A Floyd-Warshall Approach to Value Computation in Markov Decision Processes (Extended Version)

Aymeric Come, Éric Fabre, Loïc Hélouët

A Floyd-Warshall Approach to Value Computation in Markov Decision Processes (Extended Version)

Aymeric CÔME, Éric FABRE, and Loïc HÉLOUËT

Univ. Rennes, IRISA, CNRS & INRIA, Rennes, France,
aymeric.come@inria.fr, eric.fabre@inria.fr, loic.helouet@inria.fr

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.

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 resolution would incur a cubic complexity (in the number of states). More affordable solutions use linear programming or, more commonly, iterative approaches, and estimate 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 [14], Chap. 6 for details).

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, and 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 fixed 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 especially w.r.t convergence to the true value function. Due to limited space, some proofs are omitted or sketched, but they appear in the extended version [7] 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 feedbacks 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, $P(s, a, s') = P(s' \mid s, a)$ 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 are *Markovian*, i.e., only depend on the current state and not on the history before. For reasons showcased below, the reward is required to be bounded: without loss of generality, 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 to be fixed.

A MDP generates infinite *runs*, i.e. sequences of the form $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$ 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:

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

The goal is then to find a policy π_* that achieves the highest value for starting state s_0. 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 ([14], 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 $V_*(s) = V_{\pi_*}(s) = \max_{\pi \in \Pi} V_\pi(s)$. It is interesting to have the value for every state, as the value of one state will depend on the values of its neighbors; that property is formally expressed by the *Bellman operator* $\mathcal{T}_\pi : \mathbb{R}^N \to \mathbb{R}^N$:

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

Indeed, a quick calculation shows that V_π is a fixed point of $\mathcal{T}_\pi$; and $\mathcal{T}_\pi$ is a γ-contraction, so that V_π is the only fixed point ([14], Thm. 6.2.3). Similarly, V_* is the unique fixed point of the γ-contraction *optimal Bellman operator*:

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

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

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

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

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

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 fixed 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 ([14], 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 [15], Chap.3).

Another way to compute a value vector V_π is by *value iteration* (Algorithm 1), i.e., by recursively applying the Bellman operator $\mathcal{T}_\pi$. This strategy is guaranteed to converge, as V_π is the unique fixed point of the γ-contraction $\mathcal{T}_\pi$ ([14], Thm. 6.2.3; [5]). Furthermore, the stopping criterion $\|V_{k+1} - V_k\|_\infty \geq \epsilon \frac{1-\gamma}{\gamma}$ ensures that $\|V_{k+1} - V_*\|_\infty < \epsilon$ ([14], Thm 6.3.1), in other words the estimate 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. However, while being the baseline for policy evaluation, value iteration has been found to not be the most efficient when looking for an optimal policy. Even though several optimisations have been proposed (e.g. Prioritized Sweeping approaches [12]), *policy iteration* [9] is usually preferred, presented in Algorithm 2. 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 ([14], Thm. 6.4.2), and is usually faster than value iteration [18]; it also motivates the search for policy evaluation methods. More specifically, policy iteration with approximate policy evaluation has received a lot of attention ([5, 6, 11, 13]); we will discuss approximation as well in this article.

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* $(\mathbb{R}^+, +, *, 0, 1)$.

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: $\pi_{k+1} = \pi_k = \pi_*$

A *weighted automaton* $\mathcal{A}$ over the semi-ring $\mathbb{K}$ is a pair[1] $\mathcal{A} = (S, w)$ 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 [1], with the additional requirement that $\forall s \in S \backslash \{s^f\},\ \oplus_{s' \in S} w(s, s') = \bar{1}$. 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

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

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 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

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

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[2] 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.*

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

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 a more appropriate 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 that

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

So applying (7) yields $\bar{W}(s,s') = (\sum_{\sigma\in\mathcal{P}(s,s')} w(\sigma),\ \sum_{\sigma\in\mathcal{P}(s,s')} w(\sigma)f(\sigma))$. 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 [4]). Sec. 3.3 shows how to express the discounted reward in a MDP under the form of (7), and we detail in [7] how to further exploit the semi-ring flexibility to derive related quantities as its variance or its gradient.

3.2 Floyd-Warshall recursion

Let $s^1 \prec s^2 \prec \cdots \prec s^K$ be some arbitrary ordering of states 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 $0 \le k \le K$. For $s,s' \in S$, let $\mathcal{P}_k(s,s')$ denote the set of paths from s to s' in $\mathcal{A}$

[2] 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'.

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

$$W_k(s, s') = \bigoplus_{\sigma \in \mathcal{P}_k(s,s')} w(\sigma) \quad = \quad W(\mathcal{P}_k(s, s')) \tag{9}$$

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

$$\mathcal{P}_{k+1}(s, s') = \mathcal{P}_k(s, s') \uplus \mathcal{P}_k(s, s^{k+1}) \left[\biguplus_{n \geq 0} \mathcal{P}_k(s^{k+1}, s^{k+1})^n \right] \mathcal{P}_k(s^{k+1}, s') \tag{10}$$

Applying the weight operator W to this decomposition entails

$$W_{k+1}(s, s') = W_k(s, s') \oplus W_k(s, s^{k+1}) \otimes W_k(s^{k+1}, s^{k+1})^* \otimes W_k(s^{k+1}, s') \tag{11}$$

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 $\mathbb{R}^+$) 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 (11) for all pairs $s, s' \in S$ to reach the path integral (7) by $W(s, s') = W_K(s, s')$. 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, (11) 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} 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).

