POLICY ITERATION ALGORITHM FOR SHORTEST PATH PROBLEMS *

JEAN COCHET-TERRASSON[1] AND STÉPHANE GAUBERT[2]

Abstract. The shortest paths tree problem consists in finding a spanning tree rooted at a given node, in a directed weighted graph, such that for each node i, the path of the tree which goes from i to the root has minimal weight. We propose an algorithm which is a deterministic version of Howard's policy iteration scheme. We show that policy iteration is faster than the Bellman (or value iteration) algorithm. In particular, the worst case execution time of policy iteration is $O(nm)$, where n is the number of nodes, and m is the number of arcs. Policy iteration finds rapidly a circuit of negative weight when there is one.

Résumé. Le problème de l'arbre des plus courts chemins consiste à trouver un arbre couvrant de racine donnée dans un graphe orienté valué, tel que pour tout sommet i, le chemin de l'arbre qui va de i à la racine a un poids minimum. Nous proposons un algorithme qui est une version déterministe de l'itération sur les politiques d'Howard. Nous montrons que l'itération sur les politiques converge plus vite que l'algorithme de Bellman (ou itération sur les valeurs). En particulier, l'itération sur les politiques prend au plus un temps $O(nm)$, où n est le nombre de sommets et m le nombre d'arcs. L'itération sur les politiques trouve rapidement un circuit de poids négatif, quand il en existe.

AMS Subject Classification. 05C85, 52B05, 05C38, 90C39, 90C40.

Keywords and phrases: Minimal cost, shortest path, spanning tree, max-plus algebra, linear fixed point equations, deterministic optimal stopping, policy iteration, Howard algorithm.

* *This work was partially supported by the European Community Framework IV Program through the Research Network "ALAPEDES" (The Algebraic Approach to Performance Evaluation of Discrete Event Systems, RB-FMRX-CT-96-0074)*

[1] INRIA, Domaine de Voluceau, 78153 Le Chesnay Cédex, France. Email: Jean.Cochet-Terrasson@inria.fr

[2] Corresponding author. ENSTA and INRIA. Postal address: UMA, ENSTA, 32 Bd. Victor, 75739 Paris Cédex 15, France. Email: Stephane.Gaubert@inria.fr

1. INTRODUCTION

A (weighted, directed) *graph* is a 3-uple $\mathscr{G} = (\mathscr{N}, \mathscr{A}, W)$, where $\mathscr{N}$ is a finite set of *nodes*, $\mathscr{A} \subset \mathscr{N} \times \mathscr{N}$ is the set of *arcs*, and $W : \mathscr{A} \to \mathbb{R}$, $(i, j) \mapsto W_{ij}$ is the *weight* function. The weight of a path is by definition the sum of the weights of its arcs. Given a node $r \in \mathscr{N}$, the *maximal weight paths tree problem* consists in finding a spanning tree of $\mathscr{G}$ rooted at r, such that for all nodes i, the path from i to r in the spanning tree is a path of maximal weight from i to r. An essentially equivalent form of this problem consists in finding the vector of maximal weights from all nodes to r. This problem is perhaps more familiar in its dual version, the *shortest paths tree problem*, which is defined by replacing "maximal" by "minimal" in the above description. Of course, these are basic problems of Operations Research. See Gondran and Minoux [12, Ch. 2], Ahuja, Magnanti, and Orlin [1, Ch. 4,5], and Bertsekas [3, Ch. 2] for introductions. See also Gallo and Pallotino [10], and Cherkassky, Goldberg and Radzik [4], for recent surveys and detailed experimental studies.

In this paper, we propose a policy iteration algorithm to solve the maximal weight paths tree problem. This algorithm is a specialization to the deterministic case of Howard's policy iteration algorithm, which is a very classical method in Stochastic Control (see for instance Dynkin and Yushkevich [9], Whittle [18], Bertsekas [13], or Puterman [14, 15] for overviews). In our context, a policy is in essence a map $\pi : \mathscr{N} \to \mathscr{N}$, such that $(i, \pi(i)) \in \mathscr{A}$, for all $i \in \mathscr{N}$. Intuitively, π is a rule which says that when at node i, we should go to node $\pi(i)$. By applying this rule, we obtain an under estimate x^π of the vector of maximal weights for all nodes to r. The policy iteration algorithm builds a sequence of policies π_k such that the corresponding sequence of maximal weight vectors x^{π_k} is strictly increasing. When there are no circuits with positive weight, the algorithm terminates with an optimal policy, which yields a maximal weight paths tree, and the last value of x^{π_k} is the maximal weight vector. The algorithm finds a circuit of positive weight when there is one.

