Learning Optimal Edge Processing with Offloading and Energy Harvesting

Andrea Fox[*], Francesco De Pellegrini[*], Eitan Altman[◇]

[*]Avignon University, France
[◇]INRIA, Sophia Antipolis, France

TECoSA Seminar

02 November 2023

Summary

Motivation

- **Mobile computing:** integrate AI-intensive processing
  - smart city services
  - virtual/augmented reality applications
  - ...
- **IoT data processing:** periodic data updates
- **Critical:** AI apps energy consumption
- **Mitigation:** offloading + energy harvesting

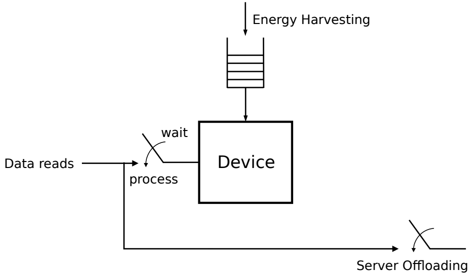
Main contributions

- **Markov model**: data freshness versus battery depletion[1]
  - processing (-)
  - energy harvesting (+)
  - offloading (+)
- **Optimal policy:** threshold structure
- **Learning:**
  - optimal policy of the single device
  - optimization of the polling probability in multi-device context

[1] A. Fox, F. De Pellegrini and E. Altman, "Learning Optimal Edge Processing with Offloading and Energy Harvesting", in proc. of ACM MSWIM 2023.

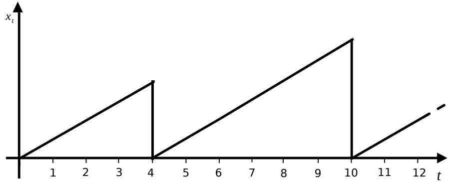
Summary

Single device model



- **dynamics:** discrete time model
- **device**: reads data batches and processes them
- **server**: polls randomly the device with fixed probability

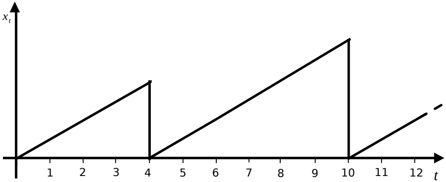
Age of Information (AoI)



AoI: metric that captures the **freshness of the information** processed

- time elapsed since last data batch processing

Age of Information (AoI)

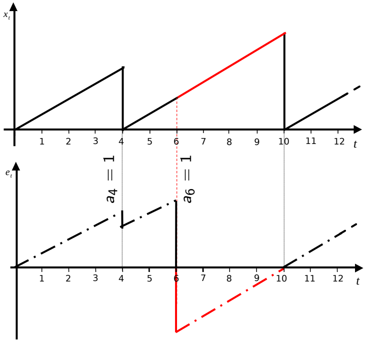


AoI: metric that captures the **freshness of the information** processed

- time elapsed since last data batch processing

Immediate reward function: AoI-decreasing
Goal: find the policy that maximizes the long-term reward

Battery level



- **harvesting rate** $\{H_t\}_{t\in\mathbb{N}}$**:** battery recharge at each timestep
- **processing cost** $\{C_t\}_{t\in\mathbb{N}}$**:** energy spent for local processing
- **processing failure:** if processing cost is higher than energy available, we wait until we have a sufficient amount of energy

Semi Markov decision process (SMDP)

Semi Markov Decision Process:

- generalization of MDPs
- considers random variable χ: time between subsequent transitions

SMDP: states and actions

- **State space:** $\mathcal{S} = \{(x, e, z)\}$ where
  - $x \in \{1, \ldots, M\}$: current age of information
  - $e \in \{0, \ldots, B\}$: current level of energy of the battery
  - $z \in \{0, 1\}$: indicates if the server has polled the device considered
- **Action space:** process (1) or wait (0)
  - if $z = 0$ then $\mathcal{A}(s) = \{0, 1\}$
  - if $z = 1$ then $\mathcal{A}(s) = \{1\}$