Remarks. Notice first that in recursion (11) the choice of $s^{k+1} \in S \setminus S_k$ can be made on the fly at step k. Secondly, if one is only interested in the specific value $W(s^i, s^f)$, it is sufficient to perform recursion (11) only for all $s \in S \setminus S_k \cup \{s^i\}$ and $s' \in S \setminus S_k \cup \{s^f\}$ at each step. 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

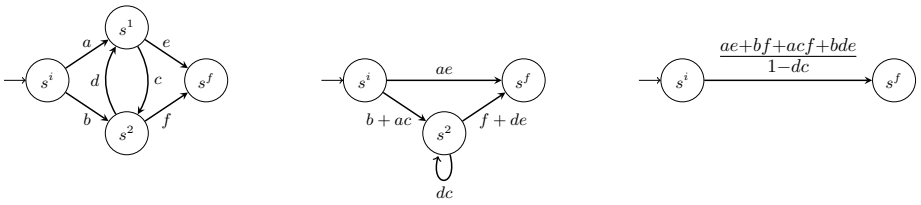

Fig. 1. Floyd Warshall path integration between s^i and s^f in the probabilistic semiring, by first eliminating s^1, then s^2.

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. 1. Finally, if one is interested in all $W(s, s^f)$ for $s \in S$ and not simply in $W(s^i, s^f)$, then (11) must be computed for all $s \in S$ and $s' \in S \setminus S_k \cup \{s^f\}$ at each step k.

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 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.

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

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi \left[\sum_{n \geq 0} \gamma^n r_n \,|\, s_0 = s \right] && (12) \\
&= \sum_{n \geq 0} \gamma^n \mathbb{E}_\pi \left[r_n \,|\, s_0 = s \right] \\
&= \sum_{n \geq 0} \gamma^n \sum_{s' \in S} R_\pi(s') P_\pi(s_n = s' | s_0 = s) \\
&= \sum_{s' \in S} R_\pi(s') \sum_{n \geq 0} \gamma^n P_\pi(s_n = s' | s_0 = s) && (13) \\
&= \sum_{s' \in S} R_\pi(s') \left(\mathbb{1}_{s'=s} + \sum_{n \geq 1} W(\mathcal{P}^{(n)}(s, s')) \right)
\end{aligned}
$$

$$= R_\pi(s) + \sum_{s' \in S} R_\pi(s')\, W(\mathcal{P}(s, s')) \tag{14}$$

$$= w(s, s^f) + \sum_{s' \in S} W(\mathcal{P}(s, s')) \cdot w(s', s^f) \tag{15}$$

$$= W\left((s, s^f) \uplus \biguplus_{s' \in S} \mathcal{P}(s, s') \cdot (s', s^f) \right) \tag{16}$$

$$= W(\mathcal{P}(s, s^f))$$

$$= W(s, s^f) \tag{17}$$

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. Relation (14) 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

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

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

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

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

The interest of this reformulation is manifold. First, by definition the value function at state s (see Eq. (12)) involves an *infinite* sum, or an expectation over *infinite* paths of the underlying Markov process rooted at s. The refactorization trick appearing at Eq. (13) 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 fixpoint 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 (Algo. 1) relies on a recursive evaluation of a fixpoint: 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. 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.

Finally, one can obtain different quantities of interest with this semi-ring approach. For example, we can compute the gradient of the V_π for parametric models/policies [18], or the variance of the discounted reward [17] (see [7] for more details).

4 Progressive Floyd-Warshall integration

Relying on Prop. 1, the expected discounted reward of policy π for MDP $\mathcal{M} = (S, A, P, R)$ starting from some specific 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 $W(s, s^f)$ is also necessary, for every $s \in S$. This section examines a specific strategy to efficiently derive these values. It relies on the interpretation of each step in the Floyd-Warshall procedure as a state elimination, where the next state s^{k+1} to process is selected on the fly.

4.1 A progressive algorithm

Let S_k be the set of vertices considered so far at the k^{th} step of recursion, 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:

$$\partial_k^+(s) = \{s' \in \{s^i, s^f\} \cup S \setminus S_k \ : s' \neq s, W_k(s, s') \neq \bar{0}\} \tag{20}$$

$$\partial_k^-(s) = \{s' \in \{s^i, s^f\} \cup S \setminus S_k \ : s' \neq s, W_k(s', s) \neq \bar{0}\} \tag{21}$$

s' is a *neighbor* of s in $\mathcal{A}_k$ iff $s' \in \partial_k^-(s) \cup \partial_k^+(s) = \delta_k(s)$. Following remarks 3.2 in Section 3.1, we know that that not every weight $W_k(s, s')$ evolves in the recursion step. This gives Algorithm 3 below. Notice that while nodes are progressively removed from the successive $\mathcal{A}_k$, s^i and s^f remain, so the desired result can be

read at the end of the procedure. If the values $W(s, s^f)$ are also desired for all $s \in S$, the update over $s \in \partial_k^-(s^{k+1})$ must be extended to states s that have already been visited and removed, *i.e.* to $s \in S_k$. The quantity $W(s, s^f)$ is then retrieved as $W_K(s, s^f)$ at the end of the procedure. Symmetrically, if the values $W(s^i, s')$ are desired for all $s' \in S$, the update over $s' \in \partial_k^+(s^{k+1})$ must also apply to S_k. Then, when $W_K(s, s^f)$ is computed for all $s \in S$, Algo 3 can be used to evaluate V_π in a policy iteration scheme (e.g. in the settings of this paper by calling Algo 3 to compute V_k at line [a] in Algo. 2).

Algorithm 3 Progressive Floyd-Warshall 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 $S \setminus S_k$ and set $S_{k+1} = S_k \cup \{s^{k+1}\}$
  set $W_{k+1} = W_k$
  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. (11)
    end for
  end for
end for
Return: $W_K(s^i, s^f)$

The choice of the next state s^{k+1} in the recursion is free and has no impact on the final result. However, observe that $W_k(s^i, s^f)$ remains the initial value $W_0(s^i, s^f)$ until there a path from s^i to s^f passing through states of S_k exists. In order to quickly grasp most rewarding paths for s^i, one should therefore choose the next state s^{k+1} to find and integrate new paths from s^i to s^f, and at least a first one if state s^i still has reward $W_0(s^i, s^f)$. This motivates structuring the recursion in Algo 3 as a path finder from s^i to s^f. Let us define the border at step k as $B_k = \partial_k^+(s^i)$. Taking $s^{k+1} \in B_k$ entails $B_{k+1} = \partial_k(s^{k+1}) \cup B_k \setminus \{s^{k+1}\}$. This choice ensures that the border "progresses" from s^i towards s^f, as in Fig. 2.

