Testing Indexability and Computing Whittle and Gittins Index in Subcubic Time

Nicolas Gast, Bruno Gaujal, Kimang Khun

Testing Indexability and Computing Whittle and Gittins Index in Subcubic Time

Nicolas Gast, Bruno Gaujal and ✉ Kimang Khun

Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP*, LIG, 38000, Grenoble, France.
*Institute of Engineering Univ. Grenoble Alpes.

Contributing authors: nicolas.gast@inria.fr; bruno.gaujal@inria.fr; khun.kimang@gmail.com;

Abstract

Whittle index is a generalization of Gittins index that provides very efficient allocation rules for restless multi-armed bandits. In this work, we develop an algorithm to test the indexability and compute the Whittle indices of any finite-state restless bandit arm. This algorithm works in the discounted and non-discounted cases, and can compute Gittins index. Our algorithm builds on three tools: (1) a careful characterization of Whittle index that allows one to compute recursively the kth smallest index from the $(k-1)$th smallest and to test indexability, (2) the use of the Sherman-Morrison formula to make this recursive computation efficient, and (3) a sporadic use of the fastest matrix inversion and multiplication methods to obtain a subcubic complexity. We show that an efficient use of the Sherman-Morrison formula leads to an algorithm that computes Whittle index in $(2/3)n^3 + o(n^3)$ arithmetic operations, where n is the number of states of the arm. The careful use of fast matrix multiplication leads to the first subcubic algorithm to compute Whittle or Gittins index: By using the current fastest matrix multiplication, the theoretical complexity of our algorithm is $O(n^{2.5286})$. We also develop an efficient implementation of our algorithm that can compute indices of Markov chains with several thousands of states in less than a few seconds.

Keywords: Whittle Index, Gittins Index, Restless Bandit, Multi-armed Bandit, Sherman-Morrison, Markov Decision Process, Fast Matrix Multiplication

1 Introduction

Markovian bandits form a subclass of multi-armed bandit problems in which each arm has an internal state that evolves over time in a Markovian manner, as a function of the decision maker's actions. In such a problem, at each time step, the decision maker observes the state of all arms and chooses which one to activate. When the state of an arm evolves only when this arm is chosen, one falls into the category of *rested* Markovian bandits for which an optimal policy (in the discounted case) was found by Gittins [1]. When the state of an arm can also evolve when the arm is not chosen, the problem is called a *restless bandit* problem, and computing an optimal policy is computationally difficult [2].

In his seminal paper [3], Whittle proposed a very efficient heuristic: For each arm, an index function maps each state of the arm to a real number. The Whittle index policy then consists in activating the arms having the highest index first. This heuristic generalizes Gittins index to restless bandits. Contrary to the rested case, the Whittle index policy is in general not optimal. Yet, this policy has been proven to be very efficient over the years: up to a condition called *indexability*, Whittle index has been shown to be (in the undiscounted case) asymptotically optimal as the number of arms grows to infinity under certain technical assumptions [4–6]. Moreover, the heuristic performs extremely well in practice [7–9]. Restless bandits and Whittle index have been applied to many scheduling and resource allocation problems such as wireless communication [10, 11], web crawling [12, 13], congestion control [14, 15], queueing systems [16–22], and clinical trials [23].

The above examples show that, when a problem is indexable, computing Whittle index is a very efficient way to construct a nearly-optimal heuristic. This raises a few important questions, that we study in this paper:

- Is testing indexability computationally hard?
- Is there an efficient algorithm to compute Whittle index?
- Is Whittle index harder to compute than Gittins index?

Related work. The computation of Gittins index has received a lot of attention in the past, see for instance [24–27] and the recent survey [28]. For a n-state arm, the algorithms having the smallest complexity perform $(2/3)n^3 + O(n^2)$ arithmetic operations [28]. Note that in page 4 of [29] the author claims that it is unlikely that this complexity can be improved. As we see later, we do improve upon this complexity.

Concerning Whittle index, to the best of our knowledge, there are very few efficient general-purpose algorithms to test indexability, see *e.g.* [30], and most papers studying Whittle index either assume that the studied model is indexable or focus on specific classes of restless bandits for which the structure of arms can be used to show indexability, see *e.g.* [17, 19, 31, 32]. Assuming indexability, the computation of Whittle index has been considered by a few papers.

The most efficient numerical algorithm to compute Whittle index is recently presented in [29]. This algorithm, called *fast-pivoting* algorithm, performs $(2/3)n^3 + O(n^2)$ arithmetic operations[1] if the initialization phase is excluded from the count. This is done by using the parametric simplex method and exploiting the special structure of this linear system to reduce the complexity of simplex pivoting steps. This fast-pivoting algorithm is an efficient implementation of adaptive-greedy algorithm [33]. Based on a geometric interpretation of Whittle index, the authors in [34] propose a refinement of the adaptive-greedy algorithm of [33] to compute Whittle indices of all indexable restless bandits. For a n-state arm, the refined algorithm achieves a $O(n^3)$ complexity by using the Sherman-Morrison formula. The authors also propose a few checkable conditions to test indexability. However, those conditions are not necessary for indexability, which means that if an arm does not verify the conditions, we cannot conclude that the arm is non-indexable and an algorithm to check indexability is still needed. Also, no detailed description is given for adapting those conditions and their algorithm to restless bandit without discount. A thorough comparison between our algorithm and [29, 34] is given in Appendix E. For continuous-time n-state restless bandits, the work of [38] proposes an algorithm to check indexability and compute Whittle index with a complexity exponential in the number of states n of each arm. According to Remark 4.1 of that paper, this complexity can be reduced to $O(n^5)$ if the restless bandit is known to be indexable and threshold-based policies are optimal. It is stated that their approach is not applicable for discounted restless bandits.

While computing the Whittle indices of a known arm's model is still a challenge, there is interesting work in trying to learn Whittle index when only the arm's simulator is given and the arm's model is unknown. For instance, [12, 35, 36] use Q-learning algorithm to estimate Whittle index as time evolves in finite-state restless bandits. Moreover, the work of [37] uses deep reinforcement learning framework to estimate Whittle indices of the arms with large state space or convoluted transition kernel, assuming a notion of strong indexability. For learning aspect, the work of [35] shows as to learn Whittle index in non-discounted case by maintaining two Q-functions, updating them using Q-learning algorithm, and deducing Whittle index from them when needed. The way that Whittle indices are computed is very close to our work but less efficient than our algorithm since the authors are more interested in learning the index.

Contributions. In this paper, we investigate Whittle index computation in restless multi-armed bandit problems and present four main contributions.

Our first contribution is to discuss the ambiguities in the classical definitions of indexability. Classical definitions assume that an arm is indexable if the optimal policy is a non-decreasing function of some penalty term λ. While this definition works for most practical cases, it is not always precise enough

[1] multiplications and additions of real numbers, regardless of their values

because the optimal policy is in general not unique. In our definition, we specify the notion of increasingness that should be used. Our definition guarantees the uniqueness of Whittle indices. Note that our definition is the same as the one used in some recent papers (*e.g.*, [29]), but the ambiguity of the classical definition seems rarely mentioned.

Our second contribution is to propose a unified algorithm that computes the Whittle indices for both discounted and non-discounted restless bandits. Our algorithm, which can be viewed as a refinement of the algorithm in [34], tests whether the input arm is indexable or not, and computes Whittle index if the arm is. As a byproduct, our algorithm can compute Gittins index in rested bandits which are a subclass of restless bandits. This algorithm computes the indices in increasing order, and relies on an efficient use of the Sherman-Morrison formula to compute Whittle index in $(2/3)n^3 + O(n^2)$ plus subcubic time [39] to solve a linear system of order n. This algorithm can detect on the fly if a computed index violates the indexability condition, which adds an extra $(1/3)n^3 + O(n^2)$ arithmetic operations. This later test is optional: the complexity of our algorithm is $n^3 + o(n^3)$ when testing indexability and $(2/3)n^3 + o(n^3)$ without the test. These two complexities are comparable to the ones excluding the common initialization phase of reduced-pivoting indexability (RPI) and fast-pivoting adaptive greedy (FPAG) algorithms in [30]. For discounted problems, our algorithm works for any finite-state arm. For non-discounted problems, our algorithm takes as an input any arm and can output three results: the arm is indexable, non-indexable, or multichain. We show the correctness of the algorithm which proves that for unichain arms, our algorithm have soundness and completeness properties. The possible outputs of our algorithm are summarized in Figure 1.

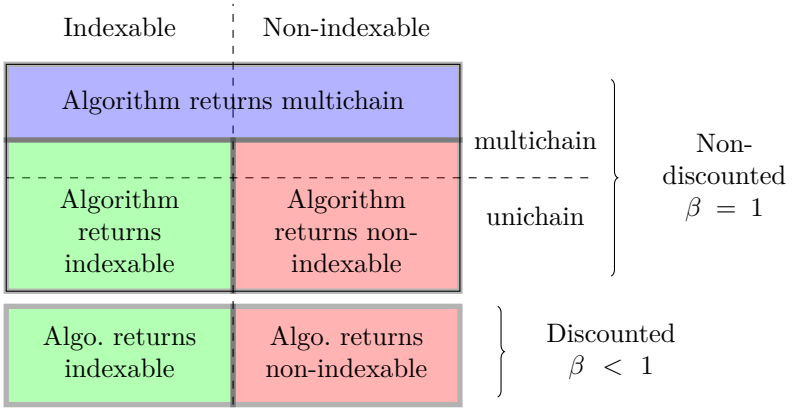


Fig. 1: Possible outputs of our algorithm: For unichain or discounted problems, our algorithm tests indexability and returns the index if and only if the problem is indexable. For some multichain problems, the algorithm can test indexability. For the others, it only returns that the problem is multichain.

Our third contribution is to show how to reduce the complexity of the above algorithm to obtain the first subcubic algorithm to compute Whittle index. This improvement is made possible by the fact that a linear system can be solved in subcubic time. By carefully reordering the computations, we show that it is possible to reduce the use of the Sherman-Morrison formula at the price of solving more linear systems. The subcubic complexity comes by striking a good balance between having too many or too few linear systems to solve. By using the current fastest matrix multiplication method, our algorithm can test indexability and compute Whittle index in $O(n^{2.5286})$. Our algorithm is also the first subcubic algorithm to compute Gittins index.

Our fourth and last contribution is to provide an open-source implementation of our algorithm in Python with Numba, and to present an empirical evaluation of the performance of our implementation. Our results show that our algorithm is very efficient in computing Whittle index and testing indexability. Moreover, our simulations indicate that the subcubic version of our algorithm not only has an asymptotically small complexity but is also faster in practice than our original $(2/3)n^3$ algorithm. Testing the indexability and computing indices takes less than one second for $n = 1000$ states and less than 10 minutes for $n = 15000$ states. This is 15 to 20 times faster than the original computation times reported in [29] (for a Matlab implementation), and about 5 times faster than an optimized implementation of [29] (for a Julia implementation).

Road map. The paper is organized as follows. We introduce the problem and the definition of Whittle index in Section 2. In Section 3, we characterize Whittle index and provide a general idea of how to compute Whittle index. Then, we show, in Section 4, how to use the Sherman-Morrison formula to compute the indices efficiently. We then show how to reduce the complexity of the algorithm by using fast matrix multiplication method in Section 5. We compare the numerical result of different variants of our algorithm in Section 6. We show how to adapt this approach to the discounted case in Section 7. Finally, we conclude in Section 8.

2 Restless bandits and indexability

2.1 Restless bandit arms and multi-armed bandit

In this paper, a restless bandit arm (that we denote later by RB) is a Markov decision process (MDP) with discrete state space $[n] := \{1, \ldots, n\}$ and binary action space $\{0, 1\}$, where 0 denotes the action "rest" and 1 denotes the action "activate". The time is discrete and the evolution is Markovian: If the MDP is in state i and action a is chosen, the decision maker earns an instantaneous reward r_i^a and the arm transitions to a new state j with probability P_{ij}^a. We denote this MDP by the pair (r, P). The name "restless" comes from the fact that an arm put at rest may still transition to a new state.

A restless multi-armed bandit (RMAB) problem is a finite collection of $M \in \mathbb{N}^*$ independent RB arms. At time t, the decision maker observes the state of all arms, and can choose up to m arms to activate, where $m \leq M$ is a fixed constant. The decision maker then earns a reward that is the sum of the rewards of all arms. The objective of the decision maker is to identify an allocation rule that maximizes the average reward earned over an infinite number of time steps. Such a problem is notoriously difficult to solve, as its complexity grows exponentially with the number of arms [2]. In his seminal paper [3], Whittle proposes the following approach: *if all arms verify a technical condition known as* indexability*, then each state i of each arm is associated with a real number λ_i, that is now known as the* Whittle index *of state i. At each time, the decision maker activates the m arms whose Whittle index of their current states are the m greatest indices.* As mentioned earlier, this heuristic performs extremely well in practice, see *e.g.*, [7–9]. This shows that if the M arms are all indexable, then one can derive a very efficient allocation rule for RMAB problems by computing the Whittle indices of all arms. The Whittle indices of an arm do not depend on the other arms. This shows that the computational cost of Whittle index policy is linear in the number of arms multiplied by the time to compute the indices for a single arm. Hence, in the remaining of the paper, we focus on a single arm and present a new algorithm to test indexability and compute the index of a given arm.

2.2 Indexability and Whittle index

For the remaining of the paper, we consider a single arm (r, P) that has n states. In this section, we introduce the notion of indexability and discuss some ambiguities that we have found when using the definition of indexability defined in previous works.

2.2.1 Policy and arm structure

An arm is a two-action MDP. Hence, a policy π is a subset of the state space, $\pi \subseteq [n]$, such that the policy chooses to activate the arm in state i if $i \in \pi$. We say that π is the set of *active* states, and we say that state i is *passive* if $i \notin \pi$. By abuse of notation, we will write $\pi_i = 1$ if $i \in \pi$ and $\pi_i = 0$ if $i \notin \pi$, and we denote by $\boldsymbol{P}^\pi$ the transition matrix corresponding to the policy π, *i.e.*, $P^\pi_{ij} = P^{\pi_i}_{ij}$.

Following the classical definitions in the literature [40], we say that:

- A policy π is *unichain* if the transition matrix $\boldsymbol{P}^\pi$ induced by π has a unique recurrent class. A policy that is not unichain is called *multichain*.
- An arm is unichain if all policies $\pi \subseteq [n]$ are unichain. An arm is multichain if it is not unichain, *i.e.*, if there exists a policy $\pi \subseteq [n]$ that is multichain.

2.2.2 Gain optimality and Bellman optimality

Following a policy π, we denote by g_i^π the long-run average reward that a decision maker would obtain when starting in state i. In the remainder of the paper, we use the term "gain" to denote the long-run average reward. Let $g_i^* = \max_\pi g_i^\pi$ be the maximal gain starting from state i. From [40, Chapter 9], $\boldsymbol{g}^*$ is uniquely defined. We say that a policy π is gain optimal if $g_i^\pi = g_i^*$ for all state i.

It is shown in [40, Chapter 9] that $\boldsymbol{g}^*$ is the optimal gain if and only if there exists a vector $\boldsymbol{h}^*$, called optimal bias vector that satisfies the Bellman *optimality* equations: for all $i \in [n]$,

$$g_i^* = \max_{a\in\{0,1\}} \Big(\sum_{j=1}^n P_{ij}^a g_j^*\Big) \tag{1}$$

$$g_i^* + h_i^* = \max_{a\in\{0,1\}} \Big(r_i^a + \sum_{j=1}^n P_{ij}^a h_j^*\Big). \tag{2}$$

We say that a policy π is *Bellman optimal* if there exists[2] a bias vector $\boldsymbol{h}^*$ that is a solution of (2) and such that π attains the maximum in (2), *i.e.*: for all i,

$$\sum_{j=1}^n P_{ij}^{\pi_i} g_j^* = g_i^* \text{ and } \pi_i \in \arg\max_{a\in\{0,1\}} \Big(r_i^a + \sum_{j=1}^n P_{ij}^a h_j^*\Big). \tag{3}$$

The notion of Bellman optimality is stronger than the notion of gain optimality: A Bellman optimal policy is gain optimal, but the converse is not true in general. Note that the distinction between gain optimal and Bellman optimal policies is only important for the average reward criterion. This distinction disappears for the discounted case that we discuss in Section 7. The notion of Bellman optimality is equivalent to the notion of canonical optimality, that characterize policies that are optimal for any finite horizon, see [**?**].

2.2.3 λ-penalized MDP and definition of indexability

For each $\lambda \in \mathbb{R}$, we define a λ-penalized MDP[3] whose transition matrices are the same as in the original MDP and whose reward at time $t \geq 0$ when taking action a_t in state s_t is $r_{s_t}^{a_t} - \lambda a_t$. The quantity λ is a penalty for taking action "activate". For λ-penalized MDPs, we define the gain and bias functions as in Section 2.2.2, but these quantities now depend on λ. Hence, we will write them as functions of λ: For instance, the optimal gain is $\boldsymbol{g}^*(\lambda)$, and we will use the notation $\boldsymbol{h}^*(\lambda)$ to denote an optimal bias and $\pi^*(\lambda)$ to denote an optimal policy.

[2] If the MDP is unichain, then the bias vector $\boldsymbol{h}^*$ is unique up to an additive constant. This is in general not the case for multichain MDPs.

[3] not to be confused with β-discounted MDPs, where the discount is on rewards and not on actions.

The classical definition of indexability use in the literature [34, 35, 37] says that an arm is indexable if and only if the optimal policy $\pi^*(\lambda)$ is non-increasing in λ (for the inclusion order). If an arm is indexable, these papers define the Whittle index of a state i as a real number λ_i such that $\pi^*(\lambda) = \{i \in [n] : \lambda_i \ > \lambda\}$. This definition is ambiguous for two reasons: First, optimal policies are in general not unique. Hence, the notion of $\pi^*(\lambda)$ being non-increasing is unclear: should all optimal policies be non-increasing or at least one? Second, the notion of optimality for a policy is also unclear: should it mean "gain optimal", "bias optimal" or another notion of optimality?

To solve these ambiguities, in this paper, we use the following definition of indexability.

Definition 1. *Given a finite-state arm, let $\Pi^*(\lambda)$ be the set of Bellman optimal policies for a penalty λ. We say that the arm is indexable if for all $\lambda < \lambda'$, and all policies $\pi \in \Pi^*(\lambda)$ and $\pi' \in \Pi^*(\lambda')$, then $\pi \supseteq \pi'$.*

This definition says that the function $\pi^*(\lambda) \supseteq \pi^*(\lambda')$ regardless of the choice of Bellman optimal policies. As we show next, it guarantees that the Whittle indices are uniquely defined when they exist. As we detail in Appendix A.1, this is not necessarily the case when we consider other interpretations of the classical definition.

Note that for discounted problems, this definition coincide with the one used in [29]. For undiscounted MDPs, we add in addition that the criterion for optimality should be the Bellman optimality.

2.3 Definition of Whittle index and characterization of indexability

The proposition below shows that Definition 1 implies that Whittle index is well defined and proposes a characterization of any indexable arm, that we will later use to derive our algorithm.

Lemma 1. *In a n-state arm, the following three properties are equivalent:*

(i) The arm is indexable.

(ii) For all state $i \in [n]$, there exists a unique penalty λ_i – called the Whittle index of state i – such that if $\pi \in \Pi^*(\lambda)$ is any Bellman optimal policy for the penalty λ, then $\pi_i = 1$ if $\lambda < \lambda_i$ and $\pi_i = 0$ if $\lambda > \lambda_i$.