A policy is a compact representation of a spanning tree rooted at r, and many path algorithms use such objects. Indeed, Gallo and Pallotino [10] remarked that an important family of "label correcting" path algorithms build, implicitly or explicitly, a sequence of policies, π_k and an associated sequence of vectors of "labels" x^{π_k}, as policy iteration does. The originality of policy iteration, by comparison with other label correcting algorithms, is that *all* the values $\pi_k(i)$, for $i \in \mathscr{N}$, are updated in parallel at each step of the algorithm.

We show that the policy iteration algorithm requires less iterations than the Bellman (or value iteration) algorithm, which is a standard method when the graph has circuits and arcs of positive weight. In particular, the algorithm performs $O(|\mathscr{N}||\mathscr{A}|)$ elementary operations. (We denote by $|X|$ the cardinality of a set X. Eliminating the trivial case where there are nodes without successors, we make the assumption that $|\mathscr{A}| \geq |\mathscr{N}|$, throughout the paper.) To our knowledge, this is the first time that a strongly polynomial bound is given for a policy iteration type algorithm. (The proof, which uses precise informations on value iteration, cannot be extended to the Stochastic Control case.)

The present study was motivated by a previous work on a slightly harder classical problem, the *maximal circuit mean* problem, which consists in finding a circuit with maximal weight to length ratio. We gave in [6] a policy iteration algorithm for this problem.

Some experiments [6], confirmed by a detailed experimental study of Dasdan, Irani, and Guptka [8] (see also Soto y Koelemeijer [16]), suggested that the policy iteration algorithm of [6] outperforms all known methods to solve the maximal circuit mean problem. Since each low level iteration of the algorithm of [6] solves essentially a maximal weight paths tree problem, it was natural to ask whether the superiority of policy iteration would remain when specializing the method to the maximal weight paths tree problem. We were disappointed to see that this is not the case.

We compared our policy iteration algorithm with the Ford-Bellman algorithm (a Gauss-Seidel refinement of the Bellman algorithm, which, according to [1, p. 156], is still the "best strongly polynomial-time algorithm for solving shortest path problems with arbitrary arcs length"). The number of iterations of both algorithms on random, full or sparse, graphs is essentially the same. We were unable to find a class of random structured graphs for which policy iteration is faster than the Bellman algorithm. For both algorithms, one iteration requires $O(|\mathscr{A}|)$ elementary operations, but one policy iteration is slightly more complex. Therefore, the Ford-Bellman algorithm seems slightly faster than policy iteration, a rather surprising conclusion since, for stochastic control problems, policy iteration is usually much faster than value iteration (the maximal weight paths tree problem appears as a degenerate "limit" deterministic case, in which this property is no longer true). However, the policy iteration algorithm is more general, since it finds without extra cost a circuit of positive weight when there is one. An added appeal of policy iteration is its incremental character: in many Operations Research problems, a good initial policy is available, and this reduces the number of policy iterations.

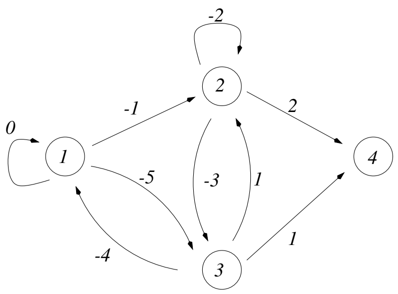
Many refinements and variants of the Ford-Bellman algorithm have appeared in the literature (see [1, § 5.4], and [4, 10]). In particular, the "FIFO label correcting algorithm", a clever implementation of the Ford-Bellman algorithm, only scans the nodes that belong to a "wave front", at each step, avoiding to make useless fixed point computations. It is possible to develop similar optimizations for the policy iteration algorithm. We hope to present these optimizations, together with refinements of the algorithm of [6], in another paper. Finally, let us note that our strong polynomiality result (Theorem 2.3 below) implies that the execution time of each low level iteration of the algorithm of [6] is strongly polynomial. Bounding accurately the number of high level iterations of the algorithm of [6] is an unsolved (harder) problem.

The paper is organized as follows. The algorithm is presented and illustrated in section 2. In section 3, we interpret the maximal weight paths tree problem in terms of deterministic optimal stopping, and we recall the classical formulation of these problems as linear fixed point problems over the max-plus semiring: this interpretation makes the proof of the algorithm more transparent, since policy iteration is nothing but the Newton algorithm, applied to the underlying fixed point problem. In section 4, we prove that the algorithm does terminate in fewer steps than the Bellman algorithm. In section 5, we present some examples and numerical experiments.