SMDP: possible transitions

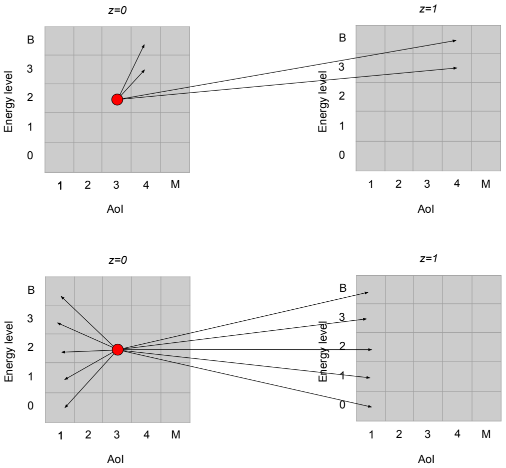


Figure: Possible transitions when $z = 0$, $a = 0$ (above) and $a = 1$ (below)

SMDP: possible transitions

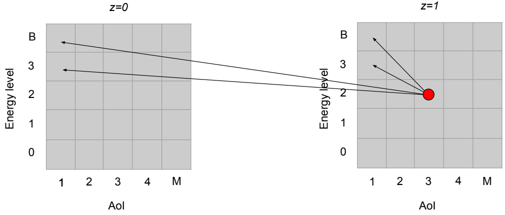


Figure: Possible transitions when $z = 1$ and $a = 1$

SMDP: sojourn time χ

Device not polled ($z = 0$):

- Wait action ($a = 0$): $\chi(s) = 1$, only the waiting timeslot
- Process locally ($a = 1$) : $\chi(s) \geq 1$, consecutive energy harvesting steps to conclude processing

Device polled ($z = 1$):

- Process remotely ($a = 1$): $\chi(s) = 1 + \delta$, processing timeslot + roundtrip delay

SMDP: reward function

Reward components:

- *utility function:* u decreasing function
  - if $x > M$, then $u(x) = u(M)$
- *disadvantage function:* d decreasing function defined only for arguments smaller than 0

SMDP: reward function

Reward components:

- *utility function:* u decreasing function
  - if $x > M$, then $u(x) = u(M)$
- *disadvantage function:* d decreasing function defined only for arguments smaller than 0

Device not polled ($z = 0$):

- Wait action ($a = 0$): $R((x, e, z = 0), a = 0) = u(x)$
- Process locally ($a = 1$): $R((x, e, z = 0), a = 1) = u(x) - d(e + h - c)$

Device polled ($z = 1$):

- Process remotely ($a = 1$): $R((x, e, z = 1), a = 1) = u(x + \delta)$

SMDP: reward function

Reward components:

- *utility function:* u decreasing function
  - if $x > M$, then $u(x) = u(M)$
- *disadvantage function:* d decreasing function defined only for arguments smaller than 0

Device not polled ($z = 0$):

- Wait action ($a = 0$): $R((x, e, z = 0), a = 0) = u(x)$
- Process locally ($a = 1$): $R((x, e, z = 0), a = 1) = u(x) - d(e + h - c)$

Device polled ($z = 1$):

- Process remotely ($a = 1$): $R((x, e, z = 1), a = 1) = u(x + \delta)$

SMDP: reward function

Reward components:

- *utility function:* u decreasing function
  - if $x > M$, then $u(x) = u(M)$
- *disadvantage function:* d decreasing function defined only for arguments smaller than 0

Device not polled ($z = 0$):

- Wait action ($a = 0$): $R((x, e, z = 0), a = 0) = u(x)$
- Process locally ($a = 1$): $R((x, e, z = 0), a = 1) = u(x) - d(e + h - c)$

Device polled ($z = 1$):

- Process remotely ($a = 1$): $R((x, e, z = 1), a = 1) = u(x + \delta)$

Value function

Bellman equation:

$$v(x,e,0) = \max\left\{u(x) - p_E(e' \mid e,1)d(e') + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,1)v(s'),\right.$$
$$\left. u(x) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,0)v(s')\right\}$$

$$v(x,e,1) = u([x+\delta]_M) + \sum_{h=1}^{\infty} p_H(h)\mathbb{E}_z[v(1,[e+h]_B, z)]$$