(iii) There is a non decreasing sequence of penalties $\mu^0_{\min} := -\infty \leq \mu^1_{\min} \leq \mu^2_{\min} \leq \cdots \leq \mu^n_{\min} \leq \mu^{n+1}_{\min} := +\infty$ and a sequence of policies $\pi^1 := [n] \supsetneq \pi^2 \supsetneq \cdots \supsetneq \pi^{n+1} := \emptyset$ such that:

- *If $\lambda \in (\mu^{k-1}_{\min}, \mu^k_{\min})$, there exists a unique Bellman optimal policy π^k.*
- *If k is such that $\mu^{k-1}_{\min} < \mu^k_{\min}$, then all Bellman optimal policies for the penalty $\mu^{k-1}_{\min}$ contain π^k, and π^k contains all Bellman optimal policies for the penalty $\mu^k_{\min}$.*

In the above lemma, we use a subscript "min" in the penalties $\mu^k_{\min}$ in order to be consistent with the same quantities used in Algorithm 1 and 2. The signification of this "min" is because it will be a minimum of values of the form μ^k_i. We should stress that these quantities (as well as the Whittle index λ_i) can either be finite or infinite. When we say that "a policy π is optimal for the penalty $+\infty$", this means "there exists a penalty $\bar{\lambda}$ such that π is optimal for all $\lambda \geq \bar{\lambda}$". Also, the last part of the lemma implies that π^k is the unique Bellman optimal policy for all penalty $\lambda \in (\mu^{k-1}_{\min}, \mu^k_{\min})$.

Proof The lemma is a direct consequence of the definition of indexability.

$(i) \Rightarrow (ii)$ – Assume first that the arm is indexable. Let $i \in [n]$ be a state and let $\lambda_i = \sup\{\lambda : \exists \pi \in \Pi^*(\lambda) \text{ such that } \pi_i = 0\}$. By Definition 1, if π' is a Bellman optimal policy for a penalty $\lambda > \lambda_i$, then $\pi' \subseteq \pi$, which in turn implies that $\pi'_i = 0$. Similarly, if $\lambda < \lambda_i$, then $\pi'_i = 1$. This implies (ii).

$(ii) \Rightarrow (iii)$ – Assume (ii) and let σ^k be the state with the kth smallest index (where ties are broken arbitrarily). Let $\mu^k_{\min} := \lambda_{\sigma^k}$ be the index of the state σ^k and let $\lambda \in (\mu^{k-1}_{\min}, \mu^k_{\min})$. By (ii), any Bellman optimal policy for the penalty $\lambda < \lambda_{\sigma^{k-1}}$ contains $\pi^k := [n] \setminus \{\sigma^1, \ldots, \sigma^{k-1}\}$. Similarly, π^k contains any Bellman optimal policy for the penalty $\lambda > \lambda_{\sigma^{k-1}}$. This implies that the policy π^k is the unique optimal policy for all $\lambda \in (\mu^{k-1}_{\min}, \mu^k_{\min})$.

$(iii) \Rightarrow (i)$ – The property (iii) implies that implies that π^k is the unique Bellman optimal policy for all $\lambda \in (\mu^{k-1}_{\min}, \mu^k_{\min})$. □

3 Condition for indexability and basic algorithm

This section aims at providing a basic algorithm to detect whether an arm is indexable or not and if it is the case, to compute the Whittle index of all states. This algorithm tries to construct a sequence of *unichain* policies $\pi^1 \supsetneq \pi^2 \supsetneq \ldots$ that satisfy the conditions of Lemma 1. We prove the correctness of our algorithm: if it can construct such a sequence, then the problem is indexable and the computed indices are correct. If the algorithm cannot compute such a sequence of policies, this is either because the problem is not indexable, or because the arm is multichain.

3.1 Condition for optimality

In this section, we provide two technical lemmas that we will use in our algorithm. They provide conditions to verify when a given unichain policy is Bellman optimal and if yes, when it is the unique Bellman optimal policy.

Let $\pi \subseteq [n]$ be a unichain policy and fix a penalty λ. By [40, Chapter 8], there exists a gain g^π and a bias vector $\boldsymbol{h}^\pi$ such that π satisfies Bellman *evaluation* equations: for all $i \in [n]$,

$$g^\pi(\lambda) + h^\pi_i(\lambda) = r^{\pi_i}_i - \lambda \pi_i + \sum_{j=1}^{n} P^{\pi_i}_{ij} h^\pi_j(\lambda). \tag{4}$$

We denote by α_i^π the *active advantage* in state i under policy π, which is the difference between the value in state i of action activate and the one of action rest. It is defined by:

$$\alpha_i^\pi(\lambda) := r_i^1 - r_i^0 - \lambda + \sum_{j=1}^{n} (P_{ij}^1 - P_{ij}^0) h_j^\pi(\lambda). \tag{5}$$

For a unichain policy, Equation (4) uniquely determines the vector $\boldsymbol{h}^\pi(\lambda)$ up to an additive constant $c\mathbf{1}$ (see [40, Chapter 8]). Hence the active advantage vector $\boldsymbol{\alpha}^\pi(\lambda)$ is uniquely determined for a unichain policy π. As we will see later, the function $\boldsymbol{\alpha}^\pi(\lambda)$ is affine in λ. Note that despite the name "advantage", $\boldsymbol{\alpha}^\pi$ can be negative.

Our algorithm computes the Whittle index in increasing order, by trying to eliminate states one by one. The following lemma shows that to compute the next Bellman optimal policy, one should look at when the active advantage of a state is equal to 0. In this lemma, $\pi \ominus \{i\}$ denotes the symmetric difference between π and $\{i\}$, *i.e.*, $\pi \ominus \{i\} = \pi \setminus \{i\}$ if $i \in \pi$ and $\pi \ominus \{i\} = \pi \cup \{i\}$ if $i \notin \pi$. Also, the active advantage provides necessary and sufficient condition for a unichain policy to be Bellman optimal, and/or to be the unique Bellman optimal policy as shown in the following lemma.

Lemma 2. *In a finite-state arm, let π be a unichain policy. Then, for any penalty λ:*

(i) π is Bellman optimal if and only if $\alpha_i^\pi(\lambda) \geq 0$ for all $i \in \pi$ and $\alpha_i^\pi(\lambda) \leq 0$ for all $i \notin \pi$.

(ii) Suppose that π is Bellman optimal and $\alpha_i^\pi(\lambda) = 0$. Then, $\pi \ominus \{i\}$ is also Bellman optimal. If, in addition, $\pi \ominus \{i\}$ is unichain, then $\boldsymbol{\alpha}^\pi(\lambda) = \boldsymbol{\alpha}^{\pi \ominus \{i\}}(\lambda)$.

(iii) π is the unique Bellman optimal policy if and only if $\alpha_i^\pi(\lambda) > 0$ for all $i \in \pi$ and $\alpha_i^\pi(\lambda) < 0$ for all $i \notin \pi$.

Proof For the first point (i), one direction of the equivalence is direct: If policy π is Bellman optimal, then $\alpha_i^\pi(\lambda) \geq 0$ for all $i \in \pi$ and $\alpha_i^\pi(\lambda) \leq 0$ for all $i \notin \pi$. This is because a bias vector $\boldsymbol{h}^\pi$ that is a solution of Bellman evaluation equations (4) satisfies Bellman optimality equations (2).

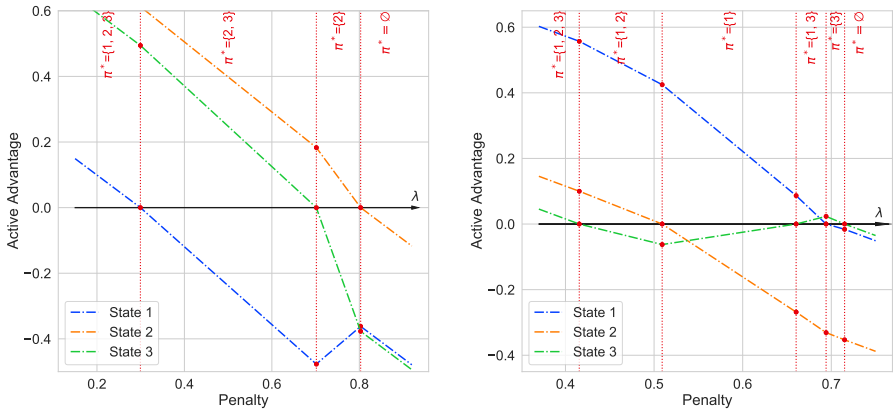
We now prove the other direction of Point (i): If $\alpha_i^\pi(\lambda) \geq 0$ for all $i \in \pi$ and $\alpha_i^\pi(\lambda) \leq 0$ for all $i \notin \pi$, then policy π is Bellman optimal. Since π is unichain, its gain g^π is state independent and satisfies the optimality equations (1). If $\alpha_i^\pi(\lambda) \geq 0$ for all $i \in \pi$ and $\alpha_i^\pi(\lambda) \leq 0$ for all $i \notin \pi$, then any bias vector $\boldsymbol{h}^\pi$ that is a solution of (4) also satisfies (2). In consequence, $g^\pi \mathbf{1}$ and $\boldsymbol{h}^\pi$ form a solution of the optimality equations (1) and (2). From [40, Chapter 9], $\boldsymbol{g}^*$ is uniquely defined by (1) and (2). Thus, $g_i^* = g^\pi$ for all $i \in [n]$ and the first condition of Bellman policy characterization equation (3) is satisfied. Finally, the fact that bias vector $\boldsymbol{h}^\pi$ satisfies (2) fulfills the second condition of (3). That concludes the proof.

For the second point (ii), since π is unichain, the optimal gain $g_i^* = g^\pi$ for all $i \in [n]$. So, $\pi \ominus \{i\}$ satisfies the first condition of (3). Moreover, $\alpha_i^\pi(\lambda) = 0$ implies

that policy $\pi \ominus \{i\}$ satisfies evaluation equations (4) for some $\boldsymbol{h}^{\pi}$. Since π is Bellman optimal, $\boldsymbol{h}^{\pi}$ is a solution of (2). So, $\pi \ominus \{i\}$ satisfies the second condition of (3). We conclude that $\pi \ominus \{i\}$ is Bellman optimal. Last but not least, if, in addition, $\pi \ominus \{i\}$ is unichain, then $\boldsymbol{h}^{\pi}$ is a solution of the evaluation equations (4) for policy $\pi \ominus \{i\}$. Consequently, $\boldsymbol{\alpha}^{\pi}(\lambda) = \boldsymbol{\alpha}^{\pi \ominus \{i\}}(\lambda)$.

Points (i) and (ii) also show one direction of the equivalence of (iii): If policy π is the unique Bellman optimal policy, then for all state i, $\alpha_i^{\pi}(\lambda) \neq 0$. The non-trivial property is the other direction of the equivalence. This is a consequence of Lemma 6 that we prove in Appendix B.1. □

Note that the main difficulty in proving Lemma 6 is that we do not assume the arm to be unichain: for a unichain arm, the bias of the optimal policy is unique up to a constant vector (see [41] and [40, Section 8.4]). This implies that if π is an optimal policy and $\alpha_i^{\pi}(\lambda) \neq 0$ for all i, then π is the unique optimal policy. The proof of Lemma 6 that we do in Appendix B.1 does not require the MDP to be unichain, but only the Bellman optimal policy π to be unichain. Lemma 6 shows that $\alpha_i^{\pi}(\lambda) \neq 0$ implies that no other policy can be Bellman optimal (not even multichain policies). In Appendix B.1, we prove that this holds not only for two-action MDPs but also for any MDP with finite state and action spaces.



(a) Indexable arm with 3 states

(b) Non-indexable arm with 3 states

Fig. 2: The active advantage as a function of penalty for two unichain examples, one is indexable (Figure 2a) and the other is not (Figure 2b). The red dots mark where the lines change their slope. The parameters of both examples are provided in Appendix A.4.

To illustrate Lemma 1 and Lemma 2, we consider two three-state arms. For each model, we plot in Figure 2 the active advantage $\alpha_i^{\pi^*(\lambda)}(\lambda)$ as a function of the penalty λ for all states $i \in \{1, 2, 3\}$. By Lemma 2, we know that an optimal policy should activate all states having a positive advantage and rest

all states having a negative advantage. Combined with the characterization of Lemma 1, this shows that:

- The model presented in Figure 2a is indexable: the optimal policy is a non-increasing function of λ and the indices are $\lambda_1 \approx 0.3$, $\lambda_2 \approx 0.8$ and $\lambda_3 \approx 0.7$.
- The model presented in Figure 2b is not indexable: the optimal policy $\pi^*(0.7) = \{3\}$ is not included in $\pi^*(0.6) = \{1\}$.

3.2 Overview of the algorithm

Our algorithm computes Whittle index in increasing order by navigating through unichain Bellman optimal policies and using the characterization provided by Lemma 1. It follows the graphical construction given in Figure 2a. It uses the following facts:

- If policy $\pi^1 := [n]$ is unichain, then $\boldsymbol{\alpha}^{\pi^1}$ is decreasing in λ
- Similarly, if policy $\pi^{n+1} := \emptyset$ is unichain, then $\boldsymbol{\alpha}^{\pi^{n+1}}$ is decreasing in λ
- For an indexable arm, computing the index can be done by a greedy algorithm that constructs a sequence of penalties $\mu^1_{\min} \leq \mu^2_{\min} \leq \cdots \leq \mu^n_{\min}$ and a sequence of unichain policies $\pi^1 \supsetneq \cdots \supsetneq \pi^n$ by looking at where $\alpha_i^{\pi^k}(\lambda)$ intersects horizontal axis for all $i \in \pi^k$.
- The arm is indexable if and only if for all k such that $\mu^{k-1}_{\min} < \mu^k_{\min}$, the constructed π^k is the largest Bellman optimal policy for the penalty $\mu^k_{\min}$.

In order to compute Whittle index and test indexability, our algorithm needs that all policies π^k constructed by the algorithm to be unichain. It does not require the arm to be unichain.

This leads to Algorithm 1, that we write in pseudo-code. This algorithm relies on two subroutines: on Line 7, to compute the next index and on Line 8 to test if a policy is Bellman optimal. We will describe later in the paper how to implement these functions in an efficient manner. Note that in all the paper, we use the superscript k (*e.g.*, π^k, μ^k, σ^k) to refer to the quantities computed at iteration k. We use the subscripts i or j (*e.g.*, $\pi_i, \lambda_i, \mu_i, \pi_j$) to refer to the quantities related to states i or j.

Note that when $\mu^k_{\min} = +\infty$, the quantity $\alpha_i^{\pi^k}(\mu^k_{\min})$ defined in Line 7 of Algorithm 1 should be understood as $\lim_{\lambda\to\infty} \alpha_i^{\pi^k}(\lambda) \in \mathbb{R} \cup \{-\infty, +\infty\}$. These limits are well defined because the functions αs are affine in λ.

To illustrate how the algorithm works, we plot in Figure 3 the values computed by the algorithm for the two arms represented in Figure 2. For both models (indexable and non-indexable), the algorithm starts with the policy $[n]$ for which the derivative of the active advantage with respect to λ is -1 for all states. It then computes $\mu^1_{\min}$ which is the potential index of State $\sigma^1 = 1$ for 3a and of State $\sigma^1 = 3$ for 3b. The algorithm then moves to iteration 2 and computes $\mu^2_{\min} > \mu^1_{\min}$ for both models and observes that for both models $\alpha^{\pi^2}_{\sigma^1}(\mu^2_{\min}) < 0$ for both models. Then the algorithm moves to iteration 3 and computes $\mu^3_{\min} > \mu^2_{\min}$. There are now two cases:

Algorithm 1 Given a n-state arm, test indexability and compute Whittle index (if indexable).

Set $\pi^1 := [n]$, $\mu^0_{\min} := -\infty$
if π^1 is multichain **then**
  return the arm is multichain
end if
for $k = 1$ to n **do**
  Compute $\boldsymbol{\alpha}^{\pi^k}(\lambda)$
  Let $\mu^k_{\min} := \inf\{\lambda \geq \mu^{k-1}_{\min} : \exists i \in \pi^k, \alpha_i^{\pi^k}(\lambda) = 0\}$
  if $\mu^{k-1}_{\min} < \mu^k_{\min}$ and for some $i \notin \pi^k$, $\alpha_i^{\pi^k}(\mu^k_{\min}) \geq 0$ **then**
    return the arm is not indexable
  end if
  if $\mu^k_{\min} = +\infty$ **then**
    Set $\lambda_i := +\infty$ for all $i \in \pi^k$
    return the arm is indexable and the indices are $\{\lambda_i\}_{i\in[n]}$.
  end if
  Let $\sigma^k \in \pi^k$ be such that $\alpha^{\pi^k}_{\sigma^k}(\mu^k_{\min}) = 0$ and $\lambda_{\sigma^k} = \mu^k_{\min}$
  Set $\pi^{k+1} := \pi^k \setminus \{\sigma^k\}$
  if π^{k+1} is multichain **then**
    return the arm is multichain
  end if
end for
return the arm is indexable and the indices are $\{\lambda_i\}_{i\in[n]}$.

- For 3a, the algorithm verifies that $\pi^4 := \emptyset$ is unichain ($\alpha_i^{\emptyset}$ is decreasing in λ for all i), terminates, and returns that the arm is indexable.
- For 3b, the algorithm realizes that $\alpha^{\pi^3}_{\sigma^1}(\mu^3_{\min}) > 0$ which shows that this model is not indexable.

Note that for the indexable example of Figure 3a, the active advantage function is not a decreasing function of λ. Hence, this example is neither PCL-indexable (defined in [29, Definition 3]) nor strongly-indexable (defined in [37]). However, this does not prevent our algorithm from working.

3.3 Correctness of Algorithm 1

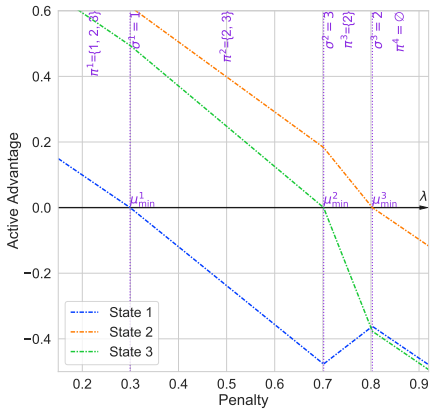
The following result shows that Algorithm 1 is correct.

Theorem 1. *Given a n-state arm:*

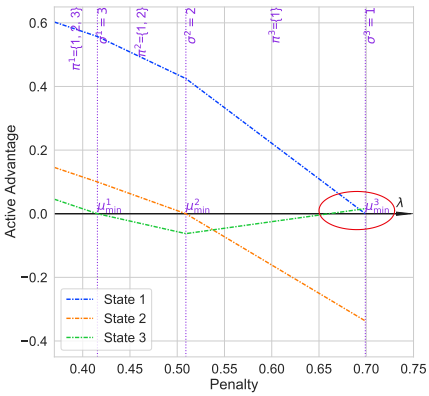
(i) if Algorithm 1 outputs "the arm is indexable and the indices are $\{\lambda_i\}_{i\in[n]}$", then the arm is indexable and each λ_i is the Whittle index of state i;

(ii) if Algorithm 1 outputs "non-indexable", then the arm is non-indexable;

(iii) if Algorithm 1 outputs "multichain", then the arm is multichain.



(a) Indexable arm with 3 states. Note that the algorithm does not compute $\boldsymbol{\alpha}^{\pi^4}$. It checks if policy $\pi^4{=}\emptyset$ is unichain or not. If it is, then $\alpha_i^{\pi^4}$ is decreasing in λ for each i.



(b) Non-indexable arm with 3 states. The algorithm stops at iteration 3 because $\alpha_3^{\pi^3}(\mu_{\min}^3) > 0$ (the green line at the zone circled with red ellipse).

Fig. 3: The active advantage $\alpha_i^{\pi^k}(\lambda)$ computed by the algorithm, for the two examples of Figure 2.

A direct consequence of Theorem 1 is that for unichain arms, Algorithm 1 provides a full characterization of indexability.

Corollary 2. *Given a unichain arm with finite states, Algorithm 1 outputs "the arm is indexable" if and only if it is indexable.*

Note that Algorithm 1 does not require the arm to be unichain to work. The required condition is that the Bellman optimal policies $\{\pi^k\}_{k\geq 1}$ that Algorithm 1 uses are all unichain. In particular, there exist examples of arms that are multichain and indexable and for which the algorithm returns "indexable". Similarly, there exist examples of arms that are multichain and non-indexable and for which the algorithm returns "non-indexable". We provide such examples in Appendix A.2.