2. Policy Iteration Algorithm

In the sequel, we shall assume that the set of nodes is $\mathscr{N} = \{1, \dots, n+1\}$, and we shall solve the maximal weight paths tree problem by taking node $n+1$ as root. We say that a node i is *co-accessible* if there is a path from i to $n+1$. We shall assume that *all nodes $i \leq n$ are co-accessible*. Eliminating this degenerate case is not a severe restriction since nodes which are not co-accessible can be found in linear time by standard linear search algorithms [1, 17]. (In fact, such degenerate nodes will be ruled out when building the initial policy, see Lemma 2.1 below.) We shall also assume that *there are no arcs with origin $n+1$*. When considering the maximal weight paths tree problem, there is no loss of generality in this assumption, which simplifies the statement of the main result.

A simple instance of our problem, where $n = 3$, is depicted at the left of Fig. 1 (the reader should wait until §3 to known the meaning of the matrices at the right of the figure).



$$A = \begin{bmatrix} 0 & -1 & -5 \\ -\infty & -2 & -3 \\ -4 & 1 & -\infty \end{bmatrix}, \quad b = \begin{bmatrix} -\infty \\ 2 \\ 1 \end{bmatrix}$$

FIGURE 1. A deterministic optimal stopping problem: graph and matrices

We call *policy* a map $\pi : \mathscr{N} \setminus \{n+1\} \to \mathscr{N}$ such that for all $i \leq n$, $(i, \pi(i))$ is an arc of $\mathscr{G}$. To a policy π, we associate the graph $\mathscr{G}(\pi)$ with nodes $\{1, \dots, n+1\}$ and set of arcs $\{(i, \pi(i)) \mid 1 \leq i \leq n\}$. We say that a policy is *admissible* if the graph of π is a spanning tree of $\mathscr{G}$ rooted at $n+1$ (we orient trees from leaves to the root, thus, π is admissible if, and only if, for all nodes $i \leq n$, there is a path from i to $n+1$ in $\mathscr{G}(\pi)$). The map $\pi \mapsto \mathscr{G}(\pi)$ sets up a one to one correspondence between admissible policies and spanning trees rooted at $n+1$ (the inverse map is obtained as follows: to a spanning tree, associate the policy which sends a child to its father).

For instance, an admissible policy for the problem of Fig. 1 is shown at the left on Fig. 2. The policy displayed at the right of the figure is non-admissible, because there is no path from 1 to 4.

We need to start the algorithm with an admissible policy. Applying standard search algorithms (see e.g. [1, §3.4]) starting from node $n+1$ in the reversed graph of $\mathscr{G}$, we obtain a spanning tree rooted at $n+1$. Therefore:

Lemma 2.1. *An admissible policy can be found in time $O(|\mathscr{A}|)$ and space $O(n)$.* $\square$

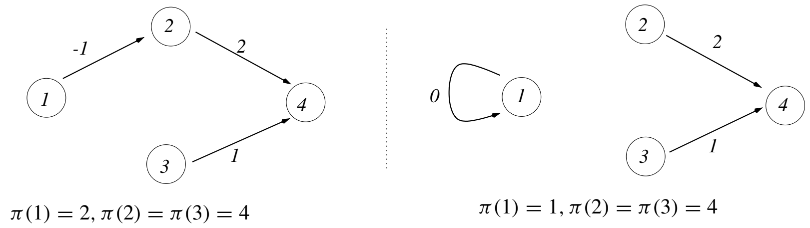


FIGURE 2. An admissible policy, and a non admissible policy

To an admissible policy π, we associate the vector x^π of dimension $n+1$, such that x_i^π is equal to the weight of the unique path from i to $n+1$ in $\mathscr{G}(\pi)$ (by convention, $x_{n+1}^\pi = 0$).

Lemma 2.2. *The vector x^π can be computed in time and space $O(n)$.*

Proof. Set $x_{n+1}^\pi = 0$. Then, make for instance a depth first search in the reverse graph of $\mathscr{G}(\pi)$, starting from $n+1$, and, when visiting a new node i, set $x_i^\pi = W_{i\pi(i)} + x_{\pi(i)}^\pi$. □

For instance, applying this algorithm to the policy at the left of Fig. 2, we get $x(4) = 0$, $x(2) = 2 + x(0) = 2$, $x(1) = -1 + x(2) = 1$, $x(3) = 1 + x(0) = 1$.

