A Floyd-Warshall Approach to Value Computation in Markov Decision Processes

Aymeric Côme, Éric Fabre, Loïc Hélouët

A Floyd-Warshall Approach to Value Computation in Markov Decision Processes

Aymeric Côme†, Eric Fabre, Loïc Hélouët

Univ Rennes, Inria, Campus de Beaulieu, 35042 Rennes cedex, France.

Contributing authors: name.surname@inria.fr;
†This work is supported by ANR MAVeriQ (ANR-20-CE25-0012).

Abstract

Value and policy iteration are classical algorithms to maximize the average discounted reward of an MDP. They rely on a breadth-first exploration strategy in the future of each state to update its value and possibly change the action policy at this state. This paper revisits this paradigm and examines a depth-first search strategy. It reformulates the average reward computation as an integral over (future) paths that is better expressed in the formalism of weighted automata. Policy evaluation can then be solved by a Floyd-Warshall algorithm, which gathers at once the rewards along possibly infinite runs. This reformulation opens the way to new approximation schemes for the value function. The same formalism also gives access to other quantities of interest, as the gradient of the average reward with respect to model or policy parameters, or the variance of the reward. The behaviors and performances of this value estimation scheme are illustrated on several benchmarks.

Keywords: MDP, value iteration, policy iteration, Floyd-Warshall, policy gradient

1 Introduction

The derivation of optimal policies for Markov Decision Processes (MDP) typically relies on two steps : an *evaluation* step, which assigns a value to every state, estimating its average future reward under the current policy, and the *policy improvement* step, which updates the action taken at each state to favor transitions towards the most rewarding future states. Under a given policy, state values satisfy a linear fix-point equation, which can also be formulated as a linear optimization problem [18]; the resolution techniques can usually be classified as direct inversion, dynamic programming or iterative algorithms. The first category involves the explicit computation of a matrix inverse, which would incur a cubic complexity (in the number of states), however some approximate methods such as singular value decomposition allow for better performance [9]. More affordable solutions use linear programming with gradient methods [17], or iterative approaches, such as the classical Value Iteration, which estimates the average reward of a state by a breadth first exploration of its future up to some bounded depth and under the current policy (see [18], Chap. 6 for details). Despite the simplicity of this approach, Value Iteration is still actively studied, to speed-up the convergence by using structural properties, randomization or even heuristics [8, 22, 10].

Inspired by this work, this paper explores a new way to derive the value function of policies in MDPs, favoring a depth first exploration along the most promising runs rooted at each state. Specifically, the computation of state values derives from

a vertex elimination technique in the MDP graph. Eliminating a vertex captures in a single step all runs (and circuits) going through this vertex, thus gathering at once the rewards of possibly infinitely many runs. This recursive vertex elimination takes the form of a Floyd-Warshall (FW) procedure.

This FW reinterpretation can be considered as an alternative way to compute the integral of a reward function over the (stochastic) space of trajectories of the underlying Markov process, once a policy is fixed. The technique was originally developed to analyze properties of weighted automata, with transition weights taken in a semi-ring [5]. We show in this paper that it is extremely versatile: the same algorithm can be used to compute the value difference of two policies, to compute the gradient of the value function with respect to parameters of a policy or to parameters of the MDP, or even to compute the variance of the value function. This is a first advantage of this reformulation. Regarding performances, the FW approach converges in bounded time to the exact value function, but with the same cubic complexity as standard techniques that solve Bellman's fix point equation relating value and reward. Nevertheless, it opens the way to new approximation schemes for the value function, as it progressively integrates weights (*i.e.*, rewards) of paths visiting a growing set of states. Integration could then stop before all states are included in path descriptions, possibly speeding up the convergence of policy iteration. This new formulation is evaluated on random instances of standard models: grid worlds (where the reward lies at a goal state in a maze) and river swim (where the biggest gain requires to swim up a river stream, which might not be worth the effort).

The paper is organized as follows. Section 2 introduces notations and recalls basic results about values and optimal policies in MDPs. Section 3 reformulates value evaluation for a fixed policy as an integration over trajectories in the MDP, performed with Floyd-Warshall (FW). It also discusses the versatility of the path integration approach. Section 4 examines a specific strategy to drive the FW algorithm, in order to quickly collect the rewards of the most contributing paths. This progressive integration suggests new approximation schemes and offers an analogy with the A* algorithm. Section 5 illustrates the behavior of the FW approach and compares it to standard value iteration. Due to limited space, some proofs and discussions are omitted or sketched. They appear in an extended version [1] of this paper.

2 Framework and notations

A finite *Markov Decision Process* (MDP) $M = (S, A, P, R)$ models an environment in which an agent stochastically moves from state to state by choosing actions, and gets rewards that will motivate the decision rule followed. S is the (finite) state space of size N, A is the finite set of actions available to the agent, $P(s, a, s') = P(s' \,|\, s, a)$ is the transition function encoding the probability to move from a state s to a state s' upon choosing action a, and $R(s, a) \in \mathbb{R}_+$ is the reward function. The transition probabilities in each state are *Markovian*, *i.e.*, only depend on the current state and not on the states visited and actions taken before (a.k.a. the *history*). For reasons explained below, the reward is required to be bounded: w.l.o.g., it is assumed that $R \in [0, 1]$. The starting state s_0 can be sampled from a distribution μ_0 over S, but w.l.o.g., we assume s_0 is fixed.

A MDP generates infinite *runs*, *i.e.*, sequences of the form $s_0, a_0, r_0, s_1, a_1, r_1, \dots$ where $r_i = R(s_i, a_i)$ and $P(s_{i+1} \mid s_i, a_i) > 0$, that depend on action choices of the agent and on state changes chosen by the environment. A stationary, Markovian and deterministic *policy* is a map $\pi : S \to A \in \Pi$; it does not depend on time nor on the history of the run, and prescribes one single action rather than a distribution on A. Fixing such a policy π for a MDP $M = (S, A, P, R)$ induces a Markov Reward Processes $M_\pi = (S, P_\pi, R_\pi)$, with state space S, transition probability $P_\pi(s' \,|\, s) = P(s' \,|\, s, \pi(s))$ and reward $R_\pi(s) = R(s, \pi(s))$ for any $\pi \in \Pi$, $(s, s') \in S^2$. Then, for a given policy π, and a discount factor $\gamma \in (0, 1)$, the *value* of a state s is defined as the expected *cumulative discounted reward* when starting a run from s and taking actions accordingly to π:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{n=0}^{+\infty} \gamma^n r_n \,|\, s_0 = s \right] \qquad (1)$$

The goal is then to find a policy π_* that achieves the highest value for starting state s_0. This policy π_* is called an *optimal policy*. We restrict ourselves to Markovian, stationary and deterministic policies, as it is well known that

using decision rules from this class is sufficient to achieve optimality in MDPs with finite sets of states and actions ([18], Thm. 9.1.8). Following the definition of optimal policies, we can also define an optimal value $V_*(s)$ for every state $s \in S$, defined as $V_*(s) = V_{\pi_*}(s) = \max_{\pi \in \Pi} V_\pi(s)$. For a given policy π, the value of a state depends on the values of its neighbors. This property is formally expressed by the *Bellman operator* $\mathcal{T}_\pi : \mathbb{R}^N \to \mathbb{R}^N$:

$$\forall V \in \mathbb{R}^N, \forall s \in S, \quad \mathcal{T}_\pi(V)(s) = R_\pi(s) + \gamma \sum_{s' \in S} P_\pi(s'|s) V(s') \tag{2}$$

A quick calculation shows that V_π is a fix point of $\mathcal{T}_\pi$; and $\mathcal{T}_\pi$ is a γ-contraction[1], so that V_π is the only fix point ([18], Thm. 6.2.3). Similarly, V_* is the unique fix point of the γ-contraction called the *optimal Bellman operator*, defined as:

$$\begin{aligned} \forall s \in S, \mathcal{T}_*(V_*)(s) = & \\ & \max_{a \in A} \{ R(s,a) \\ & \quad + \gamma \sum_{s' \in S} P(s'|s,a) V_*(s') \} \\ & = V_*(s) \end{aligned} \tag{3}$$

Given a value function $V \in \mathbb{R}_+^N$, a *V-improving policy* is a policy π_V such that

$$\begin{aligned} \forall s \in S, & \\ \pi_V(s) &= \mathcal{T}_*(V)(s) \\ &= \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right\} \end{aligned} \tag{4}$$

Obviously, one can obtain a *V-improving policy* by choosing

$$\begin{aligned} \forall s \in S, & \\ \pi_V(s) \in \arg\max_{a \in A} \{ & R(s,a) \\ & + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \} \end{aligned} \tag{5}$$

[1] An operator $\mathcal{T}_\pi$ is a γ-contraction iff, for every $V_1, V_2 \in \mathbb{R}^S$, $\|\mathcal{T}_\pi(V_1) - \mathcal{T}_\pi(V_2)\|_\infty \leq \gamma \cdot \|V_1 - V_2\|_\infty$

For an optimal policy π_* we necessarily have $\pi_{V_*} = \pi_*$. Consequently, searching for the optimal policy can be done by maximizing values.

Let us now consider *policy evaluation* methods to compute the values of a given policy. Firstly, in a finite MDP the fix point equation (2) can be rewritten in matrix form as:

$$V_\pi = R_\pi + \gamma P_\pi V_\pi \iff V_\pi = (I_d - \gamma P_\pi)^{-1} R_\pi \tag{6}$$

where $(I_d - \gamma P_\pi)$ is invertible due to the discount being strictly less than 1 and P_π being a stochastic matrix ([18], Thm 6.1.1). Hence (6) gives a straightforward way to compute V_π. However, computing V_π requires inverting the matrix $(I_d - \gamma P_\pi)$. This can be done in $O(|S|^3)$ with Gauss-Jordan elimination (see [19], Chap.3).

Another way to compute a value vector V_π is by *value iteration* (shown in Algorithm 1 below). Value iteration recursively applies the Bellman operator $\mathcal{T}_\pi$ to update the value of each state. This strategy is guaranteed to converge, as V_π is the unique fix point of the γ-contraction $\mathcal{T}_\pi$ ([18], Thm. 6.2.3; [6]). Notice that the value V_k returned by Algorithm 1 is an approximation of V_*. However, the stopping criterion $\|V_{k+1} - V_k\|_\infty \geq \epsilon \frac{1-\gamma}{\gamma}$ ensures that $\|V_{k+1} - V_*\|_\infty < \epsilon$ ([18], Thm 6.3.1), in other words, the estimate V_k is *ϵ-optimal*.

Algorithm 1 Value Iteration

Input: MDP M, policy π, error bound $\epsilon > 0$, discount γ, initial value $V_0 \in \mathbb{R}^N$
Initialization: $k = 0$, $V_1 = \mathcal{T}_\pi(V_0)$
while $\|V_{k+1} - V_k\|_\infty \geq \epsilon \frac{1-\gamma}{\gamma}$ **do**
  $V_{k+1} = \mathcal{T}_\pi(V_k)$
  Increment k
end while
Return: V_k, an ϵ-approximation of V_π

Now, when searching for optimality, one could use value iteration as well, with $\mathcal{T}_*$ to get an estimate of V_*; then, any V_k-improving policy is an ϵ-optimal one. Value iteration is the baseline for policy evaluation, but for γ close to 1, its convergence can be slow. Several optimizations have been proposed.

In the Gauss-Seidel approach, the value of each state s_j at the $k + 1^{th}$ iteration is updated with

the contribution of a state s_i either with $V_{k+1}(s_i)$ if $i < j$, that is, if the value of s_i has already been updated, or with $V_k(s_i)$ otherwise. Another close improvement is *Prioritized Sweeping* [15]), where states to update first are selected according to a measure of urgency (usually the improvement of value during the preceding iteration).

Policy iteration [12] is usually preferred to value iteration and its variants. The approach is presented in Algorithm 2 below. The idea is to alternate between improving and evaluating a current policy, until no improvement is possible anymore. It converges in a finite number of steps to an optimal policy ([18], Thm. 6.4.2), and is usually faster than value iteration [24]; it also motivates the search for policy evaluation methods. More specifically, policy iteration with approximate policy evaluation has received a lot of attention ([6, 7, 14, 16]); we will discuss approximation as well in this article.

Algorithm 2 Policy Iteration

Input: MDP M, discount γ, initial policy $\pi_0 \in \Pi$
Initialization: $k = 0,\ \pi_1 = \emptyset$
while $\pi_{k+1} \neq \pi_k$ **do**
  [a] Policy evaluation: compute the (approximated) value V_k of π_k
  [b] Policy improvement: $\pi_{k+1} \in \arg\max_{\pi\in\Pi} \{R_\pi + \gamma P_\pi V_k\}$ with $\pi_{k+1} = \pi_k$ if possible
  Increment k
end while
Return: $\pi_{k+1} = \pi_k = \pi_*$

3 Floyd-Warshall Path Integrator

3.1 Weighted automata and path integrals

A *semi-ring* $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ is a set $\mathbb{K}$ of "values" such that $(\mathbb{K}, \oplus, \bar{0})$ is a commutative monoid with $\bar{0}$ as neutral element, $(\mathbb{K}, \otimes, \bar{1})$ is a monoid with $\bar{1}$ as neutral element, $\otimes$ distributes over $\oplus$, and $\bar{0}$ is an annihilator for $\otimes$. A common example is the *probability semiring* $(\mathbb{R}^+, +, \times, 0, 1)$.