Our algorithm works by exploring solely unichain policies. It does so because (in general) the bias of multichain policy is not unique. The characterization of Bellman optimal policies is much more difficult for multichain models, and the notion of indexability becomes more elusive (see Example A1 in Appendix A.1). When Algorithm 1 returns "multichain", it means that the algorithm is unable to decide whether the arm is indexable or not (but the algorithm knows that the arm is multichain because it has just found a multichain policy).

Proof of Theorem 1 **Proof of (i)** – We first prove by induction on k that:

If Algorithm 1 completes iteration $k \geq 1$, then π^k and π^{k+1} are unichain and

(A) π^k is the unique Bellman optimal policy for all $\lambda \in (\mu^{k-1}_{\min}, \mu^k_{\min})$;

(B) π^{k+1} is Bellman optimal for $\mu^k_{\min}$ and $\boldsymbol{\alpha}^{\pi^{k+1}}(\mu^k_{\min}) = \boldsymbol{\alpha}^{\pi^k}(\mu^k_{\min})$.

Base case $k = 1$: As we prove later in (19), $\frac{\partial \alpha_i^{\pi^1}}{\partial \lambda} = -1$. So, for each $i \in \pi^1$, $\alpha_i^{\pi^1}(\lambda)$ is decreasing in λ. By definition, $\mu^1_{\min}$ is the smallest λ such that one of the $\alpha_i^{\pi^1}(\lambda) = 0$. Hence, for all $\lambda < \mu^1_{\min}$: $\alpha_i^{\pi^1}(\lambda) > 0$. By Lemma 2, this shows that π^1 is the unique Bellman optimal policy for all $\lambda \in (\mu^0_{\min}, \mu^1_{\min})$, so (A) is true. Moreover, since π^1 is Bellman optimal for the penalty $\mu^1_{\min}$ and π^2 is unichain, Lemma 2 implies that π^2 is Bellman optimal for the penalty $\mu^1_{\min}$ and $\boldsymbol{\alpha}^{\pi^2}(\mu^1_{\min}) = \boldsymbol{\alpha}^{\pi^1}(\mu^1_{\min})$. This shows (B).

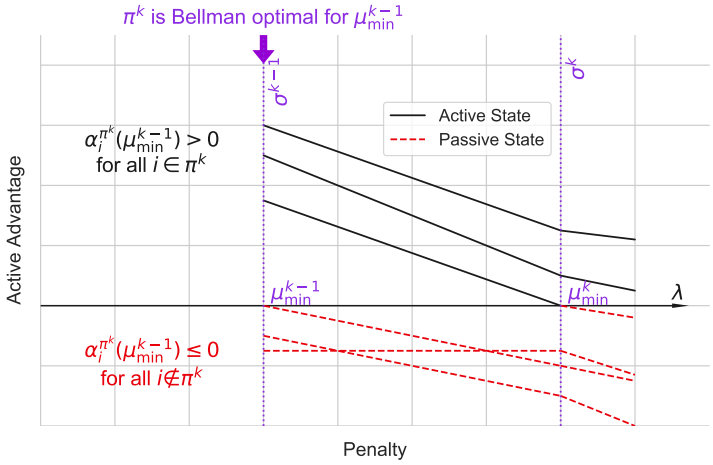


Fig. 4: Illustration of what happens when $\mu^{k-1}_{\min} < \mu^k_{\min}$ and when the test of Line 8 is successful (recall that the function $\alpha_i^{\pi^k}(\lambda)$ is affine in λ). The black lines are the advantage functions $\alpha_i^{\pi^k}(\lambda)$ of active state $i \in \pi^k$. The dashed red lines are the advantage functions $\alpha_i^{\pi^k}(\lambda)$ of passive state $i \notin \pi^k$.

Suppose that the induction is true until iteration $k-1$ and that the algorithm completes iteration k. If $\mu^{k-1}_{\min} = \mu^k_{\min}$, (A) is trivial and (B) is a direct consequence of the definition of $\mu^k_{\min}$ and of Lemma 2. Consider now that $\mu^{k-1}_{\min} < \mu^k_{\min}$ and observe what happens in Figure 4. By the induction hypothesis, π^k is Bellman optimal for the penalty $\mu^{k-1}_{\min}$. Moreover, by definition of $\mu^k_{\min}$, together with $\mu^{k-1}_{\min} < \mu^k_{\min}$, $\alpha_i^{\pi^k}(\mu^{k-1}_{\min}) \neq 0$ for all $i \in \pi^k$. Hence:

$$\alpha_i^{\pi^k}(\mu^{k-1}_{\min}) > 0 \text{ for } i \in \pi^k \qquad \text{and} \qquad \alpha_i^{\pi^k}(\mu^{k-1}_{\min}) \leq 0 \text{ for } i \notin \pi^k.$$

Finally, thanks to the test on Line 8 of the algorithm, we have:

$$\alpha_i^{\pi^k}(\mu^k_{\min}) \geq 0 \text{ for } i \in \pi^k \qquad \text{and} \qquad \alpha_i^{\pi^k}(\mu^k_{\min}) < 0 \text{ for } i \notin \pi^k. \tag{6}$$

In consequence, for each $\lambda \in (\mu^{k-1}_{\min}, \mu^{k}_{\min})$, $\alpha_i^{\pi^k}(\lambda) > 0$ for $i \in \pi^k$ and $\alpha_i^{\pi^k}(\lambda) < 0$ for $i \notin \pi^k$. Lemma 2 implies that π^k is the unique Bellman optimal policy for each $\lambda \in (\mu^{k-1}_{\min}, \mu^{k}_{\min})$. This shows (A). Also, (6) implies that π^k is Bellman optimal for the penalty $\mu^k_{\min}$. Combine this with the fact that π^{k+1} is unichain, Lemma 2 implies that π^{k+1} is Bellman optimal for $\mu^k_{\min}$ and $\boldsymbol{\alpha}^{\pi^{k+1}}(\mu^k_{\min}) = \boldsymbol{\alpha}^{\pi^k}(\mu^k_{\min})$. This shows (B). So, the induction is also true for iteration k.

This shows that the induction property is true for all $k \in [n]$. In particular, when $\pi^{n+1} := \emptyset$ is unichain, $\alpha_i^{\pi^{n+1}}$ is decreasing in λ as we prove later in (20) that $\frac{\partial \alpha_i^{\pi^{n+1}}}{\partial \lambda} = -1$. Combine this with the fact that $\alpha_i^{\pi^{n+1}}(\mu^n_{\min}) = \alpha_i^{\pi^n}(\mu^n_{\min}) \leq 0$ for all i, Lemma 2 implies that π^{n+1} is the unique Bellman optimal policy for $\lambda > \mu^n_{\min}$. We simply set $\mu^{n+1}_{\min} := +\infty$. To sum up, there are two cases for which the algorithm outputs that the arm is indexable:

1. if the algorithm goes until the end of iteration n, then the sequence of values $\{\mu^k_{\min}\}_{k\in[n+1]}$ and of policies $\{\pi^k\}_{k\in[n+1]}$ satisfies the conditions of Lemma 1(iii) and the arm is indexable.
2. if the algorithm stops at iteration k because $\mu^k_{\min} = +\infty$, then one can set $\mu^{k+1}_{\min} := \cdots := \mu^{n+1}_{\min} := +\infty$ and define a sequence of policies $\pi^{k+1} \supsetneq \cdots \supsetneq \pi^{n+1} := \emptyset$ by eliminating all states of π^k in an arbitrary order. These sequences satisfies the conditions of Lemma 1(iii) and the arm is indexable.

Proof of (ii) – Our algorithm outputs non-indexable if there exists an iteration k and a state $j \notin \pi^k$, such that $\alpha_j^{\pi^k}(\mu^k_{\min}) \geq 0$ when $\mu^{k-1}_{\min} < \mu^k_{\min}$. We know that π^k is Bellman optimal for $\mu^{k-1}_{\min}$, otherwise the algorithm would have stopped before. Assume that:

All Bellman optimal policies for any $\lambda \in (\mu^{k-1}_{\min}, \mu^k_{\min}]$ are included in π^k. (7)

We will see that this assumption leads to a contradiction. We distinguish two possibilities:

1. π^k is Bellman optimal for the penalty $\mu^k_{\min}$ – This implies that $\alpha_i^{\pi^k}(\mu^k_{\min}) \leq 0$ for all $i \notin \pi^k$ which, together with $\alpha_j^{\pi^k}(\mu^k_{\min}) \geq 0$, implies that $\alpha_j^{\pi^k}(\mu^k_{\min}) = 0$. By Lemma 2, this would imply that $\pi^k \cup \{j\}$ is Bellman optimal for $\mu^k_{\min}$. This is in contradiction with (7).
2. π^k is not Bellman optimal for $\mu^k_{\min}$ – In this case, we denote by $\tilde{\lambda}$ the smallest penalty $\lambda \in [\mu^{k-1}_{\min}, \mu^k_{\min}]$ such that there exists $\pi \subsetneq \pi^k$ that is Bellman optimal for $\tilde{\lambda}$ (it exists because π^k is not Bellman optimal for $\mu^k_{\min}$ and we assumed (7)). By definition of $\tilde{\lambda}$, π and π^k are both Bellman optimal for the penalty $\tilde{\lambda}$. Let $i \in \pi^k \setminus \pi$. By Lemma 2, this implies that $\alpha_i^{\pi^k}(\tilde{\lambda}) = 0$. The problem is that by definition, $\mu^k_{\min}$ is the smallest penalty λ for which there exists $i \in \pi^k$ such that $\alpha_i^{\pi^k}(\lambda) = 0$. This implies that $\tilde{\lambda} = \mu^k_{\min}$ which in turn implies that π^k is optimal for $\mu^k_{\min}$. This leads to a contradiction.

This shows that neither case 1 nor 2 are possible. So, (7) cannot be true. In consequence, the negation of (7) is true: there exists $\lambda > \mu^{k-1}_{\min}$ and $\pi \not\subseteq \pi^k$ such that π is Bellman optimal for λ. This contradicts Definition 1 and therefore implies that the arm is not indexable.

Proof of (iii) – if our algorithm outputs multichain, then the arm is multichain. This is straightforward based on the definition of multichain MDP. □

We should note that by Line 15, it is possible to have $\mu^k_{\min} = \mu^{k-1}_{\min}$. This happens when several states have the same value of Whittle index. This is not problematic because we are sure that $\sigma^k \neq \sigma^{k-1}$ by Line 16.

In the proof of Theorem 1(i), we showed that when policy $[n]$ is unichain, the function $\alpha_i^{\pi^1}(\lambda)$ is decreasing in λ which implies that it crosses the line 0 at some finite value μ_i^1. This implies that for an indexable arm, if $[n]$ is unichain then the Whittle index are all strictly larger than $-\infty$. A symmetric argument shows that if policy $\emptyset$ is unichain, then all Whittle index are strictly smaller than $+\infty$. This implies the following result.

Corollary 3. *Given a unichain arm with n states, if the arm is indexable, then the indices of the n states are finite: $\lambda_i \notin \{-\infty, +\infty\}$ for all $i \in [n]$.*

This is not necessarily true for multichain arms (see the discussion in Appendix A.3.)

3.4 Naive implementation of Algorithm 1 (in $O(n^4)$)

For a given penalty λ, we consider a policy π that is Bellman optimal and unichain. Recall that $g^*(\lambda)$ is the maximal gain, and $\boldsymbol{h}^\pi(\lambda) \in \mathbb{R}^n$ is a solution of (4). We consider $\boldsymbol{h}^\pi(\lambda)$ such that $h_1^\pi(\lambda) = 0$. Recall from (4) that for all $i \in [n]$:

$$g^*(\lambda) + h_i^\pi(\lambda) = r_i^{\pi_i} - \lambda\pi_i + \sum_{j=1}^{n} P_{ij}^{\pi_i} h_j^\pi(\lambda). \tag{8}$$

The above system is a system of $n+1$ linear equations with $n+1$ variables (the additional equation begins with $h_1^\pi(\lambda) = 0$). As π is unichain, the maximal gain $g^*(\lambda)$ and $\boldsymbol{h}^\pi(\lambda)$ are uniquely determined by the system of linear equations (8), together with the condition that $h_1^\pi(\lambda) = 0$. Note that in (8) the sum is for $j = 1$ to n. Since $h_1^\pi(\lambda) = 0$, it can be transformed into a sum from $j = 2$ to n.

Let us define the vector $\boldsymbol{v}^\pi(\lambda) := [g^*(\lambda)\ h_2^\pi(\lambda)\ \ldots\ h_n^\pi(\lambda)]^\top$ which is similar to the vector $\boldsymbol{h}^\pi(\lambda)$ in which we replaced $h_1^\pi(\lambda)$ by $g^*(\lambda)$. We can write Equation (8) under a matrix form as:

$$\boldsymbol{A}^\pi \boldsymbol{v}^\pi(\lambda) = \boldsymbol{r}^\pi - \lambda\boldsymbol{\pi}, \tag{9}$$

where $\boldsymbol{r}^\pi$ is the reward vector under π: $\boldsymbol{r}^\pi := [r_1^{\pi_1}\ \ldots\ r_n^{\pi_n}]^\top$, $\boldsymbol{\pi} := [\pi_1\ \ldots\ \pi_n]^\top$, and $\boldsymbol{A}^\pi$ is the following square matrix:

$$\boldsymbol{A}^\pi := \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ & & \ddots & \\ 1 & & & 1 \end{bmatrix} - \begin{bmatrix} 0 & P_{12}^{\pi_1} & \ldots & P_{1n}^{\pi_1} \\ 0 & P_{22}^{\pi_2} & \ldots & P_{2n}^{\pi_2} \\ & \vdots & & \vdots \\ 0 & P_{n2}^{\pi_n} & \ldots & P_{nn}^{\pi_n} \end{bmatrix} = \begin{bmatrix} 1 & -P_{12}^{\pi_1} & \ldots & -P_{1n}^{\pi_1} \\ 1 & 1 - P_{22}^{\pi_2} & \ldots & -P_{2n}^{\pi_2} \\ \vdots & & & \\ 1 & -P_{n2}^{\pi_n} & \ldots & 1 - P_{nn}^{\pi_n} \end{bmatrix} \tag{10}$$

As we show in Lemma 7, the matrix $\boldsymbol{A}^\pi$ is invertible if and only if policy π is unichain. In consequence, $\boldsymbol{v}^\pi$ is an affine function of λ:

$$\boldsymbol{v}^\pi(\lambda) = (\boldsymbol{A}^\pi)^{-1}(\boldsymbol{r}^\pi - \lambda\boldsymbol{\pi}) = (\boldsymbol{A}^\pi)^{-1}\boldsymbol{r}^\pi - \lambda(\boldsymbol{A}^\pi)^{-1}\boldsymbol{\pi} \tag{11}$$

For a state i, let $\delta_i := r_i^1 - r_i^0$, $\Delta_{i1} := 0$ and $\Delta_{ij} := P_{ij}^1 - P_{ij}^0$ for $j \in \{2, \ldots, n\}$. By definition of the advantage function of (5), we have:

$$\boldsymbol{\alpha}^\pi(\lambda) = \boldsymbol{\delta} - \lambda\mathbf{1} + \boldsymbol{\Delta}\boldsymbol{v}^\pi(\lambda).$$

For each active state i, we want to find the smallest penalty $\mu_i^k \geq \mu_{\min}^{k-1}$ such that $\alpha_i^{\pi^k}(\mu_i^k) = 0$. Suppose that π^{k-1} and π^k are unichain and π^{k-1} is Bellman optimal for $\mu_{\min}^{k-1}$. By Lemma 2, $\boldsymbol{\alpha}^{\pi^k}(\mu_{\min}^{k-1}) = \boldsymbol{\alpha}^{\pi^{k-1}}(\mu_{\min}^{k-1})$. Let $\boldsymbol{d}^{\pi^k} := -(\boldsymbol{A}^{\pi^k})^{-1}\boldsymbol{\pi}^k$. By (11), $\boldsymbol{\alpha}^{\pi^k}(\lambda)$ is a linear function of λ whose derivative is $-(\mathbf{1} - \boldsymbol{\Delta}\boldsymbol{d}^{\pi^k})$. In particular, $\boldsymbol{\alpha}^{\pi^k}(\lambda) = \boldsymbol{\alpha}^{\pi^{k-1}}(\mu_{\min}^{k-1}) - (\lambda - \mu_{\min}^{k-1})(\mathbf{1} - \boldsymbol{\Delta}\boldsymbol{d}^{\pi^k})$. Thus, $\alpha_i^{\pi^k}(\lambda) = 0$ if and only if

$$\alpha_i^{\pi^{k-1}}(\mu_{\min}^{k-1}) = (\lambda - \mu_{\min}^{k-1})(1 - \sum_{j=2}^{n} \Delta_{ij} d_j^{\pi^k}). \tag{12}$$

Recall that $\alpha_i^{\pi^{k-1}}(\mu_{\min}^{k-1})$ is non-negative for active state $i \in \pi^k$. The value μ_i^k is the smallest $\lambda \geq \mu_{\min}^{k-1}$ that satisfies Equation (12). There are three cases:

1. if $\alpha_i^{\pi^{k-1}}(\mu_{\min}^{k-1}) = 0$, then $\mu_i^k := \mu_{\min}^{k-1}$;
2. if $\alpha_i^{\pi^{k-1}}(\mu_{\min}^{k-1}) > 0$ and
   (a) if $1 - \sum_{j=2}^{n} \Delta_{ij} d_j^{\pi^k} > 0$, then

$$\mu_i^k := \mu_{\min}^{k-1} + \frac{\alpha_i^{\pi^{k-1}}(\mu_{\min}^{k-1})}{1 - \sum_{j=2}^{n} \Delta_{ij} d_j^{\pi^k}}; \tag{13}$$

   (b) if $1 - \sum_{j=2}^{n} \Delta_{ij} d_j^{\pi^k} \leq 0$, then $\mu_i^k := +\infty$.

This shows that, for a given k, computing $\mu_{\min}^k$ of Line 7 can be done in $O(n^3)$: A first part in $O(n^3)$ to compute the inverse of matrix $\boldsymbol{A}^\pi$ and to compute $\boldsymbol{d}^\pi$, plus some smaller order terms to compute the solutions of (12). Similarly, the test in Line 8 of Algorithm 1 can also be implemented in $O(n^3)$ by using $\boldsymbol{\alpha}^{\pi^k}(\mu_{\min}^k) = \boldsymbol{\alpha}^{\pi^{k-1}}(\mu_{\min}^{k-1}) - (\mu_{\min}^k - \mu_{\min}^{k-1})(\mathbf{1} - \boldsymbol{\Delta}\boldsymbol{d}^{\pi^k})$ with the convention that when $\mu_{\min}^k = +\infty$, $+\infty \times 0 = 0$ and $+\infty \times x = \text{sign}(x)\infty$ for any $x \neq 0$. This leads to an overall complexity of $O(n^4)$ for Algorithm 1 that contains n loops each having a $O(n^3)$ complexity. If at some iteration k the matrix $\boldsymbol{A}^{\pi^k}$ is not invertible, then Lemma 7 implies that π^k is multichain. In consequence, the algorithm outputs multichain and stops. We integrate this in the newer version of our algorithm below.

4 The $(2/3)n^3 + o(n^3)$ algorithm

This section describes a way to implement Algorithm 1 efficiently using $O(n^3)$ operations. The main idea is to use the Sherman-Morrison formula to compute in $O(n^2)$ the active advantage vector $\boldsymbol{\alpha}^{\pi^k}(\lambda)$ associated to π^k from the one associated to π^{k-1}. This leads to a $O(n^3)$ algorithm. Once this main idea is in place, we show how to avoid unnecessary computations to obtain an algorithm that performs $(2/3)n^3 + o(n^3)$ arithmetic operations.