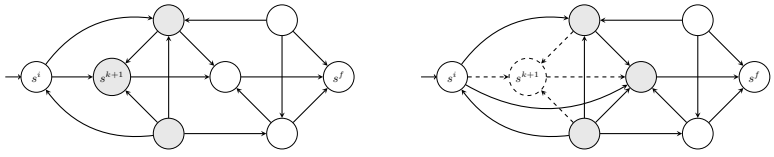

Fig. 2. Left : border B_k (in grey) at step k. Right : next border B_{k+1} after removal of state/node $s^{k+1} \in B_k$. Weights of edges to the right of the border do not change.

4.2 Approximation by early stopping

Value iteration modifies the value of states by iteratively applying a contracting operator to the current value V_k. At each step, the value gains decrease, and we have $||V_{k+1} - V_k|| \leq \lambda^k ||V_1 - V_0||$. It is hence natural to stop when the computed value is a good enough approximation of V_*. In [2], the authors use this observation to stop a Value Iteration scheme used to decide whether the value of a game can exceed a given threshold. Indeed, after a bound on the number of iterations, one can decide whether changes in a strategy can make the value of the game exceed the considered threshold. In this section, we show that the Floyd Warshall approach also give means for early stopping. $W(s^i, s^f)$ can be computed from any intermediate $\mathcal{A}_k$ and yields the same value. Observe that in $\mathcal{A}_k$, all paths of length at least 2 relating s^i to s^f must go through the border B_k. 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^i to s^f, one has:

$$\mathcal{P}_{\mathcal{A}_k}(s^i, s^f) = (s^i, s^f) \uplus \biguplus_{s \in \partial_k^+(s^i)} (s^i, s)\, \mathcal{P}_{\mathcal{A}_k}(s, s^f) \tag{22}$$

which entails through the weight operator in $\mathcal{A}_k$

$$W(s^i, s^f) = W_k(s^i, s^f) \oplus \bigoplus_{s \in \partial_k^+(s^i)} W_k(s^i, s) \otimes W(s, s^f) \tag{23}$$

This expression always holds as $W_k(s^i, s^f) = \bar{0}$ if the edge (s^i, s^f) does not exist in $\mathcal{A}_k$, and relies on the fact that path integrals computed in $\mathcal{A}_k$ and in $\mathcal{A}$ coincide. The first term $W_k(s^i, s^f)$ represents the current value of state s^i at step k, *i.e.* the expected reward of the MDP over runs starting at s^i and going through states in S_k. This term grows to reach the desired quantity $W(s^i, s^f)$. The second term $\bigoplus_{s \in \partial_k^+(s^i)} W_k(s^i, s) \oplus W(s, s^f)$ is thus the error to the true value of s^i due to ignoring some runs from s^i to s^f in the depth first graph traversal. 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 of the border, 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 as $W(s, s^f) \leq h(s)$, then a stopping criterion can be derived : the Floyd-Warshall procedure can be stopped at step k if the upper bound $\bigoplus_{s \in \partial_k^+(s^i)} g_k(s) \otimes h(s)$ on the remaining rewards to be collected is negligible compared to the already collected one $W_k(s^i, s^f)$. The reader familiar with the A* algorithm [8] 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 [3], inspired by Learning-Real-Time A* [10]. An essential difference, however, is that A* performs computations in the $(\min, +)$ semi-ring.

Nevertheless, this analogy is instructive: A* is a depth-first search algorithm, guided to the goal s^f by the heuristic function f. 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 $\partial_k^+(s^i)$ with the maximal expected contribution $W_k(s^i, s) \otimes h(s)$, in order to favor a quick visit of the 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 $W(s, s^f) \leq \gamma^{m-1} R_{max}$.*

We refer readers to [7] for a proof. This result 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 Progressive Floyd-Warshall (PFW) with Value Iteration (VI) and Linear Solving (LS). All algorithms have been implemented in Python3[3], 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 [7].

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

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

$s \in S$, 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, but also w.r.t. the *number of operations*. The latter is measured as the number of edges used at every iteration, as a tentative theoretical complexity. 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, 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 4.

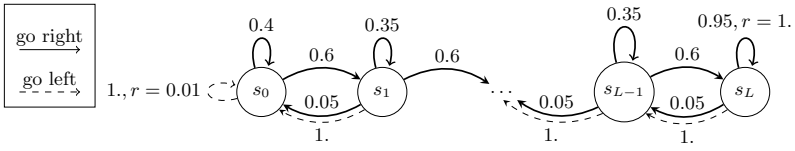

Fig. 3. The RiverSwim MDP of size $L + 1$.

RiverSwim (a.k.a. Chain MDP), drawn in Figure 3, 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 (Fig. 4, first line) from

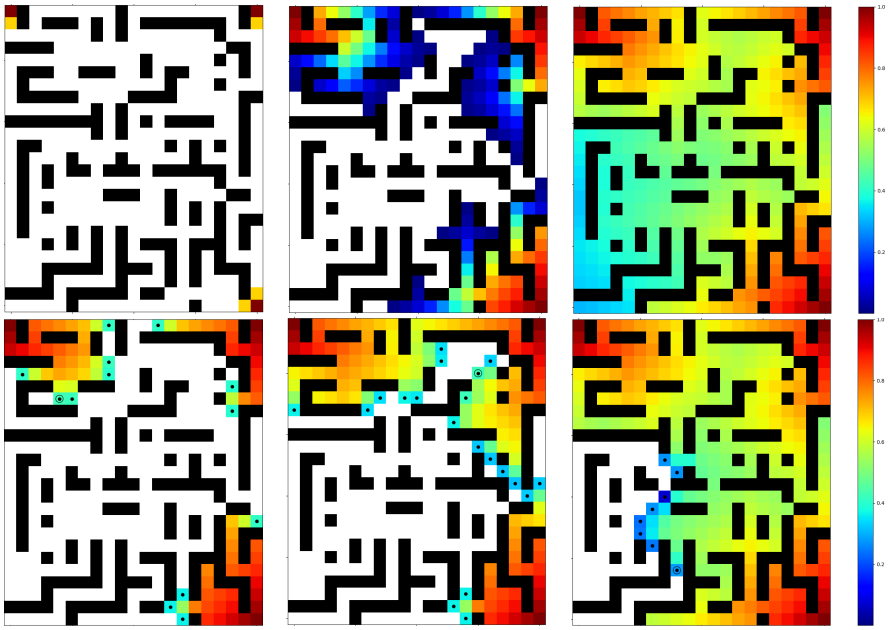

Fig. 4. Three stages of the VI (upper row) and FW (bottom row) procedures.