A *weighted automaton* $\mathcal{A}$ over the semi-ring $\mathbb{K}$ is a pair[2] $\mathcal{A} = (S, w)$ where S is the state set, with two distinguished states, the initial state s^i and the final state s^f assumed to have no successor. The weight function $w : S \times S \to \mathbb{K}$ assigns to each possible transition $(s, s') \in S \times S$ a weight $w(s, s')$ in the semi-ring $\mathbb{K}$. Weighted automata can be used to define stochastic automata [3], taking for $\mathbb{K}$ the probabilistic semi-ring and with the additional requirement that $\forall s \in S \setminus \{s^f\},\ \oplus_{s'\in S} w(s, s') = \bar{1}$. The transition set $T \subseteq S \times S$ of $\mathcal{A}$ is defined as the support of w in $S \times S$, *i.e.*, $t = (s, s') \in T$ iff $w(s, s') \neq \bar{0}$. For clarity, one may use the redundant notation $\mathcal{A} = (S, T, w)$, and for $t = (s, s')$, we denote $s = t^-$, $s' = t^+$ and $w(t)$ for $w(s, s')$. A *path* in $\mathcal{A}$ is a sequence $\sigma = t_1...t_n$ of transitions, $n > 0$, such that $t_i^+ = t_{i+1}^-$, $0 < i < n$. By extension, $\sigma^- = t_1^-$ and $\sigma^+ = t_n^+$. The *weight* of path $\sigma = t_1...t_n$ is defined as $w(\sigma) = w(t_1)\otimes\cdots\otimes w(t_n)$ and its length $|\sigma|$ as n. Path σ is a *run* if $\sigma^- = s^i$, *i.e.*, if the path is rooted at the initial state s^i. We denote by $\sigma_1\sigma_2$ the concatenation of two paths, possible whenever $\sigma_1^+ = \sigma_2^-$. Observe that $|\sigma_1\sigma_2| \geq 2$ as paths of length 0 are forbidden. We denote by $\mathcal{P}(s, s')$ the set of paths σ such that $\sigma^- = s$, $\sigma^+ = s'$, and by $\mathcal{P}^{(\geq n)}(s, s')$ the set of such paths of length at least n. So $\mathcal{P}(s, s') = \mathcal{P}^{(\geq 1)}(s, s')$. Concatenation extends to sets of paths, so one has $\mathcal{P}^{(\geq 2)}(s, s'') = \biguplus_{s'\in S} \mathcal{P}(s, s')\mathcal{P}(s', s'')$, and $\mathcal{P}(s, s'') = (s, s'') \uplus \mathcal{P}^{(\geq 2)}(s, s'')$ if transition (s, s'') exists in $\mathcal{A}$, otherwise $\mathcal{P}(s, s'') = \mathcal{P}^{(\geq 2)}(s, s'')$.

We are interested in computing integrals over paths of $\mathcal{A}$ of the form

$$W(s, s') = \bigoplus_{\sigma\in\mathcal{P}(s,s')} w(\sigma) \triangleq W(\mathcal{P}(s, s')) \quad (7)$$

and in particular $W(s^i, s^f)$. With a slight abuse of notation, in the sequel we consider W as a weight operator over sets of paths, with the convention $W(\emptyset) = \bar{0}$, $W(\{\sigma\}) = w(\sigma)$ and for sets of paths $\mathcal{P}_1, \mathcal{P}_2$, $W(\mathcal{P}_1\mathcal{P}_2) = W(\mathcal{P}_1) \otimes W(\mathcal{P}_2)$ and for disjoint sets $W(\mathcal{P}_1 \uplus \mathcal{P}_2) = W(\mathcal{P}_1) \oplus W(\mathcal{P}_2)$. Such integrals have a natural interpretation for stochastic automata. If there is no circuit containing state s', then s' is only met at the end of each

[2] Weighted automata generically assume *labeled* transitions, initial and terminal weights at states. We adopt a simplified setting here.

path in $\mathcal{P}(s,s')$, and the integral $W(s,s')$ in (7) corresponds to the probability of reaching s' in $\mathcal{A}$ when starting at state s. In the general case where circuits around s' exist, $\mathcal{P}(s,s')$ contains all repeated passages through s', and (7) diverges if s' belongs to a bottom strongly connected component (BSCC), *i.e.*, if s' returns to itself with probability one. If s' has an escape towards some s'' with no possible return to s', then $W(s,s')$ is the probability to go from s to s' conditionally[3] to not returning to s'. In the sequel, we will always use (7) in settings where the integral converges. For example by assuming that all states have a path towards the terminal state s^f.

By appropriate choices of the semi-ring, such path integrals give access to various quantities of interest. For instance, in the case of stochastic automata, one may be interested in computing the expected value of a function f over paths connecting two states s and s', *i.e.*,

$$\mathbb{E}[f(\sigma)|s,s'] = \sum_{\sigma\in\mathcal{P}(s,s')} w(\sigma)f(\sigma) \qquad (8)$$

where $w(\sigma)$ is the likelihood of path σ and assuming s' is met only at the end of paths in $\mathcal{P}(s,s')$. If function f decomposes as a product over paths by $f(\sigma_1\sigma_2) = f(\sigma_1)f(\sigma_2)$, then (8) immediately derives from (7) by taking $\bar{w}(t) = w(t)f(t)$ instead of $w(t)$ in the definition of transition weights, since one then has $\bar{w}(\sigma_1\sigma_2) = \bar{w}(\sigma_1)\otimes\bar{w}(\sigma_2)$. If function f decomposes additively over paths as $f(\sigma_1\sigma_2) = f(\sigma_1) + f(\sigma_2)$, then (8) typically represents an average path length or duration between states s and s'. This can still be captured by expression (7) provided one adopts another semi-ring. Specifically, consider as new transition weights the pairs $\bar{w}(t) = (w(t), w(t)f(t))$. In this extended semi-ring, $\oplus$ is the component wise addition with $\bar{0} = (0,0)$. The product is defined as $(u_1,v_1)\otimes(u_2,v_2) = (u_1u_2, u_1v_2 + u_2v_1)$ with $\bar{1} = (0,1)$ as unit. This choice entails

$$\begin{aligned}
&\bar{w}(\sigma_1)\otimes\bar{w}(\sigma_2)\\
&= (w(\sigma_1), w(\sigma_1)f(\sigma_1))\otimes(w(\sigma_2), w(\sigma_2)f(\sigma_2))\\
&= (w(\sigma_1)w(\sigma_2), w(\sigma_1)w(\sigma_2)(f(\sigma_1)+f(\sigma_2)))\\
&= (w(\sigma_1\sigma_2), w(\sigma_1\sigma_2)f(\sigma_1\sigma_2))\\
&= \bar{w}(\sigma_1\sigma_2) \qquad (9)
\end{aligned}$$

So (7) becomes $\bar{W}(s,s') = (\sum_{\sigma\in\mathcal{P}(s,s')} w(\sigma), \sum_{\sigma\in\mathcal{P}(s,s')} w(\sigma)f(\sigma))$. This technique generalizes, and one can actually compute all moments of function f over runs of the stochastic automaton $\mathcal{A}$, for example to derive the standard deviation of f (see [5]). Sec. 3.3 shows how to express the average discounted reward in an MDP under the form of (7). Then Sec. 3.4 further exploits the semi-ring flexibility to derive related quantities as the gradient or the variance of the value function.

3.2 Floyd-Warshall recursion

Let $s^1, s^2, \cdots, s^K$ be some arbitrary ordering of states[4] of $\mathcal{A}$. Let $S_k = \{s^1, ..., s^k\} \subseteq S$ be the subset of the first k states in S for this ordering, with $0 \le k \le K$. Let $\sigma = s\cdot s_{i_1}\dots s_{i_m}.s'$ be a path from s to s', and $X \subseteq S$ be a subset of states. We will say that σ *goes through* states of X if $\{s_{i_1},\dots,s_{i_m}\} \subseteq X$. Note that this definition allows $s,s' \notin X$. For $s,s' \in S$, let $\mathcal{P}_k(s,s')$ denote the set of paths from s to s' in $\mathcal{A}$ that only go through states in S_k, with the convention that $S_0 = \emptyset$, and $\mathcal{P}_0(s,s')$ is either empty or contains the unique transition from s to s' in $\mathcal{A}$ if it exists. Let us denote

$$W_k(s,s') = W(\mathcal{P}_k(s,s')) = \bigoplus_{\sigma\in\mathcal{P}_k(s,s')} w(\sigma) \qquad (10)$$

Then, reconsidering (7), for every pair of states $s,s' \in S$, we have $W_0(s,s') = w(s,s')$ and $W(s,s') = W_K(s,s')$. Furthermore, the set of paths $\mathcal{P}_{k+1}(s,s')$ decomposes as

$$\begin{aligned}
&\mathcal{P}_{k+1}(s,s') = \mathcal{P}_k(s,s') \uplus \qquad (11)\\
&\mathcal{P}_k(s,s^{k+1})\Big[\biguplus_{n\ge 0}\mathcal{P}_k(s^{k+1},s^{k+1})^n\Big]\mathcal{P}_k(s^{k+1},s')
\end{aligned}$$

Applying the weight operator W to this decomposition entails

$$\begin{aligned}
&W_{k+1}(s,s') = W_k(s,s') \oplus \qquad (12)\\
&W_k(s,s^{k+1})\otimes W_k(s^{k+1},s^{k+1})^*\otimes W_k(s^{k+1},s')
\end{aligned}$$

[3] To see this, let $w(s,s') = p$ and assume a self-loop at state s' with $w(s',s') = q$, then $W(s,s') = \frac{p}{1-q}$ where $1-q$ is the probability of definitely leaving s'.

[4] This enumeration of all possible state values should not be confused with s_n, the state variable at time n in a run.

where the x^* operation in $\mathbb{K}$ is defined by $x^* = \bigoplus_{m\geq 0} x^m$ and the power x^m is naturally defined by $x^0 = \bar{1}$, $x^{m+1} = x \otimes x^m$. Notice that not all semi-rings admit an x^* operation for all $x \neq \bar{1}$ (*e.g.* x^* is only defined for $x < 1$ in $(\mathbb{R}^+, +, \times, 0, 1)$) but it will be the case here for the elements x of interest in our selected semi-rings.

The Floyd-Warshall (FW) algorithm consists in applying recursion (12) for all pairs $s, s' \in S$ to reach the path integral (7) by $W(s, s') = W_K(s, s')$. For completeness, we provide this well-known algorithm in Appendix A. We shall actually use it to compute only the quantity $W(s, s^f)$, for $s \in S$. Recall that s^f has no successor, therefore it is only met at the end of paths in (7). Notice that thanks to the x^* operation, (12) incorporates at once the contribution of a possibly infinite set of paths between s and s', due to the arbitrary number of loops around state s^{k+1} when this loop exists. In the specific case of a stochastic automaton, $W_k(s^{k+1}, s^{k+1})$ is the probability of returning to s^{k+1} following a path that goes through states in S_k. So this quantity is strictly smaller than 1, except if s^{k+1} belongs to a bottom strongly connected component. We reject this situation and assume all states admit at least one path to the terminal state s^f. In the probabilistic semi-ring, $W_k(s^{k+1}, s^{k+1})^*$ exists and is equal to $\frac{1}{1-W_k(s^{k+1},s^{k+1})}$ which is the inverse probability of not returning to s^{k+1} (when only paths through S_k are allowed).

Remark 1. In recursion (12) the choice of $s^{k+1} \in S \setminus S_k$ can be made on the fly at step k.

Remark 2. With the assumption that s^f has no successor, then step k for which $s^k = s^f$ does not change the current weights and $W_k = W_{k-1}$. So it is wise to take $s^K = s^f$ and stop the recursion one step earlier.

Remark 3. The standard FW procedure computes $W(s, s')$ for all pairs (s, s'). The next subsection shows that we are mostly interested in $W(s^i, s^f)$, or more generally in the $W(s, s^f)$ for $s \in S$. Therefore recursion (12) needs not be applied to all pairs (s, s'). Section 4 explains how to organize computations to minimize the number of updates. It takes the form of a two-sweep algorithm, a forward sweep leading to $W(s^i, s^f)$, and a backward sweep deriving all the $W(s, s^f)$, following the architecture of the Kalman smoother in Bayesian estimation.

