Itération de Politique par Trajectoires

Filipo Studzinski Perotto
IRIT, Université Toulouse 1 Capitole, France
`filipo.perotto@irit.fr`

Abstract

This paper presents a dynamic programming method called *Trajectory Policy-Iteration* (TPI), which is able to find sensitive (average and multiple-order bias) optimal policies in polynomial time. TPI is designed to estimate the value function of multi-chain *Markovian Decision Processes* (MDPs) under unbounded (potentially infinite) time-horizon, when the transition function is deterministic (DMDPs). The algorithm takes advantage of the MDP structure, estimating the utility of state-action pairs by identifying circuits and transient paths inside the process, and then calculating the expected average rewards of such segments (gain and bias). The different orders of bias optimality are assessed using a very intuitive technique, where each higher order value is calculated based on the cumulated value of its immediate lower order. Such principle allows the proposition of an innovative exact policy-iteration algorithm, and constitutes an original interpretation to understand the meaning of multiple-order bias optimality. The viability of the method is supported by experimental results.

1 Introduction

Most *Dynamic Programming* (DP) and *Reinforcement Learning* (RL) methods are based on the *discounted optimality* criterion. In this setting, an optimal policy maximizes the sum of discounted rewards over time using a discount factor γ. When considering an unbounded time-horizon, the use of discounted cumulative rewards constitutes an important key on guaranteeing polynomial time convergence for such methods [2].

However, the use of discounted rewards is not always appropriate for recurrent problems (i.e. where terminal states do not exist) [20, 16, 25]. For such scenarios, maximizing the average reward per step (*average* or *gain optimality*) is, in some sense, more natural.

An example of such incongruence is the *crawling robot* problem [26]. In this problem, the rewards are proportional to the forward displacements. The behavior normally expected from an intelligent robot is moving forward as fast as possible. However, depending on the discount factor, the discounted-optimal policy will not necessarily correspond to the one that maximizes the velocity. In other words, an intuitively optimal policy can be seen as sub-optimal under the discounted framework. Other examples of this issue are given in [16].

If, on the one hand, average optimality fits better than the discounted optimality for recurrent problems, on the other hand, it has the undesirable property of being underselective. There may be several average-optimal policies which are not necessarily equivalent. Average optimality cannot distinguish between policies that present the same average reward but which have different initial transient rewards [18].

A more selective criterion called *bias optimality* can take such transient differences into account [18]. To be able to find a bias-optimal policy, iterative methods must approximate the expected average reward per step of the average-optimal policies, and then approximate the difference (or the *bias*) between the value of each state-action pair and that average. In other words, after finding the set of policies that achieve the primary objective of maximizing the long-run average reward, such methods search for that which also maximizes the short-run reward [14].

But bias optimality is still underselective. For a given MDP, several different bias-optimal policies can be found. The optimality criterion may be then refined to select a kind of *second order bias-optimal policy*, then a *third order bias-optimal policy* and so on [27, 4]. Such framework is called *sensitive-optimality* [17]. The limit order in which the optimal policies can no longer be differentiated corresponds to the *Blackwell-optimality* [3]. Such optimal policies are proved to be equivalent to the ones achieved using *discounted optimality* when the discount factor γ approaches 1 [11].

This paper proposes an original algorithm for finding optimal sensitive (average and multiple-order bias) reward policies for DMDPs (MDPs presenting deterministic transitions). The method, called *Trajectory Policy-Iteration* (TPI), works like the classic *Policy-Iteration* method [12], but instead of having a single real value associated to each state or state-action pair, TPI stocks the information relative to the trajectory within the process.

The main contribution of this work is the use of a more structured information (the *trajectory*) to represent the value of a state or state-action pair. We present the operations needed for using such structure inside a specific version of the policy-iteration method, and we show how the proposed algorithm can find multiple-order bias optimal policies.

The rest of the paper is organized as follows : Section 2 reviews related DP and MDP concepts and methods, Section 3 introduces the proposed method, Section 4 presents experimental results, and Section 5 concludes the paper.

2 Background

Dynamic Programming (DP) refers to a set of algorithms that can efficiently compute optimal policies for *Markovian Decision Processes* (MDPs), providing essential foundations for *Reinforcement Learning* (RL) methods [24, 21].

2.1 Markovian Decision Processes

The *Markovian Decision Process* (MDP) is in the center of a widely-used framework for approaching *automated control*, *sequential decision-making*, *planning*, and *computational reinforcement learning* problems [20, 24, 28, 8]. MDPs are sometimes called *Controllable Markov Chains*. *Deterministic MDPs* (DMDPs), the focus of this paper, constitute the particular set of MDPs that present deterministic transitions. A DMDP can be formally defined as a tuple $M = \{S, A, T, R\}$ where :

$S = \{s_1, s_2, ..., s_{|S|}\}$ is the finite set of states,

$A = \{a_1, a_2, ..., a_{|A|}\}$ is the finite set of actions,

$T(s, a) = s'$ is the transition function

$R(s, a, s') = r$ is the reward function

A DMDP is typically represented by a discrete finite state machine : at each time step the machine is in some state s, the agent observes that state and interacts with the process by choosing some action a to perform, then the machine changes into a new state s' and gives the agent a corresponding reward r. The transition function T defines the system dynamics by determining the next state s' given the current state s and the executed action a. The reward function R defines the immediate reward $r \in \mathbb{R}$ after transition from state s to s' with action a. The reward function can be stochastic, and in this case R can represent the mean reward. If the set of allowed actions is different for each state, a further function $E(s) \subseteq A$ must be defined, where $E(s)$ constitutes the subset of allowed actions for each state s.