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 nodes 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. 4, 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 [7].

The table 1 below shows experimental results for three algorithms: FW, 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, FW is even more efficient than VI in RiverSwim, thanks to the particular structure that keeps a low number of neighboring states throughout the iterations. On the other hand, as the size of the problem increases FW 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 environments, Riverswim being a best case. As a final note, FW always is the most efficient for small size MDPs. More discussion is available in [7].

Environment	Size	FW	VI 5%	VI 1%	VI 0.1%	LS
GridWorld	11×11	6.49 ms **13586 #**	30.3 ms **89592 #**	45.5 ms **133742 #**	67.6 ms **197023 #**	849 ms **1771561 #**
	25×25	412 ms **1029 k#**	196 ms **601 k#**	267 ms **821 k#**	372 ms **1148 k#**	118000 ms **244141 k#**
	35×35	2.26 s **5671 k#**	0.424 s **1320 k#**	0.563 s **1754 k#**	0.768 s **2392 k#**	892 s **1838266 k#**
RiverSwim	$\|S\| = 100$	5.10 ms **10198 #**	33.2 ms **97148 #**	40.3 ms **120988 #**	52.1 ms **154960 #**	480 ms **1000000 #**
	$\|S\| = 625$	47.6 ms **91.0 k#**	217 ms **697 k#**	264 ms **847 k#**	330 ms **1060 k#**	118000 ms **244141 k#**
	$\|S\| = 1225$	69.1 ms **117 k#**	423 ms **1366 k#**	514 ms **1660 k#**	644 ms **2079 k#**	894000 ms **1838266 k#**
RandomDense	$\|S\| = 100$	210 ms **498 k#**	293 ms **1034 k#**	449 ms **1585 k#**	672 ms **2376 k#**	480 ms **1000 k#**
	$\|S\| = 625$	51.2 s **122 M#**	11.2 s **40.6 M#**	17.1 s **62.1 M#**	25.7 s **93.2 M#**	118 s **244 M#**
	$\|S\| = 1225$	396 s **919 M#**	43.0 s **156 M#**	65.9 s **239 M#**	98.9 s **358 M#**	892 s **1838 M#**
RandomSparse	$\|S\| = 100$	1.69 ms **1503 #**	32.5 ms **27362 #**	48.1 ms **39892 #**	68.1 ms **57873 #**	478 ms **1000000 #**
	$\|S\| = 625$	17.6 s **45074 k#**	0.429 s **561 k#**	0.658 s **861 k#**	0.986 s **1290 k#**	118 s **244141 k#**
	$\|S\| = 1225$	250 s **619 M#**	1.48 s **2.19 M#**	2.26 s **3.35 M#**	3.37 s **5.02 M#**	892 s **1838 M#**

Table 1. Runtimes and number of operations (in bold, **k#** and **M#** mean 10^3 and 10^6 operations respectively) of PFW, VI, and LS. The results are averaged over 10 runs each time.

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 characterise the problems for which gains are obtained, using heuristics to guide the search towards most impacting runs.

Bibliography

[1] 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.

[2] Suguman Bansal, Krishnendu Chatterjee, and Moshe Y. Vardi. On satisficing in quantitative games. In *Tools and Algorithms for the Construction and Analysis of Systems: TACAS 2021, Proceedings, Part I 27*, volume 12651 of *LNCS*, pages 20–37. Springer, 2021.

[3] 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.

[4] 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.

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

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

[7] Aymeric Come, Eric Fabre, and Loïc Hélouët. A Floyd-Warshall approach to value computation in markov decision processes (extended version). Technical report, `https://inria.hal.science/hal-04619555`, 2024.

[8] 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.

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

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

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

[12] 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.

[13] 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.

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

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

[16] 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.

[17] Matthew J. Sobel. The variance of discounted markov decision processes. *Journal of Applied Probability*, 19(4):794–802, 1982.

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

A Flexibility of the semi-ring setting

This appendix shows that the "semi-ring approach" used to derive the value function in an MDP is actually quite versatile and gives access to other quantities of interest, by playing on the choice of the semi-ring. Three examples are given below.

A.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's assign to each transition (s, s') the weight:

$$\bar{w}(s,s') = (w_\pi(s,s') - w_{\pi'}(s,s'), w_\pi(s,s') + w_{\pi'}(s,s')) \quad \in \mathbb{R} \times \mathbb{R}^+ \tag{24}$$

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

$$(a,b) \oplus (c,d) = (a+c, b+d) \qquad \text{with} \quad \bar{0} = (0,0) \tag{25}$$

$$(a,b) \otimes (c,d) = \frac{1}{2}(ad+bc, ac+bd) \qquad \text{with} \quad \bar{1} = (0,2) \tag{26}$$

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 $\gamma < 1$.

A.2 Policy gradient

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. Such approaches are at the state of the art for controller synthesis of cyber-physical systems [16].