3.3 Expected discounted reward as an integral over paths

Consider MDP $\mathcal{M} = (S, A, P, R)$ and the resulting Markov reward process $\mathcal{M}_\pi = (S, P_\pi, R_\pi)$ induced by some policy π. The weighted automaton $\mathcal{A} = \Phi(\mathcal{M}_\pi, \gamma)$ is built from $\mathcal{M}_\pi$ as follows: $\mathcal{A} = (\bar{S}, w)$ where $\bar{S} = S \uplus \{s^f\}$, and $\forall s, s' \in S$, $w(s, s') = \gamma P_\pi(s'|s) = \gamma P(s'|s, a = \pi(s))$ and $w(s, s^f) = R_\pi(s) = R(s, a = \pi(s))$. The extra state s^f is an artificial sink, representing the reward collection phase, not to be confused with a target state of the MDP. Consider the example $\mathcal{M}_\pi$ in Figure 1-top, with respective rewards $R(s_0, \pi(s_0)) = 8$, $R(s_1, \pi(s_1)) = 9$, $R(s_2, \pi(s_2)) = 5$, and $R(s_3, \pi(s_3)) = 7$, and let $\gamma = 0.9$. The translation into a weighted automaton as described above gives the automaton $\mathcal{A}$ of figure 1-bottom.

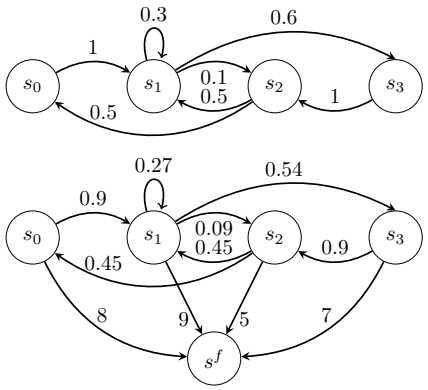

Fig. 1 Translation from a Markov reward process $\mathcal{M}_\pi$ (top) to a weighted automaton $\mathcal{A}$ (bottom).

Proposition 1. *The expected discounted reward of $\mathcal{M}_\pi$ from the initial state $s_0 = s \in S$, $V_\pi(s)$ given in (1), coincides with the path integral $W(s, s^f)$ defined in (7) for the associated weighted automaton $\mathcal{A} = \Phi(\mathcal{M}_\pi, \gamma)$.*

Proof. Let $\mathcal{P}^{(n)}(s, s')$ denote paths of length n from s to s' in $\mathcal{A}$, hence $W(s, s') = W(\mathcal{P}(s, s')) = \bigoplus_{n\geq 1} W(\mathcal{P}^{(n)}(s, s'))$. The value function (1) at initial state $s_0 = s$ writes

$$V_\pi(s) = \mathbb{E}_\pi\Big[\sum_{n\geq 0} \gamma^n r_n \,|\, s_0 = s\Big] = \sum_{n\geq 0} \gamma^n \,\mathbb{E}_\pi\left[r_n \,|\, s_0 = s\right] \tag{13}$$

$$
\begin{aligned}
&= \sum_{n\geq 0} \gamma^n \sum_{s'\in S} R_\pi(s') P_\pi(s_n = s' | s_0 = s) \\
&= \sum_{s'\in S} R_\pi(s') \sum_{n\geq 0} \gamma^n P_\pi(s_n = s' | s_0 = s) \quad (14) \\
&= \sum_{s'\in S} R_\pi(s') \Big(\mathbb{1}_{s'=s} + \sum_{n\geq 1} W(\mathcal{P}^{(n)}(s, s')) \Big) \\
&= R_\pi(s) + \sum_{s'\in S} R_\pi(s')\, W(\mathcal{P}(s, s')) \quad (15) \\
&= w(s, s^f) + \sum_{s'\in S} W(\mathcal{P}(s, s')) \cdot w(s', s^f) \\
&= W\Big((s, s^f) \uplus \biguplus_{s'\in S} \mathcal{P}(s, s') \cdot (s', s^f) \Big) \quad (16) \\
&= W(\mathcal{P}(s, s^f)) \\
&= W(s, s^f) \quad (17)
\end{aligned}
$$

These transforms rely on the fact that the initial converging series only contains positive terms, which can therefore be freely shuffled and reassembled without changing the limit. For clarity, as all computations are performed in $\mathbb{R}^+$, we use the classical sum and product notation instead of semi-ring operations[5]. Relation (15) expresses that the reward of each state s' (under policy π) contributes to the total performance of policy π by a factor which is the (discounted) probability (under π) of visiting s' from the initial state $s_0 = s$. This coefficient is a sum over a possibly infinite set of runs, which contrasts with iterative methods to estimate $V_\pi(s)$ in value iteration or policy iteration, that typically explore runs of bounded depth after s and accumulate $R_\pi(s')$ each time s' is met.

As an alternative proof of Prop. 1, notice that the set of paths from $s \in S$ to the sink state s^f also decomposes as

$$
\mathcal{P}(s, s^f) = (s, s^f) \uplus \biguplus_{s'\in S} (s, s') \cdot \mathcal{P}(s', s^f) \quad (18)
$$

which differs from the decomposition in (16) as the elementary transitions are connected as a prefix instead of a suffix of path sets. Applying the weight operator to this relation yields

$$
\begin{aligned}
W(s, s^f) &= W(\mathcal{P}(s, s^f)) \\
&= w(s, s^f) + \sum_{s'\in S} w(s, s') \cdot W(\mathcal{P}(s', s^f)) \\
&= R_\pi(s) + \sum_{s'\in S} \gamma P_\pi(s'|s) W(s', s^f) \quad (19)
\end{aligned}
$$

So the values $W(s, s^f)$ attached to each state $s \in S$ by the Floyd-Warshall procedure form a fix point of the Bellman operator. But the latter has a unique fix point, which is $V_\pi(s)$. ◇

The interest of this reformulation is manifold. First, by definition the value function at state s (see Eq. (13)) involves an *infinite* sum, or an expectation over *infinite* paths of the underlying Markov process rooted at s. The refactorization trick appearing at Eq. (14) reformulates $V_\pi(s)$ as a *finite* combination of state rewards $R_\pi(s')$ with coefficients that result of an integration over possibly many *finite* paths relating s' to the terminal state s^f. They can therefore be obtained by a Floyd-Warshall procedure. Deriving V_π in this way still has a cubic complexity (in number of states), so there is apparently no gain compared to solving the fix point equation $V_\pi = R_\pi + \gamma P_\pi V_\pi$. We show below that in practice some gains can be achieved.

A second interest relates to possible approximations of the value function. Standard value iteration (Alg. 1) relies on a recursive evaluation of a fix point: at step n, the value for each state s is estimated by collecting (discounted) rewards on paths rooted at s and of length n. The recursion augments this exploration depth to $n + 1$ by considering the estimated value of successors s' of s, and transporting it back to s by a (contracting) transition operator. This is thus a breadth first exploration of the future of each state. By contrast, the Floyd-Warshall structure of computations takes the shape of a depth first exploration of paths, as the rewards along unbounded path lengths are collected at once, as illustrated in Fig. 2. This suggests approximation schemes that would first explore the most rewarding paths and possibly neglect those leading to rewards that are too far to bring significant extra contribution (see Sec. 4.3).

Finally, interpreting the value function V_π as a path integral in a weighted automaton gives access

[5] The above derivations, from the 3rd line and on, remain valid in any semi-ring as operations on formal series. In this paper they will also make sense numerically as all the semi-rings we consider are powers of $\mathbb{R}^+$, so reshuffling terms doesn't change the limits.

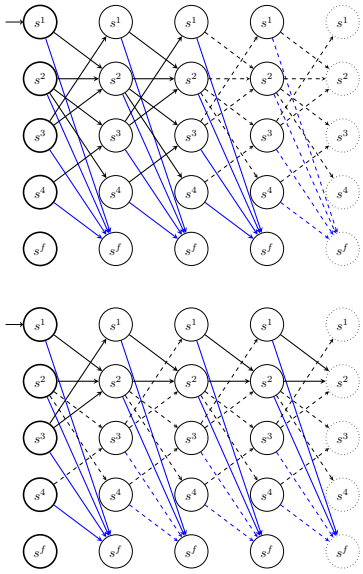

Fig. 2 Depiction of all trajectories in the automaton Fig. 1, by unfolding along time. Top: every trajectory of length 2 or less, in bold, has been integrated in Value Iteration. Bottom: every trajectory going through states $\{s^1, s^2\}$ and ending in s^f, in bold, has been integrated in Floyd-Warshall. Blue edges represent the reward collection step at the end of each path.

to other quantities of interest by simply changing the semi-ring. This is illustrated below.

3.4 Flexibility of semi-rings

3.4.1 Policy difference

As a first example, let us show that we can use this formalization to compare two policies π and π', *i.e.*, compute the difference in value of π and π'. Rather than computing the value function for π then for π' before taking the difference, one can get it in a single procedure by adopting a different semi-ring for computations. Let us assign to each transition (s, s') the weight $\bar{w}(s, s') \in \mathbb{R} \times \mathbb{R}^+$

$$\bar{w}(s, s') = \big(w_\pi(s, s') - w_{\pi'}(s, s') \,,\, w_\pi(s, s') + w_{\pi'}(s, s') \big) \tag{20}$$

where $\forall s, s' \in S$, $w_\pi(s, s') = \gamma P_\pi(s'|s)$ and $w_\pi(s, s^f) = R_\pi(s)$, and similarly for $w_{\pi'}$. As semi-ring operations over pairs in $\mathbb{R} \times \mathbb{R}^+$, consider

$$(a, b) \oplus (c, d) = (a + c, b + d) \tag{21}$$

$$(a, b) \otimes (c, d) = \frac{1}{2}(ad + bc, ac + bd) \tag{22}$$

with $\bar{0} = (0, 0)$ and $\bar{1} = (0, 2)$. Then for any path σ one has $\bar{W}(\sigma) = (W_\pi(\sigma) - W_{\pi'}(\sigma), W_\pi(\sigma) + W_{\pi'}(\sigma))$. So an integral like (7) would yield the desired value difference as the first component of the weight $\bar{W}(s^i, s^f)$. In the sequel, we examine approximate integrals that only visit paths/states that contribute the most to the final result. The interest of the above reformulation is that it enables exploring only runs that contribute the most to the difference in average reward of the two policies, while ignoring runs with similar rewards. Notice that the star operation is well defined in the above semi-ring, which enables the use of the FW algorithm. One has $(a, b)^n = \frac{1}{2^n}(\alpha^n - \beta^n, \alpha^n + \beta^n)$ with $\alpha = a + b$ and $\beta = b - a$. So $(a, b)^* = (\frac{1}{1-\alpha/2} - \frac{1}{1-\beta/2}, \frac{1}{1-\alpha/2} + \frac{1}{1-\beta/2})$ as soon as $\alpha \neq 2 \neq \beta$, which holds for $\gamma < 1$.

3.4.2 Policy gradient

This subsection assumes randomized policies $\pi : S \to \Delta(A)$ instead of deterministic policies $\pi : S \to A$, where $\Delta(A)$ represents the set of probability distributions over actions in A. Therefore the Markov Reward Process $\mathcal{M}_\pi$ associated to MDP $\mathcal{M}$ is defined by edge weights $w(s, s') = \gamma P_\pi(s'|s) = \gamma \sum_a P(s'|s, a)\pi(a|s)$ and $w(s, s^f) = R_\pi(s) = \sum_a R(s, a)\pi(a|s)$.

Consider a parametric space of policies $\{\pi_\theta\}_{\theta \in \Theta}$ where Θ is a metric space of parameters, typically $\Theta \subseteq \mathbb{R}^d$. This situation is motivated by the need to look for policies that exhibit some form of "regularity," or smooth variations from one state s to a neighbor state s'. So instead of improving policies on a state by state basis, one is more interested in gradient ascent approaches on policies. In reinforcement learning, the most efficient approaches to control synthesis for cyber-physical systems are actually based on policy gradient estimation [25], [20], [21].

For simplicity, we use subscript θ instead of π_θ for objects that depend on policy π_θ. Along with the value function, one is also interested in computing its gradient $\nabla_\theta V_\theta(s)$. Let us assume this gradient is accessible at transition scale, *i.e.*, $\nabla_\theta \, w_\theta(s, s')$ is well defined, and let us attach to each transition the composite weight $\bar{w}(s, s') \in \mathbb{R} \times \mathbb{R}^d$

$$\bar{w}(s, s') = \big(w_\theta(s, s') \,,\, \nabla_\theta \, w_\theta(s, s') \big) \tag{23}$$

As semi-ring operations over $\mathbb{R} \times \mathbb{R}^d$, consider component-wise addition for $\oplus$ as in (21), and

$$(a, b) \otimes (c, d) = (ac, ad + cb) \tag{24}$$

with $\bar{1} = (1, 0)$. Then for any path σ this choice entails $\bar{W}(\sigma) = (W_\theta(\sigma), \nabla_\theta W_\theta(\sigma))$, because the property holds for elementary transitions and extends by concatenation:

$$\begin{aligned}
&\bar{W}(\sigma_1) \otimes \bar{W}(\sigma_2) \\
&= \big(W_\theta(\sigma_1), \nabla_\theta W_\theta(\sigma_1)\big) \otimes \big(W_\theta(\sigma_2), \nabla_\theta W_\theta(\sigma_2)\big) \\
&= \big(W_\theta(\sigma_1) W_\theta(\sigma_2), \\
&\qquad W_\theta(\sigma_1) \nabla_\theta W_\theta(\sigma_2) + W_\theta(\sigma_2) \nabla_\theta W_\theta(\sigma_1)\big) \\
&= \big(W_\theta(\sigma_1) W_\theta(\sigma_2), \nabla_\theta [W_\theta(\sigma_1) W_\theta(\sigma_2)]\big) \\
&= \big(W_\theta(\sigma_1 \sigma_2), \nabla_\theta W_\theta(\sigma_1 \sigma_2)\big) \\
&= \bar{W}(\sigma_1 \sigma_2)
\end{aligned} \tag{25}$$

So again an integral like (7) yields the desired gradient of the value function as the second component of the weight $\bar{W}(s^i, s^f)$. The star operation is well defined in this new semi-ring: one has $(a, b)^n = (a^n, n a^{n-1} b)$ and so $(a, b)^* = (\frac{1}{1-a}, \frac{a}{(1-a)^2} b)$, relying on the fact that $0 \leq a < 1$.