We denote by x^* the vector of dimension $n+1$ whose i-th coordinate gives the maximal weight of a path from i to $n+1$ (by convention, $x_{n+1}^* = 0$). The algorithm is as follows:

POLICY ITERATION ALGORITHM

1. *Initialization*. Set $k = 0$. Find an admissible policy π_0.
2. *Value determination*. Compute the vector $x^k \stackrel{\text{def}}{=} x^{\pi_k}$.
3. *Policy improvement*. Compute the set of *improved states*

$$I_k = \{1 \le i \le n \mid \max_{1 \le j \le n+1} (W_{ij} + x_j^k) > x_i^k\} \enspace ,$$

$$\text{and let} \quad L_k(i) = \operatorname*{arg\,max}_{1 \le j \le n+1} (W_{ij} + x_j^k) \enspace .$$

   - If $I_k = \emptyset$, then *stop*: $x^* = x^k$, and $\mathscr{G}(\pi_k)$ is a maximal weight paths tree rooted at $n+1$.
   - If $I_k \neq \emptyset$, then set, for $i \in I_k$, $\pi_{k+1}(i) =$ any element of $L_k(i)$, and for $i \notin I_k$, $\pi_{k+1}(i) = \pi_k(i)$.
4. Check if π_{k+1} is admissible. If it is not, *stop*: the graph $\mathscr{G}(\pi_{k+1})$ has a circuit with positive weight.
5. Increment k by 1 and go to step 2.

In section 4, we shall prove the following result.

Theorem 2.3. *The policy iteration algorithm stops at a step $k \le n-1$. If all the circuits have non-positive weight, then, $x^k = x^*$, $I_k = \emptyset$, and $\mathscr{G}(\pi_k)$ is a maximal weight paths tree. Otherwise, $\mathscr{G}(\pi_{k+1})$ contains a circuit of positive weight.*

Corollary 2.4. *The policy iteration algorithm runs in time $O(n|\mathscr{A}|)$ and space $O(n)$.*

Proof. By Lemma 2.1, the initialization step takes $O(|\mathscr{A}|)$ time. By Lemma 2.2, each value determination step takes $O(n)$ time. A policy improvement step requires $O(|\mathscr{A}|)$ time because one must examine exactly once every arc $(i, j) \in \mathscr{A}$ when computing I_k (the policy improvement step is indeed the bottleneck when $|\mathscr{A}| \gg n$). Therefore, each iteration requires an $O(|\mathscr{A}|)$ time. Since there are at most n iterations, the algorithm runs in $O(n|\mathscr{A}|)$ time. One can see easily that all the operations can be done in $O(n)$ space (in particular, the set $L_k(i)$ is only a convenient notation to express the choice of π_{k+1}, and it need not be stored). □

Example 2.5. Let us apply this algorithm to the graph of Fig. 1. The weight function (represented as a $\mathscr{N} \times \mathscr{N}$ matrix) is:

$$W = \begin{bmatrix} 0 & -1 & -5 & \cdot \\ \cdot & -2 & -3 & 2 \\ -4 & 1 & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix},$$

(when $(i, j) \notin \mathscr{A}$, W_{ij} is undefined and this is represented by a "$\cdot$" in the matrix). We can start with the initial policy already displayed at the left of Fig. 2: $\pi_0 = (2, 4, 4)$ (by this abridged notation, we mean $\pi_0(1) = 2$, $\pi_0(2) = \pi_0(3) = 4$). We already determined the value after the proof of Lemma 2.2: $x^0 = [1, 2, 1, 0]^T$.

In the policy improvement step, we first compute

$$\begin{aligned} \max(W_{11} + x_1^0, W_{12} + x_2^0, W_{13} + x_3^0) &= \max(\underline{0+1}, \underline{-1+2}, -5+1) = 1 = x_1^0, \\ \max(W_{22} + x_2^0, W_{23} + x_3^0, W_{24} + x_4^0) &= \max(-2+2, -3+1, \underline{2+0}) = 2 = x_2^0, \\ \max(W_{31} + x_1^0, W_{32} + x_2^0, W_{34} + x_4^0) &= \max(-4+1, \underline{1+2}, 1+0) = 3 > x_3^0\ , \end{aligned}$$

where for $1 \leq i \leq 3$, the elements of the arg max sets $L_0(i)$ are underlined. The set of improved states is $I_0 = \{3\}$, with associated arg max $L_0(3) = \{2\}$, which yields the new policy $\pi_1 = (2, 4, 2)$

We set $k = 1$, and determine the new value $x^1 = (1, 2, 3, 0)$. Then we stop in the policy improvement step, with the solution $x^1 = x^*$.