Value function

Bellman equation:

$$v(x,e,0) = \max\left\{ u(x) - p_E(e' \mid e,1)d(e') + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,1)v(s'), \right.$$
$$\left. u(x) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,0)v(s') \right\}$$

$$v(x,e,1) = u([x+\delta]_M) + \sum_{h=1}^{\infty} p_H(h)\mathbb{E}_z[v(1,[e+h]_B\,,z)]$$

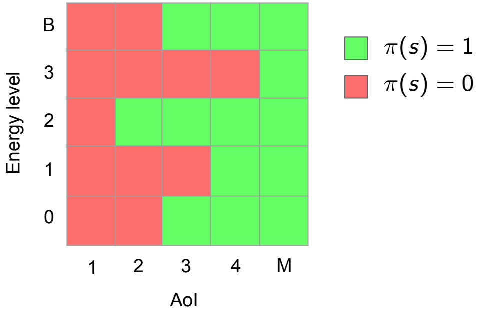
Monotony of optimal value function:

1. $v_*(x,e,z)$ is non increasing in x
2. $v_*(x,e,z)$ is non decreasing in e

Structure of the optimal policy

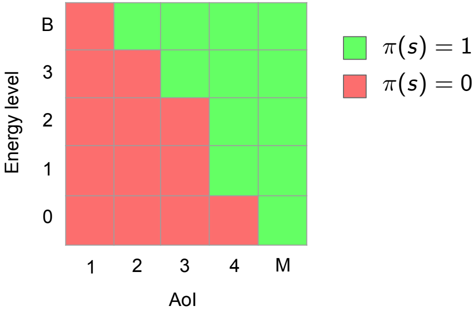
Theorem (AoI thresholds)

Let π be an optimal deterministic policy: for each energy level occupation $0 \leq e \leq B$ there exists an integer threshold $0 \leq T(e) \leq M$ such that $\pi(e, x, 0) = 1$ if and only if $x \geq T(e)$.



Structure of the optimal policy

In our experiments we have further observed that the structure of the policy is a stairway, meaning that $T(e) \geq T(e+1)\ \forall e$



Partial order on states

Natural Partial order: we can sort states as

$$(x, e+1, z) \succeq (x, e, z)$$
$$(x-1, e, z) \succeq (x, e, z)$$

***Q*-function**: $Q(s, a) := \mathbb{E}_{\pi}\left[G_t | s_0 = s, a_0 = a\right]$

Corollary

Fixed $a \in \{0, 1\}$, *then for* $s = (x, e, z) \in \mathcal{S}$

i. $q_*((x, e, z), a) \leq q_*((x, [e+1]_B, z), a)$

ii. $q_*(([x+1]_M, e, z), a) \leq q_*((x, e, z), a)$

Here: $[y]_A := \max\{0, \min\{y, A\}\}$

Summary

Q-learning

- **Q-function:**

$$Q(s,a) = \mathbb{E}_{a\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) \mid s_0 = s, a_0 = a\right]$$

- **Learning rule:**

$$Q(s_t, a_t) \leftarrow (1-\alpha_t)Q(s_t, a_t) + \alpha_t \left(R_{t+1} + \gamma \max_a Q(s_{t+1}, a)\right)$$

- $\boldsymbol{\alpha_t}$**:** decaying learning rate

Ordered Q-learning

Main idea: use the partial order on states and preserve the structure of the value function

The new learning rule is:

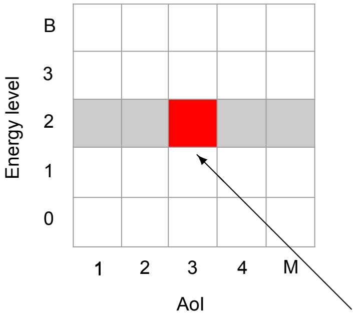
$$\begin{cases} \overline{Q}(s_t, a_t) = (1-\alpha_t) Q(s_t, a_t) + \alpha_t \left(r_t + \gamma \max_a Q(s_{t+1}, a)\right) \\ Q(s', a_t) = \Pi_{s_t}\left(\overline{Q}(s', a_t)\right) \quad \forall s' \in \mathcal{S} \end{cases}$$

Ordered Q-learning: convergence

Theorem

Consider the Ordered Q-learning algorithm and let $\gamma < 1$. Let q_* be monotone, i.e., if $s_1 \leq s_2$ according to some order on the states, then $q_*(s_1, a) \leq q_*(s_2, a)$. Then $Q_t(s, a)$ converges to $q_*(s, a)$ w.p.1. for every state $s \in \mathcal{S}$ and for every action $a \in \mathcal{A}(s)$.