A deterministic stationary policy π is a mapping between states and actions in the form $\pi(s) = a$, so as $\pi : S \to A$. The policy defines the agent behavior. The number of policies contained in Π, the set of possible policies, is exponential in the number of states, corresponding to $|\Pi| = |A|^{|S|}$. Solving an MDP means finding the policy of actions that maximizes the rewards received by the agent over time, according to some precise optimality criterion. When reward and transition functions are given, the MDP can be solved by *dynamic programming*. Such problem is generally referred to as *planning*.

2.2 Optimality Criteria

The definition of an optimal policy depends on the considered time-horizon, and on the method used to calculate utilities. In this paper, we are concerned with unbounded, potentially infinite, time-horizon. In that setting, the MDP is recurrent : there are no terminal states. The MDP corresponds to a *Finite Transition System*.

2.2.1 Discounted Optimality

When the agent has an infinite or unbounded time-horizon, the standard solution consists in evaluating policies using discounted cumulative rewards, where a *discount factor* $\{\gamma \in \mathbb{R} \mid 0 \leq \gamma < 1\}$, sometimes called *interest rate*, vanishes the utility of future rewards compared to immediate rewards.

The value (or utility) of a given state s, for a given policy π, and for a given discount factor γ, is defined as follows :

$$V^{\pi}_{\gamma}(s) = \sum_{t=1}^{\infty} \gamma^{t-1} R^{\pi}_{t}(s) \tag{1}$$

where $R^{\pi}_{t}(s)$ corresponds to the reward received in time t starting from the state s and following the policy π.

The value of the discounted cumulative rewards is always finite, which guarantees the convergence of iterative methods to an optimal solution [21]. Considering fixed discount factors, there is always at least one optimal policy that can be defined for any MDP. An optimal policy π^* is a policy that cannot be improved :

$$\forall s \in S \ . \ \forall \pi \in \Pi \ . \quad V_\gamma^{\pi^*}(s) \geq V_\gamma^\pi(s) \tag{2}$$

2.2.2 Average (or Gain) Optimality

In many domains, there is no natural interpretation for the discount factor γ. Even worst, in recurrent domains (where later rewards are as much important as earlier rewards) such criterion can distort the real utility of some sequences of actions. That situation is exemplified by the *crawling robot* problem [26]. Since the undiscounted sum of rewards can diverge on infinite time-horizon, the average reward received per time step becomes a preferable optimality criterion for this kind of problems [16, 25]. The average reward (or *gain*) starting on state s and following a policy π is :

$$\forall s \in S \ . \ \forall \pi \in \Pi \ . \quad g^\pi(s) = \lim_{k \to \infty} \frac{1}{k} \sum_{t=1}^{k} R_t^\pi(s) \tag{3}$$

The convenience of average optimality compared to discounted optimality can be observed regarding the DMDPs illustrated in the figures 1 and 2. On both problems, depending on the discount factor, the best policy under discounted-optimality can correspond to clearly worst solutions on the long run. The intuitively best policies for both DMDPs would be b, which are average-optimal for each respective problem.

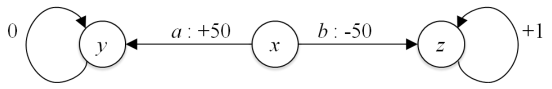


FIGURE 1 – The gain of the policy starting on the state x (on the center), and choosing the action a is $g^a = 0$, which corresponds to the reward of the recurrent state y (on the left). The alternative policy is choosing the action b, which leads to the state z (on the right). Such policy presents $g^b = +1$ and is gain-optimal. However, the discounted optimal policy for any discount factor $\gamma \leq 0.99$ is a. In fact b becomes better than a after 100 execution steps and up to the infinity.

In multi-chain MDPs, considering an infinite time-horizon, there is a convergent average reward for each communicant subset of states (i.e. for each recurrent class) within the process. Considering unichain MDPs, the average reward of a given policy π converges to a single value g^π independently of the starting state [21]. An average-optimal policy π^* is a policy that maximizes the expected gain on the long-term run :

$$\forall s \in S \ . \ \forall \pi \in \Pi \ . \quad g^{\pi^*}(s) \geq g^\pi(s) \tag{4}$$

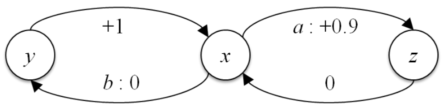


FIGURE 2 – The gain of the policy starting on the state x (on the center) and choosing the action a is $g^a = +0.45$ (the average per step on the circuit $\{x, z\}$, on the right). The action b presents $g^b = +0.5$ (the average per step on the circuit $\{x, y\}$, on the left) and is gain-optimal. However, the discounted optimal policy for any discount factor $\gamma < 0.9$ is a. In fact b becomes definitely better than a after 20 execution steps and up to the infinity.

2.2.3 Bias Optimality

In recurrent DMDPs, the use of average optimality conduces the agent to an optimal circuit within the process. However, average optimality does not take into account different initial transient rewards [18]. It means that, even though the gain g^π of a policy π is mathematically independent of the starting state s (considering infinite time-horizon and unichain MDPs), the total expected reward for a given time $t = k$, in general, is not. It is due to the fact that the rewards obtained in the transient path toward the optimal circuit disappear on the long-run averaging.

An example of that issue is given in the figure 3. Both policies a and b converge to a common average reward as time approaches infinity, but the sum of rewards corresponding to each policy for any finite time-horizon is not equivalent. Choosing a offers a better transient initial reward, and would be preferred over b.