4.1 Additional notations

In order to obtain a more efficient and compact algorithm, for an iteration k and a state i, we define $y_i^k := \sum_{j=2}^n \Delta_{ij} d_j^{\pi^k}$ and $z_i^k := \alpha_i^{\pi^k}(\mu_{\min}^k)$, where $d_j^{\pi^k}$, Δ_{ij} and $\alpha_i^{\pi^k}$ are as in (13). Equation (13) can be rewritten as

$$\mu_i^k = \mu_{\min}^{k-1} + \frac{z_i^{k-1}}{1 - y_i^k}. \tag{14}$$

The above equation can be used to compute μ_i^k and $\mu_{\min}^k$ easily from y_i^k and z_i^{k-1}. Indeed, from the previous section, we have $\alpha_i^{\pi^k}(\mu_{\min}^k) = \alpha_i^{\pi^{k-1}}(\mu_{\min}^{k-1}) - (\mu_{\min}^k - \mu_{\min}^{k-1})(1 - \sum_{j=2}^n \Delta_{ij} d_j^{\pi^k})$ which translates into

$$z_i^k = z_i^{k-1} - (\mu_{\min}^k - \mu_{\min}^{k-1})(1 - y_i^k). \tag{15}$$

This shows that the critical values to compute are the variables y_i^k. In the remainder of this section, we show that the quantity y_i^k can be computed efficiently by a recursive formula.

4.2 Application of the Sherman-Morrison formula

To compute $y_i^{k+1} := \sum_{j=2}^n \Delta_{ij} d_j^{\pi^{k+1}}$, we need to compute the quantities $\boldsymbol{d}^{\pi^{k+1}} := -(\boldsymbol{A}^{\pi^{k+1}})^{-1}\boldsymbol{\pi}^{k+1}$. This requires the inverse of $\boldsymbol{A}^{\pi^{k+1}}$. By definition of π^k, two policies π^k and π^{k+1} differ by exactly one state: $\boldsymbol{\pi}^{k+1} = \boldsymbol{\pi}^k - \boldsymbol{e}_{\sigma^k}$ where $\boldsymbol{e}_j$ denotes the column vector with a 1 in jth coordinate and 0's elsewhere. Also by definition of $\boldsymbol{A}^\pi$ in (10), the two matrices $\boldsymbol{A}^{\pi^k}$ and $\boldsymbol{A}^{\pi^{k+1}}$ differ only at the row σ^k:

$$\boldsymbol{A}^{\pi^{k+1}} = \boldsymbol{A}^{\pi^k} + \boldsymbol{e}_{\sigma^k}\boldsymbol{\Delta}_{\sigma^k} \tag{16}$$

where $\boldsymbol{\Delta}_{\sigma^k}$ is a row vector defined as in the previous section.

One can efficiently compute the inverse of matrix $\boldsymbol{A}^{\pi^{k+1}}$ from the one of $\boldsymbol{A}^{\pi^k}$ by using the Sherman-Morrison formula, which says that if $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is an invertible square matrix and $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^n$ are two column vectors, then the

matrix $\boldsymbol{A}+\boldsymbol{p}\boldsymbol{q}^\top$ is invertible if and only if $1+\boldsymbol{q}^\top\boldsymbol{A}^{-1}\boldsymbol{p}\neq 0$ and if $\boldsymbol{A}+\boldsymbol{p}\boldsymbol{q}^\top$ is invertible, then:

$$(\boldsymbol{A}+\boldsymbol{p}\boldsymbol{q}^\top)^{-1}=\boldsymbol{A}^{-1}-\frac{\boldsymbol{A}^{-1}\boldsymbol{p}\boldsymbol{q}^\top\boldsymbol{A}^{-1}}{1+\boldsymbol{q}^\top\boldsymbol{A}^{-1}\boldsymbol{p}}.$$

Let $X_{ij}^k := \boldsymbol{\Delta}_i(\boldsymbol{A}^{\pi^k})^{-1}\boldsymbol{e}_j$. Following (16), we can apply the Sherman-Morrison formula with matrix $\boldsymbol{A}^{\pi^k}$, and vectors $\boldsymbol{p}=\boldsymbol{e}_{\sigma^k}$ and $\boldsymbol{q}^\top=\boldsymbol{\Delta}_{\sigma^k}$. After some simplification, we get:

$$\begin{aligned} X_{ij}^{k+1} &:= \boldsymbol{\Delta}_i(\boldsymbol{A}^{\pi^{k+1}})^{-1}\boldsymbol{e}_j = \boldsymbol{\Delta}_i(\boldsymbol{A}^{\pi^k}+\boldsymbol{e}_{\sigma^k}\boldsymbol{\Delta}_{\sigma^k})^{-1}\boldsymbol{e}_j \\ &= X_{ij}^k - \frac{X_{i\sigma^k}^k}{1+X_{\sigma^k\sigma^k}^k}X_{\sigma^k j}^k \qquad (17) \\ \text{In particular, } X_{i\sigma^k}^{k+1} &= \frac{X_{i\sigma^k}^k}{1+X_{\sigma^k\sigma^k}^k}. \end{aligned}$$

Before computing X_{ij}^{k+1}, we need to verify that π^{k+1} is unichain. With the help of Lemma 7 and the Sherman-Morrison formula, this can be done easily: π^{k+1} is unichain if and only if $1+X_{\sigma^k\sigma^k}^k\neq 0$.

For $y_i^{k+1} := \boldsymbol{\Delta}_i\boldsymbol{d}^{\pi^{k+1}} = -\boldsymbol{\Delta}_i(\boldsymbol{A}^{\pi^{k+1}})^{-1}\boldsymbol{\pi}^{k+1}$, we use $\boldsymbol{\pi}^{k+1}=\boldsymbol{\pi}^k-\boldsymbol{e}_{\sigma^k}$ and apply the Sherman-Morrison formula to get:

$$\begin{aligned} y_i^{k+1} &= -\boldsymbol{\Delta}_i(\boldsymbol{A}^{\pi^k}+\boldsymbol{e}_{\sigma^k}\boldsymbol{\Delta}_{\sigma^k})^{-1}(\boldsymbol{\pi}^k-\boldsymbol{e}_{\sigma^k}) \\ &= -\boldsymbol{\Delta}_i(\boldsymbol{A}^{\pi^k})^{-1}(\boldsymbol{\pi}^k-\boldsymbol{e}_{\sigma^k}) + \frac{\boldsymbol{\Delta}_i(\boldsymbol{A}^{\pi^k})^{-1}\boldsymbol{e}_{\sigma^k}\boldsymbol{\Delta}_{\sigma^k}(\boldsymbol{A}^{\pi^k})^{-1}}{1+\boldsymbol{\Delta}_{\sigma^k}(\boldsymbol{A}^{\pi^k})^{-1}\boldsymbol{e}_{\sigma^k}}(\boldsymbol{\pi}_k-\boldsymbol{e}_{\sigma^k}) \\ &= y_i^k + X_{i\sigma^k}^k + \frac{X_{i\sigma^k}^k(-y_{\sigma^k}^k - X_{\sigma^k\sigma^k}^k)}{1+X_{\sigma^k\sigma^k}^k} \\ &= y_i^k + \frac{X_{i\sigma^k}^k(1-y_{\sigma^k}^k)}{1+X_{\sigma^k\sigma^k}^k} = y_i^k + (1-y_{\sigma^k}^k)X_{i\sigma^k}^{k+1} \qquad (18) \end{aligned}$$

The above formula indicate how to compute $\boldsymbol{y}^{k+1}$ from $\boldsymbol{y}^k$. To complete this analysis, let us show that $\boldsymbol{y}^1=\boldsymbol{0}$. For a given policy π, the vector $\boldsymbol{d}^\pi$ satisfies the same equation as Equation (9) but replacing $r_i^{\pi_i}-\lambda\pi_i$ by $-\pi_i$. This implies that for $\boldsymbol{\pi}=\boldsymbol{\pi}^1=[1\ \ldots\ 1]^\top$, one has $\boldsymbol{d}^\pi=[-1\ 0\ \ldots\ 0]^\top$ as d_1^π is the long-run average reward of a Markov reward process whose reward is negative one in all states and $d_2^\pi,\ldots,d_n^\pi$ is the bias of this process. This shows that for all i, one has $y_i^1 := \sum_{j=2}^n \Delta_{ij}d_j^{\pi^1}=0$. Moreover, by (11), one has

$$\boldsymbol{\alpha}^{\pi^1}(\lambda)=\boldsymbol{\delta}-\lambda\boldsymbol{1}+\boldsymbol{\Delta}(\boldsymbol{A}^{\pi^1})^{-1}\boldsymbol{r}^{\pi^1}-\lambda\underbrace{\boldsymbol{\Delta}(\boldsymbol{A}^{\pi^1})^{-1}\boldsymbol{\pi}^1}_{=:\boldsymbol{y}^1}$$

$$= \boldsymbol{\delta} - \lambda \mathbf{1} + \boldsymbol{X}^{1} \boldsymbol{r}^{\pi^{1}}. \tag{19}$$

Finally, for $\boldsymbol{\pi}^{n+1} = [0 \ \ldots \ 0]^{\top}$, one has

$$\boldsymbol{\alpha}^{\pi^{n+1}}(\lambda) = \boldsymbol{\delta} - \lambda \mathbf{1} + \boldsymbol{\Delta}(\boldsymbol{A}^{\pi^{n+1}})^{-1} \boldsymbol{r}^{\pi^{n+1}}. \tag{20}$$

4.3 Detailed algorithm

Equation (14) shows how to compute μ_i^k from the values of y_i^k and z_i^{k-1} while (18), (17) and (15) show how to compute the values of $\boldsymbol{y}$, $\boldsymbol{z}$ and $\boldsymbol{X}$ recursively in k. In order to compute $\mu_{\min}^k$ and σ^k, one only needs to compute the values μ_i^k for $i \in \pi^k$. Once $\mu_{\min}^k = \min_{i \in \pi^k} \mu_i^k$ is determined, if $\mu_{\min}^{k-1} < \mu_{\min}^k$, then Line 8 of Algorithm 1 can be performed, based on (15), by checking if $z_i^k \geq 0$ for some $i \in [n] \setminus \pi^k$.

This leads to Algorithm 2 that can be decomposed as follows:

1. In Lines 1 to 8, we initialize the various variables. The main complexity of this part is to compute the matrix $\boldsymbol{X}^1$, which is equivalent to solving the linear system $\boldsymbol{X}\boldsymbol{A}^{\pi^1} = \boldsymbol{\Delta}$. It can be done by inverting the matrix $\boldsymbol{A}^{\pi^1}$ and multiplying this by the matrix $\boldsymbol{\Delta}$. This can be done in a subcubic complexity by using for instance Strassen's algorithm [39].
2. We then enter the main loop:
   - We update the vectors $\boldsymbol{\mu}$, $\boldsymbol{z}$ by using Equations (14) and (15), and test indexability. This costs $O(n)$ operations per iteration, thus $O(n^2)$ in total.
   - We update the vector $\boldsymbol{X}$ according to (17). The "naive" way to do so is to use Subroutine 3. At iteration k this costs $2kn$ arithmetic operations if we test indexability, and $2\sum_{l=1}^{k}(n-l)$ if we do not test indexability. The total complexity of computing $\boldsymbol{X}$ is $n^3 + O(n^2)$ arithmetic operations if we test indexability and $(2/3)n^3 + O(n^2)$ if we do not. A detailed study of the arithmetic complexity is provided in Appendix C, where we also provide details on how to efficiently implement the algorithm, including how to optimize the cost of memory access.
   - In Line 11, we update the vector $\boldsymbol{y}$ by using Equation (18), which costs $O(n)$ per iteration.
3. Testing if π^{n+1} is unichain can be done in $O(n^2)$ by Tarjan's strongly connected component algorithm.

Hence, the total complexity of this algorithm is $n^3 + o(n^3)$ if we test indexability and $(2/3)n^3 + o(n^3)$ if we do not test indexability. Without testing the indexability, our algorithm has the same main complexity as [29]. However, the algorithm of [29] computes Whittle index only for an arm that is PCL-indexable. Hence we can claim that our algorithm is the first algorithm that computes Whittle index with cubic complexity for general indexable restless bandits. It is also the first algorithm with cubic complexity that tests non-indexability of restless bandits composed of unichain arms. Note that the ties are broken arbitrarily for Lines 7 and 15 of Algorithm 2.

Algorithm 2 Given a n-state arm, test indexability and compute Whittle indices (if indexable).

1: Set $\pi^1 := [n]$, $k_0 = 1$, $\Delta_{i1} := 0, \Delta_{ij} := P^1_{ij} - P^0_{ij}, \forall i \in [n], j \in \{2, \dots, n\}$, $\boldsymbol{y}^1 = \mathbf{0}$ and $\boldsymbol{A}^{\pi^1}$ is defined by (10).
2: **if** π^1 is multichain **then**
3: **return** the arm is multichain
4: **end if**
5: Set $\boldsymbol{X}^1 := \boldsymbol{\Delta}(\boldsymbol{A}^{\pi^1})^{-1}$
6: Set $\boldsymbol{\mu}^1 := \boldsymbol{r}^1 - \boldsymbol{r}^0 + \boldsymbol{X}^1 \boldsymbol{r}^1$
7: Let $\sigma^1 := \arg\min_{i \in \pi^1} \mu^1_i$, $\lambda_{\sigma^1} = \mu^1_{\min} := \mu^1_{\sigma^1}$, and $\pi^2 := \pi^1 \setminus \{\sigma^1\}$
8: Set $\boldsymbol{z}^1 = \boldsymbol{\mu}^1 - \mu^1_{\min}\mathbf{1}$
9: **for** $k = 2$ **to** n **do**
10: Update_X$(k-1)$ ▷ Here we call Subroutine 3 or Subroutine 4
11: Set $\boldsymbol{y}^k = \boldsymbol{y}^{k-1} + (1 - y^{k-1}_{\sigma^{k-1}})\boldsymbol{X}^k_{:\sigma^{k-1}}$
12: **for** $i \in \pi^k$ **do**
13: Set $\mu^k_i := \begin{cases} \mu^{k-1}_{\min}, & \text{if } z^{k-1}_i = 0 \\ \mu^{k-1}_{\min} + \dfrac{z^{k-1}_i}{1 - y^k_i}, & \text{if } z^{k-1}_i > 0 \text{ and } 1 - y^k_i > 0 \\ +\infty, & \text{otherwise} \end{cases}$
14: **end for**
15: Let $\sigma^k := \arg\min_{i \in \pi^k} \mu^k_i$ and $\mu^k_{\min} := \mu^k_{\sigma^k}$
16: Set $\boldsymbol{z}^k = \boldsymbol{z}^{k-1} - (\mu^k_{\min} - \mu^{k-1}_{\min})(\mathbf{1} - \boldsymbol{y}^k)$
17: **if** $\mu^{k-1}_{\min} < \mu^k_{\min}$ and $z^k_i \geq 0$ for some $i \in [n] \setminus \pi^k$ **then**
18: **return** the arm is not indexable
19: **end if**
20: **if** $\mu^k_{\min} = +\infty$ **then**
21: Set $\lambda_i := +\infty$ for all $i \in \pi^k$
22: **return** the arm is indexable and the indices are $\{\lambda_i\}_{i \in [n]}$.
23: **end if**
24: Set $\lambda_{\sigma^k} = \mu^k_{\min}$ and $\pi^{k+1} := \pi^k \setminus \{\sigma^k\}$
25: **end for**
26: **if** π^{n+1} is multichain **then**
27: **return** the arm is multichain
28: **end if**
29: **return** the arm is indexable and the indices are $\{\lambda_i\}_{i \in [n]}$.

Remark: Equivalently, instead of calling Subroutine 3 at Line 10, one could do the following update for all $j \in \pi^{k+1}$ and $i \in [n]$ (or $i \in \pi^{k+1}$ if we do not test indexability):

$$X^{k+1}_{ij} = X^k_{ij} - \frac{X^k_{i\sigma^k}}{1 + X^k_{\sigma^k\sigma^k}} X^k_{\sigma^k j}. \tag{21}$$

Subroutine 3 Update_X(k)

for $\ell = 1$ to $k-1$ **do**
  for $i \in [n]$ **do** ▷ or $i \in \pi^{\ell+1}$ if we do not test indexability.
    $X^{\ell+1}_{i\sigma^k} = X^{\ell}_{i\sigma^k} - X^{\ell+1}_{i\sigma^\ell} X^{\ell}_{\sigma^\ell\sigma^k}$
  end for
end for
if $1 + X^k_{\sigma^k\sigma^k} = 0$ **then**
  return the arm is multichain
end if
for $i \in [n]$ **do** ▷ or $i \in \pi^{k+1}$ if we do not test indexability.
  $X^{k+1}_{i\sigma^k} = \dfrac{X^k_{i\sigma^k}}{1 + X^k_{\sigma^k\sigma^k}}$
end for

This iterative update is very close to the one used in [29, 34] for discounted restless bandit. This results in an algorithm that has the same total complexity as Subroutine 3 (of $n^3 + O(n^2)$ or $(2/3)n^3 + O(n^2)$ with or without the indexability test) because both algorithms will have computed the same values of X^k_{ij}. The reason to use Subroutine 3 is that, as we will see in the next section, not all values of X^k_{ij} are needed at iteration k: in particular, for $\ell < k$, the computation of the value $X^{\ell}_{i\sigma^{k-1}}$ has no interest per say and is only useful because it allows to recursively compute $X^k_{i\sigma^{k-1}}$. In the section below, we show how to reduce the cost by avoiding the computation of $X^{\ell}_{i\sigma^{k-1}}$ when ℓ is much smaller than k. We comment more on the differences with [29, 34] in Appendix E and in particular we explain why our approach can be tuned into a subcubic algorithm while (21) cannot.

5 The subcubic algorithm

5.1 Main idea: recomputing $\boldsymbol{X}^k$ from $\boldsymbol{A}^{\pi^k}$ periodically

The main computational burden of Algorithm 2 is concentrated on two lines: on Line 5 where we compute $\boldsymbol{X}^1$ by solving a linear system, and on Line 10 where we compute the column vector $\boldsymbol{X}^k_{:\sigma^{k-1}}$ from $\boldsymbol{X}^1_{:\sigma^{k-1}}$. The remainder of the code runs in $O(n^2)$ operations and is therefore negligible for large matrices. In fact, the computation of $\boldsymbol{X}^1$ is done by solving a linear system, which can be computed by using a subcubic algorithm (like Strassen [39]). In this section, we show how to optimize our algorithm by reducing the complexity of the update_X() function, at the price of recomputing the full matrix $\boldsymbol{X}^{k_0}$ from $\boldsymbol{A}^{\pi^{k_0}}$ periodically.

At iteration k of Algorithm 2, the quantities $X^k_{i\sigma^{k-1}}$ are used at Line 11 to obtain the values y^k_i. The matrix $\boldsymbol{X}^k$ is defined as $\boldsymbol{X}^k := \boldsymbol{\Delta}(\boldsymbol{A}^{\pi^k})^{-1}$. It also

satisfies Equation (17), that is, for an iteration ℓ and states i and j, we have:

$$X_{ij}^{\ell+1} = X_{ij}^{\ell} - X_{i\sigma^\ell}^{\ell+1} X_{\sigma^\ell j}^{\ell}. \tag{22}$$

The way Subroutine 3 is implemented is to initialize $\boldsymbol{X}^1 := \boldsymbol{\Delta}(\boldsymbol{A}^{\pi^1})^{-1}$ and then use (22) recursively to compute the column vector $\boldsymbol{X}^k_{:\sigma^{k-1}}$ from $\boldsymbol{X}^1_{:\sigma^{k-1}}$ at each iteration.

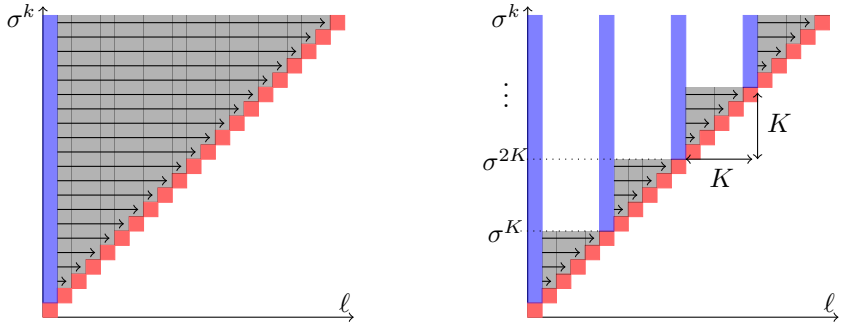
Here, we propose an alternative formulation which consists in recomputing the whole matrix $\boldsymbol{X}^k := \boldsymbol{\Delta}(\boldsymbol{A}^{\pi^k})^{-1}$ every K iterations. In the meantime, we use (22) to compute the values $X^{\ell}_{i\sigma^{k-1}}$ for $\ell \in \{k_0+1, \dots, k\}$ where k_0 is the iteration at which we recomputed the whole matrix $\boldsymbol{X}^{k_0} := \boldsymbol{\Delta}(\boldsymbol{A}^{\pi^{k_0}})^{-1}$. This can be implemented by replacing the call to Subroutine 3 at Line 10 with a call to Subroutine 4.

