Deep reinforcement learning for weakly coupled MDP's with continuous actions

Francisco Robledo, Urtzi Ayesta, Konstantin Avrachenkov

Deep reinforcement learning for weakly coupled MDP's with continuous actions

Francisco Robledo[1][0000−0003−1040−1513], Urtzi Ayesta[2][0000−0003−1761−2313], and Konstantin Avrachenkov[3][0000−0002−8124−8272]

[1] UPV/EHU, Univ. of the Basque Country, 20018 Donostia, Spain
UPPA, Université de Pau et des Pays de l'Adour, 64000 Pau, France
frrobledo96@gmail.com
[2] IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, Toulouse, France
UPV/EHU, Univ. of the Basque Country, 20018 Donostia, Spain
IKERBASQUE - Basque Foundation for Science, 48011 Bilbao, Spain
urtzi.ayesta@irit.fr
[3] INRIA Sophia Antipolis, France k.avrachenkov@inria.fr

Abstract. This paper introduces the Lagrange Policy for Continuous Actions (LPCA), a reinforcement learning algorithm specifically designed for weakly coupled MDP problems with continuous action spaces. LPCA addresses the challenge of resource constraints dependent on continuous actions by introducing a Lagrange relaxation of the weakly coupled MDP problem within a neural network framework for Q-value computation. This approach effectively decouples the MDP, enabling efficient policy learning in resource-constrained environments. We present two variations of LPCA: LPCA-DE, which utilizes differential evolution for global optimization, and LPCA-Greedy, a method that incrementally and greadily selects actions based on Q-value gradients. Comparative analysis against other state-of-the-art techniques across various settings highlight LPCA's robustness and efficiency in managing resource allocation while maximizing rewards.

Keywords: Reinforcement Learning · Weakly Coupled MDP · Continuous Actions · Lagrange Policy · Neural Networks · Differential Evolution · Resource Allocation · Policy Optimization.

1 Introduction

The exploration of optimal decision-making under uncertainty is a fundamental problem [17], with significant implications in diverse fields such as telecommunications, finance, robotics, and healthcare. At the heart of this exploration lies the restless multi-armed bandit (RMAB) problem, an extension of the classical multi-armed bandit framework [6] to scenarios where arms evolve independently of the player's actions. Introduced by [21], the RMAB problem highlights the challenge of allocating limited resources among competing projects or processes in a state of continuous change. Recently, many studies have focused on neural network approximation in restless bandit problems, such as the works of [1], [14], and [11], which use deep reinforcement learning to approximate the Whittle indices used in their heuristics.

One can generalize the restless bandits to weakly coupled MDPs, where the independent MDPs are coupled only through a constraint on the action and actions can belong to complex spaces. These problems present substantial complexity due to constraints of the actions and common resources. A key advancement in addressing such complex problems came with the introduction of Lagrangian Decomposition methods, as explored by [7]. The approach of [7] proposes a Lagrangian decomposition approach for solving the weakly coupled dynamic optimization problem, which yields upper bounds as well as heuristic solutions. Works by [16] and [10] have introduced methods for navigating these complex decision spaces, employing Gaussian processes and simulation-based algorithms, respectively, to tackle the multi-action challenges.

Other studies in weakly coupled MDPs include the work of [20], which addresses the challenges of online learning in this specific MDP setting and presents an algorithm with a tight $O(\sqrt{t})$ regret and constraint violations simultaneously. Additionally, [5] introduces the LP-update policy, which generalizes the classical restless bandit problems and demonstrates asymptotic optimality at various rates depending on problem characteristics.

Significant advances in deep reinforcement learning include the development of Deep Deterministic Policy Gradient (DDPG) [9] and Twin Delayed DDPG (TD3) [4], algorithms that have significantly advanced complex control tasks by solving MDPs with continuous actions. Building on the capabilities of these frameworks, the OptLayer architecture was introduced [12], specifically designed to generate safe, constraint-compliant actions. OptLayer integrates an additional layer that solves a constraint optimization problem applicable to both DDPG and TD3 architectures. This extension ensures that the actions taken by the learning models adhere to predefined constraints. [8] explores the online learning landscape for discrete multi-action RMABs and presents a Q-learning Lagrange policy algorithm tailored for restless multi-armed bandits with multiple discrete actions. Similarly, [15] uses this Lagrangian decomposition to train separate subagents for each individual MDP problem, and a general network to combine these results, also in the context of discrete multi-action RMABs.

In this work, we introduce the Lagrange Policy for Continuous Actions (LPCA) algorithm, a reinforcement learning algorithm specifically designed for weakly coupled MDP problems with continuous action spaces. To the best of our knowledge, this is the first paper proposing an algorithm to solve weakly coupled MDPs with continuous actions. LPCA integrates a neural network-based framework to study weakly coupled MDP using the Lagrange relaxation introduced in [7] to decouple the projects of the MDP, being able to study their dynamics independently of one another and effectively balancing resource constraints and individual project decisions. Continuous actions allow for a more accurate representation of real-world scenarios, such as adjusting resource levels or control parameters, without the limitations of discretization. This flexibility enhances the algorithm's ability to optimize performance by better managing trade-offs between competing processes, ultimately leading to more robust and efficient policy learning.