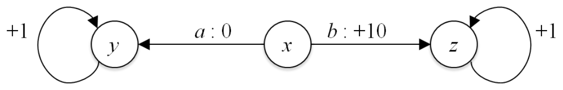


FIGURE 3 – The gain of both policies is equivalent ($g^a = g^b = +1$) but the transient paths correspond to different bias ($h^a = 0 - 1 = -1$ and $h^b = +10 - 1 = +9$). In this case, both policies a and b are gain-optimal, but only b is bias-optimal.

That difference (or *bias*) can be used as an auxiliary optimality criterion in order to compare states outside the optimal circuit. Bias-optimal policies are, among all gain-optimal policies, the ones which also optimize such transient initial rewards [14]. The combination of bias and gain enables the agent to optimally reach an optimal circuit from every state in a DMDP.

The *bias* corresponds to the sum of the differences between the value of each state-action pair and the average [14]. For that reason it is sometimes called *averaged adjusted sum of rewards* [16]. Since states in the recurrent class will be visited forever, the expected average reward cannot differ across these states. Since the transient states will never be reentered, they can at most accumulate a finite total reward before reaching a recurrent state.

The bias $h^{\pi}(s)$ of a given policy π starting from state s is the cumulative difference between the rewards received following the given policy and its average reward :

$$\forall s \in S \;.\; \forall \pi \in \Pi \;. \quad h^{\pi}(s) = \sum_{t=1}^{\infty} (R_t^{\pi}(s) - g^{\pi}) \qquad (5)$$

A policy π^* is *bias-optimal* if it is *gain-optimal* and also maximizes bias values for all states compared to all other possible policies :

$$\forall s \in S \;.\; \forall \pi \in \Pi \;. \begin{cases} g^{\pi^*}(s) > g^{\pi}(s) & \vee \\ g^{\pi^*}(s) = g^{\pi}(s) & \wedge \;\; h^{\pi^*}(s) \geq h^{\pi}(s) \end{cases} \qquad (6)$$

The bias of a transient path represents its total reward until reaching a recurrent class. This has the effect of treating the recurrent class as a single state and confining the bias analysis to the transient states. But distinguishing the time inside the recurrent cycle at which the rewards are received is necessary [14].

2.2.4 Sensitive Optimality

Even the relative bias optimality can be underselective, and higher order bias can be necessary to reach the maximum sensitivity. Such issue can be observed regarding the example in figure 4.

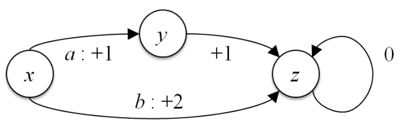


FIGURE 4 – The gain of both policies is equivalent ($g^a = g^b = 0$). The constant bias of both policies is also equivalent ($h^a = h^b = +2$). The relative bias of both policies is necessarily equivalent, given the fact that both policies enter into the same circuit at the same state. The cumulated reward following both policies is equivalent at any time, except in time $t = 2$, when policy b presents a bigger cumulated reward compared to a.

In the same way that bias-optimality constitutes a complementary criterion in order to select among all average optimal policies, it is possible to select *second order bias-optimal policies*, then *third order bias-optimal policies* and so on [4, 14]. Such framework is called *sensitive-optimality* [17]. The limit order in which the optimal policies cannot be longer distinguished corresponds to the *Blackwell-optimality* [3] and such optimal policies are proved to be equivalent to the ones achieved using discount-optimality when the discount factor γ approaches 1 [11].

In fact, for a given MDP, there is a discount factor γ^* from which the optimal policies do not change. Blackwell-optimal policies are discount-optimal policies for any $\gamma \geq \gamma^*$.

Sensitive optimality can be assessed by discounting discount factors that tend to 1. It is called *n-discount-optimality*, where $\{n \in \mathbb{Z} \mid n \geq -1\}$ [27]. In such approach, -1-discount optimality corresponds to the average optimality, 0-discount optimality corresponds to the first order bias optimality, 1-discount optimality corresponds to the second order bias optimality, and so on.

There is an n^* from which the optimal policies do not change. Blackwell-optimal policies are n-discount-optimal policies for all $n \geq n^*$. The number of bias orders necessary to distinguish all policies cannot be greater than the number of states in the MDP, so as $n^* < |S|$ [13].

A similar approach is the *n-average optimality* [23], which is equivalent to the *n-discount optimality* [13]. In this setting, the n^{th}-order gain $g^{\pi,n}$ of a given policy π over infinite time-horizon is defined as :

$$\lim_{t\to\infty} g_t^{\pi,n} = \begin{cases} 1/t \sum_{k=1}^{t} R_k^{\pi} & \textit{for } n = -1 \\ 1/t \sum_{k=1}^{t} g_k^{\pi,n-1} & \textit{for } n \geq 0 \end{cases} \qquad (7)$$

2.3 Dynamic Programming

Dynamic Programming (DP) refers to optimization methods which can be used to efficiently compute optimal policies of Markov Decision Processes (MDPs) when a model is given [1, 2, 21]. Classic DP methods typically presents polynomial-time convergence, even if the space of possible policies is exponential in relation to the number of state-action pairs. DP methods can be adapted as model-based reinforcement learning (RL) methods, and provide essential intuitions for model-free RL methods. The key idea of DP, and of RL in general, is the use of value (or utility) functions to organize and structure the search for optimal policies [24].

Value-Iteration (VI) [1] and *Policy-Iteration* (PI) [12] are the two fundamental and widely used DP algorithms. It had been demonstrated that PI converges at least as quickly as VI [20], and, in practice, PI has been remarkably successful and shown to be most effective [15].