Subroutine 4 Update_X_FMM(k)

1: **if** k is an iteration at which we recompute the whole $\boldsymbol{X}^{k+1}$ **then**
2: Set $k_0 := k+1$
3: **if** $\boldsymbol{A}^{\pi^{k_0}}$ is not invertible **then**
4: **return** the arm is multichain
5: **end if**
6: Set $\boldsymbol{X}^{k_0} := \boldsymbol{\Delta}(\boldsymbol{A}^{\pi^{k_0}})^{-1}$
7: **else**
8: **for** $\ell = k_0$ to $k-1$ **do**
9: **for** $i \in [n]$ **do** ▷ or $i \in \pi^{\ell+1}$ if we do not test indexability.
10: $X^{\ell+1}_{i\sigma^k} = X^{\ell}_{i\sigma^k} - X^{\ell+1}_{i\sigma^\ell} X^{\ell}_{\sigma^\ell \sigma^k}$
11: **end for**
12: **end for**
13: **if** $1 + X^k_{\sigma^k \sigma^k} = 0$ **then**
14: **return** the arm is multichain
15: **end if**
16: **for** $i \in [n]$ **do** ▷ or $i \in \pi^{k+1}$ if we do not test indexability.
17: $X^{k+1}_{i\sigma^k} = \dfrac{X^k_{i\sigma^k}}{1 + X^k_{\sigma^k \sigma^k}}$
18: **end for**
19: **end if**

To see why this can be more efficient, we illustrate in Figure 5 the pairs (ℓ, σ^{k-1}) for which we compute the vector $\boldsymbol{X}^{\ell}_{:\sigma^{k-1}}$, either for Subroutine 3 or Subroutine 4. In each case, a vertical blue line indicates that we recompute the whole matrix $\boldsymbol{X}^k$ by solving a linear system. The gray zone corresponds to the values (ℓ, σ^{k-1}) for which we compute $\boldsymbol{X}^{\ell}_{:\sigma^{k-1}}$ using Equation (22) and the red squares represent the vector $\boldsymbol{X}^k_{:\sigma^{k-1}}$ used at Line 11 of Algorithm 2.

For Subroutine 3, we do one matrix inversion at the beginning and then compute for all (ℓ, σ^{k-1}) with $\ell \leq k$ because the red square at value (k, σ^{k-1}) is computed starting from the vertical blue line at value $(1, \sigma^{k-1})$. For Subroutine 4, we do $\lfloor n/K \rfloor$ (here $\lfloor n/K \rfloor = 4$) full recomputation of $\boldsymbol{X}^k$, which correspond to the vertical blue lines. We gain in terms of operations because the surface of the gray zone to compute is divided by $\lfloor n/K \rfloor$.



(a) Computation load of Subroutine 3. (b) Computation load of Subroutine 4

Fig. 5: Illustration of the improvement proposed by replacing Subroutine 3 with Subroutine 4 when we do not test the indexability. For Subroutine 4, we solve more linear systems (each vertical blue line corresponds to solving a linear system) but we reduce the gray zone to compute. A linear arrow corresponds to the internal loop of Line 8 of Subroutine 4.

Note that the y-axis of Figure 5 is ordered by increasing value of σ^k (and not by increasing value of k). The value of σ^k is computed at iteration k but unknown before iteration k. This explains why in Subroutine 4, when we recompute the matrix (X^k_{ij}) at an iteration k (vertical blue lines in Figure 5(b)), we recompute it for all i, j and not just $i, j \in \{\sigma^k, \ldots, \sigma^K\}$ (which are the only values that we will use): Indeed, $\sigma^k, \ldots, \sigma^K$ are unknown at iteration k.

5.2 A subcubic algorithm for Whittle index

We now assume to have access to a subcubic matrix multiplication algorithm that satisfies the following property:

(FMM) There exists an algorithm to multiply a matrix of size $n \times n$ by a matrix of size[4] $n \times n^\gamma$ that runs in $O(n^{\omega(\gamma)})$, where $\omega : [0,1] \to [2,3]$ is a non-decreasing function.

Going back to Algorithm 2 where Line 10 is Subroutine 4, we now assume that we recompute the whole matrix $\boldsymbol{X}^k$ every $O(n^\gamma)$ iterations. The new algorithm has a subcubic complexity:

[4]We write n^γ which is possibly non-integer. For the sake of simplicity, we write n^γ and $n^{1-\gamma}$ but they should be understood as $\lfloor n^\gamma \rfloor$ and $\left\lfloor n^{1-\gamma} \right\rfloor$ respectively.

Theorem 4. *Given a n-state arm, Algorithm 2 with Subroutine 4 checks indexability and computes Whittle (and Gittins) index in time at least* $\Omega(n^{2.5})$ *and at most* $O(n^{2.5286})$ *when choosing* $\gamma = 0.5286$.

We believe that Theorem 4 is the first theoretical result that shows that Whittle index can be computed in subcubic time. As we show in Section 7, this algorithm can be directly extended to discounted index. As a byproduct, we also obtain the first subcubic algorithm to compute Gittins index.

Proof The algorithm starts by computing $\boldsymbol{X}^1$ which can be done in $O(n^{\omega(1)})$. Then, there are $n^{1-\gamma}$ times that we do:

1. We fill the "gray" mini matrices by using (22). This amounts to three for loops of size n (for i), n^γ (for k) and n^γ (for ℓ). Hence, each small gray matrix costs $O(n^{1+2\gamma})$.
2. At the end of a cycle, we recompute the full inverse by updating $(\boldsymbol{A}^{\pi^k})^{-1}$ from $(\boldsymbol{A}^{\pi^{k-n^\gamma}})^{-1}$. As we show in Lemma 3 (stated below), this can be done in $O(n^{\omega(\gamma)})$.

This implies that the algorithm has a complexity:

$$O(n^{\omega(\gamma)}) + n^{1-\gamma}\left(O(n^{1+2\gamma}) + O(n^{\omega(\gamma)})\right) = O(n^{\max\{2+\gamma, 1-\gamma+\omega(\gamma)\}}).$$

To compute the optimal value of γ minimizing this expression requires the knowledge of the function $\omega(\gamma)$ which is not known. The current state of the art only gives a lower bound ($\omega(\gamma) \geq 2$) and an upper bound described in [42].

It is shown in [42, Table 3] that $\gamma = 0.5286$ is the smallest currently known value of γ for which $\omega(\gamma) < 1 + 2\gamma$. This implies that the complexity is at most $O(n^{2.5286})$.

As for the lower bound, $\omega(\gamma) \geq 2$ implies that the complexity of the algorithm is at least $\Omega(n^{2.5})$. □

In the next lemma, $\boldsymbol{B}$ plays the role of $\boldsymbol{A}^{\pi^k}$ and $\boldsymbol{A}$ the role of $\boldsymbol{A}^{\pi^{k-n^\gamma}}$. Note that as required in the lemma, exactly n^γ rows and columns are changed between the two.

Lemma 3. *Assume (FMM). Let* $\boldsymbol{A}$ *be a square matrix whose inverse* $\boldsymbol{A}^{-1}$ *has already been computed, and let* $\boldsymbol{B}$ *be an invertible square matrix such that* $\boldsymbol{A} - \boldsymbol{B}$ *is of rank smaller than* n^γ. *Then, it is possible to compute the inverse of* $\boldsymbol{B}$ *in* $O(n^{\omega(\gamma)})$.

Proof The matrix $\boldsymbol{B}$ can be written as $\boldsymbol{B} = \boldsymbol{A} + \boldsymbol{UCV}$ where $\boldsymbol{U}$ is a $n \times n^\gamma$ matrix, $\boldsymbol{C}$ is $n^\gamma \times n^\gamma$ and $\boldsymbol{V}$ is $n^\gamma \times n$. The Sherman–Morrison–Woodbury formula [43] states that

$$\boldsymbol{B}^{-1} = (\boldsymbol{A} + \boldsymbol{UCV})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}\left(\boldsymbol{C}^{-1} + \boldsymbol{VA}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{VA}^{-1}.$$

This shows that $\boldsymbol{B}^{-1}$ can be computed by:

- Computing $\boldsymbol{D} := \boldsymbol{A}^{-1}\boldsymbol{U}$ and $\boldsymbol{E} := \boldsymbol{VA}^{-1}$: this takes $O(n^{\omega(\gamma)})$.

- Computing $\boldsymbol{F} := \left(\boldsymbol{C}^{-1} + \boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U}\right)^{-1}$: as this is the inversion of a $n^\gamma \times n^\gamma$ matrix, it can be done in $O(n^{\gamma\omega(1)})$ where $\gamma\omega(1) \leq \omega(\gamma)$.
- Computing $\boldsymbol{G} := \boldsymbol{D}\boldsymbol{F}$ and then $\boldsymbol{G}\boldsymbol{E}$: this again takes $O(n^{\omega(\gamma)})$.

Hence, computing $\boldsymbol{B}^{-1}$ can be done in $O(n^{\omega(\gamma)})$ operations for the inversion and all multiplications plus an additional $O(n^2)$ term for the subtraction and the addition. As $\omega(\gamma) \geq 2$, this concludes the proof of the lemma. □

5.3 The subcubic algorithm in practice

The complexity of $O(n^{2.5286})$ given in Theorem 4 is mainly of theoretical interest. The value $\gamma = 0.5286$ is obtained by using the best upper bound on $\omega(\gamma)$ known today which is based on the Coppersmith-Winograd algorithm and its variants. The Coppersmith-Winograd algorithm (or its variants) are, however, known as a *galactic algorithm*: the hidden constant in the $O()$ is so large that their runtime is prohibitive for any reasonable value of n. Hence, the existence of these algorithms is of theoretical interest but has limited applicability.

This does not discard the practical improvement provided by Subroutine 4 which is based on the mere fact that multiplying two matrices (or inverting a matrix) is faster than three nested loops even for matrices of moderate size. To verify this, we launched a detailed profiling of the code of Algorithm 2 with the non-optimized Subroutine 3. It shows that for a problem of dimensions 5000, the update of Line 10 takes more than 90% of the computation time, the initialization of $\boldsymbol{X}^1$ on Line 5 takes about 5% of the time and the rest of the code takes less than 1% of the running time.

Now, if inverting the full matrix takes about 5% of the execution time, and updating the gray zone takes 95%, then by doing 5 updates, one can hope to obtain an algorithm whose running time is roughly $5 \times 5 + 95/5 \approx 43\%$ the one of the original implementation. As we observe in Section 6, this is close to the gain that we obtain in practice. A general way to choose the best number of updates is used in the numerical section. It is based on the following reasoning. For large matrices (say $n \geq 10^3$), the fastest implementations of matrix multiplication and inversion are based on Strassen's algorithm [44, 45]. As we report in Appendix D, the time to solve a linear system of size n by using the default installation of `scipy` seems to run in $O(n^{2.8})$. By replacing the function $\omega(\gamma)$ used in Theorem 4 by a more practical bound ($\omega(\gamma) = 2.8$), the best value for γ becomes $\gamma = 0.9$. This indicates that our algorithm can be implemented in $O(n^{2.9})$ by doing $O(n^{0.1})$ recomputation of $\boldsymbol{X}^k$ from $\boldsymbol{A}^{\pi^k}$. Note that even for very large values of n (like $n = 15000$), $n^{0.1}$ remains quite small, *e.g.*, $15000^{0.1} \approx 2.6$. In practice, we observe that updating $\text{int}(2n^{0.1})$ times (the notation $\text{int}(x)$ indicates that it is rounded to the closest integer) gives the best performance among all algorithms, as reported in the next section.

6 Numerical experiments

In complement to our theoretical analysis, we developed a python package that implements Algorithm 2 and gives the choice of using the variant of Subroutine 3 or of Subroutine 4 to do the "update_X()" function. This package relies on three python libraries: `scipy` and `numpy` for matrix operations, and `numba` to compile the python code. To facilitate its usage, this package can be installed by using `pip install markovianbandit-pkg`.

All experiments were conducted on a laptop (Macbook Pro 2020) with an Intel Core i9 CPU at 2.3 GHz with 16GB of Memory using Python 3.6.9 :: Anaconda custom (64-bit) under macOS Big Sur version 11.6.2. The version of the packages are `scipy` version 1.5.4, `numpy` version 1.19.5 and `numba` version 0.53.1. The code of all experiments is available at https://gitlab.inria.fr/markovianbandit/efficient-whittle-index-computation.

In all of our experiments, the way we generated arms guarantees that they are almost surely unichain because all the elements on their diagonal as well as on the upper and lower diagonals are positive.

6.1 Time to compute Whittle indices

To test the implementation of our algorithm, we randomly generate restless bandit arms with n states where $n \in \{100, 1000, \dots, 15000\}$. In each case, both transition matrices are uniform probabilistic matrices: for each row of each matrix, we generate n i.i.d. entries following the exponential distribution and divide the row by its sum. This means that all matrices are dense. We use dense matrix since it is the worst case for computational interest. By running our algorithms on sparse matrix, we would expect to have faster running time. Note that all tested matrices are indexable. This is coherent with [33, 46] that report that for uniform matrices, the probability of finding a non-indexable example decreases very rapidly with the dimension n. Finally, reward vectors were generated from random Uniform[0,1) entries.

We record the runtime of the different variants of our algorithm and report the results in Table 1. Note that these results present the whole execution time of the algorithm, including the initialization phase in which $\boldsymbol{X}^1$ is computed. For each value of n, we run Algorithm 2 with four variants:

- The first two columns correspond to the $O(n^3)$ algorithm (that uses Subroutine 3 for Line 10), either (a) with the indexability test, or (b) without the indexability test.
- The last two columns correspond to the subcubic algorithm (that uses Subroutine 4 for Line 10 with $\text{int}(2n^{0.1})$ updates), either (c) with the indexability test, or (d) without the indexability test.

Our numbers show that our algorithm can compute the Whittle index in less than one second for $n = 1000$ states and slightly less than 7 minutes for $n = 15000$ states with variant (d). As expected, not doing the indexability test does improve the performance compared to doing the indexability test (here by a factor approximately 1/3 for Subroutine 3 and 1/5 for Subroutine 4).

Table 1: Running time (in seconds) of the variants of Algorithm 2

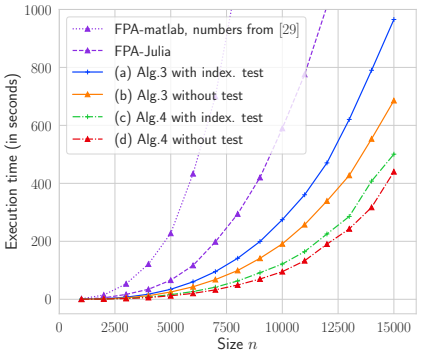
	$O(n^3)$ algorithm (Subroutine 3)		Subcubic algorithm (Subroutine 4)	
n	(a) With index. test	(b) Without test	(c) With index. test	(d) Without test
100	0.006	0.004	0.007	0.005
1000	0.2	0.2	0.2	0.2
2000	1.9	1.2	1.1	1.0
3000	7.2	5.0	3.2	2.6
4000	17	12	8	6
5000	34	25	16	12
6000	60	43	27	20
7000	95	68	42	33
8000	142	99	63	49
9000	199	141	92	70
10000	275	191	122	95
11000	361	257	164	133
12000	471	339	225	190
13000	620	428	286	243
14000	790	553	408	317
15000	965	685	501	403

More importantly, this table shows that the time when using the subcubic variant, Subroutine 4, diminishes the computation time by about 40% to 50% compared to when using Subroutine 3. Note that for $n = 100$, using the Subroutine 3 is slightly faster than using the Subroutine 4 (while both takes only a few milliseconds). This indicates that the subcubic algorithm becomes interesting when n is large enough (say $n \geq 2000$).

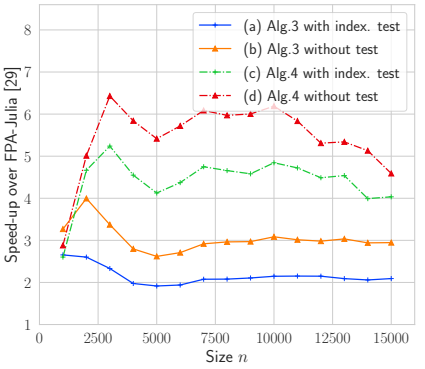
To give a visual idea of how the various variants of the algorithms compare, we plot in Figure 6a the runtime of the four variants along with two variants of the algorithms of [29]: FPA-Matlab (the original matlab implementation), and FPA-Julia: a Julia's implementation of the algorithm provided by the authors. We choose to compare to this algorithm as it was the one with the smallest complexity up to now. The numbers for FPA-Matlab are the ones reported in [29] and they are comparable to the ones that we obtained on our machine with the same algorithm. FPA-Julia is significantly faster. Hence, we plot in Figure 6b the runtime of each variant divided by the runtime of FPA-Julia. For large n, our best implementation is about 4 to 6 times faster than the best one of [29]. For instance, for $n = 15000$, our implementation takes about 7 minutes to compute the index (or 9 minutes when checking indexability) whereas FPA-Julia takes 33 minutes (and does not check indexability on the fly). In our implementation, not testing indexability reduces the computation time of $15 - 20\%$ for Subroutine 4 or $25 - 30\%$ for Subroutine 3 compared to the version that tests indexability.

It should be clear that the comparison of the computation times of our implementation versus the ones of FPA-Matlab or FPA-Julia has its limits, because we do not use the same programming language. The influence of the choice of programming language is clear when comparing FPA-Matlab and FPA-Julia: while both codes are similar, the compiled Julia code is about 5 times faster. To obtain a fairer comparison with our algorithm, we tried to

rewrite the algorithm of [29] in Python with Numba but our implementation was significantly slower than the one of FPA-Julia. We do not know if this is by lack of optimization of our code or because Julia is indeed faster. Regardless of the choice programming langages, our implementation has nevertheless a few advantages that can explain why it is faster: our algorithm has some technical advantage (a simpler internal loop, and the use of Subroutine 4), and our implementation is optimized to improve data access pattern (which consists mostly in sorting on the fly the array according to the permutation σ, see Appendix C).



(a) Runtime of our implementations as a function of the state size n.



(b) Speedup of each implementation over the data from [29] as a function of the state size n.

Fig. 6: Numerical result over 7 simulations: in each simulation, we run the algorithm over randomly generated RBs with the state size ranging over $\{1000, \ldots, 15000\}$. We plot the average runtime over 7 simulations. The solid lines represent the result of Subroutine 3 and the dashed-dot lines represent the one of Subroutine 4. The marker "+" indicates that algorithms test indexability and the triangles indicates that algorithms do not test indexability.

6.2 Statistics of indexable problems

To the best of our knowledge, our algorithm provides the first indexability test that scales well with the dimension n. We used this to answer a very natural question: given a randomly generated arm, how likely is it to be indexable? This question was partially answered in [33] that shows that when generating dense arms, the probability of generating a non-indexable arm is close to 10^{-n} for $n \in \{3, \ldots, 7\}$. This suggests that most arms are indexable. Below, we answer two questions: what happens for larger values of n, and more importantly, what happens when the state transition matrices are not dense?

To answer these questions, we consider randomly generated arms where the matrices $\boldsymbol{P}^0$ and $\boldsymbol{P}^1$ are b-diagonal matrices with b non-null diagonals. In particular, $b = 3$ corresponds to tridiagonal matrices, $b = 5$ corresponds to pentadiagonal matrices and $b = 7$ corresponds to septadiagonal matrices. We also compare with the classical case of dense matrices (which corresponds to $b = 2n - 1$). For each model, the entries are generated from the exponential distribution for each row and we divide the row by the sum of generated entries for this row. We vary n from 3 to 50 and for each case, we generate 100000 arms. We report in Table 2 the number of indexable arms for each case. Note that pentadiagonal matrices are dense matrices for $n = 3$ and septadiagonal matrices are dense matrices for $n = 4$ and do not make sense for $n = 3$, which is why no numbers are reported.