2 Problem Formulation

In our approach to the weakly coupled MDPs with continuous actions, we consider an environment consisting of N projects, each characterized by its unique state, action, and the resulting reward. Specifically, the state of the system is given by $\mathbf{s} = (s_1, ..., s_N) \in \mathbf{S}$, where each project is represented as s_i, an element from the finite state space S_i, $i = 1, ..., N$. Correspondingly, the actions taken in each project are denoted as elements a_i belonging to the compact action space A_i, and the complete system action is denoted with bold font $\mathbf{a} = (a_1, ..., a_N) \in \mathbf{A}$. The rewards obtained from these actions are encapsulated as elements r_i in the reward vector $\mathbf{r}$. The cost associated with each action a_i is expressed as $c(a_i)$, and the cumulative cost for all actions is given by $C(\mathbf{a}) = \sum_i c(a_i)$.

The system dynamics are governed by a transition probability kernel $T : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to [0, 1]$, which specifies the probabilities of transitioning to new states given particular state and action vector. Given the values of actions, T has a product form. A discount factor $\gamma \in (0, 1)$ is used to balance immediate and future rewards.

The long-term discounted reward can be expressed through the Bellman value function $V(\mathbf{s})$, which is the expected sum of discounted rewards accumulated over time, starting from the state $\mathbf{s}$ and satisfying the Bellman dynamic programming equation:

$$V(\mathbf{s}) = \max_{\mathbf{a}\in\mathbf{A},\, C(\mathbf{a})=B} \left[\sum_{i=1}^{N} r_i(s_i, a_i) + \gamma \mathbb{E}[V(\mathbf{s}') \mid \mathbf{s}, \mathbf{a}] \right]. \tag{1}$$

The complexity of the problem comes primarily from the constraint imposed on the actions, which are dictated by a common pool of resources. Specifically, each project must select a continuous action $a_i \in [0, a_i^{\max}]$ whose activation cost, represented by the total cost $C(\mathbf{a})$, directly consumes a predefined total pool of available resources B. This shared resource pool constraint means that actions across projects are inherently coupled, which significantly increases the complexity of the decision space as the number of projects increases. The exponential growth in decision space complexity due to this coupling underscores the challenge of resource allocation and emphasizes the need for efficient use of the shared resource pool [2].

To manage this complexity, we can relax the value function using a Lagrange multiplier λ. This transforms the original problem into a Lagrangian form:

$$J(\mathbf{s}, \lambda) = \max_{\mathbf{a}\in\mathbf{A}} \left[\sum_{i=1}^{N} r_i(s_i, a_i) + \lambda \left(B - \sum_{i=1}^{N} c(a_i) \right) + \gamma \mathbb{E}[J(\mathbf{s}', \lambda) \mid \mathbf{s}, \mathbf{a}] \right]. \tag{2}$$

Here, λ is the Lagrange multiplier associated with the resource constraint B. By adjusting λ, we effectively balance the immediate cost of actions against their long-term rewards, allowing for a decoupling of the projects' decisions. If we assume the additive structure of the value function with respect to the projects of the weakly coupled MDP, the equation (2) can be rewritten as:

$$J(\mathbf{s}, \lambda) = \frac{\lambda B}{1 - \gamma} + \sum_{i=1}^{N} \max_{a_i \in A_i} Q_i(s_i, a_i, \lambda), \tag{3}$$

where

$$Q_i(s_i, a_i, \lambda) = r_i(s_i, a_i) - \lambda c(a_i) + \gamma \sum_{s_i'} T(s_i, a_i, s_i') \max_{a_i' \in A_i} Q_i(s_i', a_i', \lambda). \quad (4)$$

In this decoupled framework, the Lagrange multiplier λ is instrumental in determining the optimal policy for each project. Under the budget constraint B, λ acts as a trade off parameter by introducing a penalty term $\lambda c(a_i)$ for the actions taken. A higher λ parameter places more emphasis on minimizing the cost (i.e., staying within the resource limit B), while a lower λ value shifts the focus towards maximizing rewards with less emphasis on the cost implementations. As λ rises, the preferred policy for each project will increasingly favor actions that offer the highest "value-to-cost" ratio. Thus, the function (3) is a measure of the total expected reward, adjusted for the cost of the actions taken under that policy. To balance the expected rewards with the cost of actions, we need to find λ^* such that

$$\lambda^*(\mathbf{s}) = \arg\min_{\lambda} J(\mathbf{s}, \lambda). \quad (5)$$

This term is defined as the best trade-off between maximizing rewards and minimizing the cost of actions. It is at this point that the policy aligns with the time-averaged resource constraints, ensuring that the actions selected are not only rewarding but also resource-efficient.