Instead of computing the value function of some policy π_θ, one may rather be interested in computing its gradient with respect to parameter θ, *i.e.* $\nabla_\theta V_{\pi_\theta}(s)$. Let us assume this gradient is accessible at transition scale, *i.e.* $\nabla_\theta\, w_{\pi_\theta}(s,s')$ is well defined, and let us attach to each transition the composite weight

$$\bar{w}(s,s') = (w_{\pi_\theta}(s,s'), \nabla_\theta\, w_{\pi_\theta}(s,s')) \quad \in \mathbb{R} \times \mathbb{R}^d \tag{27}$$

As semi-ring operations over $\mathbb{R} \times \mathbb{R}^d$, consider component-wise addition for $\oplus$ as in (25), and

$$(a,b) \otimes (c,d) = (ac, ad+cb) \qquad \text{with} \quad \bar{1} = (1,0) \tag{28}$$

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

$$\begin{aligned}
\bar{W}(\sigma_1) \otimes \bar{W}(\sigma_2) &= (\, W_{\pi_\theta}(\sigma_1), \nabla_\theta W_{\pi_\theta}(\sigma_1)\,) \otimes (\, W_{\pi_\theta}(\sigma_2), \nabla_\theta W_{\pi_\theta}(\sigma_2)\,) \\
&= (\, W_{\pi_\theta}(\sigma_1) W_{\pi_\theta}(\sigma_2)\,,\, W_{\pi_\theta}(\sigma_1) \nabla_\theta W_{\pi_\theta}(\sigma_2) + W_{\pi_\theta}(\sigma_2) \nabla_\theta W_{\pi_\theta}(\sigma_1)\,) \\
&= (\, W_{\pi_\theta}(\sigma_1) W_{\pi_\theta}(\sigma_2)\,,\, \nabla_\theta [W_{\pi_\theta}(\sigma_1) W_{\pi_\theta}(\sigma_2)]\,) \\
&= (\, W_{\pi_\theta}(\sigma_1\sigma_2), \nabla_\theta W_{\pi_\theta}(\sigma_1\sigma_2)\,) \\
&= \bar{W}(\sigma_1\sigma_2)
\end{aligned} \tag{29}$$

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 $(a,b)^n = (a^n, na^{n-1}b)$ 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 a classical value iteration equation for the gradient. Specifically,

$$\bar{W}(s,s') = \bar{w}(s,s') \oplus \bigoplus_{s' \in S} \bar{w}(s,s') \otimes \bar{W}(s', s^{K+1}) \tag{30}$$

yields

$$\begin{aligned}
&(V_{\pi_\theta}(s), \nabla_\theta\, V_{\pi_\theta}(s)) \\
&= (R_{\pi_\theta}(s), \nabla_\theta\, R_{\pi_\theta}(s)) \,\oplus\, \sum_{s' \in s} \gamma\, (P_{\pi_\theta}(s'|s), \nabla_\theta\, P_{\pi_\theta}(s'|s)) \,\otimes\, (V_{\pi_\theta}(s'), \nabla_\theta\, V_{\pi_\theta}(s'))
\end{aligned} \tag{31}$$

This shows that pairs (value,gradient) form a fixpoint of a linear operator in $\mathbb{R}^{K\times(d+1)}$. Its norm is not anymore bounded by γ as it depends on the gradients of the transition probability P_{π_θ} and of the reward function R_{π_θ}. But for small enough values of the discount factor γ, this linear operator can become contracting, which opens the way to a value iteration approach for computing the gradient of V_{π_θ}.

A.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, for example to select among two policies with equal value the one with least variance. Variance computation actually amounts to computing the second moments of the discounted reward, for all possible starting states $s \in S$:

$$V_\pi^{(2)}(s) = \mathbb{E}[\,(\sum_{n\geq 0}\gamma^n r_n)^2 \mid s_0 = s\,] \tag{32}$$

Similar to the recursive form of the value function

$$V_\pi(s) = R_\pi(s) + \gamma \sum_{s'\in S} V_\pi(s')P_\pi(s'|s) \tag{33}$$

straightforward calculations yield

$$V_\pi^{(2)}(s) = R_\pi(s)^2 + 2\gamma R_\pi(s)\sum_{s'\in S} V_\pi(s')P_\pi(s'|s) + \gamma^2 \sum_{s'\in S} V_\pi^{(2)}(s')P_\pi(s'|s) \tag{34}$$

which readily shows that first and second moments of the discounted reward must be computed jointly. (33)-(34) can actually be assembled into a coupled fixpoint equation (see Eq. (2,3,4) in [17]). The necessity of this joint computation suggests that if a semi-ring approach 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 (32). This is due to the expansion of the square, that produces cross-product terms $R_\pi(s)R_\pi(s')$ which complex 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. 5, which avoids using Proposition 1.

A second difficulty appears with the discount factor γ. As all runs are now finite, let us first assume $\gamma = 1$ for simplicity, and let us attach the reward of each state s to all transitions leaving s (see Fig. 5). This yields a stochastic automaton with edge rewards, for which one wants to compute the first and second moments of run reward, 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

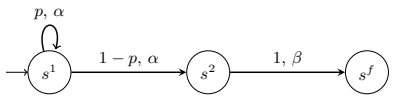

Fig. 5. A simple Markov process $\mathcal{M}_\pi$ with finite trajectories. First label on edges represent its probability. State rewards α at s^1 and β at s^2 have been "pushed" to outgoing transitions of these states.

this form in [4] (Sec. IV), and it can be recast in the semi-ring formalism by assigning to each transition (s, s') a weight in $(\mathbb{R}^+)^3$

$$\bar{w}(s,s') \triangleq (P_\pi(s'|s), P_\pi(s'|s)R_\pi(s), P_\pi(s'|s)R_\pi(s)^2) \;=\; P_\pi(s'|s) \cdot (1, R_\pi(s), R_\pi(s)^2) \quad (35)$$