The above formalism applies to another interesting situation. Assume policy π is fixed, but the reward function $R_{\pi,\mu}$ and the transition probability $P_{\pi,\mu}$ both depend on some parameter μ. One may be interested in the gradient $\nabla_\mu W_{\pi,\mu}(s, s^f)$ of the value function as a measure of the policy sensitivity to parameter changes in the model. Clearly this can be computed exactly as above. Further, one can combine the two situations with parameters on both the policy and the model. It then becomes possible to perform gradient ascent on a policy for a criterion that would combine performance (the value function) and sensitivity to model parameters.

As a final remark, consider again property (19) on the value function. It applies here and reveals that the gradient also satisfies a Bellman equation. Specifically,

$$\bar{W}(s, s^f) = \bar{w}(s, s^f) \oplus \bigoplus_{s' \in S} \bar{w}(s, s') \otimes \bar{W}(s', s^f)$$

yields

$$\begin{aligned}
(V_\theta(s), \nabla_\theta V_\theta(s)) = (R_\theta(s), \nabla_\theta R_\theta(s)) \oplus \\
\sum_{s' \in S} \gamma \left(P_\theta(s'|s), \nabla_\theta P_\theta(s'|s)\right) \otimes \left(V_\theta(s'), \nabla_\theta V_\theta(s')\right)
\end{aligned} \tag{26}$$

This shows that pairs $(value, gradient)$ form a fix point of an affine operator in $\mathbb{R}^{K \times (d+1)}$. On the first component, one recovers the usual fix point equation on the value function V_θ. On the second component, one gets

$$\begin{aligned}
&\nabla_\theta V_\theta(s) \\
&= \nabla_\theta R_\theta(s) + \sum_{s' \in S} \gamma \nabla_\theta P_\theta(s'|s) V_\theta(s') \\
&\quad + \sum_{s' \in S} \gamma P_\theta(s'|s) \nabla_\theta V_\theta(s') \\
&= \sum_{a \in A} \nabla_\theta \pi_\theta(a|s) Q_\theta(s, a) + \sum_{s' \in S} \gamma P_\theta(s'|s) \nabla_\theta V_\theta(s')
\end{aligned} \tag{27}$$

where $Q_\theta(s, a) = R(s, a) + \sum_{s' \in S} \gamma P(s'|a, s) V_\theta(s')$ is the state-action value function (or the quality function), and satisfies $V_\theta(s) = \sum_{a \in A} \pi_\theta(a|s) Q_\theta(s, a)$. Once V_θ has converged, the first term in the last line of (27) stabilizes and $\nabla_\theta V_\theta$ appears to satisfy a standard Bellman type equation. In other words, the affine operator in (26) is contracting, so the pair $(value, gradient)$ could also be estimated by standard value iteration. Appendix B further exploits properties of this semi-ring to recover a closed-form expression of the value gradient.

To illustrate the use of policy gradient, we consider the River Swim exemple (see Fig. 8), a chain of states with rewards located at the extremities. The rightmost state yields the biggest reward but is hard to reach, so the optimal action might depend on the distance to this rewarding state. We search an optimal policy in the space of logistic policies, defined using the logistic function as follows:

$$p_{(k,x_0)}(\mathsf{go\ right} \,|\, s_i) = \frac{1}{1 + \exp(-k(i - x_0))}$$

where $k \in \mathbb{R}$ is the curve steepness and $x_0 \in \mathbb{R}$ is the cutoff point. These parameters define a family of parametric policies, meaning one can use the Floyd-Warshall approach to jointly compute the policy values and gradients for any pair $\theta = (k, x_0)$. Using the gradient to update the

parameters in a gradient ascent fashion [2], we can iteratively improve the policy:

$$\theta_{new} \leftarrow \theta_{old} + \alpha \cdot \nabla_\theta V_\theta(s_0)$$

for some learning coefficient $\alpha > 0$. We optimize on the starting state s_0. Running the experiment for a 50-states long River Swim, with $\gamma = 0.95$, and starting parameters $\theta_0 = (0.05, 20)$, we obtain the successive policies shown in Fig. 3.

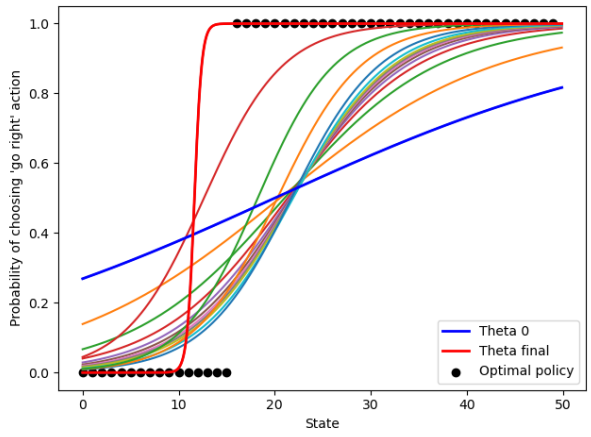

Fig. 3 Policy gradient ascent on logistic policies in River Swim. The state-dependent probability of choosing action 'go right' is plotted for different policies. The bold blue and red lines correspond to the initial and last iterations respectively, while the black dots are for the optimal deterministic policy; the remaining curves represent the intermediate steps to converge to the final result.

In this figure, the optimal policy shows that there is a critical state (s_{15} here) before which it is more beneficial to always go left, and for states closer to the big reward, going right is the optimal action. The gradient ascent successively pushed θ towards a bigger steepness, getting closer to the shape of the optimal policy. At the same time, the cutoff point has been pushed back from 20 to roughly 11, so a bit further than the optimal one: for those states around the critical one, the difference in value between going left or right is small, and so is the gradient. Overall, we have been able to converge to a locally optimal policy inside the family of logistic policies thanks to a Floyd-Warshall computation of the policy gradient.

3.4.3 Variance of the discounted reward

Under some fixed policy π, the total discounted reward of an infinite run $\sum_{n\geq 0}\gamma^n r_n$ is a random variable which (conditional) expectation defines the value function V_π. One may be interested in the variance of this random variable [23], [26], for example to select among two policies with equal value the one with least variance from the intial state. Variance computation actually amounts to computing the second moment of the discounted reward, for all possible starting states $s \in S$:

$$V_\pi^{(2)}(s) = \mathbb{E}[\,(\sum_{n\geq 0}\gamma^n r_n)^2 \,|\, s_0 = s\,] \qquad (28)$$

Similar to the recursive form of the value function

$$V_\pi(s) = R_\pi(s) + \gamma \sum_{s'\in S} V_\pi(s')P_\pi(s'|s) \qquad (29)$$

straightforward calculations yield

$$V_\pi^{(2)}(s) = R_\pi(s)^2 + 2\gamma R_\pi(s)\sum_{s'\in S} V_\pi(s')P_\pi(s'|s) + \gamma^2 \sum_{s'\in S} V_\pi^{(2)}(s')P_\pi(s'|s) \qquad (30)$$

which readily shows that first and second moments of the discounted reward must be computed jointly. (29)-(30) can actually be assembled into a coupled fix point equation (as already evidenced by Sobel, see Eq. (2,3,4) in [23]). The necessity of this joint computation suggests that if a semi-ring formulation exists it should handle tuples of values. Several difficulties arise however in the construction of such a semi-ring.

First of all, the refactorization technique used in Prop. 1 to transform an infinite discounted sum into an integral over finite runs does not work as easily on (28). This is due to the expansion of the square, as better seen in (30) by further expanding the term $V_\pi(s')$ as in (29): this produces cross-product terms $R_\pi(s)R_\pi(s')$ which coefficients do not appear as straightforward path integrals. To go around this difficulty, let us assume that runs of M_π are finite with probability one, such as in Fig. 4, which avoids using Proposition 1.

A second difficulty appears with the discount factor γ. As all runs are now finite, let us first

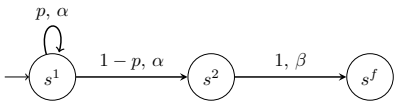

Fig. 4 A simple Markov process $\mathcal{M}_\pi$ with finite trajectories. The first label on each edge represents its probability. State rewards α at s^1 and β at s^2 have been "pushed" to outgoing transitions from these states, and appear as a second edge label.

assume $\gamma = 1$ for simplicity, and let us attach the reward of each state s to all transitions leaving s (see Fig. 4). This yields a stochastic automaton with edge rewards, for which one wants to compute the first and second moments of rewards after each state, where obviously the reward of a run is the sum of all transition rewards crossed along that run. The problem has been considered exactly under this form in [5] (Sec. IV), and it can be recast in the semi-ring formalism by assigning to each transition (s, s') a weight in $(\mathbb{R}^+)^3$

$$\bar{w}(s, s') \;=\; P_\pi(s'|s) \cdot \big(1,\, R_\pi(s),\, R_\pi(s)^2\,\big) \qquad (31)$$

Consider then as semi-ring operations on these triples component-wise addition for $\oplus$ with $\bar{0} = (0, 0, 0)$, and

$$\begin{aligned} &(a, b, c) \otimes (\tilde{a}, \tilde{b}, \tilde{c}) \\ &= (a\tilde{a},\; a\tilde{b} + b\tilde{a},\; a\tilde{c} + 2b\tilde{b} + c\tilde{a}) \end{aligned} \qquad (32)$$

with unit $\bar{1} = (1, 0, 0)$. These operations make $\mathbb{K} = \big((\mathbb{R}^+)^3, \oplus, \otimes, \bar{0}, \bar{1}\big)$ a commutative semi-ring. For some path σ in $\mathcal{M}_\pi$, let p be its probability, *i.e.*, the product of all its transition probabilities, and let r be its total reward, *i.e.*, the sum of all its transition rewards. With the above semi-ring operations, one has

$$\bar{W}(\sigma) \;=\; p \cdot (1, r, r^2) \qquad (33)$$

The proof is by recursion. (33) readily holds for transitions by definition (31), and it extends by concatenation: for path $\sigma\tilde{\sigma}$ one has

$$\begin{aligned} &\bar{W}(\sigma) \otimes \bar{W}(\tilde{\sigma}) \\ &= (p, pr, pr^2) \otimes (\tilde{p}, \tilde{p}\tilde{r}, \tilde{p}\tilde{r}^2) \\ &= (p\tilde{p},\; p \cdot \tilde{p}\tilde{r} + \tilde{p} \cdot pr,\; p \cdot \tilde{p}\tilde{r}^2 + 2pr \cdot \tilde{p}\tilde{r} + \tilde{p} \cdot pr^2) \\ &= (p\tilde{p},\; p\tilde{p}(r + \tilde{r}),\; p\tilde{p}(r + \tilde{r})^2) \\ &= \bar{W}(\sigma\tilde{\sigma}) \end{aligned} \qquad (34)$$

The star operation is well defined in this semi-ring: for $0 \le a < 1$, one has $(a, b, c)^n = (a^n,\; na^{n-1}b,\; na^{n-1}c + n(n-1)a^{n-2}b^2)$ whence

$$(a, b, c)^* \;=\; \big(A,\; bA^2,\; cA^2 + 2b^2A^3\big) \qquad (35)$$

with $A = \frac{1}{1-a}$. The Floyd-Warshall approach thus applies to sum the path weights (33) over all finite runs. For the example in Fig. 4, this yields

$$\begin{aligned} &\bar{W}(s^1, s^f) \\ &= \Big[p \cdot (1, \alpha, \alpha^2)\Big]^* \otimes (1 - p) \cdot (1, \alpha, \alpha^2) \otimes (1, \beta, \beta^2) \\ &= \left(1,\; \frac{\alpha p}{1-p},\; \frac{\alpha^2 p(1+p)}{(1-p)^2}\right) \otimes (1,\; \alpha + \beta,\; (\alpha + \beta)^2) \\ &= \left(1,\; \frac{\alpha}{1-p} + \beta,\; \frac{\alpha^2(1+p)}{(1-p)^2} + \frac{2\alpha\beta}{1-p} + \beta^2\right) \\ &= (1,\; V(s^1),\; V^{(2)}(s^1)) \end{aligned} \qquad (36)$$

from which one derives the variance of run rewards as $V^{(2)}(s^1) - V(s^1)^2 = p\alpha^2/(1-p)^2$.