Then, in a continuous action framework, at each time step t we aim to solve the following Knapsack-like optimization problem:

$$\max_{\mathbf{a} \in \mathbf{A}} \quad \sum_{i=1}^{N} Q_i(s_i(t), a_i, \lambda^*(s_i)) \qquad s.t. \quad \sum_{i=1}^{N} c(a_i) = B. \quad (6)$$

In the LPCA algorithm, described in detail next, we interpolate the curve of the Q-values $Q(s, a, \lambda)$ as functions of the Lagrange multiplier λ through a neural network. This curve is a convex function with respect to λ [7], making the minimization of (3) a simple one-dimensional convex optimization problem once the neural network is trained. For the optimization (6) we explore two approaches as outlined in Sections 3.1 and 3.2.

3 LPCA Algorithm

In numerous practical applications, the model parameters, particularly expected rewards and transition probabilities, are often unknown or inaccessible. To address this, traditional reinforcement learning methods have been employed to learn those parameters [17]. However, a significant challenge arises in environments where the projects of the MDP are coupled. In these cases, the complexity of solving the problem increases exponentially with the number of projects. To address this challenge, we introduce LPCA, a reinforcement learning algorithm that extends Q-learning by incorporating neural networks for approximating Q-values for constrained continuous actions. This section details the operation and implementation of LPCA.

The core methodology of the LPCA algorithm involves a two-timescale process centered around learning and optimization. Initially, LPCA focuses on training a neural

Algorithm 1 LPCA Training Process

Require: Environment, N_{iter}, Update frequency N, Batch size M, Policy method
Ensure: Train LPCA Model, Update Policy Dictionary
1: Initialize Q-value neural network, policy dictionary, experience memory
2: **for** iteration $= 1$ **to** N_{iter} **do**
3: Select and execute action $\mathbf{a}$, store $(\mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}'$, done)
4: **if** memory $\geq M$ **then**
5: Update Q-values with mini-batch of M (Algorithm 2)
6: **end if**
7: **if** iteration $\mod N = 0$ **then**
8: Update policy with Differential Evolution or Greedy (Algorithm 3)
9: **end if**
10: **end for**

network to accurately approximate the Q-values as defined in Equation (4). This process involves learning the balance between immediate rewards, action costs, and future rewards based on the transition dynamics of the system. Once the neural network is effectively trained, in online fashion, for the current coupled state $\mathbf{s}$, LPCA computes the value function $J(\mathbf{s}, \lambda)$ as described in Equation (3). The objective is to determine the optimal Lagrange multiplier λ^* that minimizes $J(\mathbf{s}, \lambda)$ as formulated in Equation (5). Finally, LPCA addresses the optimization problem set out in Equation (6) through two possible methods: a differential evolution optimizer (Algorithm 4) or a greedy optimizer (Algorithm 5).

The general training process of LPCA, as outlined in Algorithm 1, is a key aspect of our approach. The algorithm begins by utilizing a policy dictionary to interact with the environment. This dictionary is a mapping of states to actions, where each state corresponds to a unique action vector. During each interaction, an action is selected based on the current policy, and the environment responds accordingly. The response, including the state transition and reward information, is stored as a transition sample. Notably, each process of the weakly coupled MDP is treated individually, with the transition sample from each project recorded separately in a memory buffer. This memory serves as a repository for experiences, which are later used to update the neural network that approximates Q-values.

The training of the neural network, as detailed in Algorithm 2, is central to learning the Q-values from Equation (4) associated with state transitions (s, a, r, s') across a range of test λ values. These test values are selected as a random subset from 'lambda_grid', which encompasses a discretized set of λ values in the range of a problem-dependent $[-\lambda_{\max}, \lambda_{\max}]$, using 1000-point discretization.

During each iteration of the training process, the algorithm samples a batch of experiences from the memory. Each experience comprises the current state s, the action taken a, the reward received r, the subsequent state s', and a boolean flag indicating the terminal status of s', i.e. whether s' is the last state in an epoch, for a given individual project. For each experience, the algorithm computes the target Q-values for the state-action pair (s, a) using a random subset of λ values from 'lambda_grid'. This step involves evaluating the Q-value function for different levels of resource utilization

