, and R. K. Gupta, "Aces–automatic configuration of energy harvesting sensors with reinforcement learning," *arXiv preprint arXiv:1909.01968*, 2019.

[10] Z. Fei, B. Li, S. Yang, C. Xing, H. Chen, and L. Hanzo, "A survey of multi-objective optimization in wireless sensor networks: Metrics, algorithms, and open problems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 550–586, 2016.

[11] P. V. R. Ferreira, R. Paffenroth, A. M. Wyglinski, T. M. Hackett, S. G. Bilén, R. C. Reinhart, and D. J. Mortensen, "Multiobjective reinforcement learning for cognitive satellite communications using deep neural network ensembles," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 5, pp. 1030–1041, 2018.

[12] Y. Rioual, Y. Le Moullec, J. Laurent, M. I. Khan, and J.-P. Diguet, "Reward function evaluation in a reinforcement learning approach for energy management," in *2018 16th Biennial Baltic Electronics Conference (BEC)*. IEEE, 2018, pp. 1–4.

[13] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 636–14 647.

[14] M. Pirotta, S. Parisi, and M. Restelli, "Multi-objective reinforcement learning with continuous pareto frontier approximation," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[15] S. Shresthamali, M. Kondo, and H. Nakamura, "Power management of wireless sensor nodes with coordinated distributed reinforcement learning," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 2019, pp. 638–647.

[16] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," *arXiv preprint arXiv:2007.10319*, 2020.

[17] F. Restuccia and T. Melodia, "Deepwierl: Bringing deep reinforcement learning to the internet of self-adaptive things," pp. 844–853, 2020.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[19] Japan Meteorological Agency, "Japan meteorological agency," 2019, [Online; accessed 6-July-2019]. [Online]. Available: http://www.jma.go.jp/jma/index.html

[20] "Waspmote: The sensor platform to develop iot projects — libelium," https://www.libelium.com/iot-products/waspmote/, (Accessed on 01/22/2021).