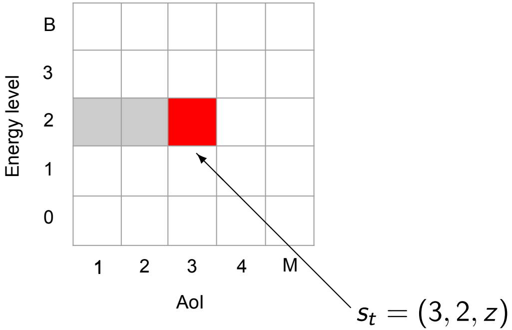
Threshold Q-learning



$s_t = (3, 2, z)$

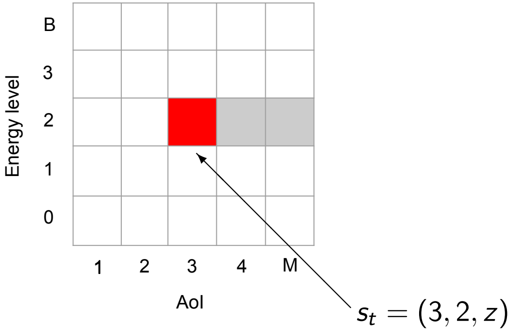
Partial order considered: $(x-1, e, z) \succeq (x, e, z)$

Threshold Q-learning (larger states)



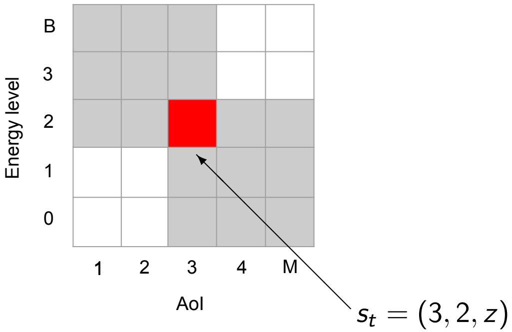
$$Q\left(s', a_t\right) = \Pi_{s_t}\left(Q(s', a_t)\right) = \max\{\overline{Q}(s_t, a_t), Q(s', a_t)\} \quad \forall s' \succeq s_t$$

Threshold Q-learning (smaller states)



$$Q\left(s', a_t\right) = \Pi_{s_t}\left(Q(s', a_t)\right) = \min\{\overline{Q}(s_t, a_t), Q(s', a_t)\} \quad \forall s' \preceq s_t$$

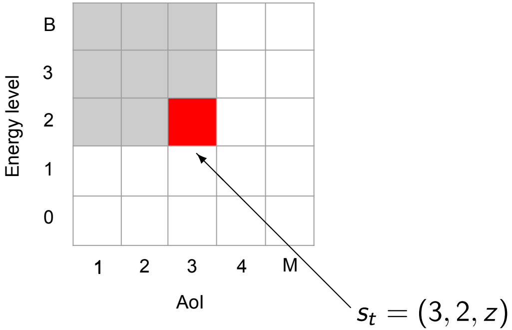
Stairway Q-learning



Partial order considered:

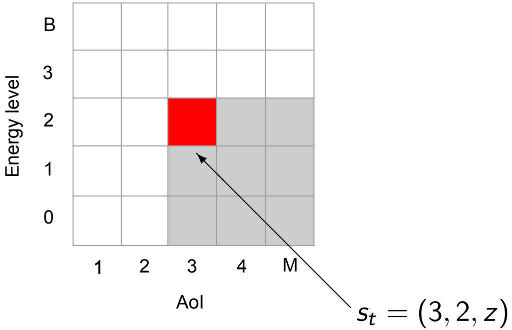
$$\begin{cases} (x, e+1, z) & \succeq (x, e, z) \\ (x-1, e, z) & \succeq (x, e, z) \end{cases}$$

Stairway Q-learning (larger states)