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

$$(a,b,c) \otimes (\tilde{a},\tilde{b},\tilde{c}) = (a\tilde{a},\; a\tilde{b} + b\tilde{a},\; a\tilde{c} + 2b\tilde{b} + c\tilde{a}) \quad (36)$$

with unit $\bar{1} = (1,0,0)$. These operations make $\mathbb{K} = (\mathbb{R}^+)^3$ 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

$$\bar{W}(\sigma) = p \cdot (1, r, r^2) \quad (37)$$

The proof is by recursion. (37) readily holds for transitions by definition (35), and it extends by concatenation: for path $\sigma\tilde{\sigma}$ one has

$$\begin{aligned} \bar{W}(\sigma) \otimes \bar{W}(\tilde{\sigma}) &= (p, pr, pr^2) \otimes (\tilde{p}, \tilde{p}\tilde{r}, \tilde{p}\tilde{r}^2) \\ &= (p\tilde{p},\; p \cdot \tilde{p}\tilde{r} + \tilde{p} \cdot pr,\; p \cdot \tilde{p}\tilde{r}^2 + 2pr \cdot \tilde{p}\tilde{r} + \tilde{p} \cdot pr^2) \\ &= (p\tilde{p},\; p\tilde{p}(r + \tilde{r}),\; p\tilde{p}(r + \tilde{r})^2) \\ &= \bar{W}(\sigma\tilde{\sigma}) \end{aligned} \quad (38)$$

The star operation is well defined in this semi-ring: for $0 \leq a < 1$, one has

$$(a,b,c)^n = (a^n,\; na^{n-1}b,\; na^{n-1}c + n(n-1)a^{n-2}b^2) \quad (39)$$

whence

$$(a,b,c)^* = \left(\frac{1}{1-a},\; \frac{b}{(1-a)^2},\; \frac{c}{(1-a)^2} + \frac{2b^2}{(1-a)^3}\right) \quad (40)$$

The Floyd-Warshall approach thus applies to sum the weights (37) over all finite runs. For the example in Fig. 5, this yields

$$\begin{aligned}\bar{W}(s^1, s^f) &= \big[\, p \cdot (1, \alpha, \alpha^2) \,\big]^* \otimes (1-p) \cdot (1, \alpha, \alpha^2) \otimes (1, \beta, \beta^2) \\ &= \frac{1}{1-p} \cdot \left(1,\, \frac{\alpha p}{1-p},\, \frac{\alpha^2 p(1+p)}{(1-p)^2}\right) \otimes (1-p) \cdot (1,\, \alpha+\beta,\, (\alpha+\beta)^2) \\ &= \left(1,\, \frac{\alpha}{1-p} + \beta,\, \frac{\alpha^2(1+p)}{(1-p)^2} + \frac{2\alpha\beta}{1-p} + \beta^2\right) \\ &= (1,\, V(s^1),\, V^{(2)}(s^1)) \end{aligned} \tag{41}$$

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

To take the discount factor into account, some extra complications must be incorporated into the construction above. 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 $r + \rho\tilde{r}$, where $\rho = \gamma^n$ and 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

$$\bar{w}(s, s') = P_\pi(s'|s) \cdot (\, 1, \gamma, \gamma^2, R_\pi(s), \gamma R_\pi(s), R_\pi(s)^2 \,) \tag{42}$$

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

$$\begin{aligned}&(a_0, a_1, a_2, b_0, b_1, c_0) \otimes (\tilde{a}_0, \tilde{a}_1, \tilde{a}_2, \tilde{b}_0, \tilde{b}_1, \tilde{c}_0) \\ &= (a_0\tilde{a}_0,\, a_1\tilde{a}_1,\, a_2\tilde{a}_2,\, b_0\tilde{a}_0 + a_1\tilde{b}_0,\, b_1\tilde{a}_1 + a_2\tilde{b}_1,\, c_0\tilde{a}_0 + 2b_1\tilde{b}_0 + a_2\tilde{c}_0\,) \end{aligned} \tag{43}$$

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 $\rho = \gamma^n$) 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

$$\begin{aligned}\bar{W}(\sigma) \otimes \bar{W}(\tilde{\sigma}) &= p \cdot (1, \rho, \rho^2, r, \rho r, r^2) \otimes \tilde{p} \cdot (1, \tilde{\rho}, \tilde{\rho}^2, \tilde{r}, \tilde{\rho}\tilde{r}, \tilde{r}^2) \\ &= p\tilde{p} \cdot (1, \rho\tilde{\rho}, (\rho\tilde{\rho})^2, r + \rho\tilde{r}, \rho\tilde{\rho}(r + \rho\tilde{r}), (r + \rho\tilde{r})^2) \\ &= \bar{W}(\sigma\tilde{\sigma}) \end{aligned} \tag{44}$$

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

$$\begin{aligned}&(a_0, a_1, a_2, b_0, b_1, c_0)^n \\ &= (a_0^n, a_1^n, a_2^n,\, b_0 \sum_{\substack{i+j \\ = n-1}} a_0^i a_1^j,\, b_1 \sum_{\substack{i+j \\ = n-1}} a_1^i a_2^j,\, c_0 \sum_{\substack{i+j \\ = n-1}} a_0^i a_2^j + 2b_0 b_1 \sum_{\substack{i+j+k \\ = n-2}} a_0^i a_1^j a_2^k\,) \end{aligned} \tag{45}$$

and consequently

$$(a_0, a_1, a_2, b_0, b_1, c_0)^* = (a_0^*, a_1^*, a_2^*, b_0 a_0^* a_1^*, b_1 a_1^* a_2^*, c_0 a_0^* a_2^* + 2 b_0 b_1 a_0^* a_1^* a_2^*) \quad (46)$$

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. 5, this yields

$$\begin{aligned}
&\bar{W}(s^1, s^f) \\
&= \left[p \cdot (1, \gamma, \gamma^2, \alpha, \gamma\alpha, \alpha^2) \right]^* \otimes (1-p) \cdot (1, \gamma, \gamma^2, \alpha, \gamma\alpha, \alpha^2) \otimes (1, \gamma, \gamma^2, \beta, \gamma\beta, \beta^2) \\
&= p^* \cdot \left(1, \frac{(p\gamma)^*}{p^*}, \frac{(p\gamma^2)^*}{p^*}, p\alpha(p\gamma)^*, p\alpha\gamma \frac{(p\gamma)^*(p\gamma^2)^*}{p^*}, \alpha^2 (p\gamma)^* (p\gamma^2)^* p(1+p\gamma) \right) \\
&\qquad \otimes (1-p) \cdot \left(1, \gamma^2, \gamma^4, \alpha + \beta\gamma, \gamma^2(\alpha + \beta\gamma), (\alpha + \beta\gamma)^2 \right) \\
&= \left(1, \cdots, \cdots, \frac{\alpha + \beta\gamma(1-p)}{1-p\gamma}, \cdots, \frac{\alpha^2(1+p\gamma)}{(1-p\gamma)(1-p\gamma^2)} + \frac{2\alpha\beta\gamma(1-p)}{(1-p\gamma)(1-p\gamma^2)} + \frac{\beta^2\gamma^2(1-p)}{1-p\gamma^2} \right) \quad (47)
\end{aligned}$$

(only values of interest represented) which coincides with (41) 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 ρ, 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 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 (43), which reveals that all coordinates can now be properly propagated to a composite path $\sigma\tilde{\sigma}$. ◇

B Proof of Proposition 2

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 upperbound computed before, this entails $W(s, s^f) = W_\mathcal{A}(s, s^f) \leq R_{max}\,\gamma^{m-1}$.