There is a significant research effort for understanding the complexity of PI. The demonstration of its tight upper and lower bounds is still an open problem. Considering stochastic MDPs under discounted optimality, with a fixed discount rate $0 \leq \gamma < 1$, PI is proved to be *strongly polynomial* [29, 22], i.e. the number of operations required to compute an optimal policy has an upper bound that is polynomial in the number of state-action pairs. Inside each iteration, PI uses at most $O(|E| \cdot |S|) = O(|S|^2 \cdot |A|)$ arithmetic operations. Unfortunately, the convergence time increases with rate $\frac{1}{1-\gamma}$ [29]. It constitutes a major impediment for using high discount factors ($\gamma \to 1$) in practice. Typically, average optimization is a more difficult problem than discounted optimization [6]. PI can need an exponential number of iterations under average-optimality for stochastic MDPs [7].

Deterministic MDPs can be viewed as *weighted directed graphs*, and solving such DMDPs is essentially equivalent to find *minimum mean-cost cycles*. For this reason, and contrarily to stochastic MDPs, deterministic MDPs under average-optimality can be solved in strongly polynomial-time, $O(|E| \cdot |S|) = O(|S|^2 \cdot |A|)$ [19]. Experimental studies suggest that PI works very efficiently in this context [5]. The number of iterations performed by PI when applied to a weighted directed graph seems to be at most the number of edges in the graph [10], then limited to $O(|E|)$. In addition, for deterministic MDPs, the number of arithmetic operations inside each iteration is at most $O(|E|)$ [10].

3 Proposed Solution

Within every finite DMDP M, every policy $\pi \in \Pi(M)$ contains at least one closed cyclic set of states-action pairs (i.e. at least one recurrent class of states). Given a policy π, any state $s \in S$ must be, either part of a circuit, or part of a transient path leading to a circuit. It can be demonstrated by the fact that any directed graph having one exiting edge per node (like a policy) must present at least one cycle [9]. If M is unichain, an optimal policy π^* will present a single *optimal circuit*, where the other states lead to, like illustrated in figure 5.

For each state-action pair, *Trajectory Policy-Iteration* (TPI) estimates a *cruise value* (g), representing the average reward per time step in the periodic part of the walk, and a *sprint value* (f), representing the average reward per time step in the transient part of the walk. The *cruise value* is the average in the circuit, just another name for the *gain*. The *sprint value* is the average in the transient path, then representing something slightly different from the *bias*. The *bias* is the sum of the differences between the reward of each state in the transient path and the average in the circuit. The *sprint* is just the initial average in the transient part of the walk, regardless of the average in the circuit.

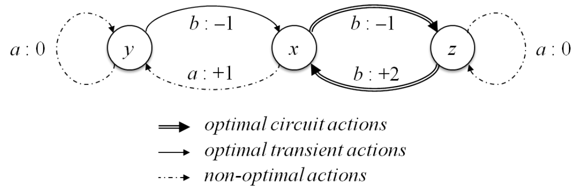


FIGURE 5 – The optimal circuit is the one with the best average reward per cycle within the MDP. The optimal transient path is the best one to reach the optimal circuit.

Such values are represented inside a special structure called *trajectory*. The basic TPI main loop, `PolicyIteration` is described in the algorithm 1. Given any initial policy π_0, the algorithm iterates calculating the value of the current policy π_i, and then greedily improving it for all states where an improvement is possible, as described by the algorithm 2, `ImprovePolicy`.

Algorithm 1 PolicyIteration(S, A, R, T)
```
$\pi_0 \leftarrow InitializePolicy()$
repeat
  $Q_i \leftarrow EvaluatePolicy(\pi_i)$
  $\pi_{i+1} \leftarrow ImprovePolicy(\pi_i, Q_i)$
until $\pi_{i+1} = \pi_i$
```

Algorithm 2 ImprovePolicy(π_i,Q_i)
```
for all $s \in S$ do
  $V_i(s) = Q_i(s, \pi_i(s))$
  if $\max_a[Q_i(s,a)] > V_i(s)$ then
    $\pi_{i+1}(s) \leftarrow \arg\max_a[Q_i(s,a)]$
  else
    $\pi_{i+1}(s) \leftarrow \pi_i(s)$
  end if
end for
```

The initial policy is typically initialized using an uniform distribution over the possible actions for each state. At this level, the proposed method (defined by algorithms 1 and 2) corresponds exactly to the original Policy-Iteration method [12]. The novelty introduced by TPI is the use of a structured value to represent the set of utilities.

In the rest of the section, we define such structure (the *trajectory*), as well as the necessary methods and operations in order to implement the *Trajectory Policy-Iteration* method (i.e. how to evaluate a given policy, how to compare, and how to compose trajectories).

3.1 Trajectory

A trajectory corresponds to a structure in the form $\tau^{\pi}(x) = \langle v/n : z : w/m \rangle$. It represents the trajectory τ which starts in the state x and follows the policy π. Such trajectory reaches the state z after n steps through a transient path where a total reward v is cumulated, and then enters into a circuit with period m where a total reward w is cumulated. A trajectory is exemplified in the figure 6. In TPI, $V(s)$ values (the state utilities) as well as $Q(s, a)$ values (the state-action utilities) are trajectories.

Considering a given trajectory $\tau = \langle v/n : z : w/m \rangle$, the *cruise* (or *gain*) corresponds to the average reward per step within the circuit, if the circuit is defined, otherwise the cruise is zero. Similarly, the *sprint* corresponds to the average reward per step on the transient path.