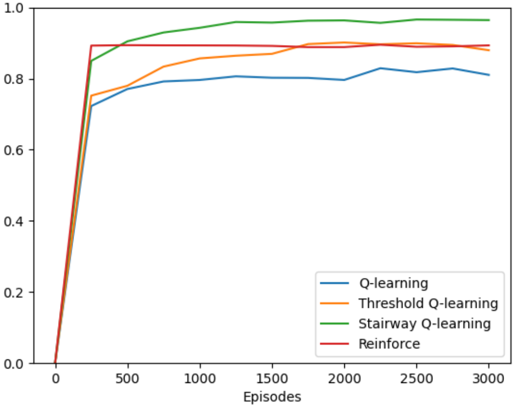
$$Q\left(s', a_t\right) = \Pi_{s_t}\left(Q(s', a_t)\right) = \max\{\overline{Q}(s_t, a_t), Q(s', a_t)\} \quad \forall s' \succeq s_t$$

Stairway Q-learning (smaller states)



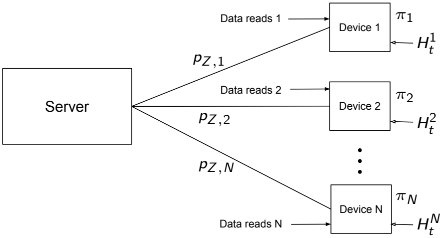
$$Q\left(s', a_t\right) = \Pi_{s_t}\left(Q(s', a_t)\right) = \min\{\overline{Q}(s_t, a_t), Q(s', a_t)\} \quad \forall s' \preceq s_t$$

Numerical results



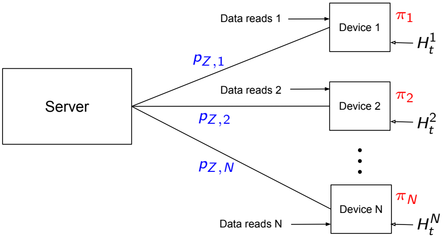
Summary

Multi device model



- N **devices:** each server-polled with probability $p_{Z,k}$
- **device:** takes decision regardless state of other devices

Multi device model



- N **devices:** each server-polled with probability $p_{Z,k}$
- **device:** takes decision regardless state of other devices
- **Goal:** optimize both $p_{Z,k}$ (i.e. a polling distribution) and π_k $\forall k$

Discounted reward function

For each device k: compute the discounted reward function

$$R_{\gamma_k,\pi}(p_{Z,k}) = \mathbb{E}_{s_0\sim\rho,a\sim\pi}\left[\sum_{t=0}^{\infty}\gamma_k^t r(s_t,a_t) \mid S_0 = s_0\right]$$

Properties of $\boldsymbol{R_{\gamma_k,\pi}(p_{Z,k})}$: (for fixed policy π)

- differentiable
- concave (proved only for a simple case)

Optimization algorithm

Goal: maximize the objective function wrt $p_Z = (p_{Z,1}, \ldots, p_{Z,N})$

$$R_\gamma(p_Z) = \sum_{k=1}^{N} R_{\gamma_k, \pi_k^\star}(p_{Z,k})$$

Optimization algorithm

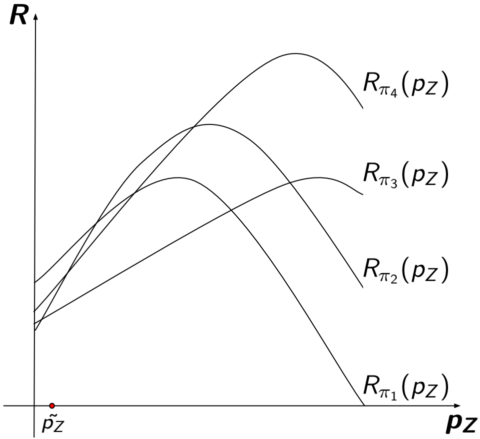
Goal: maximize the objective function wrt $p_Z = (p_{Z,1}, \dots, p_{Z,N})$