3. Optimal Stopping, Max-Plus Algebra, and Bellman-Dirichlet Problems

Optimal path problems are nothing but special discrete, deterministic, Optimal Control problems. Since policy iteration looks more intuitive in this context, we next briefly recall this classical interpretation.

Consider a one player game to be played on a weighted graph $\mathscr{G} = (\mathscr{N}, \mathscr{A}, W)$, with $\mathscr{N} = \{1, \dots, n+1\}$. Nodes represent states, $\mathscr{C} = \{1, \dots, n\}$ is the set of states in which the game continues, and $n + 1$ is a final state. Initially, there is a token at a given node $s_0 = i \in \mathscr{C}$. At time $t \in \mathbb{N}$, the token being in a state $s_t \in \mathscr{C}$, the player must move the token to a node s_{t+1} such that $(s_t, s_{t+1}) \in \mathscr{A}$. Then, he receives the reward $W_{s_t s_{t+1}}$. If the node that is chosen is the final state $s_{t+1} = n + 1$, then, the game ends, and we call

$\tau = t$ the *stopping time*. In this context, the assumption that for each node $i \in \mathscr{N}$, there is at least one arc in $\mathscr{A}$ with origin i means that the player has at least one allowed action at each state.

The deterministic optimal stopping problem consists in maximizing the total reward, i.e., in finding a path $(s_0, \dots, s_\tau, n+1)$ starting from $s_0 = i$, which maximizes the weight

$$W_{s_0 s_1} + \cdots + W_{s_{\tau-1}s_\tau} + W_{s_\tau, n+1} \quad . \tag{1}$$

Therefore, solving for all initial states i the deterministic optimal stopping problem is equivalent to solving the maximal weight paths tree problem. The idea of policy is natural in terms of optimal control: a policy is a feedback rule which gives the decision to take, as a function of the current state. For instance, for the deterministic optimal stopping problem of Fig. 1, the final state is 4. When in states 1, 3, it is optimal to move to state 2, and when in state 2, it is optimal to go to the final state 4.

To characterize the optimal reward of the above game, it is convenient to introduce the (completed) max-plus semiring $\overline{\mathbb{R}}_{\max}$, which is the set $\mathbb{R} \cup \{\pm\infty\}$, equipped with the addition $(a, b) \mapsto a \oplus b = \max(a, b)$ and the multiplication $(a, b) \mapsto a \otimes b = a + b$, with the convention $(-\infty) + (+\infty) = -\infty$. In the sequel, we shall use with out further mention the familiar algebraic notations for max-plus algebraic objects. For instance, if $A \in (\overline{\mathbb{R}}_{\max})^{p\times q}$ and $B \in (\overline{\mathbb{R}}_{\max})^{q\times r}$, AB denotes the $p \times r$ matrix such that $(AB)_{ij} = \max_{1\leq k\leq q}(A_{ik} + B_{kj})$, the identity matrix I has 0 on the diagonal and $-\infty$ outside the diagonal, etc.

To a weighted graph $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ with $\mathscr{N} = \{1, \dots, n + 1\}$, as in section 2, we associate a $n \times n$ matrix A and a vector b, defined by $A_{ij} = W_{ij}$ for all $i, j \leq n$ such that $(i, j) \in \mathscr{A}$, $A_{ij} = -\infty$ if $(i, j) \notin \mathscr{A}$, $b_i = W_{i,n+1}$ if $(i, n + 1) \in \mathscr{A}$, and $b_i = -\infty$ otherwise. Examples of matrices A, b are shown in Fig. 1.