C Experiments & further discussion

C.1 Setup

numpy[4], standard Python library for mathematical operations, has been used extensively in the implementation, in particular the array structure for vectors/matrices/tensors, so that accessing an element of an array takes $O(1)$, and not $0(n)$ with n being the number of elements, as would have been the case with lists. However, to make experimental comparisons as fair as possible and not benefit from the advanced optimisations numpy offers, the basic vector and matrix operations (addition, multiplication, inversion) have been recoded with the $+$, $\times$ and access operations as basic operations.

The algorithms have been implemented as described; note again that for VI we sticked to the base algorithm, thus ignoring multiple optimisations possible. In particular, the value update computation is still done at states where it doesn't change (states with successors with current value 0), unlike what is done in PFW to keep track of only the relevant edges. This choice was made to focus on FW and not VI, and because VI is a steady diffusion process from all states simultaneously, with incremental updates; but keep in mind that in the following results VI is not at its best.

To generate RandomDense and RandomSparse MDPs, for every state-action pair (s, a) the number of successors is drawn from a Poisson law with parameter $0.7 \times N$ and $0.01 \times N$ respectively, at least one successor. Rewards are generated similarly, with a 0.02 chance per (s, a) to give a reward of 1.; if no rewards have been generated, one is attached to a random state-action pair.

Regarding the random generation of GridWorld environments, firstly a 2D grid of walls with odd width and height is generated, then a depth-first search is done to cut a path from the starting state to every state with even coordinates. As the resulting maze was too constrained, offering little choice during the exploration, random lines of length 4 were additionally cut through the walls to open up the maze. As for the rewards, multiple configurations have been tried out, the rewards at the three other corners have been kept, for showing the most

[4] https://numpy.org/

interesting diffusion behavior. Random rewards spread around the GridWorld have also been considered: an example can be seen in Figure 6, along with two default generations.

Then, an implementation of Modified Policy Iteration ([14], section 6.5.1) is used to derive an optimal policy for the current MDP, to be evaluated in the experiments. FW is ran first, to get the exact value of the policy and set the error parameter ϵ for VI 5%, 1% and 0.1%. The process of randomly generating the MDP, compute an optimal policy and evaluate it with all 5 algorithms is done ten times per environment and size; the results are then averaged out and displayed in Table 1.

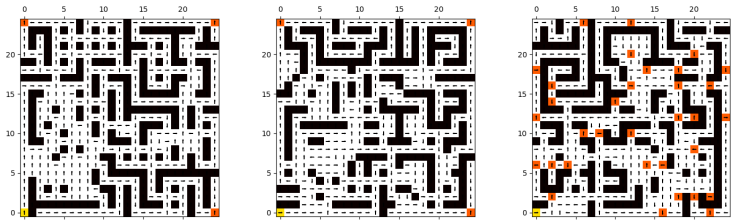

Fig. 6. Three examples of the randomly generated GridWorld environment. The yellow square indicates the starting state, orange ones the reward states.

C.2 Algorithms comparison

As disclaimed earlier in the article, one goal would be to use the PFW procedure as an approximate evaluation method, by early stopping; indeed, our intuition is that the one-step integration of circuits around a state can speed up reward diffusion. To support this idea, further experiments have been conducted, and the results are shown in Figures 7 and 8.

Four variants of the GridWorld environment are showcased:

1. with loop: taking any action on goal states gives a reward of 1 and teleports the agent back to the starting state
2. without loop: no teleportation mechanism, the agent can keep moving as usual around goal states
3. with absorbing state: collecting the reward on goal states sends the agent to an absorbing state with no reward
4. with random rewards: random rewards of value 1 spread around, with no teleportation nor absorbing state

Without teleportation nor absorbing state, the value of reward state quickly explodes, as the optimal strategy is then to stay around to collect the reward almost every iteration. The absorbing state reduces the problem to a shortest

path one, as the final results is $\gamma^c \times 1$, where c is the number of steps needed to reach a goal state. The teleportation mechanism makes it a repeated shortest path problem, but forcing VI to keep iterating to propagate the reward through the teleportation loop.

VI is run with an arbitrary error parameter $\epsilon = 1e^{-3}$.

Firstly, in Figure 7 is plotted the evolution of the value estimates for VI and FW along the total number of operations performed so far. There is a clear difference between the two approaches: while VI slowly but steadily backpropagates the reward, FW alternates between big jumps and plateaus. One can analyse it as, for a state s: a first plateau, to build up the paths towards s from the reward until the border reaches it, then a few steps to integrate the neighbors of s, getting it to its final value or almost, and finally a second plateau to finish with all of the leftover states. More specifically, on the last row, with random rewards multiple, smaller jumps can be noticed: this phenomenon corresponds to rewards coming from several sources around s, multiple border reaching it at different times.

Those figures support our intuition for early stopping in some cases, for instance it is very visible that for without loop the algorithm has reached a very good estimate way before the last iteration, and significantly faster than VI. However, other cases reflects how the FW can be worse, and not fit for early stopping. When there are no circuits for the reward to diffuse through, as in with absorbing state, it is clear that VI is significantly more efficient; and in with loop we have to wait for the border to get to s_0 to get a good estimate. The latter observation isn't final though, rather it calls the integration order into question; maybe if s_0 was reached earlier, the process could then be stopped and yield a good estimate already.