$$R_\gamma(p_Z) = \sum_{k=1}^{N} R_{\gamma_k,\pi_k^\star}(p_{Z,k})$$

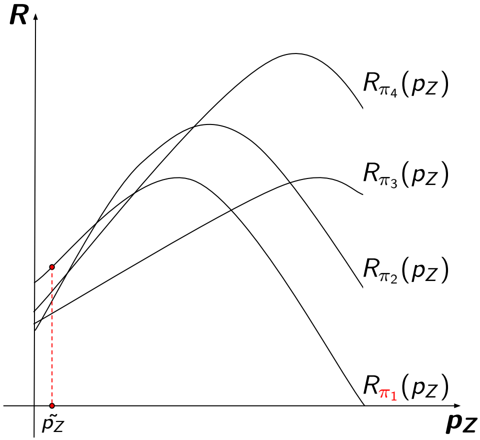
Issues:

- the function to maximize is neither concave nor differentiable
- finding the optimal policy requires RL (slow operation)

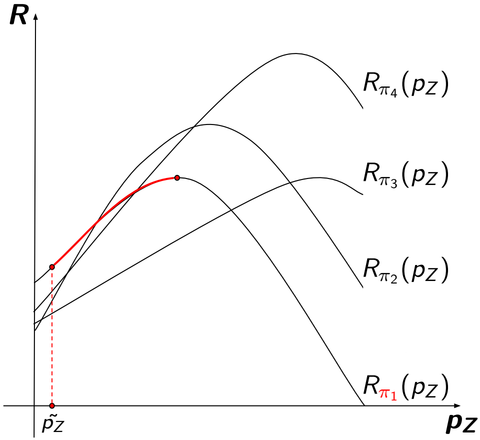
APPI algorithm



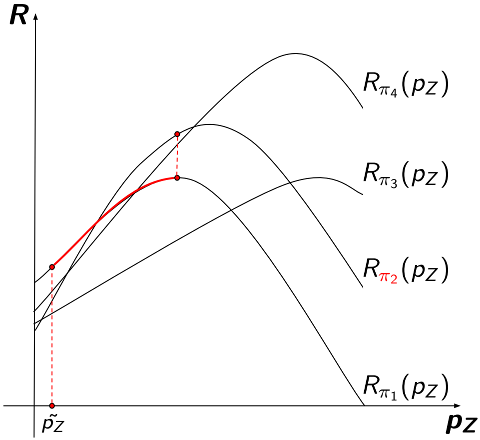
APPI algorithm



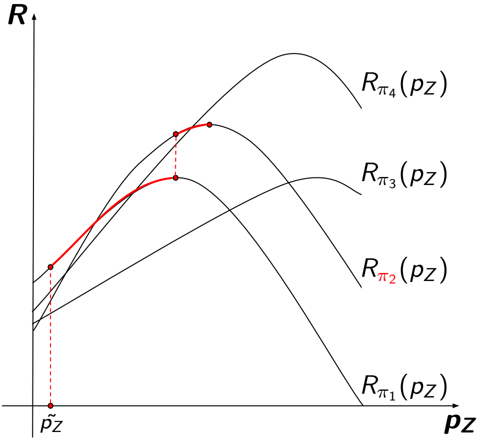
APPI algorithm



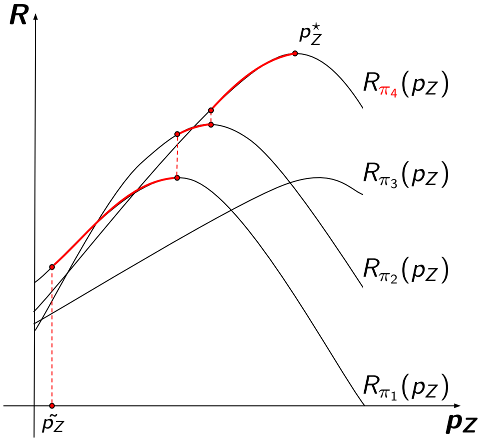
APPI algorithm



APPI algorithm



APPI algorithm



APPI algorithm

- **APPI:** alternates between **two stages**:
  1. **Policy learning:** we fix the polling probability and learn the optimal policy of each device using Stairway Q-learning
  2. **Polling distribution optimization:** SPSA algorithm to compute the approximate gradient of the total reward function $R_\pi(p_{Z,k})$