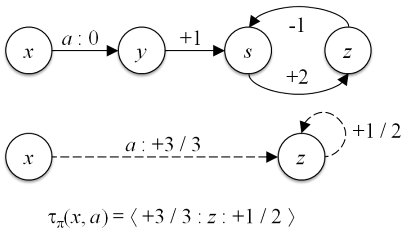


FIGURE 6 – On the top we can observe a sample trajectory within some DMDP, where, starting from the state x, with action a, and following the policy π afterward, the process reaches state z in $n = 3$ steps, cumulating rewards up to a value of $v = +3$, then entering in a circuit composed by $m = 2$ states where a reward equivalent to $w = +1$ can be cumulated at each completed lap. That trajectory is represented as $\tau(x, a) = \langle +3/3 : z : +1/2 \rangle$, and corresponds to a *cruise* of $g = +0.5$ and a *sprint* of $f = +1$. Note that the chosen reference state is z and not s, because it is the state which represents the better utility to this trajectory, given that $\langle +3/3 : z : +1/2 \rangle > \langle +1/2 : s : +1/2 \rangle$.

3.2 Policy Evaluation

The value of a given policy π is iteratively calculated by the algorithm 4, based on both the reward and the transition functions (R and T). The value $Q^{\pi}_{j}(s, a)$ represents the trajectory calculated at the iteration j for the state s. The trajectory starts on the state s, where the action a is executed. The policy π determines the actions on the other states. The algorithm converges to the exact Q at worst in $2 \cdot |S|$ iterations. $|S|$ steps can be necessary to find the biggest circuit (if a circuit that passes by all states exists in the process), and then more $|S|$ steps can be necessary to find the reference state.

At the beginning, when $j = 0$, all the state-action pairs are initialized with the corresponding immediate transition and reward, as explained in algorithm 3. At each new iteration, such values are recalculated by incrementing the immediate trajectory with the next step trajectory, which implies either in growing the transient path, or in defining a circuit. Such composition of trajectories is defined in the algorithm 5, and explained in the figure 7. Thus, before convergence, the trajectory corresponding to the utility of some state-action pair will be in one of the following situations : (a) $\tau(x, a) = \langle v/n : z : 0/0 \rangle$: an open transient path toward some yet unknown circuit, (b) $\tau(x, a) = \langle 0/0 : x : w/m \rangle$: the starting point of a circuit, or (c) $\tau(x, a) = \langle v/n : z : w/m \rangle$: a path leading to a circuit.

At the convergence, all the state-action pairs must be closed, being either in the case (b) or in the case (c), i.e. all the state-action pairs must know a circuit. Furthermore, the circuits as well as the paths must be optimal.

Algorithm 3 InitializeUtilities()
```
for all ($x \in S, a \in A$) do
    $y = T(x, a)$
    $r = R(x, a, y)$
    $Q_0(x, a) \leftarrow \begin{cases} \langle 0/0 : x : r/1 \rangle & \text{if } (y = x) \\ \langle r/1 : y : 0/0 \rangle & \text{if } (y \neq x) \end{cases}$
end for
```

Algorithm 4 PolicyEvaluation(π)
```
$Q_0 \leftarrow \mathit{InitializeUtilities}()$
repeat
    for all ($x \in S, a \in A$) do
        $\tau \leftarrow \mathit{IncrementTrajectory}(x, a, \pi)$
        $Q_{j+1}(x, a) \leftarrow \max\{Q_j(x, a), \tau\}$
    end for
until $Q_{j+1} = Q_j$
```

Algorithm 5 IncrementTrajectory(x,a,π)

```
$y = T(x, a)$
$r = R(x, a, y)$
$V_j(y) = Q_j(y, \pi(y)) = \langle v/n : z : w/m \rangle$
$\tau \leftarrow \begin{cases} \langle 0/0 : x : r/1 \rangle & \text{if } (y = x) \\ \langle 0/0 : x : (v+r)/(n+1) \rangle & \text{if } (y \neq x = z) \\ \langle (v+r)/(n+1) : z : w/m \rangle & \text{if } (y \neq x \neq z) \end{cases}$
$Q_{j+1}(x, a) \leftarrow \tau$
```

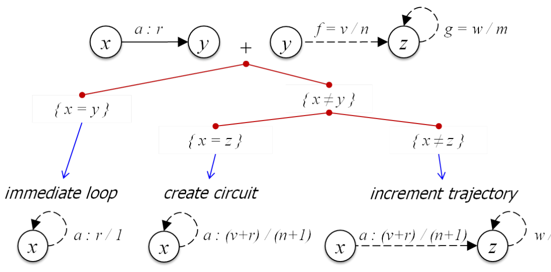


FIGURE 7 – The intuition behind the `IncrementTrajectory` method.

3.3 Trajectory Comparison

To be able to execute the algorithms 1 and 4, we must define how the function `max` can be applied to trajectories. It means defining how two trajectories can be compared. Two given trajectories τ_a and τ_b can be compared based on their relative quality to each other, which determines if the utility represented by τ_a is greater, fewer or equivalent to the utility of τ_b. The quality is not an absolute property of a given trajectory, but a comparative value relative to another trajectory. It is given by the algorithm 6.