Table 2: Number of indexable problems among 100 000 randomly generated problems.

Problem size n	Tridiagonal	5-diagonal	7-diagonal	Dense
3	98 731	–	–	99 883
4	95 067	99 655	–	99 931
5	89 198	99 309	99 902	99 969
10	54 129	90 377	98 914	100 000
30	7 094	29 699	66 143	100 000
50	1 823	9 332	32 069	100 000

Based on these results, we can assert that dense models are essentially always indexable which conforms with the data reported in [33]. The situation is, however, radically different for sparse models: the number of indexable problems decreases quickly with the number of states. For instance, there are only 1 823 indexable 50-state problems among 100 000 generated tridiagonal models (*i.e.* around 1.8% are indexable). Note that a tridiagonal model is a birth-death Markov chain which is frequently used for queueing systems. Hence, it is very important to check the indexability of the problem because it is not a prevalent property for sparse models. This also calls for new efficient policies in restless multi-arm bandit problems that are not based on Whittle indices.

7 Extension to the discounted case

The model described in Section 2 corresponds to the definition of Whittle index for a time-average criterion, for which Whittle index is known to be asymptotically optimal [6] for restless bandits. Yet, Whittle index can also be defined for the discounted case [29, 34]. Notably, the discounted Whittle index simplifies into Gittins index when the bandit is rested (*i.e.*, when $\boldsymbol{P}^0 = \boldsymbol{I}$ and $\boldsymbol{r}^0 = \boldsymbol{0}$). In this section, we show how to adapt our algorithm to the discounted case. As a by product, we obtain the first subcubic algorithm to compute Gittins index rested bandit.

7.1 Discounted Whittle index

We now consider a λ-penalized MDP in which the instantaneous reward received at time $t \geq 0$ is discounted by a factor β^t, where $\beta \in (0,1)$ is called the discount factor: when executing action a in state i at time $t \geq 0$, the decision maker earns a reward $\beta^t(r_i^a - \lambda a)$. For a given policy π, we denote by $u_i^\pi(\lambda)$ the expected sum of discounted rewards earned by the decision maker when the MDP starts in state i at time 0. The vector $\boldsymbol{u}^\pi(\lambda) = [u_1^\pi(\lambda) \ \ldots \ u_n^\pi(\lambda)]^\top$ is called the value function of the policy π. From [40], it satisfies Bellman's equation, that is, for all state i we have:

$$u_i^\pi(\lambda) = r_i^{\pi_i} - \lambda\pi_i + \beta \sum_{j=1}^{n} P_{ij}^{\pi_i} u_j^\pi(\lambda). \tag{23}$$

The above equation is a linear equation, whose solution is unique because $\beta < 1$. It is given by:

$$\boldsymbol{u}^\pi(\lambda) = (\boldsymbol{I} - \beta \boldsymbol{P}^\pi)^{-1}(\boldsymbol{r}^\pi - \lambda\boldsymbol{\pi}). \tag{24}$$

For a given penalty λ and a state i, we denote by $u_i^*(\lambda) := \max_\pi u_i^\pi(\lambda)$ be the optimal value of state i. A policy π is optimal for the penalty λ, *i.e.*, $\pi \in \Pi^*(\lambda)$, if for all state $i \in [n]$, $u_i^*(\lambda) = u_i^\pi(\lambda)$. By [40] such a policy exists, $|\Pi^*(\lambda)| > 0$. As mentioned in Section 2.2.3, the distinction between Bellman optimal and gain optimal disappears in discounted MDP in which we are concerned with maximizing the value function. Similarly to the time-average criterion studied before, a β-discounted RB is called indexable if for all penalty $\lambda < \lambda'$, all $\pi \in \Pi^*(\lambda)$ and $\pi' \in \Pi^*(\lambda')$, one has $\pi \supseteq \pi'$.

7.2 Analogy between the time-average and the discounted versions

Let π be a policy and $i \in \pi$ be an active state. Similarly to average reward model studied before, the advantage of action activate over action rest in state i right before following policy π is given by, $\alpha_i^\pi(\lambda) := r_i^1 - r_i^0 - \lambda + \beta \sum_{j=1}^n (P_{ij}^1 - P_{ij}^1) u_j^\pi(\lambda)$. Then $\alpha_i^\pi(\lambda) = 0$ if and only if $\lambda = \delta_i + \sum_{j=1}^n \tilde{\Delta}_{ij} u_j^\pi(\lambda)$. where δ_i is defined as for the average reward model and $\tilde{\boldsymbol{\Delta}}$ is such that for all[5] states $i, j \in [n]$: $\tilde{\Delta}_{ij} := \beta(P_{ij}^1 - P_{ij}^0)$.

To finish the derivation of the algorithm, one should note that the value function $\boldsymbol{u}(\lambda)$ plays the same role as the vector $\boldsymbol{v}(\lambda)$ defined for the average reward model. In particular, the definition of $\boldsymbol{u}$ in Equation (24) is the analogue of the definition of $\boldsymbol{v}$ in (11) up to the replacement of the matrix $\boldsymbol{A}^\pi$ in (11) by the matrix $\boldsymbol{I} - \beta \boldsymbol{P}^\pi$. This means that similarly to $\boldsymbol{v}$, the value function $\boldsymbol{u}(\lambda)$ is affine in λ.

[5] Note that the definition of $\tilde{\boldsymbol{\Delta}}$ is identical to $\boldsymbol{\Delta}$ except when $j = 1$, for which $\Delta_{i1} := 0$ but $\tilde{\Delta}_{i1} := \beta(P_{i1}^1 - P_{i1}^0)$.

Hence, following the same development in Section 4, we can modify Algorithm 2 to compute the discounted Whittle index by modifying only the initialization phase:

$$\tilde{\boldsymbol{\Delta}} := \beta(\boldsymbol{P}^1 - \boldsymbol{P}^0) \text{ and } \boldsymbol{X}^1 := \tilde{\boldsymbol{\Delta}}(\boldsymbol{I} - \beta \boldsymbol{P}^{\pi^1})^{-1}.$$

Note that we still have $\boldsymbol{y}^1 = \boldsymbol{0}$ because $(\boldsymbol{I} - \beta \boldsymbol{P}^{\pi^1})^{-1}\boldsymbol{\pi}^1 = \frac{1}{1-\beta}\boldsymbol{1}$ (value function in a β-discounted Markov reward process with reward equals to 1 in all states) and $\tilde{\boldsymbol{\Delta}}\boldsymbol{1} = \boldsymbol{0}$. Also, if Subroutine 4 is used, Line 6 should be changed to $\boldsymbol{X}^{k_0} := \tilde{\boldsymbol{\Delta}}(\boldsymbol{I} - \beta \boldsymbol{P}^{\pi^{k_0}})^{-1}$. Last but not least, in the discounted case, we no longer need to check if the optimal policies are unichain because the matrix $(\boldsymbol{I} - \beta \boldsymbol{P}^{\pi})$ is invertible for any policy π as long as $\beta < 1$ (from Perron-Frobenius' Theorem).

7.3 Gittins index

The notion of "restless" bandit comes from the fact that even when the action "rest" is taken, the Markov chain can still change state and generate rewards. When this is not the case (*i.e.*, when $\boldsymbol{P}^0 = \boldsymbol{I}$ and $\boldsymbol{r}^0 = \boldsymbol{0}$), an arm is no longer restless and is simply called a Markovian bandit (or a *rested* Markovian bandit if one wants to emphasize that it is not restless).

In a discounted rested bandit, the notion of Whittle index coincides with the notion of Gittins index (In fact, Whittle index was first introduced as a generalization of Gittins index to restless bandit in [3]). In such a case, there is no notion of indexability: a discounted rested bandit is always indexable. Its index can be computed by Algorithm 2 without testing indexability. The best known algorithms to compute Gittins index runs in $(2/3)n^3+O(n^2)$ [28]. When using fast multiplication, our algorithm computes Gittins index in $O(n^{2.5286})$ which makes it the first algorithm to compute Gittins index in subcubic time.

Note that when $\boldsymbol{P}^1 = \boldsymbol{I}$, it is possible to compute $\boldsymbol{X}^1 = \tilde{\boldsymbol{\Delta}}/(1-\beta)$ without having to solve a linear system which means that, when $\boldsymbol{P}^1 = \boldsymbol{I}$, our algorithm with the variant Subroutine 3 has complexity $(2/3)n^3+O(n^2)$. This shows that our cubic algorithm can compute the Gittins index also in $(2/3)n^3 + O(n^2)$ instead of $(2/3)n^3 + o(n^3)$.

8 Conclusion

In this paper, we propose a univocal definition of indexability and present an algorithm that is efficient for detecting the non-indexability and computing the Whittle index of all indexable finite-state restless bandits whose arms are all unichain. With no assumptions on the structure of arms, this algorithm can still test the indexability and compute the Whittle index of some arms that are multichain and remains efficient if it is able to do so. Our algorithm is based on the efficient application of the Sherman-Morrison formula. This is a unified algorithm that works for both discounted and non-discounted restless

bandits, and can be used for Gittins index computation. We present a first version of our algorithm that runs in $n^3 + o(n^3)$ arithmetic operations (or in $(2/3)n^3 + o(n^3)$ if we do not test the indexability). So, we conclude that Whittle index is not harder to compute than Gittins index. The second version of our algorithm uses the fastest matrix multiplication method and has a complexity of $O(n^{2.5286})$. This makes it the first subcubic algorithm to compute Whittle index or Gittins index. We provide numerical simulations that show that our algorithm is very efficient in practice: it can test indexability and compute the index of a n-state restless bandit arm in less than one second for $n = 1000$, and in a few minutes for $n = 15000$. These numbers are provided for dense matrices. One might expect to have more efficient algorithms if the arm has a sparse structure. We leave this question for future work.

Acknowledgments. This work is supported by the French National Research Agency (ANR) through REFINO Project under Grant ANR-19-CE23-0015.

Declarations

- Funding: the French National Research Agency (ANR) through REFINO project under Grant ANR-19-CE23-0015
- Conflict of interest/Competing interests: Not applicable
- Ethics approval: No ethical concerns
- Availability of data and materials: Not applicable
- Code availability: https://gitlab.inria.fr/markovianbandit/efficient-whittle-index-computation

Appendix A Examples and counterexamples

In this section, we provide a few examples to illustrate the ambiguities in the classical definition of indexability, and to illustrate what can happen for some multichain arms. We also provide the parameters of arms presented in Figure 2.

A.1 Discussion on the definition of indexability

The classical notion of indexability used in the literature is to say that the optimal policy $\pi^*(\lambda)$ should be non-increasing in λ. Yet, we argue that this definition has two problems:

1. What does "increasing" mean when $\pi^*(\lambda)$ is not unique? Two possibilities are: for all penalties $\lambda < \lambda'$:
   (∃) there exist policies π, π' with π optimal for λ and π' optimal for λ' such that $\pi \supseteq \pi'$;
   (∀) for all policies π, π' such that π is optimal for λ and π' is optimal for λ', we have $\pi \supseteq \pi'$.
2. What notion of "optimality" should be used? Two possibilities are:

(GO) "optimal" means gain optimal.
(BO) "optimal" means Bellman optimal.

The most problematic choice is the notion of increasingness: Interpretation (∃) is more permissive: For instance, consider an arm with two states and assume that the optimal policy is $\{1, 2\}$ for $\lambda < 0$ and is either $\{1\}$ or $\emptyset$ for $\lambda > 0$. Interpretation (∃) says that the arm is indexable while interpretation (∀) says that this arm is not indexable. If the arm is indexable, what should the index of state 1 be? Any choice $\lambda_1 \in [0, +\infty]$ seems reasonable. Saying that the arm is not indexable clarifies the situation. This is why we choose interpretation (∀) in our paper.

In our paper, we choose the combination (∀-BO) because we believe that, for a problem that has transient state, the notion of Bellman optimality is more meaningful than the notion of gain optimality. Also, our combination (∀-BO) allows for more problems to be indexable compared to (∀-GO) and is easier to characterize.

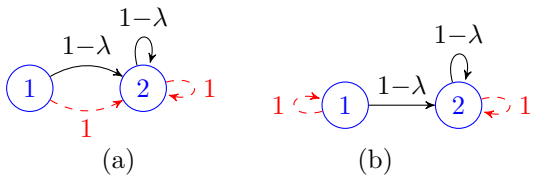


Fig. A1: Ambiguous examples: All transitions are deterministic and labels on transitions indicate rewards. Solid black arrows correspond to the action "activate" and dashed red arrows to the action "rest".

We illustrate these different definitions in Figure A1. For example (a):

- The gain optimal policies are $\{1, 2\}$ and $\{2\}$ for $\lambda < 0$, and $\{1\}$ and $\emptyset$ for $\lambda > 0$: According to the interpretation (∃), the problem should be indexable but the index for state 1 is unclear. According to the interpretation (∀), the problem should not be indexable.
- The Bellman optimal policy is $\{1, 2\}$ for $\lambda < 0$, and $\emptyset$ for $\lambda > 0$. According to our definition, (∀-BO), the problem is indexable and the indices are $\lambda_1 = \lambda_2 = 0$.

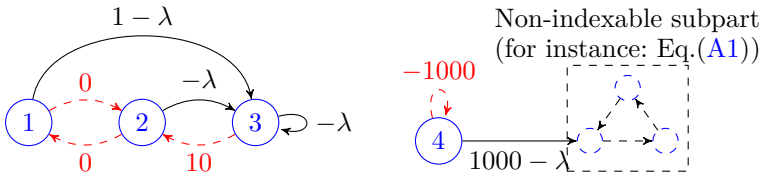
For example (b), the Bellman optimal and gain optimal policies are identical and equal to the gain optimal policies of example (a). Hence, example (b) is not indexable according to our definition. The output of our algorithm for this problem is "multichain".

Note that if the distinction between (BO) and (GO) disappears for discounted problems, the distinction between (∀) and (∃) remains.

A.2 Possible outputs for multichain arms

When running our algorithm on a multichain arm, it outputs "multichain" if one of the policies π^k is multichain. This suggests that our algorithm will not

necessarily output "multichain" for all multichain arms because it explores only a small subset of the policies. In Figure A2, we provide two examples that illustrate this case. The two are multichain arms for which our algorithm is able to identify the indexability.



(a) indexable arm (b) non-indexable arm.

Fig. A2: Two examples of multichain arms for which our algorithm does not return "multichain" but returns "indexable" (a) or "non-indexable" (b).

In the first example shown in Figure A2(a), the arm is multichain because the policy $\{3\}$ has two recurrent classes: $\{1, 2\}$ and $\{3\}$. Yet, this arm is indexable and the indices are $\{11, 8, -10\}$. Our algorithm will output that this arm is indexable because it will explore the sequence of policies $\pi^1, \pi^2, \pi^3, \pi^4$, where

- $\pi^1 = \{1, 2, 3\}$ is the unique Bellman optimal policy for $\lambda < -10$;
- $\pi^2 = \{1, 2\}$ is the unique Bellman optimal policy for $\lambda \in (-10, 8)$;
- $\pi^3 = \{1\}$ is the unique Bellman optimal policy for $\lambda \in (8, 11)$;
- $\pi^4 = \emptyset$ is the unique Bellman optimal policy for $\lambda > 11$.

All these policies are unichain, and the policy $\{3\}$ will never be explored. Hence, our algorithm will output "indexable" for this case and will compute the indices.

In the second example, shown in Figure A2(b), we construct a non-indexable arm by taking the non-indexable 3-state example shown in Figure 3b (parameters are given in (A1)) to which we add an extra state "4". For this state, the active action has a very high reward (1000) and leads to the non-indexable recurrent class. The passive action has a very low reward and stays in state 4. Any policy that does not activate 4 is multichain. The algorithm will start by exploring policies that activate the state 4. As for the original example presented in Figure 3b, our algorithm will realize that the arm is non-indexable when exploring values around $\lambda \approx 0.70$. The algorithm will stop and answer "not indexable" before trying the passive action for state 4 because the active avantage for state 4 is larger than $2000 - \lambda$. The output of the algorithm is thus "non-indexable".

(a) Our algorithm returns "indexable".

(b) Our algorithm returns "multichain".

Fig. A3: Example of an indexable multichain problem with infinite Whittle index. Transitions are deterministic and labels on edges indicate rewards (for the λ-penalized arm). Solid black transitions correspond to action "activate" and dashed red to the action "rest".

A.3 Multichain arms and infinite indices

Consider the two examples of Figure A3. The examples are multichain because policy $\emptyset$ has two irreducible classes for example (a) and policy $\{1,2\}$ has two irreducible classes for example (b). These two problems are indexable:

- For (a), the Bellman optimal policy for $\lambda < 0$ is $\{1,2\}$ and $\{1\}$ for $\lambda > 0$. The indices are $\lambda_2 = 0$ and $\lambda_1 = +\infty$.
- For (b), the Bellman optimal policy is $\{2\}$ for $\lambda < 0$ and $\{1,2\}$ for $\lambda > 0$. The indices are $\lambda_1 = 0$ and $\lambda_2 = -\infty$.

For the first example, our algorithm returns the correct indices because the constructed policies are $\pi^1 := \{1,2\} \supsetneq \pi_2 := \{1\}$ and they are both unichain. For the second example, our algorithm will start with the policy $\{1,2\}$ and will stop by saying that this example is multichain.

A.4 Parameters for the example of Figure 2

The numerical data of the indexable arm presented in Figure 2a is

$$\boldsymbol{P}^0 = \begin{bmatrix} 0.363 & 0.503 & 0.134 \\ 0.082 & 0.754 & 0.164 \\ 0.246 & 0.029 & 0.724 \end{bmatrix} \boldsymbol{P}^1 = \begin{bmatrix} 0.172 & 0.175 & 0.653 \\ 0.055 & 0.931 & 0.014 \\ 0.155 & 0.627 & 0.218 \end{bmatrix} \boldsymbol{r}^1 = \begin{bmatrix} 0.441 \\ 0.803 \\ 0.426 \end{bmatrix} \boldsymbol{r}^0 = \mathbf{0}$$

The numerical data of the non-indexable arm presented in Figure 2b is

$$\boldsymbol{P}^0 = \begin{bmatrix} 0.005 & 0.793 & 0.202 \\ 0.027 & 0.558 & 0.415 \\ 0.736 & 0.249 & 0.015 \end{bmatrix} \boldsymbol{P}^1 = \begin{bmatrix} 0.718 & 0.254 & 0.028 \\ 0.347 & 0.097 & 0.556 \\ 0.015 & 0.956 & 0.029 \end{bmatrix} \boldsymbol{r}^1 = \begin{bmatrix} 0.699 \\ 0.362 \\ 0.715 \end{bmatrix} \boldsymbol{r}^0 = \mathbf{0} \tag{A1}$$

Appendix B Technical lemmas

B.1 Unicity of Bellman optimal policy

B.1.1 Definition and notation

We are given a MDP $\langle \mathcal{S}, \mathcal{A}, r, P \rangle$ with finite state and action spaces. As shown in [40, Chapter 9], for such a MDP, the optimal gain $\boldsymbol{g}^*$ is a vector that satisfies the *multichain optimality equations*: for each $i \in \mathcal{S}$:

$$\max_{a \in \mathcal{A}} \Big(\sum_{j \in \mathcal{S}} P_{ij}^a g_j^* - g_i^* \Big) = 0 \tag{B2}$$

$$\max_{a \in \mathcal{A}} \Big(r_i^a - g_i^* + \sum_{j \in \mathcal{S}} P_{ij}^a h_j - h_i \Big) = 0. \tag{B3}$$

This system uniquely determines the optimal gain that we denote by $\boldsymbol{g}^*$. However, vector $\boldsymbol{h}$ is not uniquely determined by the system. In the following, we denote by H the set of bias vector $\boldsymbol{h}$ such that $(\boldsymbol{g}^*, \boldsymbol{h})$ is a solution of the optimality equations (B2)–(B3). A policy π is Bellman optimal if there exists a bias $\boldsymbol{h} \in H$ such that policy π attains the maximum (B3), *i.e.*, for all $i \in \mathcal{S}$:

$$\pi_i \in \arg\max_{a \in \mathcal{A}} \Big(r_i^a - g_i + \sum_{j \in \mathcal{S}} P_{ij}^a h_j - h_i \Big) = \arg\max_{a \in \mathcal{A}} \Big(r_i^a + \sum_{j \in \mathcal{S}} P_{ij}^a h_j \Big).$$

For a given policy $\pi : \mathcal{S} \mapsto \mathcal{A}$, we denote by $\boldsymbol{r}^\pi$ and $\boldsymbol{P}^\pi$ the reward vector and state transition matrix under policy π: $r_i^\pi = r_i^{\pi_i}$ and $P_{ij}^\pi = P_{ij}^{\pi_i}$. Let $\boldsymbol{g}^\pi, \boldsymbol{h}^\pi \in \mathbb{R}^{|\mathcal{S}|}$ be a solution of the following system:

$$\boldsymbol{g}^\pi - \boldsymbol{P}^\pi \boldsymbol{g}^\pi = \boldsymbol{0} \tag{B4}$$

$$\boldsymbol{r}^\pi - \boldsymbol{g}^\pi + \boldsymbol{P}^\pi \boldsymbol{h}^\pi - \boldsymbol{h}^\pi = \boldsymbol{0}. \tag{B5}$$

The vector $\boldsymbol{g}^\pi$ is uniquely determined by this system of equations and is called the long-run average reward or gain of policy π. The vector $\boldsymbol{h}^\pi$ is unique up to an element of the null space of $(\boldsymbol{I} - \boldsymbol{P}^\pi)$. Such a vector $\boldsymbol{h}^\pi$ is called a bias of policy π.