APPI algorithm

- **APPI:** alternates between **two stages**:
  1. **Policy learning:** we fix the polling probability and learn the optimal policy of each device using Stairway Q-learning
  2. **Polling distribution optimization:** SPSA algorithm to compute the approximate gradient of the total reward function $R_\pi(p_{Z,k})$
- **Convergence:** guaranteed only to a local maximum

Numerical results

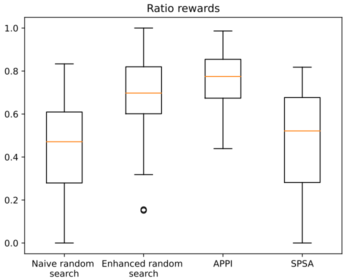


Figure: Performance ratio.

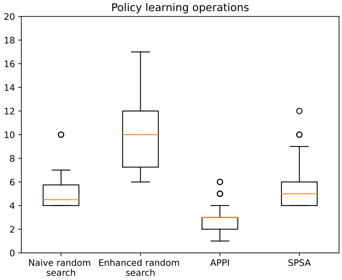


Figure: Number of policy-learning operations.

Numerical results

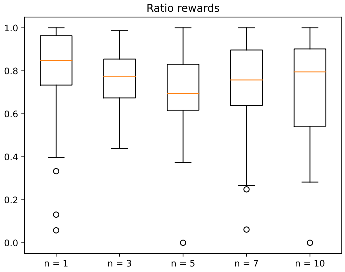


Figure: Performance ratio.

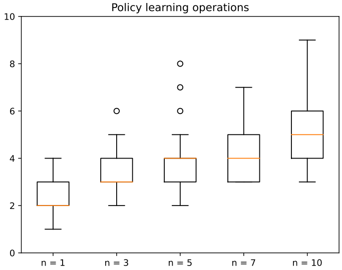


Figure: Number of policy-learning operations.

Conclusions:

- model minimizing **AoI** versus **battery consumption**, while considering **offloading** and **energy harvesting**
- **structure** of the optimal policy
- **model-based** learning method
- **offloading optimization** (state-agnostic case)

Conclusions

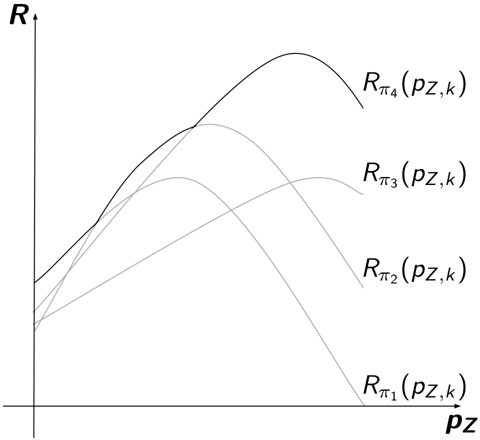
Conclusions:

- model minimizing **AoI** versus **battery consumption**, while considering **offloading** and **energy harvesting**
- **structure** of the optimal policy
- **model-based** learning method
- **offloading optimization** (state-agnostic case)

Future works:

- state-aware offloading (issue of stability)
- correlation among devices states

Discounted reward function



APPI algorithm: learning the optimal policy

- fix the polling probability $p_{Z,k}$ for each device
- we use the learning methods (Stairway Q-learning) for each device independently
- bottleneck of the method

APPI algorithm: optimizing the polling distribution

We use SPSA algorithm:

- computes the approximate gradient of the total reward function, considering positive and negative increments in the argument of the function
- all the devices are independent to each other
- $$(\hat{g}_n)_k = \frac{\hat{R}_{\gamma_k,\pi_k}\left(p_{Z,k} + c_n\left(\Delta_n\right)_k\right) - \hat{R}_{\gamma_k,\pi_k}\left(p_{Z,k} - c_n\left(\Delta_n\right)_k\right)}{2c_n(\Delta_n)_k}$$
- for fixed policies for each device, it converges to the optimal polling distribution

APPI algorithm

Algorithm 1 Alternating Polling and Policy Improvement (APPI)