For all $1 \leq i \leq n$, let v_i^τ denote the maximum of the total reward (1) over all trajectories $s_0, \dots, s_\tau$ with initial state $s_0 = i$. Using the classical graph interpretation of the matrix product (see for instance [12, Ch. 3, Prop. 1]), we see that the vector $v^\tau = (v_i^\tau)_{1\leq i\leq n}$ satisfies $v^\tau = A^\tau b$, where A^τ denotes the τ-th power of A in the max-plus semiring. Hence, the value function of the game, v, whose i-th entry v_i is, by definition, the supremum of all the sums (1), is given by $v = \max_{\tau\geq 0} v^\tau = b \oplus Ab \oplus A^2 b \oplus \cdots$ Alternatively, the dynamic programming principle shows that v satisfies the Bellman-Dirichlet equation

$$v_i = \max(\max_{1\leq j\leq n}(A_{ij} + v_j), b_i), \quad \text{for } i = 1, \dots, n,$$

which can be written as $v = Av \oplus b$ in the max-plus semiring. A natural idea to compute v is to use a fixed point algorithm, i.e., to define the sequence:

$$y^0 = [-\infty, \dots, -\infty]^T, \quad y^k = Ay^{k-1} \oplus b \ \text{, for } k \geq 1. \tag{2}$$

This algorithm is known as value iteration, or Bellman algorithm (see e.g. [12, Ch. 2,Algo 5]). It is immediate to check that for all $k \geq 1$,

$$y^k = (I \oplus A \oplus \cdots \oplus A^{k-1})b = \max_{0\leq\tau\leq k-1} v^\tau \quad .$$

We shall need a basic result on the convergence of the Bellman algorithm. When $y, y' \in (\overline{\mathbb{R}}_{\max})^n$, we use the notation $y \le y'$ when $y_i \le y'_i$ for all i, and $y < y'$ when $y \le y'$ and $y_j < y'_j$ for some j. Since we assumed that all nodes are co-accessible, for all $i \le n$, there is at least one j such $(i, j) \in \mathscr{A}$, which means that $A_{ij} > -\infty$ or $b_i > -\infty$. Therefore, the map $y \mapsto Ay \oplus b$, which is a priori defined $(\overline{\mathbb{R}}_{\max})^n \to (\overline{\mathbb{R}}_{\max})^n$, admits a restriction $\mathbb{R}^n \to \mathbb{R}^n$. The following theorem is an immediate consequence of standard results (see Theorem 3.17 of [2] and the subsequent results).

Theorem 3.1. *If all the circuits in $\mathscr{G}$ have non-positive weight, then, $y^{n-1} = y^n = v$ is the minimal fixed point of the map $y \mapsto Ay \oplus b$. Otherwise, $y^{n-1} < y^n$, and $y \mapsto Ay \oplus b$ has no fixed points in $\mathbb{R}^n$.* □

Example 3.2. For the optimal stopping problem depicted in Fig. 1, the maximal weight of the circuits is zero. Applying the Bellman algorithm, we get

$$y^1 = \begin{bmatrix} -\infty \\ 2 \\ 1 \end{bmatrix}, \; y^2 = y^3 = v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} .$$

Here, the solution of $x = Ax \oplus b$ is not unique (because the circuit $1 \to 1$ has weight zero, see [2, Theorem 3.17]). E.g., $x' = [1 + \lambda, 2, 3]^T$ is a solution, for all $\lambda \ge 0$.

A popular algorithm is the Gauss-Seidel variant of the Bellman algorithm, or Ford-Bellman algorithm (see e.g. [12, Ch.2,Algo. 6]). Rather than $y^k = Ay^{k-1} \oplus b$, the Ford-Bellman algorithm builds the sequence $y_1^k = A_{11}y_1^{k-1} \oplus \cdots \oplus A_{1n}y_n^{k-1} \oplus b_1$, $y_2^k = A_{21}y_1^k \oplus A_{22}y_2^{k-1} \oplus \cdots \oplus A_{1n}y_n^{k-1} \oplus b_2, \ldots, y_n^k = A_{n1}y_1^k \oplus \ldots A_{n,n-1}y_{n-1}^k \oplus A_{nn}y_n^{k-1} \oplus b_n$. The Ford-Bellman algorithm always converge more rapidly than the Bellman algorithm (see the discussion of the max-plus Jacobi and Gauss-Seidel algorithms in [12, Ch. 3, §4.1,4.2]), but the number of iterations, which depends on the numbering of nodes, is more difficult to bound accurately.

4. Proof of the Algorithm

The convergence of the policy iteration algorithm can be proved by elementary graph theory arguments [5], but we prefer to give an analytical proof which sheds more light on the properties needed for policy iteration to converge (see [7, 11] for other applications of these ideas). We associate to a policy π the affine map

$$f^\pi : \mathbb{R}^n \to \mathbb{R}^n, \; f_i^\pi(x) = \begin{cases} A_{i\pi(i)} + x_{\pi(i)} & \text{if } \pi(i) \le n, \\ b_i & \text{if } \pi(i) = n+1. \end{cases}$$

The map $f : \mathbb{R}^n \to \mathbb{R}^n$, $f(x) = Ax \oplus b$, satisfies

$$f(x) = \max_\pi f^\pi(x) \quad , \tag{3}$$

where the max is taken over all policies. The proof of the algorithm relies on the following (immediate) monotonicity, nonexpansiveness, and uniqueness properties.