If $(\boldsymbol{g}^\pi, \boldsymbol{h}^\pi)$ is a solution of (B4) and (B5), then the advantage of action a over the action π_i when the MDP is in state i is given by:

$$\begin{aligned} B_i^a(\boldsymbol{h}^\pi) &:= r_i^a + \sum_{j \in \mathcal{S}} P_{ij}^a h_j^\pi - g_i^\pi - h_i^\pi \\ &= r_i^a - r_i^{\pi_i} + \sum_{j \in \mathcal{S}} (P_{ij}^a - P_{ij}^{\pi_i}) h_j^\pi \end{aligned}$$

We recall the two notions of optimality:

- A policy π is gain optimal if its gain $\boldsymbol{g}^\pi$ equals $\boldsymbol{g}^*$.
- A policy π is Bellman optimal if $\boldsymbol{g}^* = \boldsymbol{P}^\pi \boldsymbol{g}^*$ and if there exists a bias $\boldsymbol{h}^\pi$ that is a solution of (B5) and also (B3).

Recall that a Bellman optimal policy is also gain optimal.

Note that by the definition the advantage $B(\cdot)$, a policy is Bellman optimal if $\boldsymbol{g}^* = \boldsymbol{P}^\pi \boldsymbol{g}^*$ and if there exists $\boldsymbol{h} \in H$ such that $B_i^{\pi_i}(\boldsymbol{h}) = 0$.

Useful notations

Let $\pi : \mathcal{S} \to \mathcal{A}$ be a policy. We say that a state is recurrent for π if it is recurrent for the Markov chain whose transition matrix is $\boldsymbol{P}^\pi$. In other words, a state $i \in \mathcal{S}$ is recurrent if when the chain starts in i at time 0, it almost surely visits the state i at some time $t \geq 1$. We denote by $\mathcal{R}^\pi$ the set of recurrent states of policy π.

We also define $\bar{\boldsymbol{P}}^\pi$ as the Cesaro limit of the sequence $\{(\boldsymbol{P}^\pi)^t\}_{t=1}$:

$$\bar{\boldsymbol{P}}^\pi := \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} (\boldsymbol{P}^\pi)^{t-1}.$$

From [40, Section A.4 of Appendix A], the matrix $\bar{\boldsymbol{P}}^\pi$ exists and has the following properties:

- $\bar{\boldsymbol{P}}^\pi$ is a stochastic matrix, and satisfies $\boldsymbol{P}^\pi \bar{\boldsymbol{P}}^\pi = \bar{\boldsymbol{P}}^\pi \boldsymbol{P}^\pi = \bar{\boldsymbol{P}}^\pi$.
- For all state i, j, if $j \notin \mathcal{R}^\pi$, then $\bar{P}_{ij}^\pi = 0$.
- If π is unichain, then the rows of $\bar{\boldsymbol{P}}^\pi$ are identical.

B.1.2 Characterization of gain optimal policies

The following lemma characterizes gain optimal policy by showing that the policy must satisfies (B3) on their recurrent states.

Lemma 4. *Let $\pi : \mathcal{S} \mapsto \mathcal{A}$ be a policy and recall that $\mathcal{R}^\pi$ is the set of recurrent states of policy π. The three properties below are equivalent.*

(i) $\boldsymbol{P}^\pi \boldsymbol{g}^* = \boldsymbol{g}^*$ *and for all* $\boldsymbol{h} \in H$, $B_i^{\pi_i}(\boldsymbol{h}) = 0$ *for all* $i \in \mathcal{R}^\pi$

(ii) $\boldsymbol{P}^\pi \boldsymbol{g}^* = \boldsymbol{g}^*$ *and for some* $\boldsymbol{h} \in H$, $B_i^{\pi_i}(\boldsymbol{h}) = 0$ *for all* $i \in \mathcal{R}^\pi$

(iii) π *is gain optimal.*

Proof (i) $\Rightarrow$ (ii) is trivial.

(ii) $\Rightarrow$ (iii): By definition of $B_i^a(\boldsymbol{h})$, we have $r_i^\pi - g_i^* = h_i - \sum_{j\in\mathcal{S}} P_{ij}^\pi h_j$ for any recurrent state i of π. Multiply this with $\bar{P}_{ki}^\pi$ and sum over $i \in \mathcal{S}$ (if i is not recurrent, then $\bar{P}_{ki}^\pi = 0$) gives

$$\sum_{i\in\mathcal{S}} \bar{P}_{ki}^\pi (r_i^\pi - g_i^*) = \sum_{i\in\mathcal{S}} \bar{P}_{ki}^\pi h_i - \underbrace{\sum_{i\in\mathcal{S}} \bar{P}_{ki}^\pi \sum_{j\in\mathcal{S}} P_{ij}^\pi h_j}_{=\sum_{j\in\mathcal{S}} \bar{P}_{kj}^\pi h_j \text{ since } \bar{\boldsymbol{P}}^\pi \boldsymbol{P}^\pi = \bar{\boldsymbol{P}}^\pi.} = \mathbf{0}.$$

By Theorem 8.2.6 of [40], the average reward of π is $\bar{\boldsymbol{P}}^\pi r^\pi$. The above equation shows that $\bar{\boldsymbol{P}}^\pi \boldsymbol{r}^\pi = \bar{\boldsymbol{P}}^\pi \boldsymbol{g}^*$. Moreover, the assumption $\boldsymbol{P}^\pi \boldsymbol{g}^* = \boldsymbol{g}^*$ implies that $\bar{\boldsymbol{P}}^\pi \boldsymbol{g}^* = \boldsymbol{g}^*$ which in turn implies that $\bar{\boldsymbol{P}}^\pi \boldsymbol{r}^\pi = \boldsymbol{g}^*$. This shows that the average reward of π is $\boldsymbol{g}^*$ and therefore π is gain optimal.

(iii) ⇒ (i): If π is gain optimal, then $\boldsymbol{P}^\pi \boldsymbol{g}^* = \boldsymbol{g}^*$ and $\bar{\boldsymbol{P}}^\pi(\boldsymbol{r}^\pi - \boldsymbol{g}^*) = \boldsymbol{0}$. The latter rewrites as $\sum_{i\in\mathcal{S}} \bar{P}^\pi_{ki}(r^\pi_i - g^*_i) = 0$ for all state k. Let $\boldsymbol{h} \in H$ be an optimal bias. For all state k, we have

$$\sum_{i\in\mathcal{S}} \bar{P}^\pi_{ki} B^{\pi_i}_i(\boldsymbol{h}) = \sum_{i\in\mathcal{S}} \bar{P}^\pi_{ki}(r^\pi_i - g^*_i + \sum_{j\in\mathcal{S}} P^\pi_{ij} h_j - h_i) = 0. \tag{B6}$$

As $\boldsymbol{h}$ satisfied (B3), for all action a, we have $B^a_i(\boldsymbol{h}) \le 0$ for all states $i \in \mathcal{S}$ and in particular $B^{\pi_i}_i(\boldsymbol{h}) \le 0$. This shows that for any state i such that $\bar{P}^\pi_{ki} > 0$, one must have $B^{\pi_i}_i(\boldsymbol{h}) = 0$. Such state i are the recurrent states of π. This shows that $B^{\pi_i}_i(\boldsymbol{h}) = 0$ for all $i \in \mathcal{R}^\pi$. □

B.1.3 Characterization of Bellman optimal policies

The previous lemma shows that a policy is gain optimal if and only if the actions for the recurrent states of the policy satisfy (B3). The following lemma shows the relationship between two policies that are unichain and satisfy (B3) on all states.

Lemma 5. *Suppose that two policies π and θ are Bellman optimal, unichain and have at least one common recurrent state: $\mathcal{R}^\pi \cap \mathcal{R}^\theta \neq \emptyset$.*

Then for any $\boldsymbol{h}^\pi$ and $\boldsymbol{h}^\theta$ solutions of (B5) for π and θ respectively, there exists a constant c such that for all state i: $h^\pi_i - h^\theta_i = c$. Moreover, in this case, $B^{\theta_i}_i(\boldsymbol{h}^\pi) = B^{\pi_i}_i(\boldsymbol{h}^\theta) = 0$ for all i.

Proof Since π and θ are Bellman optimal, $\boldsymbol{h}^\pi, \boldsymbol{h}^\theta \in H$. In consequence, we have

$$\boldsymbol{h}^\pi \ge \boldsymbol{r}^\theta - \boldsymbol{g}^* + \boldsymbol{P}^\theta \boldsymbol{h}^\pi.$$

By Lemma 4 (i), the above inequality is an equality for all $i \in \mathcal{R}^\theta$ because θ is gain optimal.

As $\boldsymbol{h}^\theta$ satisfies (B5), we have

$$\boldsymbol{h}^\theta - \boldsymbol{h}^\pi \le \boldsymbol{r}^\theta - \boldsymbol{g}^* + \boldsymbol{P}^\theta \boldsymbol{h}^\theta - (\boldsymbol{r}^\theta - \boldsymbol{g}^* + \boldsymbol{P}^\theta \boldsymbol{h}^\pi) = \boldsymbol{P}^\theta(\boldsymbol{h}^\theta - \boldsymbol{h}^\pi),$$

with equality for all state $i \in \mathcal{R}^\theta$. This shows that for all t, $\boldsymbol{h}^\theta - \boldsymbol{h}^\pi \le (\boldsymbol{P}^\theta)^t(\boldsymbol{h}^\theta - \boldsymbol{h}^\pi)$ which implies that $\boldsymbol{h}^\theta - \boldsymbol{h}^\pi \le \bar{\boldsymbol{P}}^\theta(\boldsymbol{h}^\theta - \boldsymbol{h}^\pi)$ with equality for all states $i \in \mathcal{R}^\theta$. Similarly, $\boldsymbol{h}^\pi - \boldsymbol{h}^\theta \le \bar{\boldsymbol{P}}^\pi(\boldsymbol{h}^\pi - \boldsymbol{h}^\theta)$ with equality for any state $i \in \mathcal{R}^\pi$.

Let $c^\pi_i = \sum_{j\in\mathcal{S}} \bar{P}^\pi_{ij}(h^\pi_j - h^\theta_j)$ and $c^\theta_i = \sum_{j\in\mathcal{S}} \bar{P}^\theta_{ij}(h^\pi_j - h^\theta_j)$. By what we have just shown, for all state i, we have

$$c^\theta_i \underbrace{\le}_{\text{equality if } i\in\mathcal{R}^\theta} h^\pi_i - h^\theta_i \underbrace{\le}_{\text{equality if } i\in\mathcal{R}^\pi} c^\pi$$

As both policies are unichain, c^π_i and c^θ_i do not depend on i. Moreover, if there exists $i \in \mathcal{R}^\theta \cap \mathcal{R}^\pi$, we have $c^\pi_i = c^\theta_i =: c$. In consequence, $h^\pi_i - h^\theta_i = c$ for all state i. □

B.1.4 Unicity of Bellman optimal policy

Lemma 6. *Let π be a Bellman optimal policy that is unichain. If π is not the unique Bellman optimal policy, then there exists a state i and an action $a \neq \pi_i$ such that $B^a_i(\boldsymbol{h}^\pi) = 0$.*

Proof Let $\theta \neq \pi$ be another Bellman optimal policy. Since θ is gain optimal and $\boldsymbol{h}^{\pi} \in H$, Lemma 4 implies that $B_i^{\theta_i}(\boldsymbol{h}^{\pi}) = 0$ for all $i \in \mathcal{R}^{\theta}$. If there exists $i \in \mathcal{R}^{\theta}$ such that $\theta_i \neq \pi_i$, then the proof is concluded. Otherwise, $\theta_i = \pi_i$ for all $i \in \mathcal{R}^{\theta}$. This show that π and θ coincide for all recurrent states of θ and that $\mathcal{R}^{\theta} = \mathcal{R}^{\pi}$. Moreover, as π is unichain, θ is also unichain. Hence, Lemma 5 implies that $B_i^{\theta_i}(\boldsymbol{h}^{\pi}) = 0$ for all i. Since $\theta \neq \pi$, there exists at least one state $i \in \mathcal{S}$ such that $\theta_i \neq \pi_i$. □

B.2 Unichain property

Lemma 7. *Given a two-action MDP* $\langle [n], \{0,1\}, r, P \rangle$*, let* $\boldsymbol{P}^{\pi}$ *be the transition matrix under a Bellman optimal policy* π*. Policy* π *is* unichain *if and only if the matrix*

$$\boldsymbol{A}^{\pi} = \begin{bmatrix} 1 & -P_{12}^{\pi} & \dots & -P_{1n}^{\pi} \\ 1 & 1 - P_{22}^{\pi} & \dots & -P_{2n}^{\pi} \\ \vdots & & & \\ 1 & -P_{n2}^{\pi} & \dots & 1 - P_{nn}^{\pi} \end{bmatrix}$$

is invertible.

Proof $\boldsymbol{A}^{\pi}$ is not invertible if there exists a column vector $\boldsymbol{u} \neq \boldsymbol{0}$ such that $\boldsymbol{u}^{\top}\boldsymbol{A}^{\pi} = \boldsymbol{0}$. We prove that such $\boldsymbol{u}$ does not exist when policy π is unichain. Let $\boldsymbol{u} \in \mathbb{R}^n$ be an arbitrary vector such that $\boldsymbol{u}^{\top}\boldsymbol{A}^{\pi} = \boldsymbol{0}$. Then, we have

$$\begin{cases} \sum_{i=1}^{n} u_i & = 0 \\ u_i - \sum_{j=1}^{n} u_j P_{ji}^{\pi} & = 0, \text{ for } 2 \leq i \leq n \end{cases}$$

Combining the above equation with $\sum_j P_{ji}^{\pi} = 1$, we get:

$$u_1 = -\sum_{i=2}^{n} u_i = -\sum_{i=2}^{n}\sum_{j=1}^{n} u_j P_{ji}^{\pi} = -\sum_{j=1}^{n} u_j (1 - P_{j1}^{\pi}) = \sum_{j=1}^{n} u_j P_{j1}^{\pi},$$

where we used that $\sum_{i=1}^{n} u_i = 0$ to obtain the last equality. This shows that

$$\begin{cases} \sum_{i=1}^{n} u_i & = 0 \\ \boldsymbol{u}^{\top}\boldsymbol{P}^{\pi} & = \boldsymbol{u}^{T} \end{cases}$$

The set of vector $\boldsymbol{u}$ such that $\boldsymbol{u}^{\top}\boldsymbol{P}^{\pi}$ is a vector space. It is of dimension 1 if and only if π is unichain, in which case the vector $\boldsymbol{u}$ verifying $\boldsymbol{u}^{\top}\boldsymbol{P}^{\pi} = \boldsymbol{u}^{\top}$ are multiples of a stationary distribution under policy π [40]. Thus, if the policy π induces a unichain Markov chain, then $\sum_{i=1}^{n} u_i = 0$ implies $\boldsymbol{u} = \boldsymbol{0}$. If policy π is not unichain, there exists $\boldsymbol{u} \neq \boldsymbol{0}$ such that $\sum_{i=1}^{n} u_i = 0$. □

Appendix C Implementations

C.1 Arithmetic complexity of Subroutine 3 and memory usage

Recall that in Subroutine 3, we compute the values X_{ij}^{ℓ} by doing the update (for all iteration k, for all $\ell = 1$ to k and for all $i \in [n]$ or all $i \in \pi^{\ell+1}$ if we do not test indexability):

$$X_{i\sigma^k}^{\ell+1} = X_{i\sigma^k}^{\ell} - \frac{X_{i\sigma^\ell}^{\ell}}{1 + X_{\sigma^\ell\sigma^\ell}^{\ell}} X_{\sigma^\ell\sigma^k}^{\ell} \tag{C7}$$

If we test indexability, there are $\sum_{k=1}^{n} kn = n^3/2 + O(n^2)$ such updates. If we do not test indexability, there are $\sum_{k=1}^{n}\sum_{\ell=1}^{k}(n-\ell) = n^3/3 + O(n^2)$ such updates. Below, we show each update of Equation (C7) can be done in two arithmetic operations (one addition and one multiplication), which leads to the complexity of $n^3 + O(n^2)$ (or $(2/3)n^3 + O(n^2)$) arithmetic operations for the computation of all the needed X_{ij}^{k}. We also show how to reduce the memory size to $O(n^2)$.

Let $W_{i\ell} := X_{i\sigma^\ell}^{\ell}/(1 + X_{\sigma^\ell\sigma^\ell}^{\ell})$ and $V_i := X_{i\sigma^k}^{\ell}$. Using this, Equation (C7) can be rewritten as:

$$V_i = V_i - W_{i\ell} V_{\sigma^\ell}. \tag{C8}$$

This results in the following loop at iteration k:

- Initialize V_i from $X_{i\sigma^k}^{1}$.
- For all $\ell \in \{1, \ldots, k-1\}$, and all $i \in [n]$ (or $i \in \pi^{\ell+1}$), apply (C8).
- Compute $W_{ik} = V_i/(1 + V_{\sigma^k})$

Note that the value of $\boldsymbol{V}$ is not necessary for iteration k (only the values of $W_{i\ell}$ are needed). This shows that the algorithm can be implemented with a memory $O(n^2)$.

C.2 Speedup when not checking the indexability: First found go last

When the indexability is not tested, the update (C8) is computed for all $i \in \pi^{\ell+1}$. This creates inefficiencies (due to inefficient cache usage) because the elements V_i are not accessed sequentially.

To speedup the memory accesses, our solution is to sort the items during the execution of the algorithm. At iteration k, the algorithm computes σ^k. When this is done, our implementation switches all quantities in positions σ^k and $n-k+1$. These quantities are $\delta, y, z, \boldsymbol{W}$ and $\boldsymbol{X}$. For instance, once σ^1 is found, we know that the state at position n is state n and we do the following switches:

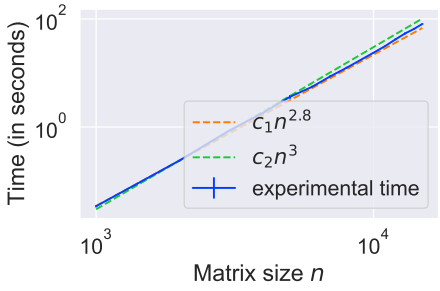
$$\delta_{\sigma^1}, \delta_n \to \delta_n, \delta_{\sigma^1}$$
$$y_{\sigma^1}^{1}, y_n^{1} \to y_n^{1}, y_{\sigma^1}^{1}$$

$$z^1_{\sigma^1}, z^1_n \to z^1_n, z^1_{\sigma^1}$$
$$\boldsymbol{W}_{\sigma^1:}, \boldsymbol{W}_{n:} \to \boldsymbol{W}_{n:}, \boldsymbol{W}_{\sigma^1:}$$
$$\text{and } \boldsymbol{X}_{\sigma^1:}, \boldsymbol{X}_{n:} \to \boldsymbol{X}_{n:}, \boldsymbol{X}_{\sigma^1:}.$$

To do so, we need an array to store all states such that at iteration k, the first $n-k$ states of the array are the active states. We will need to track the position of each state in such array.