Algorithm 2 Update Q-values in LPCA Neural Network Model

for each random sample in memory **do**
  Extract s, a, r, s', $is\_terminal$ from sample {$is\_terminal$ indicates if s' is a terminal state}
  $Q \leftarrow$ Calculate target Q-values for s and a using a subset of λ values lambda_grid
  $V_{expected} \leftarrow$ Calculate expected value functions for s' using target network for each $\lambda \in$ lambda_grid
  if $is\_terminal$ **then**
    $Q_{target}(s, a, \lambda) \leftarrow r(s) - \lambda c(a)$
  else
    $Q_{target}(s, a, \lambda) \leftarrow r(s) - \lambda c(a) + \gamma \cdot V_{expected}$
  end if
  Perform a gradient descent step on $(Q_{target}(s, a, \lambda) - Q(s, a, \lambda))^2$ to update network weights
end for
Perform soft-update on target network weights $\theta' \leftarrow \theta\tau + (1 - \tau)\theta'$

and cost. By using a random subset of λ values, the algorithm optimizes computation, reducing the number of evaluations needed for each update. Additionally, this approach helps to avoid overfitting by selecting different λ points each time, ensuring that the model does not become too specialized to specific values of λ. The computation of the target Q-values $Q_{target}(s, a, \lambda)$ utilizes a target network, which is a lagged version of the primary neural network, to provide stable targets for learning [18].

Through this training process, the LPCA algorithm efficiently learns the Q-values for various state transitions under different levels of resource constraints, as dictated by the varying λ values.

Having trained the neural network to generate accurate approximations of Equation (4), we proceed with Algorithm 3 to compute the value function $J(\mathbf{s}, \lambda)$ for a given state $\mathbf{s}$ as in Equation (3). This computation involves evaluating $\sum_{i=1}^{N} \max_{a_i} Q(s_i, a_i, \lambda)$ for every λ within the discretized set 'lambda_grid'.

Once this term is calculated, obtaining the optimal λ^* is a one-dimensional convex optimization problem, as shown in Equation (5).

A key technical contribution of our work is how we explore the action space to solve the knapsack problem described in equation (6). This problem is challenging in neural networks due to the existence of many local minima, where traditional gradient optimization methods get stuck.

We propose two different strategies to explore this action space in order to make the best use of the available resources and select the best action based on our Q-value estimates. The first strategy, presented in Section 3.1, is an evolutionary algorithm (LPCA-DE). It uses mechanisms similar to natural selection to iteratively search for the optimal solution, effectively avoiding local minima by exploring a wider range of solutions.

The second strategy, presented in Section 3.2, is a greedy algorithm (LPCA-Greedy). It focuses on choosing the action based on the gradient of the Q-values with respect to the actions for each project, selecting the action that promises the highest increase in

Algorithm 3 Computation of Lagrange term λ^*

Require: method
Ensure: Updated policy dictionary $\pi(\mathbf{s})$
function PolicyDictUpdate(method)
for all $\mathbf{s} \in \mathbf{S}$ **do**
  q_table $\leftarrow$ Zero Matrix of size [n_lambda, N]
  for $i \in 1 : N$ **do**
    q_table$[:, i] \leftarrow \max_{a_i} Q(s_i, a_i, \lambda), \forall \lambda \in$ lambda_grid
  end for
  $J(\mathbf{s}, \lambda) \leftarrow$ Compute value functions as (3)
  $\lambda^*(\mathbf{s}) \leftarrow \arg\min_\lambda J(\mathbf{s}, \lambda)$
  if method = Evolution **then**
    $\mathbf{a}^* \leftarrow$ DifferentialEvolution($\mathbf{s}, \lambda^*(\mathbf{s}), a_{\max}$)
  else
    $\mathbf{a}^* \leftarrow$ Greedy($\mathbf{s}, \lambda^*(\mathbf{s}), a_{\max}, \delta$)
  end if
  $\pi(\mathbf{s}) \leftarrow \mathbf{a}^*$
end for
end function

the Q-value per unit of resource expended. This method is simpler and faster, and helps to quickly identify actions that increase payoff, even if it does not explore as widely.

3.1 Differential Evolution Optimization (LPCA-DE)

The first method (Algorithm 4) employs a differential evolution algorithm, renowned for its effectiveness in identifying global optima and circumventing local optima traps. This method is particularly adept at exploring the search space comprehensively [3].

A critical aspect of this approach is the integration of a penalty mechanism to ensure that action selection remains within resource constraints. Actions leading to resource utilization beyond the available limit are subjected to a significant penalty. This mechanism is in line with the role of the λ term in the Q-value definition (see Equation (4)). Given the $\lambda c(a)$ term in Equation (4), the derived optimal policy tends towards cost-effectiveness. However, it may not always coincide with the optimal policy of the original constrained problem (see Equation (1)) particularly if a higher action's benefit does not justify its cost in the relaxed problem, leading to potential underutilization of resources. This leads to a policy that may not fully utilize the available resources as defined in Equation (6). To address this, we introduce an additional penalty, proportional to the amount of unused resources, into the differential evolution optimization problem. This modification guides the optimizer towards actions that maximize resource usage, ensuring the algorithm not only pursues cost-effective solutions but also fully utilizes the available resources.