Lemma 4.1. *For all policies π,*

1. *f^π is monotone, i.e. $x \leq x' \implies f^\pi(x) \leq f^\pi(x')$;*
2. *f^π is nonexpansive for the sup-norm, i.e. $|f^\pi(x) - f^\pi(x')| \leq |x - x'|$, where $|x| = \max_{1\leq i\leq n} |x_i|$;*
3. *the policy π is admissible, if, and only if, the map f^π has the unique fixed point $(x_i^\pi)_{1\leq i\leq n} \in \mathbb{R}^n$.* □

(Recall that the vector x^π was defined just before Lemma 2.2.)

For all $k \geq 0$, we set $z^k = (x_i^k)_{1\leq i\leq n}$, where x^k is the vector defined in the policy iteration algorithm. The classical proof technique, for policy iteration algorithms, is to show that the sequence z^k is increasing. Here, we show a more precise sandwich lemma, which compares the sequence of vectors z^k with the sequence y^k produced by the Bellman algorithm (see Eqn (2)).

Lemma 4.2. *We have at each step $k \geq 0$,*

$$y^k \leq z^k \ , \tag{4a}$$
$$z^k \leq y^{n-1} = (I \oplus \cdots \oplus A^{n-1})b \ , \tag{4b}$$

and, if the algorithm does not stop at step k,

$$z^k < z^{k+1} \ . \tag{4c}$$

Proof. The policy π_{k+1} built at step k of the algorithm is such that:

$$f(z^k)_i = (Az^k \oplus b)_i = \max_{1\leq j\leq n+1}(W_{ij} + x_j^k) = W_{i\pi_{k+1}(i)} + x^k_{\pi_{k+1}(i)} \ , \tag{5}$$

for all $1 \leq i \leq n$, therefore,

$$f(z^k) = f^{\pi_{k+1}}(z^k) \ . \tag{6}$$

If the algorithm does not stop at step k, we have $I_k \neq \emptyset$, i.e.

$$\max_{1\leq j\leq n+1}(W_{ij} + x_j^k) > x_i^k \ , \tag{7}$$

for some $1 \leq i \leq n$. Combining (7), (5), and (6), we get that

$$f(z^k) = f^{\pi_{k+1}}(z^k) > z^k \ . \tag{8}$$

Since $f^{\pi_{k+1}}(z^k) \geq z^k$ and since $f^{\pi_{k+1}}$ is monotone (see Lemma 4.1,1) the sequence $((f^{\pi_{k+1}})^l(z^k))_{l\geq 1}$ is nondecreasing (we denote by $(f^{\pi_{k+1}})^l = f^{\pi_{k+1}} \circ \cdots \circ f^{\pi_{k+1}}$ the l-th iterate of $f^{\pi_{k+1}}$). When the algorithm does not stop at step k, the policy π_{k+1} is admissible, and by Lemma 4.1,3 $f^{\pi_{k+1}}$ has a unique fixed point, z^{k+1}. Since $f^{\pi_{k+1}}$ is nonexpansive (see Lemma 4.1,2) we get $|(f^{\pi_{k+1}})^l(z^k) - z^{k+1}| = |(f^{\pi_{k+1}})^l(z^k) - (f^{\pi_{k+1}})^l(z^{k+1})| \leq |z^k - z^{k+1}|$ for all $l \geq 1$. Therefore, the sequence $((f^{\pi_{k+1}})^l(z^k))_{l\geq 1}$ is bounded, and since it is non-decreasing, it converges to a limit z, which, by continuity of $f^{\pi_{k+1}}$, is such that $f^{\pi_{k+1}}(z) = z$. By uniqueness of the fixed point of $f^{\pi_{k+1}}$, $z = z^{k+1}$, and we get from (8)

$$z^{k+1} = \lim_{l\to\infty} (f^{\pi_{k+1}})^l(z^k) \geq f^{\pi_{k+1}}(z^k) > z^k \quad ,$$

which shows (4c).