Algorithm 6 Quality(τ_a,τ_b)

```
$\tau_a = \langle v_a/n_a : z_a : w_a/m_a \rangle$
$\tau_b = \langle v_b/n_b : z_b : w_b/m_b \rangle$
$k \leftarrow \max(n_a + m_a, n_b + m_b)$
$u_a \leftarrow n_a \cdot f_a + (k - n_a) \cdot g_a$
$q_a \leftarrow \begin{cases} +\infty & \text{if } (m_a = 0) \wedge (m_b > 0) \\ -\infty & \text{if } (m_a > 0) \wedge (m_b = 0) \\ f_a & \text{if } (m_a = 0 = m_b = 0) \\ g_a & \text{if } (m_a, m_b > 0) \wedge (g_a \neq g_b) \\ u_a & \text{if } (m_a, m_b > 0) \wedge (g_a = g_b) \wedge (u_a \neq u_b) \\ -n_a & \text{if } (m_a, m_b > 0) \wedge (g_a = g_b) \wedge (u_a = u_b) \end{cases}$
```

When two given trajectories are closed (i.e. have defined circuits), they can be compared regarding their *cruise*. In the long run, the trajectory with better cruise must always be preferred. In case of equivalent *cruise*, the *sprint* is used for tie-breaking. However, simply comparing sprints is not sufficient to correctly compare two trajectories with equivalent cruise. The example illustrated in the figure 8 presents two trajectories with equivalent cruise, and where the trajectory with smaller sprint is better. It is because both sprints have a worst average compared to the cruise, and in this case it is better to chose the shortest transient path.

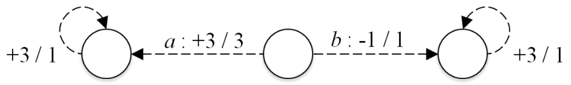


FIGURE 8 – Both trajectories $\tau_a = \langle +3/3 : z : +3/1 \rangle$ and $\tau_b = \langle -1/1 : z : +3/1 \rangle$ have the same cruise ($g_a = g_b = +3$), but the first one presents a better sprint ($f_a = +1 > f_b = -1$). Despite this, the reward cumulated by τ_b will be ever superior to the rewards cumulated by τ_a from the moment which both have completed at least one lap within their respective circuits. In the example, it corresponds to four steps ($k = 4$). At that time, the runner in τ_a will have traveled all the transient path, entered into the circuit, and just completed its first lap, and the runner in τ_b will be completing its third lap. It means that τ_b is better than τ_a after four steps, even if its sprint is lower : $\sum_{t=1}^{4} r_t^{\tau_a} = 6 < \sum_{t=1}^{4} r_t^{\tau_b} = 8$.

When the cruise of both trajectories is the same $g_a = g_b = g$, it is possible to take the cumulated value (based on the averages f and g) of each trajectory considering a common time-horizon, $k = \max(n_a + m_a, n_b + m_b)$. At time k, both trajectories will have completed at least one lap in the periodic part of the walk (in the circuit). Because the cruise is the same, when both trajectories are running in the circuit, their average distance does not change. Thus, the best trajectory is the one that arrives in time k with a greater cumulated reward. In other words, comparing such trajectories on the infinite time-horizon is equivalent to comparing them in time k.

When the operation `max` is called inside the `PolicyIteration` method (algorithm 1), only closed trajectories are compared. The trajectories are complete at that level. But to execute the `PolicyEvaluation` method (algorithm 4), the comparison must be done also considering open trajectories. In order to guarantee that the construction of a trajectory will not stop at the first circuit found, open trajectories must be preferred over closed trajectories. If both trajectories are still open at a given iteration, they can be compared regarding their sprint.

3.4 High-order values

The first-order cruise $g_\pi(s)$ of a given state s represents the average reward of the policy π in the limit of infinite time. After convergence, that value must correspond to the average reward into an optimal-circuit. The first-order sprint $f_\pi(s)$ represents the average reward on the transient path, and it must represent the best path toward an optimal-circuit.

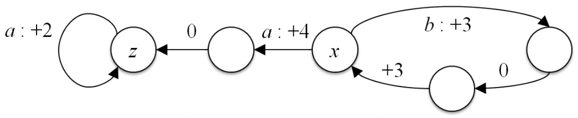
The intuition for computing high-order values comes from the idea of comparing two different trajectories by placing two hypothetical runners on the starting state of each trajectory, and then simply comparing who must necessarily be the winner in a long race, i.e. the one that cumulates more reward given a common horizon. The first criterion, as explained, is the cruise. If trajectory τ_a presents a greater cruise than trajectory τ_b, then there exist a time $k^\star$ from when $\sum_{t=1}^{k} r_t^{\tau_a} > \sum_{t=1}^{k} r_t^{\tau_b}$ for all $k > k^\star$, where r_t^τ is the expected immediate reward in time t inside the trajectory τ. Such situation is exemplified in figures 1 and 2.

If the cruise of both compared trajectories is equivalent, then the distance between the two runners (the average difference considering their positions) will not change once they are both running within their respective (and cruise equivalent) circuits. In this case, the transient initial running difference must be compared, but it cannot be done by a simple sprint comparison. We must verify who is further away in the moment when both runners have completed at least one lap within their respective circuits. In the particular DMDP presented in figure 3, the sprint can be used alone to capture such difference. This is possible because, in that example, both transient paths have the same size, and both runners (corresponding to the two alternative policies) enter in their circuits (of equivalent cruise) at the same time. The figure 8 presents an example where the trajectory having the higher sprint is not the best one.

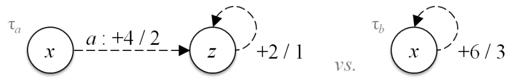
To understand the meaning of such different orders, an analogy can be made with the relation between the different displacement derivatives (position, velocity, acceleration, jerk, etc.). When we consider a single order of values, the successive rewards represent the different velocities at each elapsed time step, and the cumulated reward corresponds to the traveled distance. If some object a presents a bigger average velocity than some object b, then a must have traveled a bigger distance than b. However, having equivalent average velocities does not imply traveling the same distance. It is for this reason that higher-order values are necessary to distinguish two trajectories.