To further the analysis of the FW behavior, in Figure 8 are presented the evolution of the value estimate by FW for different states. Once again, it supports the idea for early stopping for some states and cases, such as state 3 (top left) in random rewards. Also displayed is the asynchronous update of different states: while every state value estimate is updated similarly, by big jumps, such changes occur at different times. This is consequence of the sequential aspect of FW, visiting states one by one.

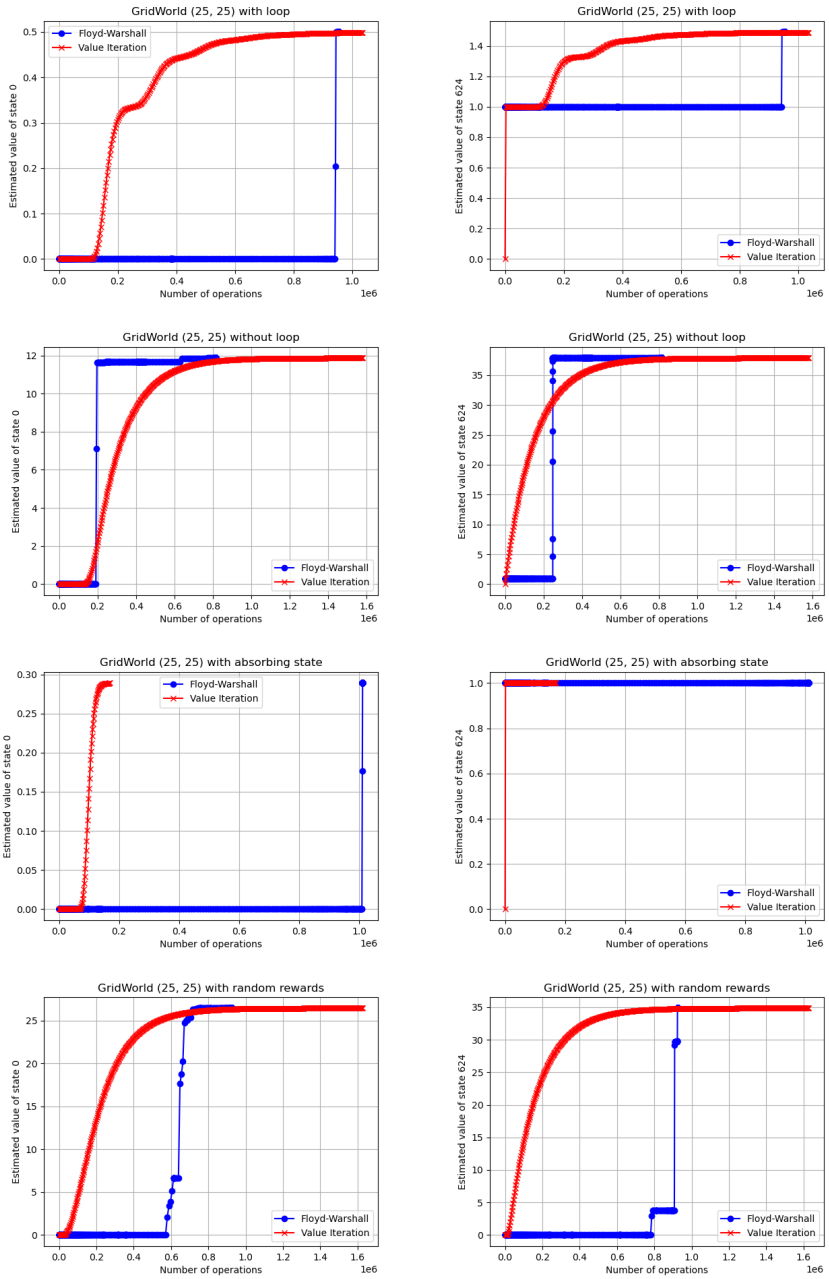

Fig. 7. Comparison of the number of operations needed between VI (red) and Floyd-Warshall (blue), monitored for the starting state (left column) and the upper right goal state (right column).

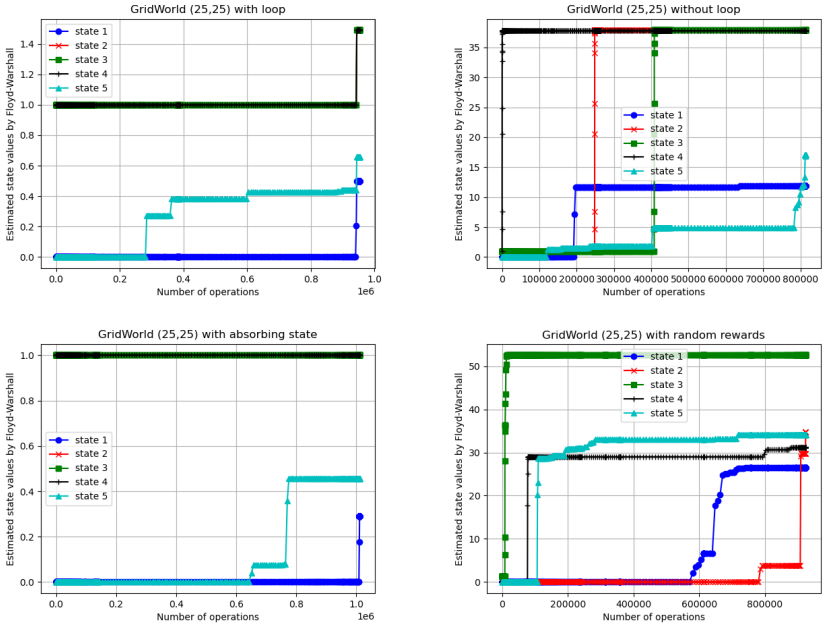

Fig. 8. Evolution of the value for different states throught the FW procedure. States labeled 1, 2, 3, 4, 5 correspond to bottom left (starting state), top right, top left, bottom right and middle respectively.