Appendix C completes the variance computation in the general case of discounted rewards, showing that the semi-ring approach remains valid, although the appropriate semi-ring is of dimension at least 6.

4 Progressive Floyd-Warshall integration

Relying on Prop. 1, the expected discounted reward of policy π for MDP $\mathcal{M} = (S, A, P, R)$ with initial state s^i can be computed as $W(s^i, s^f)$ in the associated weighted automaton $\mathcal{A} = (\bar{S}, w) = \Phi(\mathcal{M}_\pi, \gamma)$. If one further wishes to improve this policy, then knowing $W(s, s^f)$ for every $s \in S$ is also necessary in principle, as the optimal policy necessarily opts for the most rewarding action at every state. But of course not all states s are likely to be visited from s^i when applying the optimal policy, so $W(s, s^f)$ needs not be computed for all states, and some computational effort can be saved. While the standard Floyd-Warshall procedure computes $W(s, s')$ for every pair of states $s, s' \in S$, this section examines more efficient strategies to derive $W(s^i, s^f)$ and compute the value $W(s, s^f)$ only if s is a relevant state. A connection is made with the A* algorithm, and further with forward-backward algorithms that appear in Bayesian estimation for Markov chains.

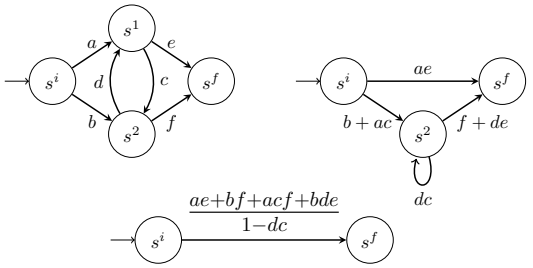

Fig. 5 Floyd Warshall path integration between s^i and s^f in the probabilistic semi-ring, by first eliminating s^1, then s^2. Final result is independent of the elimination order.

4.1 Forward sweep

This section focuses on the derivation of $W(s^i, s^f)$. For simplicity, it is assumed that each state $s \in S$ is reachable from s^i and co-reachable from s^f. All other states have no contribution to $W(s^i, s^f)$ as they participate to no path in $\mathcal{P}_{\mathcal{A}}(s^i, s^f)$.

If one is only interested in the specific value $W(s^i, s^f)$, the Floyd-Warshall recursion (12) needs not be performed for all pairs (s, s') but only for all $s \in S \setminus S_k \cup \{s^i\}$ and all $s' \in S \setminus S_k \cup \{s^f\}$ at each step k. To see this, let us examine the very first step of the FW algorithm. $W_1(s, s')$ corresponds to the weight of a new direct transition from s to s', and it is non-zero as soon as (i) (s, s') is already a transition in $\mathcal{A}$ or (ii) s and s' are connected through s^1, *i.e.*, $(s, s^1) \in T$ and $(s^1, s') \in T$. For pairs (s, s') not satisfying condition (ii), the weight remains unchanged: one has $W_1(s, s') = W_0(s, s') = w(s, s')$. Once W_1 is computed, state s^1 does not appear in subsequent computations (provided $s^1 \notin \{s^i, s^f\}$). This can be expressed differently: W_1 corresponds to the weight function of a new automaton $\mathcal{A}_1$ with $S \setminus \{s^1\}$ as state set (still for $s^1 \notin \{s^i, s^f\}$). Starting the FW algorithm from $\mathcal{A}_1$ instead of $\mathcal{A}$ yields the same final integral $W(s^i, s^f)$. So the full FW procedure can be interpreted as a recursive state elimination in $\mathcal{A}$, as illustrated in Fig. 5.

Algorithm 3 Computation of $V_\pi(s^i) = W(s^i, s^f)$

Input: weighted automaton $\mathcal{A} = (S, w)$, with $|S| = K$ and distinguished states s^i, s^f
Output: $W(s^i, s^f)$
Initialization: $W_0 = w$, $S_0 = \emptyset$
for k=0..K-1 **do**
  select $s^{k+1} \in B_k^+(s^i)$
  set $S_{k+1} = S_k \cup \{s^{k+1}\}$
  set $W_{k+1} = W_k$ restricted to $\big(\{s^i\} \cup (S \setminus S_{k+1})\big) \times \big((S \setminus S_{k+1}) \cup \{s^f\}\big)$
  for all $s \in \partial_k^-(s^{k+1})$ **do**
    for all $s' \in \partial_k^+(s^{k+1})$ **do**
      update $W_{k+1}(s, s')$ by Eq. (12)
    end for
  end for
end for
Return: $W_K(s^i, s^f)$

The above observation forms the basis of a more efficient implementation of the FW procedure aiming at $W(s^i, s^f)$ only. Let S_k be the set of vertices taken into account at the k^{th} step, and let $\mathcal{A}_k = (\{s^i, s^f\} \cup S \setminus S_k, T_k, W_k)$ be the weighted automaton at step k defined by weight function W_k. We denote by $\partial_k^+(s)$ the set of successors of s in $\mathcal{A}_k$, and by $\partial_k^-(s)$ its set of predecessors:

$$\partial_k^+(s) = \{s' \in \{s^i, s^f\} \cup S \setminus S_k : W_k(s, s') \neq \bar{0}\} \quad (37)$$
$$\partial_k^-(s) = \{s' \in \{s^i, s^f\} \cup S \setminus S_k : W_k(s', s) \neq \bar{0}\} \quad (38)$$

We denote as $B_k^+ = \partial_k^+(s^i)$ and $B_k^- = \partial_k^-(s^i)$ the successors and predecessors of s^i, respectively, that together form the border $B_k \triangleq B_k^- \cup B_k^+$ in our computation procedure. The idea of the forward sweep is to select s^{k+1} in B_k^+ in order to quickly reach s^f or create new paths to it, and thus increase the current estimate $W_k(s^i, s^f)$ of $W(s^i, s^f)$.

Remark 4. Few weights change at step k. And less and less vertices are covered by the weight function W_k, until only s^i and s^f remain.

Remark 5. The neighboring relations δ_k^+ and δ_k^- (and thus the border sets B_k^+, B_k^-) evolve in accordance with W_k. For example $\partial_{k+1}^+(s) = \big(\partial_k^+(s) \cup \partial_k^+(s^{k+1})\big) \setminus \{s^{k+1}\}$ for $s \in \partial_k^-(s^{k+1})$, and $\partial_{k+1} = \partial_k$ elsewhere in $(S \setminus S_{k+1}) \cup \{s^i, s^f\}$. This is not detailed in Alg. 3 for simplicity. Consequently, the border B_k progresses from s^i towards s^f, as illustrated in Fig. 6.

Remark 6. $W_k(s^i, s^f)$ remains at its initial value $w(s^i, s^f)$, possibly $\bar{0}$, until the border subset B_k^+ absorbs s^f, *i.e.*, until a first path is found from s^i to s^f through vertices in S_k.

Remark 7. As soon as node s^i has been added to S_k by $s^k = s^i$, subset B_k^- of the border becomes empty and s^i only has successors (no way back to s^i) in the next reduced automata $\mathcal{A}_l$, for $l \geq k$.

Remark 8. Alg. 3 remains valid if the next node s^{k+1} is simply taken in $S \setminus S_k$ instead of B_k^+. The interest of selecting it in the border is to shape the derivation of $W(s^i, s^f)$ into a path finding strategy, aiming at exploring first the most rewarding paths. Section 4.3 elaborates further on this idea through an analogy with the A* algorithm, where computations are stopped as soon as a good approximation of $W(s^i, s^f)$ is obtained.

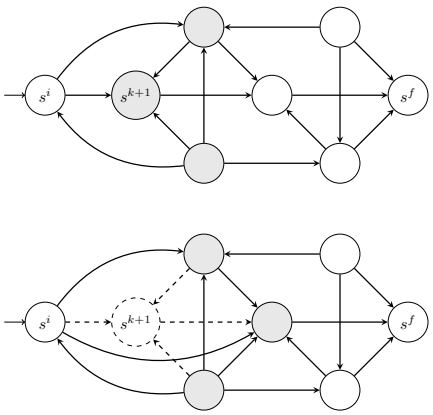

Fig. 6 Top: border B_k (in grey) at step k. Bottom: next border B_{k+1} after removal of state/node $s^{k+1} \in B_k^+$. Weights of edges to the right of B_{k+1} do not change: they remain equal to their initial value.

4.2 Backward sweep

At step k, Alg. 3 knows $W_k(s, s')$ for all $s \in \big(S \setminus S_k\big) \cup \{s^i\}$ and all $s' \in \big(S \setminus S_k\big) \cup \{s^f\}$. So $W_k(s^{k+1}, s^f)$ is known, but at subsequent steps $l > k$, $W_l(s^{k+1}, s^f)$ will not be updated anymore and the value function at s^{k+1}, $V_\pi(s^{k+1}) = W_K(s^{k+1}, s^f)$, will be missing. In a value iteration scheme, it is necessary to know the value function at *all* states in order to improve the current policy π, so one needs a way to update each intermediate estimate $W_k(s^{k+1}, s^f)$ to obtain $W_K(s^{k+1}, s^f)$.

$W(s, s^f)$ can be computed from any intermediate automaton $\mathcal{A}_k$ for $s \in (S \setminus S_k) \cup \{s^i\}$, so specifically for $s = s^{k+1}$. Observe that in $\mathcal{A}_k$, all paths of length at least 2 relating s to s^f must go through its immediate successors $\partial_k^+(s)$. Denoting $\mathcal{P}_{\mathcal{A}_k}(s, s')$ the set of paths relating s to s' in $\mathcal{A}_k$, and assuming an edge already exists from s to s^f, one has:

$$\mathcal{P}_{\mathcal{A}_k}(s^{k+1}, s^f) = (s^{k+1}, s^f) \uplus \biguplus_{s \in \partial_k^+(s^{k+1})} (s^{k+1}, s)\, \mathcal{P}_{\mathcal{A}_k}(s, s^f) \tag{39}$$

Applying the weight operator W in $\mathcal{A}_k$ yields

$$W(s^{k+1}, s^f) = W_k(s^{k+1}, s^f) \oplus \bigoplus_{s \in \partial_k^+(s^{k+1})} W_k(s^{k+1}, s) \otimes W(s, s^f) \tag{40}$$

which mimics the fix point equation (19) already stated in $\mathcal{A}$.

(40) forms the basis of a backward procedure to update $W_k(s^{k+1}, s^f)$ into its final value $W(s^{k+1}, s^f)$ as soon as the final values $W(s, s^f)$ are available at all its immediate successors $s \in \partial_k^+(s^{k+1})$ in $\mathcal{A}_k$. Two cases appear to derive $W(s^{k+1}, s^f)$

1. either $s^{k+1} \notin \partial_k^+(s^{k+1})$, *i.e.*, there is no self-loop at s^{k+1} in $\mathcal{A}_k$, and (40) applies directly,
2. or (40) is indeed a fix point equation as $W(s^{k+1}, s^f)$ appears on both sides, which resolves as

$$W(s^{k+1}, s^f) = W_k(s^{k+1}, s^{k+1})^* \otimes \Big(W_k(s^{k+1}, s^f) \oplus \bigoplus_{s \in \partial_k^+(s^{k+1}), s \neq s^{k+1}} W_k(s^{k+1}, s) \otimes W(s, s^f)\Big) \tag{41}$$

Notice that (41) reduces to (40) when $W_k(s^{k+1}, s^{k+1}) = \bar{0}$.

To organize the backward sweep, let $G = (S, E)$ be the directed graph over vertices S resulting from a run of Alg. 3, defined by $(s^{k+1}, s) \in E$ iff $s \in \partial_k^+(s^{k+1}) \setminus \{s^{k+1}\}$. By construction, G is a DAG and thus defines a partial ordering over S. Any linear extension of G, taken backwards, forms a valid order to apply the retrograde equation (40). In particular, $s^K, s^{K-1}, ..., s^1$ is a possible ordering for this backward sweep. As a result, while Alg. 3 focuses on the derivation of

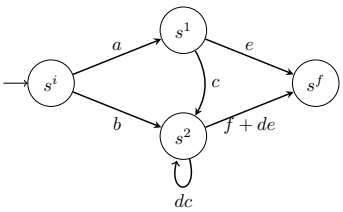

Fig. 7 Directed graph and edge values that forms the support of the backward sweep, after Alg. 3 is run on the example in Fig. 5.

$W(s^i, s^f)$, by doubling its complexity one gets $W(s, s^f) = V\pi(s)$ for every $s \in S$.