Require: $\epsilon > 0$, episode length T

1: initial polling probability $p_{Z,\text{new}}$
2: **while do** $|p_{Z,\text{old}} - p_{Z,\text{new}}| > \epsilon$
3: $p_{Z,\text{old}} \leftarrow p_{Z,\text{new}}$
4: **Optimal policy learning:** find optimal policy $\pi_k^*(p_{Z,\text{old}})$ $\forall k$ for episode length T
5: **Polling optimization:** find $p_{Z,\text{new}} \geq 0$ such that

$$\begin{cases} p_{Z,\text{new}} = \arg\max_{p_Z} \sum_k R_{\gamma_k, \pi_k^*(p_{Z,k,\text{old}})}(p_Z) \\ \sum_k p_{Z,\text{new}} = 1 \end{cases}$$

6: **end while**
7: **return** $p_{Z,\text{new}}$

Random search algorithms

Value iteration algorithm

Algorithm 2 Value iteration algorithm

Require: $\epsilon \geq 0$
Require: $\gamma \in (0, 1)$
Require: $v_0 \in \mathcal{V}$

$n \leftarrow 0$
while $\|v_{n+1} - v_n\|_\infty > \frac{\epsilon(1-\gamma)}{2\gamma}$ **do**
  for each $s \in \mathcal{S}$ **do**
    $v_{n+1}(s) \leftarrow \max_a\{r(s, a) + \gamma \sum_{j \in \mathcal{S}} p_{s,a}(j) v_n(j)\}$
  end for
end while
for each $s \in \mathcal{S}$ **do**
  $\pi^\star(s) \in \text{argmax}_a\{r(s, a) + \gamma \sum_{j \in \mathcal{S}} p_{s,a}(j) v_{n+1}(j)\}$
end for

Transition probabilities when $a = 0$

$$p(s'|s,0) = \begin{cases} p_Z(z' \mid 0) p_H(h) & s = (x, e, 1), \\ & s' = ([x+1]_M, e+h, z'), \\ & e + h < B \\ p_Z(z' \mid 0) \sum_{h=B-e}^{\infty} p_H(h) & s = (x, e, 1), \\ & s' = ([x+1]_M, B, z') \\ 0 & \text{otherwise} \end{cases}$$

Transition probabilities when $a = 1$

$$p(s' \mid s, 1) = \begin{cases} p_Z(z' \mid 0)\Big(\sum_{r=0}^{\infty} \Gamma_{1,r} p_C(e + r - e') + \\ + \sum_{k=2}^{\infty} \sum_{r=k-1}^{\infty} \Gamma_{k-1,r} \sum_{t=e+r+1}^{\infty} p_C(c) p_H(e' + c - r - e)\Big) & s = (x, e, 0), \; s' = (1, e', z'), \; 0 \le e' < B \\ p_Z(z \mid 0) \sum_{c=1}^{\infty} p_C(c) \sum_{h=B-e+c}^{\infty} p_H(h) & s = (x, e, 0), \; s' = (1, B, z') \\ p_Z(z' \mid 1) p_H(h) & s = (x, e, 1), \; s' = (1, e + h, z'), \; e' < B \\ p_Z(z' \mid 1) \sum_{h=B-e}^{\infty} p_H(h) & s = (x, e, 1), \; s' = (1, B, z') \\ 0 & \text{otherwise} \end{cases}$$

Average execution time when finishing in e'

$$\tau(e') = p_Z(1)\sum_{e=0}^{B} p_E(e)p_H(e'-e)+$$
$$+(1-p_Z(0))\left(\sum_{e=0}^{B} p_E(e)\sum_{r=1}^{\infty}\Gamma_{1,r}p_C(e+h_1-e')+\right.$$
$$\left.+\sum_{k=2}^{\infty} k\sum_{e=0}^{B} p_E(e)\sum_{r=1}^{\infty}\Gamma_{k-1,r}\sum_{c=e+\sum_{j=1}^{k-1}h_j+1}^{\infty} p_C(c)p_H(e'+c-\sum_{j=1}^{k-1}h_j-e)\right)$$

where

$$\Gamma_{k,r} = \mathbb{P}\left(\sum_{j=1}^{k} h_j = r\right), \quad h_j \text{ i.i.d.}, h_j \sim p_H$$