3.2 Greedy Optimization Strategy (LPCA-Greedy)

The second method (Algorithm 5) is a greedy optimization strategy. This approach is characterized by its iterative process of evaluating the gradient of the Q-values with

Algorithm 4 Action Selection through Differential Evolution Optimization

Require: State vector $\mathbf{s}$, fixed Lagrange multiplier λ_{fix}, maximum action $a_{\max}$
Ensure: Optimal actions maximizing Q-values under resource constraints
function DifferentialEvolution($\mathbf{s}, \lambda_{\text{fix}}, a_{\max}$)
  Bounds $\leftarrow [0, a_{\max}]$
  function ObjectiveFunction($\mathbf{a}, \mathbf{s}, \lambda^*$)
    $Q_{\text{total}} \leftarrow \sum_{i=1}^{N} Q(s_i, a_i, \lambda^*)$
    $C_{\text{total}} \leftarrow \sum_{i=1}^{N} C(s_i, a_i)$
    if $C_{\text{total}} > B$ **then**
      Penalty $\leftarrow$ Large constant value
      $Q_{\text{total}} \leftarrow Q_{\text{total}} -$ Penalty
    else if $C_{\text{total}} < B$ **then**
      Penalty $\leftarrow B - C_{\text{total}}$
      $Q_{\text{total}} \leftarrow Q_{\text{total}} -$ Penalty
    end if
    return $-Q_{\text{total}}$
  end function
  $\mathbf{a}^* \leftarrow$ Apply Differential Evolution optimization with $(ObjectiveFunction, Bounds)$
  return $\mathbf{a}^*$
end function

respect to the actions for each project and then allocating resources to the project with the highest gradient. The process continues until all resources are exhausted.

This strategy prioritizes complete resource utilization, assigning resources to the projects that promise the highest increase in the Q-value per unit of resource expended. Unlike the differential evolution method, which searches for an optimal policy and then adjusts for resource utilization, the greedy approach begins with the premise of full resource allocation and does so in a manner that maximizes the benefit derived from each project.

The choice between these methods can be guided by the specific characteristics of the problem at hand, such as the nature of the resource constraints and the desired balance between resource utilization and reward maximization.

Algorithm 5 Greedy Action Selection for Continuous MDP

Require: State $\mathbf{s}$, λ_{fix}, max action a_{max}, increment δ
Ensure: Optimal actions maximizing Q-values, maximum action $a_{\max}$
function Greedy($\mathbf{s}, \lambda_{\text{fix}}, a_{\text{max}}, \delta$)
  Initialize action vector $\mathbf{a}$ to zeros, $B_{\text{remaining}} = B$
  while $B_{\text{remaining}} > 0$ **do**
    $i \leftarrow \arg\max_i \frac{\partial Q}{\partial a_i}$
    $a_i \leftarrow a_i + \delta$, ensure $a_i \leq a_{\max}$
    $B_{\text{remaining}} \leftarrow B - \sum_{i=1}^{N} c(s_i, a_i)$
  end while
  return $\mathbf{a}$
end function

4 Experimental Results

To evaluate the effectiveness of our algorithms, we rely on measuring the average discounted rewards that their policies yield. Given a discount factor of $\gamma = 0.9$, we examine the rewards that each algorithm's policy yields over $t \in [0, 50]$ iterations, starting from every possible state in our problem space. The evaluation process involves computing the discounted sum of the rewards using the equation

$$R = \sum_{t=0}^{50} \sum_{i=1}^{N} \gamma^t r(s_i(t), a_i(t)),$$

where $r(s_i(t), a_i(t))$ represents the reward received at time t for being in state $s_i(t)$ and taking action $a_i(t)$, for each MDP i. To ensure statistical robustness and to derive confidence intervals for our performance metrics, we repeat this evaluation 100 times. The results are shown in our figures, with the mean performance represented by bold lines and the confidence intervals represented by the shaded areas surrounding these lines.

Our experimental framework encompasses three distinct types of problems: Type A and Type B, each representing a continuous action version of challenges similar to those discussed in [8], and the speed scaling problem inspired from [22]. Types A and B feature two states per project with $a \in [0, 2]$, with a reward function $R(s) = s$ and a cost function $C(a) = a$. The key difference between Type A and Type B lies in their transition probability matrices:

$$P_A(a) = \begin{pmatrix} 0.02a^2 - 0.09a + 0.8 & -0.02a^2 + 0.09a + 0.2 \\ 0.75e^{-0.947a} & 1 - 0.75e^{-0.947a} \end{pmatrix}$$
$$P_B(a) = \begin{pmatrix} 0.95e^{-2.235a} & 1 - 0.95e^{-2.235a} \\ 0.3347e^{-1.609a} & 1 - 0.3347e^{-1.609a} \end{pmatrix}.$$

Additionally, we introduce a mixed environment where half of the projects follow the transition probabilities of Type A and the other half those of Type B.