Fig. 7 illustrates this backward sweep for the example in Fig. 5, where Alg. 3 is run with the ordering $s^0 = s^i, s^1, s^2, s^3 = s^f$, and where $W_k(s^i, s^f)$ is not recursively updated but delayed to the backward sweep. The starting point is a directed weighted graph defined by choosing edges of the form (s^{k+1}, s) such that $s \in \delta_k^+(s^{k+1})$ with $w((s^{k+1}, s)) = W_k(s^{k+1}, s)$. Removing self-loops in this graph yields the DAG G above. Starting from $W(s^f, s^f) = 1$, one gets $W(s^2, s^f) = \frac{f+de}{1-cd}$ by (41) at s^2. Then (40) applies at s^1 and yields $W(s^1, s^f) = e + cW(s^2, s^f) = \frac{e+cf}{1-cd}$. Finally, (40) applies again at $s^0 = s^i$ and yields $W(s^i, s^f) = aW(s^1, s^f) + bW(s^2, s^f) = \frac{ae+acf+bf+bde}{1-cd}$, which is the value computed by Alg. 3 for the exmaple in Fig. 5.

4.3 Approximation by early stopping

This subsection shows that the Floyd-Warshall approach in Alg. 3 gives criteria for an early stopping, resulting in a bounded error approximation of the desired $W(s^i, s^f)$.

The fix point relation (40) derived in Sec. 4.2 actually holds for all states s in $\mathcal{A}_k$, not just s^{k+1}. Applying it to $s = s^i$ yields

$$W(s^i, s^f) = W_k(s^i, s^f) \oplus \bigoplus_{s \in B_k^+} W_k(s^i, s) \otimes W(s, s^f) \quad (42)$$

where for simplicity we assume $s^i \notin B_k^+$, *i.e.*, s^i has already been incorporated into S_k.

In the RHS of (42), the first term $W_k(s^i, s^f)$ represents the current estimate of the value function at state s^i, *i.e.*, the expected reward of the MDP over runs starting at s^i and going only through states in S_k. This term is still at $\bar{0}$ if there is no reward at state s^i (*i.e.*, no direct link to s^f), and states in S_k are not sufficient to establish a path from s^i to s^f. $W_k(s^i, s^f)$ can only grow with k as more and more paths to s^f are considered, until the desired quantity $W(s^i, s^f)$ is reached.

The second term $\bigoplus_{s \in B_k^+} W_k(s^i, s) \otimes W(s, s^f)$ is thus the estimation error due to ignoring some runs from s^i to s^f in the depth first graph traversal run over $\mathcal{A}$. Each contribution to the error term decomposes in turn as $g_k(s) = W_k(s_i, s)$, the current expected reward on paths from s^i to state s in the border B_k^+, which is known at step k, and a remaining "reward to come after s" $W(s, s^f)$, which is unknown. If one can bound from above this future reward by $W(s, s^f) \leq h(s)$, then a stopping criterion can be derived: Alg. 3 can be stopped at step k if the upper bound $\bigoplus_{s \in B_k^+} g_k(s) \otimes h(s)$ on the remaining rewards to be collected is negligible compared to the already collected ones $W_k(s^i, s^f)$.

The reader familiar with the A* algorithm [11] computing a shortest path between s^i and s^f will notice a striking similarity: $g_k(s)$ resembles the current shortest distance from s^i to node s of the active set, and $h(s)$ resembles the "heuristic" function giving a lower bound on the remaining distance to s^f. Even more, the approach can be connected to the use of a heuristic for asynchronous Dynamic Programming in [4], inspired by Learning-Real-Time A* [13]. An essential difference is that A* - just like the original Floyd-Warshall - is designed to perform computations in the $(\mathbb{R}^+, \min, +)$ semi-ring instead of $(\mathbb{K}, \oplus, \otimes)$. Nevertheless, this analogy is instructive: A* is a depth-first search algorithm, guided to the goal s^f by the heuristic function h. It selects as next state s to cross the one minimizing $g_k(s) + h(s)$. Here, this suggests a strategy to select s^{k+1}: one should pick state s in B_k^+ with the maximal expected contribution $W_k(s^i, s) \otimes h(s)$, in order to favor visiting the (possibly) most rewarding runs in $\mathcal{A}$.

The existence of sharp heuristic functions bounding $W(s, s^f)$ from above is clearly dependent on the MDP domain (see Sec. 5 for examples). However, one has the following result:

Proposition 2. *Let R_{max} be the maximal reward in MDP $\mathcal{M}$ for policy π, let $\mathcal{A} = \Phi(\mathcal{M}_\pi, \gamma)$ be the associated weighted automaton for discount factor*

γ. If state s is at least m transitions away from s^f in $\mathcal{A}$, then $W(s, s^f) \leq \gamma^{m-1} R_{max}$.

Proof. We first turn the weighted automaton $\mathcal{A} = \Phi(\mathcal{M}_\pi, \gamma)$ into a stochastic automaton $\mathcal{B}$ by adding one extra trap state τ. Recall that in $\mathcal{A}$ one has $w(s, s') = \gamma P_\pi(s'|s)$ for $s, s' \in S$, whence a loss of $1 - \gamma$ of probability mass at each s. To make $\mathcal{B}$ a proper stochastic automaton, we need to compensate for this loss. Let us first consider states $s \in S$ that bear a reward under π, *i.e.*, such that $w(s, s^f) = R_\pi(s) \neq 0$, and let us replace this value by $1-\gamma$. So $w_\mathcal{A}(s, s^f) \leq w_\mathcal{B}(s, s^f) R_{max}/(1-\gamma)$ for such states s. For states $s \in S$ such that $w(s, s^f) = 0$, let us add an extra transition to τ by $w(s, \tau) = 1 - \gamma$. This compensates the probability loss at all states $s \in S$. To complete the procedure, let us add in $\mathcal{B}$ self loops at s^f and at τ with probability one.

Let us now consider $W(s, s^f)$ in $\mathcal{A}$, denoted $W_\mathcal{A}(s, s^f)$ for the sake of clarity. It gathers the weight of all paths in $\mathcal{A}$ relating s to s^f. These paths are the same in $\mathcal{B}$, except that the weight of their last step (s', s^f) replaces the reward $w(s', s^f) = R_\pi(s')$ by $1 - \gamma$. This entails that $W_\mathcal{A}(s, s^f) \leq W_\mathcal{B}(s, s^f) * R_{max}/(1 - \gamma)$.

Let us now consider some state $s \in S$ in $\mathcal{A}$ which is m steps (transitions) away from s^f in $\mathcal{A}$. This property is structurally preserved in stochastic automaton $\mathcal{B}$, and notice that $W_\mathcal{B}(s, s^f)$ is the probability of reaching s^f from s in $\mathcal{B}$. On the way from s to s^f there are at least $m - 1$ states with an escape to the trap state τ, *i.e.*, one progresses to s^f with at most probability γ on the first $m-1$ steps, and the last step of each path to s^f is through some (s', s^f) with probability $1-\gamma$. This results in $W_\mathcal{B}(s, s^f) \leq \gamma^{m-1}(1 - \gamma)$. Given the upper bound computed before, this entails $W(s, s^f) = W_\mathcal{A}(s, s^f) \leq R_{max}\, \gamma^{m-1}$. ◇

Proposition 2 suggests that taking $h(s) = \gamma^{m-1} R_{max}$ as upper bound on the reward to come will favor the exploration of states s that are the closest to the point s^f where rewards are collected. But again tighter domain specific upper bounds $h(s)$ can exist.

Last, consider an MDP $\mathcal{M}$ and policy π such that few states of S bear a non-null reward, and for example s^i is m transitions away from the first state bearing a reward. Then the current value $W_k(s^i, s^f)$ of the accumulated rewards will remain at 0 for the first $k < m$ steps and will only make a sudden jump when s^f is met for the first time. This suggests that the computation of $W(s^i, s^f)$ in $\mathcal{A}$ should not be made starting at s^i and progressively accumulating runs to s^f, but conversely starting at s^f to which all rewards point, and progressing backwards to reach s^i. Algebraically, this simply corresponds to inverting the orientation of all edges without changing their weight. Doing so, one can propagate the highest $W_k(s, s^f)$, *i.e.*, the most promising accumulated rewards, back to s^i. These $W_k(s, s^f)$ are non-null by construction. As upper bound $h(s)$ for $W(s^i, s)$, one has this time γ^m if s is at least m transitions away from s^i.

5 Case study

This section compares the Progressive Floyd-Warshall (PFW) algorithm introduced in Section 4.3 with Value Iteration (VI) and Linear Solving (LS). All algorithms have been implemented in Python3[6], and we use a discount factor $\gamma = 0.98$ in all cases. We test FW and VI to evaluate the value of the optimal policy π^* (calculated beforehand). Note however that PFW and VI can be used in Policy Iteration (line [a] of algorithm 2) to find π^*, and that PFW can be used for other applications than value computation, as shown in Section 3.4.

In our implementation of PFW, the next chosen state s_{k+1} is the state of B_k with the highest current value $W_k(s, s^f)$. As we are interested in the whole value vector $W(s, s^f)$ (needed for policy improvement in Policy Iteration), we update the relevant edges. On the other hand, starting from $V_0(s) = 0$ for all $s \in S$, we speed up calculus of the new estimate V_{k+1} in VI by looking only at values of direct successors of each state. The algorithms will be compared w.r.t. their *runtime* in seconds. Note however that the *number of operations*, *i.e.*, the number of edges updated at every iteration differ in PFW and VI. For PFW, at each iteration, the number of operations is $|\partial_k^-(s_{k+1}) \times (\partial_k^+(s_{k+1}) \backslash (S_k \cup \{s^f\}))|$, and for VI, $\sum_{s \in S} |\partial_k^+(s)|$. PFW and VI are compared on four standard test cases: GridWorld, RiverSwim, RandomDense and RandomSparse.

GridWorld is a 2D maze, with a starting state s_0, few goal states with rewards, and walls,

[6] Code available at https://gitlab.com/aymcome/phd/-/tree/main/code/FWPI.

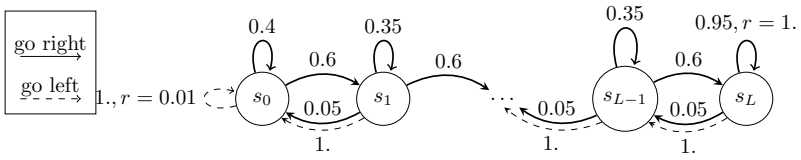

Fig. 8 The RiverSwim MDP of size $L+1$. Thick edges are transitions for action `go right` and dashed edges are for action `go left`.

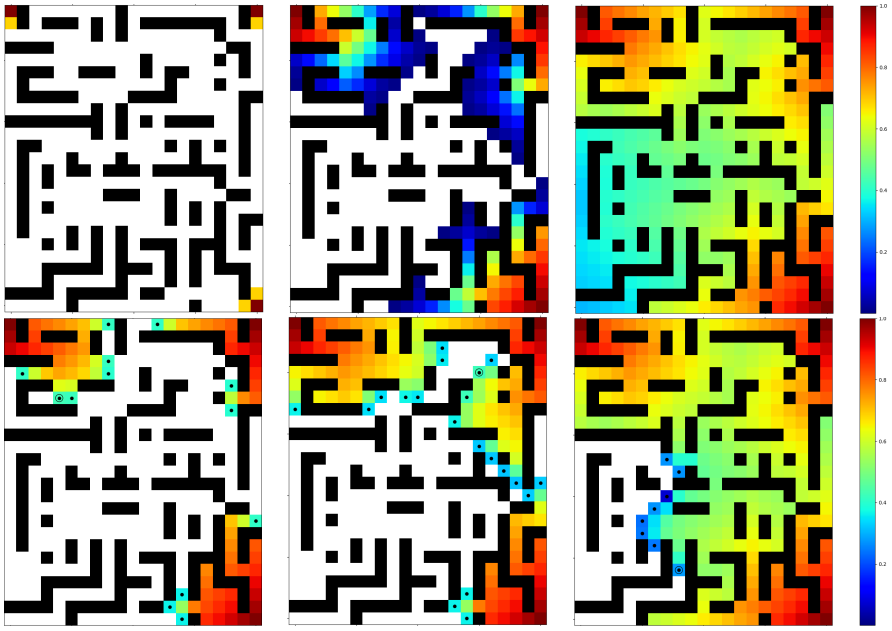

Fig. 9 Three stages of the VI (upper row) and FW (bottom row) procedures.

which are absorbing states with no predecessor. The four available actions `Up, Right, Bottom, Left` lead to the corresponding adjacent state with a probability 0.7, but might slip to one of the three other adjacents states with a probability 0.3; if there is a wall, the agent stays at the same state. Multiple variants may be used, by default any action taken from a goal state gives the corresponding reward and teleports back to s_0 with probability 1. This mechanism hardens the exploration, compared with a shortest-path problem. The GridWorlds we use are randomly generated mazes with s_0 being located at the bottom-left corner, and the three goal states (always accessible) at the three other corners with a reward of 1 each; an example can be seen in Figure 9.

RiverSwim (a.k.a. Chain MDP), drawn in Figure 8, is another standard example. It simulates a swimmer trying to go upstream. Going right (upstream) is hard, while going left is ensured; and yet the reward at s_L is a hundred times higher than at s_0. A policy has to decide between fighting for a high reward or going for the smaller, easier one. Even with a small number of states, backpropagation of the reward is long, making this case study hard to solve.