With two orders of values, the successive rewards can be compared to the acceleration. In a race, the object with the biggest average acceleration must be the winner. However, two objects with the same average acceleration will not necessarily present the same average velocity. Even if that objects present the same average acceleration and also the same average velocity, it does not mean that both will travel the same distance.

In the worst case, two trajectories may require $|S|$-order values to be distinguished. To be able to evaluate and find policies in a o-order bias quality, each trajectory must stock o pairs of v and w values. In the figure 9 we can see how two trajectories can be compared using the second-order values.



($k = 3$)	τ_a			τ_b		
$\boldsymbol{t}$	1	2	3	1	2	3
$\boldsymbol{r_t}$	4	0	2	3	0	3
$\Sigma \boldsymbol{r_t}$	4	4	**6**	3	3	**6**
$\Sigma\Sigma \boldsymbol{r_t}$	4	8	**14**	3	6	**12**



$\tau_a^1 = \langle +4 / 2 : z : +2 / 1 \rangle$ $\tau_b^1 = \langle 0 / 0 : x : +6 / 3 \rangle$

$\tau_a^2 = \langle +8 / 3 : z : +2 / 1 \rangle$ $\tau_b^2 = \langle 0 / 0 : x : +12 / 6 \rangle$

FIGURE 9 – Comparing two trajectories with different path sizes and circuit periods using the second-order values.

4 Experimental Results

In a first set of experiments, we verify the convergence of the `PolicyEvaluation` method (algorithm 4). The number of iterations (j) necessary for convergence (max and average) is given for experiments with DMDPs of size $|S|$ varying from 2 to 100, and fixed $|A| = 1$. Because there is a unique possible action for each state, the process constitutes a *Markov Chain*, which is equivalent to a policy. For each setting, 10000 DMDPs have been randomly generated, with integer rewards varying between 0 and $|S|$. We can see in figure 10 that, in practice, the worst cases are always below the suggested theoretical limit. The number of iterations is never greater than $2 \cdot |S|$. Following the results, bigger is the number of states in the DMDP, lower is the number of iterations necessary to evaluate a policy. This is due to the way the DMDPs are generated, where the period of the created circuits tends to be much smaller than the number of states.

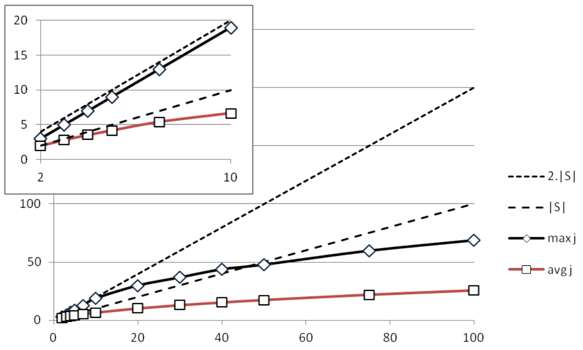


FIGURE 10 – Convergence time for the `PolicyEvaluation` method.

In a second set of experiments, we verify the convergence of the proposed `TPI` method. The same kind of DMDPs have been generated, this time using $|A| = 2$. As expected, polynomial convergence performances are shown in figure 11.

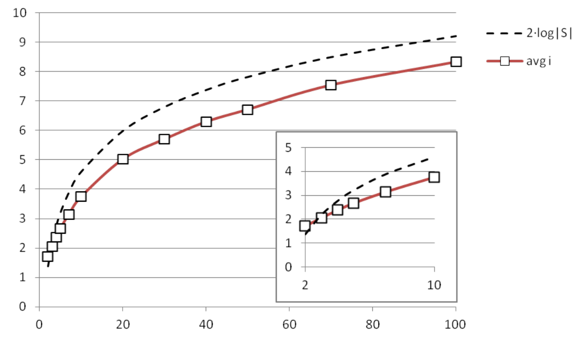


FIGURE 11 – Average number of iteration necessary for convergence using `TPI`, experimenting with DMDPs of size $|S|$ varying from 2 to 100, and fixed $|A| = 2$. For each setting, 10000 DMDPs have been randomly generated, with integer rewards varying between 0 and $|S|$.

The performance of `TPI` using 1st and 3rd-order values has been compared to a discounted version of the policy-iteration algorithm using a discount factor $\gamma = 0.9$. Figure 12 shows the number of times when `TPI`(3rd-order) beats `PI`($\gamma = 0.9$), the number of times when `TPI`(1st-order) beats `PI`($\gamma = 0.9$) (which means the number of times `PI`($\gamma = 0.9$) was not able to find an average-optimal policy), and the number of times `PI`($\gamma = 0.9$) beats `TPI`(3rd-order) (which means the number of times `TPI`(3rd-order) was not able to find a Blackwell-optimal policy). These performances are based on the cumulated reward of each policy, over a time-horizon equivalent to $|S|^2$, starting on each possible different state.

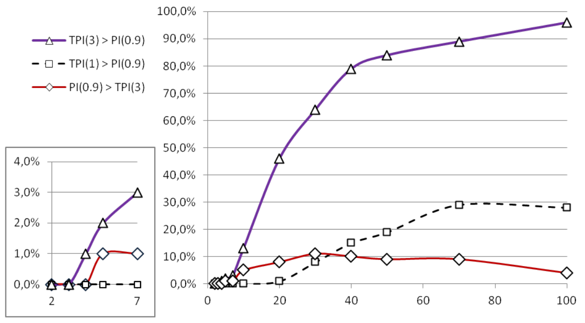


FIGURE 12 – Compared performances for `TPI`(3rd-order), `TPI`(1st-order), and `PI`($\gamma = 0.9$).