The speed scaling environment involves projects with six states, and $a \in [0, 2]$. We apply the uniformization technique [13] to construct an equivalent discrete time version of the continuous time problem. The transition probabilities are given by

$$P(a) = \begin{pmatrix} 1 - \frac{\alpha}{\nu} & \frac{\alpha}{\nu} & 0 & 0 & 0 & 0 \\ \frac{\mu_a}{\nu} & 1 - \frac{\alpha+\mu_a}{\nu} & \frac{\alpha}{\nu} & 0 & 0 & 0 \\ 0 & \frac{\mu_a}{\nu} & 1 - \frac{\alpha+\mu_a}{\nu} & \frac{\alpha}{\nu} & 0 & 0 \\ 0 & 0 & \frac{\mu_a}{\nu} & 1 - \frac{\alpha+\mu_a}{\nu} & \frac{\alpha}{\nu} & 0 \\ 0 & 0 & 0 & \frac{\mu_a}{\nu} & 1 - \frac{\alpha+\mu_a}{\nu} & \frac{\alpha}{\nu} \\ 0 & 0 & 0 & 0 & \frac{\mu_a}{\nu} & 1 - \frac{\mu_a}{\nu} \end{pmatrix},$$

where $\alpha = 0.9$ is the arrival rate, $\mu(a) = \sqrt{a}$ is the controlled departure rate, $\nu = \max_a(\alpha + \mu(a))$ is the normalization factor, β is the continuous discount factor related

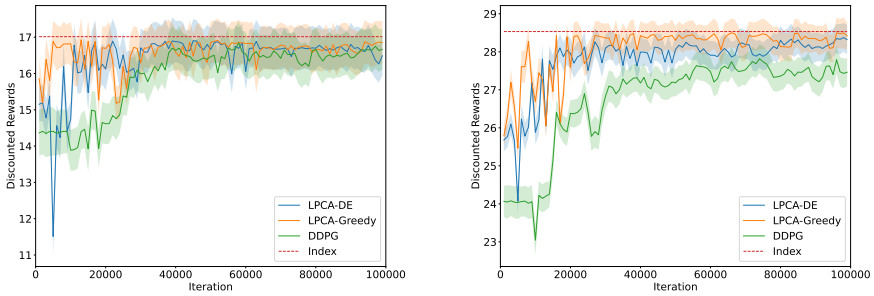

Fig. 1. Experimental results for Type A environment: (Left) 4 projects and 2 units of resources, (Right) 6 projects and 4 units of resources.

to the discrete factor γ as $\beta = \frac{\nu}{\gamma} - \nu$. The reward function is defined as:

$$R(s) = \frac{-s}{\nu+\beta} + \frac{C_r}{\nu+\beta} = \begin{cases} \frac{-s}{\nu+\beta} & \text{if } s < s_{\max} \\ \frac{-s_{\max}-10}{\nu+\beta} & \text{if } s = s_{\max} \end{cases}$$

where $C_r = -10$ is the rejection cost that occurs in the final state $s_{\max}$. The cost function is defined as $C(s,a) = \frac{a}{\nu+\beta}$ if $s > 0$, otherwise 0.

For Types A and B, we conducted experiments with both 4 projects with 2 units of resources and 6 projects with 4 units of resources. The mixed environment, combining Types A and B, was tested with 6 projects and 4 units of resources. The Speed Scaling experiment involved 4 projects with 1.5 units of resources, equivalent to fully activating two of the four projects.

To benchmark our algorithm, we choose DDPG (Deep Deterministic Policy Gradient) [9] augmented with OptLayer [12] as the baseline. OptLayer enhances DDPG by incorporating a constraint optimization layer in the actor network, enabling the generation of actions that respect the constraints outlined in the original problem formulation (Equations (1) and (6)).

In addition to this, we have benchmarked Whittle's index heuristic for continuous actions. These indices are computed through the algorithm proposed by [19] for discrete multi-action ($a \in [0, 1, 2, \dots]$) and adapted for an arbitrary discretization δ_a of the action ($a \in [0, \delta_a, 2\delta_a, \dots]$). For an approximation of a fully continuous action, we use a discretization of $\delta_a = 0.001$, leading to a total of 2001 possible actions. Due to the large amount of indices to compute, a tabular learning algorithm for those indices would not be feasible.

In the 4 projects and 2 resources configuration, both LPCA-DE and LPCA-Greedy demonstrated a clear advantage over DDPG, particularly in Type B environment (Figure 2 left), where the gap between the performance of both versions of LPCA and DDPG is larger and both LPCA algorithms converge to the Whittle Index policy performance. In Figure 1 left, although DDPG achieves a similar level of performance to LPCA, the latter converges to a performance level similar to Whittle indices' much faster, while DDPG takes around 40000 iterations.