Finally, **RandomDense** and **RandomSparse** are MDPs with randomly generated transitions and reward locations. Both MDPs have four actions. In the dense case, each state-action pair has a non-zero probability transition towards 70% of the state space, and only 1% in the sparse case. 2% of the state-action pairs (and at least one in the MDP) give a reward of value 1.

Let us first compare the propagation of values in VI and PFW when the two algorithms compute the value for V_*. We show snapshots of current state values at three different stages of the procedures, after an almost identical number of operations. VI shows a classical diffusion behavior

Environment	Size	PFW	VI 5%	VI 1%	VI 0.1%	LS
GridWorld	11×11	**6.49 ms**	30.3 ms	45.5 ms	67.6 ms	849 ms
	25×25	412 ms	**196 ms**	267 ms	372 ms	118000 ms
	35×35	2.26 s	**0.424** s	0.563 s	0.768 s	892 s
RiverSwim	$\|S\| = 100$	**5.10 ms**	33.2 ms	40.3 ms	52.1 ms	480 ms
	$\|S\| = 625$	**47.6 ms**	217 ms	264 ms	330 ms	118000 ms
	$\|S\| = 1225$	**69.1 ms**	423 ms	514 ms	644 ms	894000 ms
RandomDense	$\|S\| = 100$	**210 ms**	293 ms	449 ms	672 ms	480 ms
	$\|S\| = 625$	51.2 s	**11.2 s**	17.1 s	25.7 s	118 s
	$\|S\| = 1225$	396 s	**43.0 s**	65.9 s	98.9 s	892 s
RandomSparse	$\|S\| = 100$	**1.69 ms**	32.5 ms	48.1 ms	68.1 ms	478 ms
	$\|S\| = 625$	17.6 s	**0.429 s**	0.658 s	0.986 s	118 s
	$\|S\| = 1225$	250 s	**1.48 s**	2.26 s	3.37 s	892 s

Table 1 Runtimes of PFW, VI, and LS. The results are averaged over 10 runs each time. Number in bold highlight the most efficient algorithm.

(Fig. 9, first line) from the reward states. At each iteration, all states are updated, but the value of some states remain at 0 until the diffusion process reaches them. The diffusion is isotropic from each reward node and would progress in growing circles if there were no obstacles in the grid. Notice that the final value is not reached when all nodes have been visited once, since the process converges asymptotically : this breadth first exploration needs more iterations to properly capture the effect of circuits in the underlying state diagram. For the FW procedure (Fig. 9, second line), we used a backwards implementation starting at s^f and progressing towards s^i, and the current values $W_k(s, s^f)$ are displayed, although one would only need those of the border to derive the final $W(s^i, s^f)$ representing the performance of policy π. Nodes of the border are depicted by black dots, and the one selected as the next state to integrate over, s^{k+1}, appears as a circled dot. At each step of the recursion, we chose to expand the state s in the border with maximal $W_k(s, s^f)$. With such a strategy, the procedure takes also the shape of an isotropic diffusion of rewards in the grid. Observe however that for similar stages of VI and FW (center), values are higher with FW at nodes that are first hit (nodes of the border), since these values correspond to rewards over all paths within the already colored region (corresponding to S_k), whereas VI needs several more iterations to correctly capture the rewards due to the circuits in this region. We provide more insights on the differences in value propagation in [1].

The table 1 above shows experimental results for three algorithms: PFW, VI and Linear Solving (LS). LS solve (6) using Gaussian elimination. The algorithms are compared on instances of Gridworld, RiverSwim, RandomDense and RandomSparse of varying sizes. Since VI is an approximate approach, we compare PFW and LS to instances of VI stopping when current estimate of the value vector is close at $5\%, 1\%$ and 0.1% to the optimal value, in infinite norm $\|\cdot\|_\infty$.

Though PFW has a theoretical cubic worst case complexity, in practice it seems to require less time and operations than LS, on all selected MDPs. This shows the impact of updating only the necessary edges in W_k, hence greatly reducing the number of operations per iteration. In fact, PFW is even more efficient than VI in RiverSwim. Ths is due to the particular structure of the example, that allows keeping the number of neighbor states low throughout the iterations. On the other hand, as the size of the problem increases PFW seems to fall off for RandomDense and RandomSparse MDPs: the number of edges scaling with the size quickly builds up the border, meanwhile vertices in GridWorld and RiverSwim never have more than 5 successors. Hence it appears that the Floyd-Warshall approach benefits from more structured MDPs, Riverswim being a best case. As a final note, FW always is the most efficient for small size MDPs, and does not depend on the value of the discount factor, unlike VI. More discussion is available in [1].

6 Conclusion

This work has proposed a reformulation of the value estimation problem in MDPs as an integral over runs of a weighted automaton, that can be computed by a Floyd-Warshall procedure. This new perspective offers several advantages: it recasts under the same algebraic roof several

apparently disconnected problems, as the computation of the gradient of the value, or its variance. But it also opens the way to several strategies to collect run rewards more efficiently, by properly processing cycles and by enabling a depth-first exploration. The first experiments reveal performance gains on some (but not all) benchmarks, and show that this approach could be turned into an approximation scheme for policy iteration. Future work will characterize the problems for which gains are obtained, using heuristics to guide the search towards most impacting runs.

References

[1] Technical report.

[2] Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research*, 22(98):1–76, 2021.

[3] Borja Balle and Mehryar Mohri. Learning weighted automata. In Andreas Maletti, editor, *Algebraic Informatics - 6th International Conference, CAI 2015, Proceedings*, volume 9270 of *LNCS*, pages 1–21. Springer, 2015.

[4] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial intelligence*, 72(1-2):81–138, 1995.

[5] Hugo Bazille, Eric Fabre, and Blaise Genest. Diagnosability degree of stochastic discrete event systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5726–5731, 2017.

[6] Dimitri P. Bertsekas. Dynamic programming and optimal control. 1995.

[7] Dimitri P. Bertsekas. Approximate policy iteration: A survey and some new methods. *Journal of Control Theory and Applications*, 9(3):310–335, 2011.

[8] P. Dai, Mausam, D. S. Weld, and J. Goldsmith. Topological Value Iteration Algorithms. *J. Artif. Intell. Res.*, 42:181–209, October 2011.

[9] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[10] Eric A Hansen and Shlomo Zilberstein. Lao⋆: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

[11] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[12] Ronald A. Howard. Dynamic programming and markov processes. 1960.

[13] Richard E. Korf. Real-time heuristic search. *Artificial intelligence*, 42(2-3):189–211, 1990.

[14] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[15] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Mach. Learn.*, 13(1):103–130, oct 1993.

[16] Rémi Munos. Error bounds for approximate policy iteration. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 560–567. AAAI Press, 2003.

[17] John L Nazareth. Conjugate gradient method. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(3):348–353, 2009.

[18] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994.

[19] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[20] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *32nd International Conference on Machine Learning, PMLR*, volume 37, pages 1889–1897, 2015.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[22] Aaron Sidford, Mengdi Wang, Xian Wu, and Yinyu Ye. Variance reduced value iteration and faster algorithms for solving markov decision processes. *Naval Research Logistics (NRL)*, 70(5):423–442, 2023.

[23] Matthew J. Sobel. The variance of discounted markov decision processes. *Journal of Applied*

Probability, 19(4):794–802, 1982.

[24] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16:285–286, 2005.

[25] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[26] Li Xia. Mean–variance optimization of discrete time discounted markov decision processes. *Automatica*, 88:76–82, 2018.

A The Floyd Warshall Algorithm

The Floyd Warshall algorithm is usually applied to compute iteratively shortest path between pairs of vertices, considering at each iteration a new unexplored vertex s_k. However, it can be easily adapted to computes iteratively weights of paths between pairs of vertices. The standard algorithm computes the length of the shortest path from each $s_i, i \in 1..N$ to each $s_j, j \in 1..N$. The calculus is iterative, and introduces at each step a new unexplored vertex s_k. Floyd Warshall can be easily adapted to compute the sum of weights of all path from s_i to s_j as shown in Algorithm 4.

Algorithm 4 Floyd Warshall on a Weighted graph (FWWG)

Input: A matrix W_0 associating a weight to every pair of states s, s'
for $s_k \in S$ **do**
  for $s_i \in S$ **do**
    for $s_j \in S$ **do**
      $W_{n+1}(s_i, s_j) = W_n(s_i, s_j) + W_n(s_i, s_k) \cdot W_n(s_k, s_k)^* \cdot W_n(s_k, s_j))$
    end for
  end for
end for
Return: W_N

B Closed form expression for the value function gradient

Path decompositions in the Markov reward process $\mathcal{M}$ led to the fix point equation (27) for the gradient. Recall that $|S| = K$ and $\theta \in \mathbb{R}^d$, so by stacking all gradients $\nabla_\theta V_\theta(s)$ into a single vector $\nabla_\theta V_\theta$ of dimension Kd, and similarly for the $Q_\theta(s,a)\nabla_\theta \pi_\theta(a|s)$ into vector $Q_\theta(\cdot, a)\nabla_\theta \pi_\theta(a|\cdot)$, (27) becomes

$$\nabla_\theta V_\theta = \sum_{a \in A} Q_\theta(\cdot, a)\nabla_\theta \pi_\theta(a|\cdot) + \gamma\big(P_\theta \odot I_d\big)\nabla_\theta V_\theta$$

where $\odot$ denotes the Kronecker product. This Bellman type equation resolves as

$$\nabla_\theta V_\theta = (I - \gamma P_\theta \odot I_d)^{-1}\big(\sum_{a \in A} Q_\theta(., a)\nabla_\theta \pi_\theta(a|\cdot)\big)$$

Focusing on a single starting state s, this becomes

$$\begin{aligned} &\nabla_\theta V_\theta(s) \\ &= \sum_{s' \in S} d_\theta(s, s') \sum_{a \in A} Q_\theta(s', a)\nabla_\theta \pi_\theta(a|s') \end{aligned} \tag{43}$$

where

$$d_\theta(s, s') = (I - \gamma P_\theta)^{-1}(s, s') = \sum_{k \geq 0} \gamma^k P_\theta^k(s, s')$$

denoting by $P_\theta^k(s, s')$ the probability to go from s to s' in k steps through policy π_θ. (43) was already derived in [25] (see Thm. 1) as the basis of reinforcement learning methods, where both the quality of state action pairs $Q_\theta(s', a)$ and the random walk (discounted) distance $d_\theta(s, s')$ need to be estimated from rollouts rooted at s.

Observe that quantities $d_\theta(s, s')$ and $Q_\theta(s', a)$ are path integrals. It is therefore natural to wonder if they could appear more simply from the semiring formalism and the FW integration procedure. This is what we present now.

Let us first refine the definition of the weighted automaton $\mathcal{A}$ attached to the Markov reward process $\mathcal{M}_\pi$ by adding all pairs (s, a) as extra middle states. The state set of $\mathcal{A}$ becomes $\bar{S} = S \cup (S \times A) \cup \{s^f\}$. Transition weights follow accordingly (Fig. 10) by $w(s, (s, a)) = \pi_\theta(a|s)$, $w((s, a), s') = P(s'|s, a)$ and $w((s, a), s^f) = R(s, a)$. These weights are then augmented by

their gradient as $\bar{w}(q,q') = (w(q,q'), \nabla_\theta w(q,q'))$, so only edges of the form $(s,(s,a))$ have a non-vanishing weight gradient. In this extended model, Prop. 1 still holds so $\bar{W}(s,s^f) = (V_\theta(s), \nabla_\theta V_\theta(s))$, and one easily checks that $\bar{W}((s,a),s^f) = (Q_\theta(s,a), \nabla_\theta Q_\theta(s,a))$.

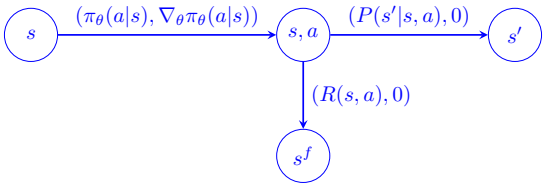

Fig. 10 Adding extra intermediate states (s,a) to the weighted automaton $\mathcal{A}$ attached to a Markow reward process.