Appendix D Analysis of the experimental time to solve a linear system

In this section, we report in Figure D4 the time taken by the default implementation to solve a linear system of the form $\boldsymbol{AX} = \boldsymbol{B}$ where $\boldsymbol{A}$ and $\boldsymbol{B}$ are two square matrices. To obtain this figure, we generated random (full) matrices where each entry is between 0 and 1 and use the function `scipy.linalg.solve` from the library `scipy`. The reported numbers suggest that the complexity of the solver is closer to $O(n^{2.8})$ than to $O(n^3)$, although we agree that the difference between the $O(n^{2.8})$ and the $O(n^3)$ curves is small. Note that this is in accordance with the papers [44, 45] that claim that the fastest implementations of matrix multiplication and inversion are based on Strassen's algorithm and should therefore be in $O(n^{2.8})$.



n	Time (in second)
1000	0.03 ± 0.02
2000	0.24 ± 0.01
4000	1.83 ± 0.05
5000	3.6 ± 0.3
7000	8.7 ± 0.5
10000	23.7 ± 0.6
13000	54.0 ± 2.5
15000	80.8 ± 1.8

Fig. D4: Time taken of the default implementations `scipy.linalg.solve` of scipy to solve a linear system $\boldsymbol{AX} = \boldsymbol{B}$ where $\boldsymbol{A}$ and $\boldsymbol{B}$ are two square $n \times n$ matrices.

Appendix E Detailed comparison with [34] and [29]

In this section, we compare our algorithm with two main related works for finite-state restless bandits problem.

E.1 Comparison with [34]

The paper presents an algorithm that computes Whittle indices in $O(n^3)$ (no explicit constant before n^3 is given) for all indexable problems. Despite following a different approach, our algorithm for computing Whittle index can be viewed as a refinement of this work. Let us recall once again that our approach also allows one to check the indexability of general restless bandits.

In the following, we show how we can refine the work of [34] to obtain an algorithm that is exactly the same as ours. Let D^π and N^π be two vectors defined as in [34] (we use the same notation, D^π and N^π, as the cited paper),

$$D^\pi = (1-\beta)(\boldsymbol{I} - \beta \boldsymbol{P}^\pi)^{-1}\boldsymbol{r}^\pi, \quad \text{and} \quad N^\pi = (1-\beta)(\boldsymbol{I} - \beta \boldsymbol{P}^\pi)^{-1}\boldsymbol{\pi}.$$

Then, we have $D^\pi - \lambda N^\pi = (1-\beta)\boldsymbol{u}^\pi(\lambda)$ where $\boldsymbol{u}^\pi(\lambda)$ is defined as in (24). In our proposition, at each iteration k, we compute μ_i^k by Line 13. Instead, it is defined in [34] by two steps:

1. for all state $j \in [n]$ such that $N_j^{\pi^k \setminus \{i\}} \neq N_j^{\pi^k}$, one needs to compute $\mu_{ij}^k = \dfrac{D_j^{\pi^k \setminus \{i\}} - D_j^{\pi^k}}{N_j^{\pi^k \setminus \{i\}} - N_j^{\pi^k}}$
2. compute $\mu_i^k = \underset{j \in [n]: N_j^{\pi^k \setminus \{i\}} \neq N_j^{\pi^k}}{\arg\min} \mu_{ij}^k$.

From [34, Theorem 2], in an indexable problem, for state σ^k, there exists a state $j \in [n]$ such that $N_j^{\pi^k \setminus \{\sigma^k\}} \neq N_j^{\pi^k}$. Now, suppose that for any active state $i \in \pi^k$, there exists $j \in [n]$ such that $N_j^{\pi^k \setminus \{i\}} \neq N_j^{\pi^k}$. Using the Sherman-Morrison formula, we have[6]

$$D^{\pi^k \setminus \{i\}} - D^{\pi^k} = -\frac{(1-\beta)\delta_i + \tilde{\boldsymbol{\Delta}}_i D^{\pi^k}}{1 + \tilde{\boldsymbol{\Delta}}_i [(\boldsymbol{I} - \beta \boldsymbol{P}^{\pi^k})^{-1}]_{:i}} [(\boldsymbol{I} - \beta \boldsymbol{P}^{\pi^k})^{-1}]_{:i},$$

$$\text{and} \quad N^{\pi^k \setminus \{i\}} - N^{\pi^k} = -\frac{(1-\beta) + \tilde{\boldsymbol{\Delta}}_i N^{\pi^k}}{1 + \tilde{\boldsymbol{\Delta}}_i [(\boldsymbol{I} - \beta \boldsymbol{P}^{\pi^k})^{-1}]_{:i}} [(\boldsymbol{I} - \beta \boldsymbol{P}^{\pi^k})^{-1}]_{:i}.$$

Then, for any $j \in [n]$ such that $N_j^{\pi^k \setminus \{i\}} \neq N_j^{\pi^k}$, $\mu_{ij}^k = \dfrac{(1-\beta)\delta_i + \tilde{\boldsymbol{\Delta}}_i D^{\pi^k}}{(1-\beta) + \tilde{\boldsymbol{\Delta}}_i N^{\pi^k}}$ which does not depend on j. Then, we simply have $\mu_i^k = \dfrac{(1-\beta)\delta_i + \tilde{\boldsymbol{\Delta}}_i D^{\pi^k}}{(1-\beta) + \tilde{\boldsymbol{\Delta}}_i N^{\pi^k}}$. Also, we have

$$\tilde{\boldsymbol{\Delta}}_i N^{\pi^k} = (1-\beta)\tilde{\boldsymbol{\Delta}}_i (\boldsymbol{I} - \beta \boldsymbol{P}^{\pi^k})^{-1}\boldsymbol{\pi}^k = -(1-\beta)y_i^k \quad \text{and}$$
$$\tilde{\boldsymbol{\Delta}}_i D^{\pi^k} = (1-\beta)\tilde{\boldsymbol{\Delta}}_i \boldsymbol{u}^{\pi^k}(\mu_{\min}^{k-1}) + \mu_{\min}^{k-1}\tilde{\boldsymbol{\Delta}}_i N^{\pi^k} = (1-\beta)z_i^{k-1} - (1-\beta)\mu_{\min}^{k-1} y_i^k.$$

[6] the expression of $D^{\pi \setminus \{i\}}$ and $N^{\pi \setminus \{i\}}$ given by Equation (18) in [34] are erroneous.

So, replacing these terms in μ_i^k, we get the formula in Equation (14) of our work.

Note that the algorithm of [34] was only developed for the discounted case. Our approach for the time-average reward case is different because we use the active advantage function defined in (5) instead of working with the expected discounted total reward D^{π^k} and total number of activations N^{π^k} under policy π^k. Note that the counterpart of D^{π^k} in undiscounted MDP is the average reward g^{π^k} and as we have seen in Appendix A.1, utilizing average reward optimality is not rich enough for undiscounted MDPs with transient states. In addition, our code is also optimized to avoid unnecessary computation and to reduce memory usage. Finally, the way we do the update of our matrix $\boldsymbol{X}$ makes it possible to obtain a subcubic algorithm whereas their approach does not (see also below).

E.2 Comparison with the algorithm of [29]

The algorithm [29] has the best complexity up to date for discounted restless bandit. There is a square matrix $\boldsymbol{A}$ that plays a similar role as the square matrix $\boldsymbol{X}$ in our proposed algorithm. The most costly operations in the algorithm of [29] is to update their matrix $\boldsymbol{A}$ at each iteration and it is done by Equation (21) that we recall here (using the same notation $\boldsymbol{A}$ as the cited paper):

$$\text{for } i,j \in \pi^k,\ \boldsymbol{A}_{ij}^{k+1} = \boldsymbol{A}_{ij}^{k} - \frac{\boldsymbol{A}_{i\sigma^k}^{k}}{\boldsymbol{A}_{\sigma^k\sigma^k}^{k}} \boldsymbol{A}_{\sigma^k j}^{k}. \tag{E9}$$

This incurs a total complexity of $(2/3)n^3 + O(n^2)$ arithmetic operations. As mentioned in Section 4.3, if we updated $\boldsymbol{X}^{k+1}$ as given by (21), our algorithm would also have a $(2/3)n^3+O(n^2)$ complexity but this version of update cannot be optimized by using fast matrix multiplication.

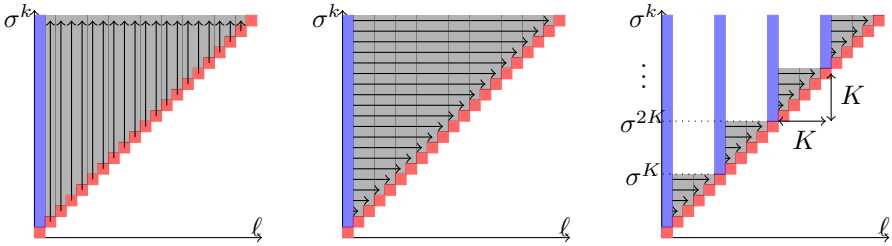
E.3 Their approach cannot be directly transformed into a subcubic algorithm

In addition to all the previously cited differences, one of the major contribution of our algorithm with respect to [29, 34] is that the most advanced version of our algorithm runs in a subcubic time. The approach[7] used in [29, 34] is to update the full matrix $\boldsymbol{X}^{k+1}$ at iteration k, by using (E9). This idea is represented in Figure E5(a): for a given ℓ, their algorithm compute $\boldsymbol{X}_{:\sigma^k}^{\ell}$ for all k (i.e., the full vertical lines represented by arrows). Our first Subroutine 3 uses an horizontal approach based on (22), which we recall here:

$$X_{i\sigma^k}^{\ell+1} = X_{i\sigma^k}^{\ell} - \frac{X_{i\sigma^\ell}^{\ell}}{1 + X_{\sigma^\ell\sigma^\ell}^{\ell}} X_{\sigma^\ell\sigma^k}^{\ell}.$$

[7] Equation (18) of [34], which is central to their algorithm is the same as the above equation (E9).

At iteration $k+1$, we use $\boldsymbol{X}^1_{:\sigma^k}$ to compute all values of $\boldsymbol{X}^\ell_{:\sigma^k}$ up to $\ell = k+1$. This is represented in Figure E5(b). Our approach can be used to obtain the subcubic algorithm illustrated in Figure E5(c) by using subcubic algorithms for multiplication.



(a) [29, 34] use (E9). (b) Subroutine 3. (c) Subroutine 4.

Fig. E5: Comparison of the computation load of (E9) used in [29, 34] with the one of Subroutine 3 and Subroutine 4.

This leads to the next fundamental question: why should the computation of Whittle index be harder than matrix inversion (or multiplication)? To us, the main difference is that when computing Whittle indices, the permutation σ is not known a priori but discovered as the algorithm progresses: σ^k is only known at iteration k. Hence, while all terms of the matrices X^k_{ij} are not needed, it is difficult to know a priori which ones are needed and which ones are not. Hence, a simple divide and conquer algorithm cannot be used. This is why when recomputing $\boldsymbol{X}^{k+1}$ in Subroutine 4, we recompute the whole matrix (the vertical blue line) and not just the part that will be used to compute the gray zone: we do not know *a priori* what part of X^{k+1}_{ij} will be useful or not.

References

[1] Gittins, J.C.: Bandit processes and dynamic allocation indices. Journal of the Royal Statistical Society: Series B (Methodological) **41**(2), 148–164 (1979)

[2] Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of optimal queueing network control. In: Proceedings of IEEE 9th Annual Conference on Structure in Complexity Theory, pp. 318–322 (1994). IEEE

[3] Whittle, P.: Restless bandits: Activity allocation in a changing world. Journal of applied probability **25**(A), 287–298 (1988)

[4] Verloop, I.M.: Asymptotically optimal priority policies for indexable and nonindexable restless bandits. The Annals of Applied Probability **26**(4), 1947–1995 (2016)

[5] Lott, C., Teneketzis, D.: On the optimality of an index rule in multichannel allocation for single-hop mobile networks with multiple service classes. Probability in the Engineering and Informational Sciences **14**(3), 259–297 (2000)

[6] Weber, R.R., Weiss, G.: On an index policy for restless bandits. Journal of applied probability, 637–648 (1990)

[7] Glazebrook, K.D., Ruiz-Hernandez, D., Kirkbride, C.: Some indexable families of restless bandit problems. Advances in Applied Probability **38**(3), 643–672 (2006)

[8] Ansell, P., Glazebrook, K.D., Nino-Mora, J., O'Keeffe, M.: Whittle's index policy for a multi-class queueing system with convex holding costs. Mathematical Methods of Operations Research **57**(1), 21–39 (2003)

[9] Glazebrook, K., Mitchell, H.: An index policy for a stochastic scheduling model with improving/deteriorating jobs. Naval Research Logistics (NRL) **49**(7), 706–721 (2002)

[10] Aalto, S., Lassila, P., Taboada, I.: Whittle index approach to opportunistic scheduling with partial channel information. Performance Evaluation **136**, 102052 (2019)

[11] Liu, K., Zhao, Q.: Indexability of restless bandit problems and optimality of Whittle index for dynamic multichannel access. IEEE Transactions on Information Theory **56**(11), 5547–5567 (2010)

[12] Avrachenkov, K.E., Borkar, V.S.: Whittle index based Q-learning for restless bandits with average reward. Automatica **139**, 110186 (2022)

[13] Niño-Mora, J.: A dynamic page-refresh index policy for web crawlers. In: International Conference on Analytical and Stochastic Modeling Techniques and Applications, pp. 46–60 (2014). Springer

[14] Avrachenkov, K., Ayesta, U., Doncel, J., Jacko, P.: Congestion control of tcp flows in internet routers by means of index policy. Computer Networks **57**(17), 3463–3478 (2013)

[15] Avrachenkov, K., Piunovskiy, A., Zhang, Y.: Impulsive control for G-AIMD dynamics with relaxed and hard constraints. In: 2018 IEEE Conference on Decision and Control (CDC), pp. 880–887 (2018). IEEE

[16] Scully, Z., Harchol-Balter, M., Scheller-Wolf, A.: SOAP: One clean analysis of all age-based scheduling policies. Proceedings of the ACM on Measurement and Analysis of Computing Systems **2**(1), 1–30 (2018)

[17] Aalto, S., Ayesta, U., Righter, R.: Properties of the Gittins index with application to optimal scheduling. Probability in the Engineering and Informational Sciences **25**(3), 269–288 (2011)

[18] Aalto, S., Ayesta, U., Righter, R.: On the Gittins index in the M/G/1 queue. Queueing Systems **63**(1-4), 437 (2009)

[19] Borkar, V.S., Pattathil, S.: Whittle indexability in egalitarian processor sharing systems. Annals of Operations Research, 1–21 (2017)

[20] Larrañaga, M., Ayesta, U., Verloop, I.M.: Asymptotically optimal index policies for an abandonment queue with convex holding cost. Queueing systems **81**(2), 99–169 (2015)

[21] Archibald, T.W., Black, D., Glazebrook, K.D.: Indexability and index heuristics for a simple class of inventory routing problems. Operations research **57**(2), 314–326 (2009)

[22] Glazebrook, K.D., Kirkbride, C., Ouenniche, J.: Index policies for the admission control and routing of impatient customers to heterogeneous service stations. Operations Research **57**(4), 975–989 (2009)

[23] Villar, S.S., Bowden, J., Wason, J.: Multi-armed bandit models for the optimal design of clinical trials: benefits and challenges. Statistical science: a review journal of the Institute of Mathematical Statistics **30**(2), 199 (2015)

[24] Chen, Y.R., Katehakis, M.N.: Linear programming for finite state multi-armed bandit problems. Mathematics of Operations Research **11**(1), 180–183 (1986)

[25] Katehakis, M.N., Veinott Jr, A.F.: The multi-armed bandit problem: decomposition and computation. Mathematics of Operations Research **12**(2), 262–268 (1987)

[26] Niño-Mora, J.: A $(2/3)n^3$ fast-pivoting algorithm for the Gittins index and optimal stopping of a markov chain. INFORMS Journal on Computing **19**(4), 596–606 (2007)

[27] Sonin, I.M.: A generalized Gittins index for a markov chain and its recursive calculation. Statistics & Probability Letters **78**(12), 1526–1533 (2008)

[28] Chakravorty, J., Mahajan, A.: Multi-armed bandits, Gittins index, and its calculation. Methods and applications of statistics in clinical trials: Planning, analysis, and inferential methods **2**(416-435), 455 (2014)

[29] Niño-Mora, J.: A fast-pivoting algorithm for Whittle's restless bandit index. Mathematics **8**(12), 2226 (2020)

[30] Niño-Mora, J.: Characterization and computation of restless bandit marginal productivity indices. In: 1st International ICST Workshop on Tools for Solving Structured Markov Chains (2010)

[31] Akbarzadeh, N., Mahajan, A.: Restless bandits with controlled restarts: Indexability and computation of Whittle index. In: 2019 IEEE 58th Conference on Decision and Control (CDC), pp. 7294–7300 (2019). IEEE

[32] Akbarzadeh, N., Mahajan, A.: Maintenance of a collection of machines under partial observability: Indexability and computation of Whittle index. arXiv preprint arXiv:2104.05151 (2021)

[33] Niño-Mora, J.: Dynamic priority allocation via restless bandit marginal productivity indices. Top **15**(2), 161–198 (2007)

[34] Akbarzadeh, N., Mahajan, A.: Conditions for indexability of restless bandits and an $O(K^3)$ algorithm to compute Whittle index. arXiv (2020)

[35] Gibson, L.J., Jacko, P., Nazarathy, Y.: A novel implementation of Q-learning for the Whittle index. In: EAI International Conference on Performance Evaluation Methodologies and Tools, pp. 154–170 (2021). Springer

[36] Fu, J., Nazarathy, Y., Moka, S., Taylor, P.G.: Towards Q-learning the Whittle index for restless bandits. In: 2019 Australian & New Zealand Control Conference (ANZCC), pp. 249–254 (2019). IEEE

[37] Nakhleh, K., Ganji, S., Hsieh, P.-C., Hou, I., Shakkottai, S., et al.: NeurWIN: Neural Whittle index network for restless bandits via deep RL. Advances in Neural Information Processing Systems **34** (2021)

[38] Ayesta, U., Gupta, M.K., Verloop, I.M.: On the computation of Whittle's index for markovian restless bandits. Mathematical Methods of Operations Research **93**(1), 179–208 (2021)

[39] Strassen, V.: Gaussian elimination is not optimal. Numerische mathematik **13**(4), 354–356 (1969)

[40] Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming, 1st edn. John Wiley & Sons, Inc., USA (1994)

[41] Schweitzer, P.J., Federgruen, A.: The functional equations of undiscounted Markov renewal programming. Mathematics of Operations Research **3**(4), 308–321 (1978)

[42] Gall, F.L., Urrutia, F.: Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1029–1046 (2018). SIAM

[43] Woodbury, M.A.: Inverting modified matrices. Statistical Research Group (1950)

[44] Huang, J., Smith, T.M., Henry, G.M., Van De Geijn, R.A.: Strassen's algorithm reloaded. In: SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 690–701 (2016). IEEE

[45] Huang, J., et al.: Practical fast matrix multiplication algorithms. PhD thesis (2018)

[46] Gast, N., Gaujal, B., Yan, C.: Exponential convergence rate for the asymptotic optimality of Whittle index policy. arXiv preprint arXiv:2012.09064 (2020)