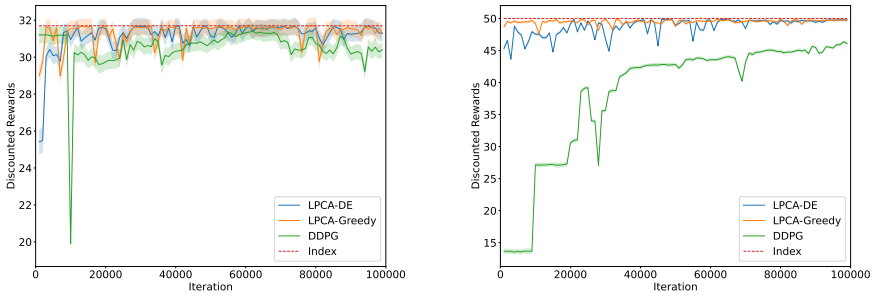

Fig. 2. Experimental results for Type B environment: (Left) 4 projects and 2 units of resources, (Right) 6 projects and 4 units of resources.

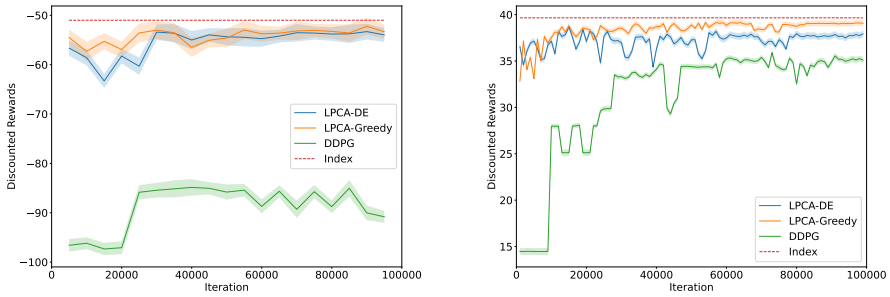

Fig. 3. (Left) Speed Scaling with 4 projects and 1.5 units of resources, (Right) Mixed Type A and B environments with 6 projects and 4 units of resources.

This gap widened significantly in the 6 projects and 4 resources setting. In Type A (Figure 1 right), the optimality gap between both versions of LPCA and DDPG widens. A similar pattern shows in Type B (Figure 2 right), with DDPG having subpar performance. In the mixed environment (Figure 3 right), DDPG's performance reflects the issues observed in the previous scenarios. On the other hand, LPCA-DE and specially LPCA-Greedy are able to obtain a better policy, close to the Whittle index policy performance.

In the Speed Scaling experiment (Figure 3 left), the performance of both LPCA-DE and LPCA-Greedy algorithms converges to a similar performance to the Whittle index policy, while DDPG's performance lags behind.

Overall, the LPCA algorithms consistently outperformed DDPG with OptLayer across various settings and environments. Notably, LPCA's superiority became increasingly pronounced in more complex scenarios involving a greater number of processes and limited resources.

5 Conclusion

In this study, we introduced the LPCA (Lagrange Policy for Continuous Actions) algorithm, a reinforcement learning approach for weakly coupled MDPs with continuous actions and resource constraints. Our experimental results demonstrate that LPCA, in both its Differential Evolution (DE) and Greedy variants, consistently outperforms the DDPG algorithm augmented with OptLayer across various scenarios. Notably, LPCA exhibits superior scalability with an increasing number of projects.

As a direction for future research, we aim to test the LPCA algorithm in larger-scale environments featuring more states per projects. This expansion will allow us to further evaluate LPCA's scalability and effectiveness in even more complex and dynamic settings, potentially broadening its applicability to a wider array of practical problems in operations research and beyond. The exploration of LPCA's performance in these extended scenarios is expected to yield valuable insights into its capabilities and limitations, guiding future enhancements and adaptations of the algorithm.

Acknowledgments. F. Robledo has received funding from the Department of Education of the Basque Government through the Consolidated Research Group MATHMODE (IT1456-22). Research partially supported by the French "Agence Nationale de la Recherche (ANR)" through the project ANR-22-CE25-0013-02 (ANR EPLER) and DST-Inria Cefipra project LION.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Avrachenkov, K.E., Borkar, V.S.: Whittle index based Q-learning for restless bandits with average reward. Automatica **139**, 110186 (May 2022). https://doi.org/10.1016/j.automatica.2022.110186, https://www.sciencedirect.com/science/article/pii/S0005109822000310
2. Bertsekas, D.: Dynamic Programming and Optimal Control: Volume I. Athena Scientific (2012), google-Books-ID: qVBEEAAAQBAJ
3. Das, S., Suganthan, P.N.: Differential Evolution: A Survey of the State-of-the-Art. IEEE Transactions on Evolutionary Computation **15**(1), 4–31 (Feb 2011). https://doi.org/10.1109/TEVC.2010.2059031, https://ieeexplore.ieee.org/abstract/document/5601760
4. Fujimoto, S., Hoof, H., Meger, D.: Addressing Function Approximation Error in Actor-Critic Methods. pp. 1587–1596. PMLR (Jul 2018), https://proceedings.mlr.press/v80/fujimoto18a.html
5. Gast, N., Gaujal, B., Yan, C.: The LP-update policy for weakly coupled Markov decision processes. Tech. rep. (Nov 2022). https://doi.org/10.48550/arXiv.2211.01961, http://arxiv.org/abs/2211.01961, arXiv:2211.01961 [math] type: article
6. Gittins, J.C.: Bandit processes and dynamic allocation indices. Journal of the Royal Statistical Society: Series B (Methodological) **41**(2), 148–164 (1979). https://doi.org/10.1111/j.2517-6161.1979.tb01068.x, https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1979.tb01068.x