5 CONCLUSION

This paper presented *Trajectory Policy-Iteration* (TPI), an original dynamic programming method able to find exact optimal policies for deterministic MDPs over unbounded time-horizon using average and multiple-order bias optimality. To the best of our knowledge, the proposed method is the first dynamic programming algorithm based on the representation of trajectories inside a DMDP.

This paper also presents an original meaning for the multiple-order bias optimality. Firstly, interpreting the *gain* and the *bias* of two different policies as, respectively, the *cruise* velocity and the initial *sprint* velocity of two runners in a race. A second analogy compares the different orders of bias to the different derivatives of displacement in kinematics (velocity, acceleration, jerk, etc.). In a fixed-time race, if a runner presents an average acceleration superior to all the other runners, he/she must be the winner. If two runners present the same average acceleration, the winner is the one with biggest average velocity.

The experiments presented on this paper have demonstrated promising results, indicating that the ideas implemented in `TPI` can constitute a viable way for dealing with infinite DMDPs. The next steps of research include formally proving the convergence bounds and extend the method to stochastic MDPs and factored MDPs. In the same way, `TPI` can be converted into a RL method.

Références

[1] R.E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[2] D. P. Bertsekas. *Dynamic programming and optimal control*, volume I. Athena Scientific, Belmont, Mass., 3 edition, 2005.

[3] D. Blackwell. Discrete dynamic programming. *Ann. Math. Stat.*, 33(2) :719–726, 1962.

[4] Xi-Ren Cao and Junyu Zhang. The n^{th}-order bias optimality for multichain markov decision processes. *Automatic Control, IEEE Transactions on*, 53(2) :496–508, 2008.

[5] A. Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Trans. Design Autom. Electr. Syst.*, 9(4) :385–418, 2004.

[6] J. Huang E. A. Feinberg. Strong polynomiality of policy iterations for average-cost mdps modeling replacement and maintenance problems. *Operations Research Letters*, 41(3) :249–251, 2013.

[7] J. Fearnley. Exponential lower bounds for policy iteration. In *ICALP (2)*, volume 6199 of *LNCS*, pages 551–562. Springer, 2010.

[8] E.A. Feinberg and A. Shwartz. *Handbook of Markov Decision Processes : methods and applications.* Kluwer, 2002.

[9] J. L. Gross, J. Yellen, and P. Zhang. *Handbook of Graph Theory.* Chapman & Hall/CRC, 2nd edition, 2013.

[10] T. D. Hansen and U. Zwick. Lower bounds for howard's algorithm for finding minimum mean-cost cycles. In *Proceedings of the 21st ISAAC*, volume 6506 of *LNCS*, pages 415–426. Springer, 2010.

[11] A. Hordijk and A. Yushkevich. Blackwell optimality. In E. Feinberg and A. Shwartz, editors, *The Handbook of Markov Decision Processes*, chapter 8, pages 231–268. Kluwer, 2002.

[12] R.A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA, 1960.

[13] L. Kallenberg. Finite state and action mdps. In Eugene A. Feinberg and Adam Shwartz, editors, *The Handbook of Markov Decision Processes : Methods and Applications*, chapter 2, pages 21–87. Springer US, Boston, MA, 2002.

[14] Mark E. Lewis and Martin L. Puterman. Bias optimality. In E. Feinberg and A. Shwartz, editors, *The Handbook of Markov Decision Processes*, chapter 3, pages 89–111. Kluwer, 2002.

[15] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the 11th UAI)*, pages 394–402, 1994.

[16] S. Mahadevan. Average reward reinforcement learning : Foundations, algorithms, and empirical results. *Mach. Learn.*, 22(1-3) :159–195, 1996.

[17] S. Mahadevan. Sensitive discount optimality : Unifying discounted and average reward reinforcement learning. In *Proceedings of the 13th ICML*, pages 328–336. Morgan Kaufmann, 1996.

[18] S. Mahadevan. Learning representation and control in markov decision processes : New frontiers. *Foundations and Trends in Mach. Learn.*, 1(4) :403–565, 2009.

[19] C. Papadimitriou and J. N. Tsitsiklis. The complexity of markov decision processes. *Math. Oper. Res.*, 12(3) :441–450, 1987.

[20] M.L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming.* Wiley, 1 edition, 1994.

[21] M.L. Puterman and J. Patrick. Dynamic programming. In C. Sammut and G.I. Webb, editors, *Encyclopedia of Machine Learning*, pages 298–308. Springer, 2010.

[22] U. Mall S. Kalyanakrishnan and R. Goyal. Batch-switching policy iteration [to appear]. In *Proceedings of the 25th IJCAI.* AAAI Press, 2016.

[23] K. Sladký. On the set of optimal controls for markov chains with rewards. *Kybernetika*, 10(4) :350–367, 1974.

[24] R.S. Sutton and A.G. Barto. *Introduction to Reinforcement Learning.* MIT Press, 1998.

[25] P. Tadepalli. Average-reward reinforcement learning. In C. Sammut and G.I. Webb, editors, *Encyclopedia of Machine Learning*, pages 64–68. Springer, 2010.

[26] M. Tokic, J. Fessler, and W. Ertel. The crawler, a class room demonstrator for reinforcement learning. In *Proceedings of the 22nd FLAIRS*, pages 160–165. AAAI Press, 2009.

[27] A. Veinott. Discrete dynamic programming with sensitive discount optimality criteria. *Ann. Math. Stat.*, 40(5) :1635–1660, 1969.

[28] M. Wiering and M. Otterlo. Reinforcement learning and markov decision processes. In *Reinforcement Learning : State-of-the-Art*, pages 3–42. Springer, 2012.

[29] Y. Ye. The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate. *Math. Oper. Res.*, 36(4) :593–603, 2011.