We are interested in a closed form expression for the second component in $\bar{W}(s^i,s^f)$. It derives from a specific property of the semi-ring defined by (24). Observe that in $(A,B) = (a_1,b_1) \otimes \ldots \otimes (a_n,b_n)$ one has $B = \sum_i a_1 \ldots a_{i-1} b_i a_{i+1} \ldots a_n$, *i.e.* the "$b$" term (or the gradient) is taken only once, for each possible position, or each edge if (A,B) is a path weight. Consider now all paths from q to q'. One has $\mathcal{P}(q,q') = \bigcup_{\alpha,\beta\in\bar{S}} \mathcal{P}(q,\alpha)(\alpha,\beta)\mathcal{P}(\beta,q')$ (which is not a partition). The edge (α,β) is the one for which the b term will be considered, while for all other edges the a term will be used. Then the above property of the product generalizes into

$$\bar{W}(q,q') = \big(W(q,q'), \nabla_\theta W(q,q')\big) = \big(W(q,q'), \sum_{\alpha,\beta\in\bar{S}} W(q,\alpha)\,\nabla_\theta w(\alpha,\beta)\,W(\beta,q')\big) \tag{44}$$

Taking into account that only edges of the form $(s,(s,a))$ have a non-vanishing gradient, this yields

$$\begin{aligned} &\nabla_\theta V_\theta(s^i) = \nabla_\theta W(s^i,s^f) \\ &= \sum_{s\in S, a\in A} W(s^i,s)\,\nabla_\theta \pi_\theta(a|s)\,W((s,a),s^f) \\ &= \sum_{s\in S, a\in A} d_\theta(s^i,s)\,\nabla_\theta \pi_\theta(a|s)\,Q_\theta(s,a) \end{aligned} \tag{45}$$

which coincides with (43). As noted by several authors, it is remarkable that the gradient of the value function does not depend on derivatives of the quality function Q_θ or of the distribution d_θ.

C Variance of the discounted reward

This appendix extends Sec. 3.4.3 by removing the simplifying assumptions made there: finite runs and no discount factor on future rewards.

To take the discount factor into account, some extra complications must be incorporated into the construction of Sec. 3.4.3. In essence, when connecting two paths σ and $\tilde{\sigma}$ with respective (discounted) rewards r and $\tilde{r}$, one obtains the reward of path $\sigma\tilde{\sigma}$ as $r + \gamma^n \tilde{r}$, where n is the length of prefix σ. To reflect this extra discount applied to the reward of suffix $\tilde{\sigma}$, computations now require a semi-ring structure over $(\mathbb{R}^+)^6$, by attaching to each transition (s,s') the weight

$$\bar{w}(s,s') = P_\pi(s'|s) \cdot \big(1, \gamma, \gamma^2, R_\pi(s), \gamma R_\pi(s), R_\pi(s)^2\big) \tag{46}$$

So the quantities of interest will appear in the 4th and 6th positions, for the first and second moments respectively of the discounted rewards, the other components appearing only to enable consistent computations. As semi-ring operations, $\oplus$ remains component-wise, and $\otimes$ is given by

$$\begin{aligned} &(a_0,a_1,a_2,b_0,b_1,c_0) \otimes (\tilde{a}_0,\tilde{a}_1,\tilde{a}_2,\tilde{b}_0,\tilde{b}_1,\tilde{c}_0) \\ &= (a_0\tilde{a}_0,\ a_1\tilde{a}_1,\ a_2\tilde{a}_2,\ b_0\tilde{a}_0 + a_1\tilde{b}_0, \\ &\quad b_1\tilde{a}_1 + a_2\tilde{b}_1,\ c_0\tilde{a}_0 + 2b_1\tilde{b}_0 + a_2\tilde{c}_0\,) \end{aligned} \tag{47}$$

with $\bar{1} = (1,1,1,0,0,0)$ as unit. These operations make $(\mathbb{R}^+)^6$ a *non-commutative* semi-ring, as by construction the discount of the prefix applies to the suffix in the product. If path σ has probability p, a length n (thus a discount $\rho = \gamma^n$ for its suffixes) and a total contribution r to the discounted reward, its weight will be $\bar{W}(\sigma) = p \cdot (1,\rho,\rho^2,r,\rho r,r^2)$. Again the proof is by recursion. It readily holds for transitions, and for path $\sigma\tilde{\sigma}$, one has

$$\begin{aligned} &\bar{W}(\sigma) \otimes \bar{W}(\tilde{\sigma}) \\ &= p \cdot (1,\rho,\rho^2,r,\rho r,r^2) \otimes \tilde{p} \cdot (1,\tilde{\rho},\tilde{\rho}^2,\tilde{r},\tilde{\rho}\tilde{r},\tilde{r}^2) \\ &= p\tilde{p} \cdot (1,\rho\tilde{\rho},(\rho\tilde{\rho})^2, r+\rho\tilde{r}, \rho\tilde{\rho}(r+\rho\tilde{r}), (r+\rho\tilde{r})^2) \\ &= \bar{W}(\sigma\tilde{\sigma}) \end{aligned} \tag{48}$$

which reflects that the total (discounted) reward of path $\sigma\tilde{\sigma}$ has to be $r + \rho\tilde{r}$.

Again the star operation is well defined in this semi-ring: for $0 \leq a_0, a_1, a_2 < 1$ one has

$$\begin{aligned}
&(a_0, a_1, a_2, b_0, b_1, c_0)^n \\
&= (a_0^n, a_1^n, a_2^n,\ b_0 \sum_{\substack{i+j \\ =n-1}} a_0^i a_1^j,\ b_1 \sum_{\substack{i+j \\ =n-1}} a_1^i a_2^j, \\
&\quad c_0 \sum_{\substack{i+j \\ =n-1}} a_0^i a_2^j + 2b_0 b_1 \sum_{\substack{i+j+k \\ =n-2}} a_0^i a_1^j a_2^k\ ) \qquad (49)
\end{aligned}$$

and consequently

$$\begin{aligned}
&(a_0, a_1, a_2, b_0, b_1, c_0)^* \ = \qquad (50) \\
&(a_0^*, a_1^*, a_2^*, b_0 a_0^* a_1^*, b_1 a_1^* a_2^*, c_0 a_0^* a_2^* + 2b_0 b_1 a_0^* a_1^* a_2^*)
\end{aligned}$$

denoting $x^* = \frac{1}{1-x}$ for $0 \leq x < 1$. The Floyd-Warshall algorithm is thus applicable to derive the first and second moments of the discounted rewards. For the simple example in Fig. 4, this yields

$$\begin{aligned}
&\bar{W}(s^1, s^f) \\
&= \left[p\,(1, \gamma, \gamma^2, \alpha, \gamma\alpha, \alpha^2) \right]^* \\
&\quad \otimes (1-p)(1, \gamma, \gamma^2, \alpha, \gamma\alpha, \alpha^2) \otimes (1, \gamma, \gamma^2, \beta, \gamma\beta, \beta^2) \\
&= p^* \big(1, \frac{(p\gamma)^*}{p^*}, \frac{(p\gamma^2)^*}{p^*}, p\alpha(p\gamma)^*, \\
&\qquad p\alpha\gamma \frac{(p\gamma)^*(p\gamma^2)^*}{p^*}, \alpha^2 (p\gamma)^* (p\gamma^2)^* p(1+p\gamma)\big) \\
&\quad \otimes (1-p)\big(1, \gamma^2, \gamma^4, \alpha+\beta\gamma, \gamma^2(\alpha+\beta\gamma), (\alpha+\beta\gamma)^2\big) \\
&= \Big(1, ..., ..., \frac{\alpha+\beta\gamma(1-p)}{1-p\gamma}, ..., \frac{\alpha^2(1+p\gamma)}{(1-p\gamma)(1-p\gamma^2)} + \\
&\qquad \frac{2\alpha\beta\gamma(1-p)}{(1-p\gamma)(1-p\gamma^2)} + \frac{\beta^2\gamma^2(1-p)}{1-p\gamma^2}\Big) \qquad (51)
\end{aligned}$$

(only values of interest represented) which coincides with (36) for $\gamma = 1$.

It is rather amazing that introducing a simple discount factor γ imposes to go as far as 6 as semi-ring dimension in order to perform the necessary computations, while dimension 3 is sufficient when there is no discount. One can actually prove that:

Proposition 3. *To compute the second moment of the discounted reward as an integral over runs of a weighted automaton, one needs a semi-ring of dimension at least 6 (this semi-ring being an extension of $\mathbb{R}^+$).*

Proof. Let $(\mathbb{K}, \oplus, \otimes)$ be a semi-ring and consider $\mathbb{M} = \mathbb{K}^d$ for some integer d. Taking a field for $\mathbb{K}$ would make $\mathbb{M}$ a vector space, taking a ring would make it a module, and here $\mathbb{M}$ is simply a semi-module over $\mathbb{K}$. As a semi-module, $\mathbb{M}$ readily enjoys an addition, the component-wise addition inherited from $\mathbb{K}$, denoted as $\boxplus$ for the sake of clarity. This explains that all additions are pointwise in the composite semi-rings of this paper. To turn $\mathbb{M}$ into a semi-ring, one must define a product $\boxtimes$ over its elements, and numerous choices are possible as we already saw.

Step 1: the product $\boxtimes$ in $\mathbb{M}$ must be bilinear, *i.e.* $\forall a_1, ..., a_n, b \in \mathbb{M}$, $\forall k_1, ..., k_n \in \mathbb{K}$, $(k_1 \otimes a_1 \boxplus ... \boxplus k_n \otimes a_n) \boxtimes b = k_1 \otimes (a_1 \boxtimes b) \boxplus ... \boxplus k_n \otimes (a_n \boxtimes b)$, and symmetrically over the b term. Let us first observe that $\boxtimes$ must distribute over $\boxplus$ in $\mathbb{M}$, so one only has to prove that $(k \otimes a) \boxtimes b = k \otimes (a \boxtimes b)$ for all $k \in \mathbb{K}$. Distributivity also entails that $(a \boxplus a) \boxtimes b = (a \boxtimes b) \boxplus (a \boxtimes b)$ which we rewrite as $(2 \otimes a) \boxtimes b = 2 \otimes (a \boxtimes b)$, with the convention that $2 = 1 \oplus 1$ in $\mathbb{K}$, and similarly $(n \otimes a) \boxtimes b = n \otimes (a \boxtimes b)$. This relation extends to all $k \in \mathbb{K}$ with minor assumptions. Typically when $(\mathbb{K}, \oplus)$ as a semi-group has a finite number of generators, or possibly a denumerable number of generators by any element being a finite combination of them. More generally, it extends to $\mathbb{K}$ when $\otimes$ in $\mathbb{K}$ can be expressed as successive $\oplus$ operations, as it is the case in $\mathbb{Q}^+$, or in $\mathbb{R}^+$ with a continuity argument.

Step 2: as $\boxtimes : \mathbb{M} \times \mathbb{M} \to \mathbb{M}$ is a bilinear operator, it decomposes through the tensor product $\mathbb{M} \odot \mathbb{M}$ (universal property of the tensor product of semi-modules). Specifically, there exists a unique linear function $\phi : \mathbb{M} \odot \mathbb{M} \to \mathbb{M}$ such that $\boxtimes = \phi \circ \odot$, or equivalently $a \boxtimes b = \phi(a \odot b)$. Let us denote $a = (a_1, ..., a_d)$ and $b = (b_1, ..., b_d)$ and let $c = a \boxtimes b$. The above property means that each component of c decomposes as a linear combination (in $\mathbb{K}$) of terms $a_i \otimes b_j$.

Step 3. Consider now $\mathbb{K} = \mathbb{R}^+$, and let us characterize path σ in our target weighted automaton by its likelihood p, its accumulated (discounted) reward r and its discount factor $\rho = \gamma^{|\sigma|}$, and similarly for $\tilde{\sigma}$. We look for the structure of the semi-ring $\mathbb{M} = (\mathbb{R}^+)^d$ used to assign weights to transitions and paths. The quantity of interest for σ is pr^2 that, integrated over all finite runs would yield the second moment of the discounted reward, so pr^2 should appear as one coordinate of the semi-ring $\mathbb{M}$. Considering the concatenation $\sigma\tilde{\sigma}$ of two paths, this quantity must be

$$p\tilde{p}(r + \rho\tilde{r})^2 = (pr^2) \cdot (\tilde{p}) + 2(p\rho r) \cdot (\tilde{p}\tilde{r}) + (p\rho^2) \cdot (\tilde{p}\tilde{r}^2)$$

If this expression has to be a linear combination of components in the weights of σ and of $\tilde{\sigma}$, then one clearly needs the terms p, $p\rho r$, pr, $p\rho^2$ to appear as other components of the weight of σ, and similarly for $\tilde{\sigma}$. So $\bar{W}(\sigma)$ should look like $(p, p\rho^2, pr, p\rho r, pr^2)$, which shows that $\mathbb{M}$ must be of dimension 5 at least. But one should also be able to propagate these 5 terms by the product $\boxtimes$ in $\mathbb{M}$. The third term, pr, does not yet have all its ingredients. Considering again $\sigma\tilde{\sigma}$, this third term must be

$$p\tilde{p}(r + \rho\tilde{r}) = (pr) \cdot (\tilde{p}) + (p\rho) \cdot (\tilde{p}\tilde{r})$$

which reveals that $p\rho$ must also appear as another coordinate of the weight for σ, pushing up the dimension to $d = 6$. This results in weight $\bar{W}(\sigma) = (p, p\rho, p\rho^2, pr, p\rho r, pr^2)$ for path σ and in the product definition (47) for $\boxtimes$, which reveals that all coordinates can now be properly propagated to a composite path $\sigma\tilde{\sigma}$ by the semi-ring product $\boxtimes$ in $\mathbb{M}$, *i.e.* $\bar{W}(\sigma\tilde{\sigma}) = \bar{W}(\sigma) \boxtimes \bar{W}(\tilde{\sigma})$. $\diamond$