7. Hawkins, J.T.: A Langrangian decomposition approach to weakly coupled dynamic optimization problems and its applications. Ph.D. thesis, Massachusetts Institute of Technology (2003)
8. Killian, J.A., Biswas, A., Shah, S., Tambe, M.: Q-Learning Lagrange Policies for Multi-Action Restless Bandits. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. pp. 871–881. KDD '21, Association for Computing Machinery, New York, NY, USA (Aug 2021). https://doi.org/10.1145/3447548.3467370, https://doi.org/10.1145/3447548.3467370
9. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. Tech. rep. (Jul 2019). https://doi.org/10.48550/arXiv.1509.02971, http://arxiv.org/abs/1509.02971, arXiv:1509.02971 [cs, stat] type: article
10. Meshram, R., Kaza, K.: Simulation Based Algorithms for Markov Decision Processes and Multi-Action Restless Bandits. Tech. rep. (Jul 2020). https://doi.org/10.48550/arXiv.2007.12933, http://arxiv.org/abs/2007.12933, arXiv:2007.12933 [cs, eess] type: article
11. Nakhleh, K., Ganji, S., Hsieh, P.C., Hou, I.H., Shakkottai, S.: NeurWIN: Neural Whittle Index Network For Restless Bandits Via Deep RL. In: Advances in Neural Information Processing Systems. vol. 34, pp. 828–839. Curran Associates, Inc. (2021), https://proceedings.neurips.cc/paper/2021/hash/0768281a05da9f27df178b5c39a51263-Abstract.html
12. Pham, T.H., De Magistris, G., Tachibana, R.: OptLayer - Practical Constrained Optimization for Deep Reinforcement Learning in the Real World. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 6236–6243 (May 2018). https://doi.org/10.1109/ICRA.2018.8460547, https://ieeexplore.ieee.org/abstract/document/8460547, iSSN: 2577-087X
13. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons (Aug 2014), google-Books-ID: VvBjBAAAQBAJ
14. Robledo, F., Borkar, V.S., Ayesta, U., Avrachenkov, K.: Tabular and Deep Learning of Whittle Index. In: EWRL 2022 - 15th European Workshop of Reinforcement Learning. Milan, Italy (Sep 2022), https://hal.science/hal-03767324
15. Shar, I.E., Jiang, D.: Weakly Coupled Deep Q-Networks. In: Oh, A., Neumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems. vol. 36, pp. 43931–43950. Curran Associates, Inc. (2023)
16. Srinivas, N., Krause, A., Kakade, S.M., Seeger, M.: Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. IEEE Transactions on Information Theory **58**(5), 3250–3265 (May 2012). https://doi.org/10.1109/TIT.2011.2182033, http://arxiv.org/abs/0912.3995, arXiv:0912.3995 [cs]
17. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
18. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence. vol. 30 (2016), https://ojs.aaai.org/index.php/AAAI/article/view/10295
19. Weber, R.: Comments on: Dynamic priority allocation via restless bandit marginal productivity indices. Top **15**(2), 211–216 (2007)
20. Wei, X., Yu, H., Neely, M.J.: Online Learning in Weakly Coupled Markov Decision Processes: A Convergence Time Study. Proceedings of the ACM on Measurement and Analysis of Computing Systems **2**(1), 12:1–12:38 (Apr 2018). https://doi.org/10.1145/3179415, https://dl.acm.org/doi/10.1145/3179415
21. Whittle, P.: Restless bandits: Activity allocation in a changing world. Journal of Applied Probability **25**(A), 287–298 (Jan 1988).

https://doi.org/10.2307/3214163, https://www.cambridge.org/core/journals/journal-of-applied-probability/article/abs/restless-bandits-activity-allocation-in-a-changing-world/DDEB5E22AFFEFF50AA97ADC96B71AE35

22. Wierman, A., Andrew, L.L.H., Tang, A.: Power-Aware Speed Scaling in Processor Sharing Systems. In: IEEE INFOCOM 2009. pp. 2007–2015 (Apr 2009). https://doi.org/10.1109/INFCOM.2009.5062123, https://ieeexplore.ieee.org/abstract/document/5062123, iSSN: 0743-166X