To prove (4a), we shall assume by induction that $y^k \leq z^k$ (the inequality holds trivially when $k = 0$). Then, using (6) and the monotonicity of f, we get

$$z^{k+1} = \lim_{l\to\infty} (f^{\pi_{k+1}})^l(z^k) \geq f(z^k) \geq f(y^k) = y^{k+1} \quad , \tag{9}$$

which shows that the inequality (4a) holds for all k.

Finally, we note that by construction each entry of z^k is equal to the weight of a path of length at most n with final node $n+1$ in $\mathcal{G}$ therefore, $z^k \leq (I \oplus A \oplus \cdots \oplus A^{n-1})b = y^{n-1}$, which shows (4b). □

Lemma 4.3. *If the algorithm finds a non admissible policy π_{k+1}, then, the graph of this policy contains a circuit with positive weight.*

Proof. If π_k is not admissible, we can find a circuit $(i_1 \to \cdots \to i_l \to i_1)$ which is such that $\pi_{k+1}(i_1) = i_2, \ldots , \pi_{k+1}(i_l) = i_1$. Using (6), we get

$$\begin{array}{rcl} f(x^k)_{i_1} = A_{i_1 i_2} + x^k_{i_2} & \geq & x^k_{i_1} \\ f(x^k)_{i_2} = A_{i_2 i_3} + x^k_{i_3} & \geq & x^k_{i_2} \\ & \vdots & \\ f(x^k)_{i_l} = A_{i_l i_1} + x^k_{i_1} & \geq & x^k_{i_l} \quad . \end{array} \tag{10}$$

We claim that one of these inequalities is strict. Indeed, if it were note the case, we would have $\pi_{k+1}(i_j) = \pi_k(i_j)$ for $j = 1, \ldots , l$, due to the conservative rule at the end of step (3) of the algorithm, which sets $\pi_{k+1}(i) = \pi_k(i)$ when the equality holds in $f(x^k)_i = (x^k)_i$. Thus, the circuit $(i_1, \ldots , i_l)$ would also belong to the graph of the previous policy π_k, which would non admissible, and thus, the algorithm would have stopped before the current step, a contradiction.

Therefore, summing all the inequalities in (10), using the fact that one is strict, and canceling $x_{i_1} + \cdots + x_{i_l}$, we get $A_{i_1 i_2} + \cdots + A_{i_l i_1} > 0$, which shows that the graph of π_k contains a circuit of positive weight. □

We finally prove Theorem 2.3. If the algorithm reaches step $k = n$, we get from (4) that $y^{n-1} \leq z^{n-1} < z^n \leq y^{n-1}$, a contradiction. Therefore, one of the two stopping conditions must be satisfied before step n.

First, if there are no circuits with positive weight, it follows from Lemma 4.3 that all the policies generated by the algorithm are admissible, therefore, stopping must occur due to the first stopping condition $I_k = \emptyset$, which means precisely that $f(z^k) = z^k$. Since by Theorem 3.1, y^{n-1} is the minimal fixed point of f, $z^k \geq y^{n-1}$, and since we proved the opposite inequality (see Eqn (4b)), we get $z^k = y^{n-1} = v$, which shows that the algorithm computes the vector of maximal weights $x^* = (v, 0)^T$.

Last, if there are circuits with positive weight, by Theorem 3.1, the map f does not have a finite fixed point, therefore, the algorithm which cannot stop with $I_k = \emptyset$, must stop by finding a non admissible policy, and, by Lemma 4.3, the graph of this policy contains a circuit a positive weight. □

5. Examples and Numerical Tests

To show that Theorem 2.3 is accurate, we next give an example where the max-plus policy iteration algorithm stops at step $n-1$.

Example 5.1. Consider the $n \times n$ matrix and the n-dimensional vector:

$$A = \begin{bmatrix} -\infty & 0 & -\infty & \dots \\ \vdots & -\infty & \ddots & \dots \\ \vdots & & \ddots & 0 \\ -\infty & \dots & & -\infty \end{bmatrix}, \quad b = \begin{bmatrix} -1 \\ \vdots \\ -1 \\ 0 \end{bmatrix} .$$

Starting from the initial policy $\pi_0(i) = n+1$ for all $i \leq n$, we get the vectors

$$x^0 = \begin{bmatrix} -1 \\ \vdots \\ -1 \\ 0 \end{bmatrix}, \; x^1 = \begin{bmatrix} -1 \\ \vdots \\ -1 \\ 0 \\ 0 \end{bmatrix}, \quad x^{n-1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

When $n = 3$, the corresponding graph and policies are depicted in Fig. 3. In this special

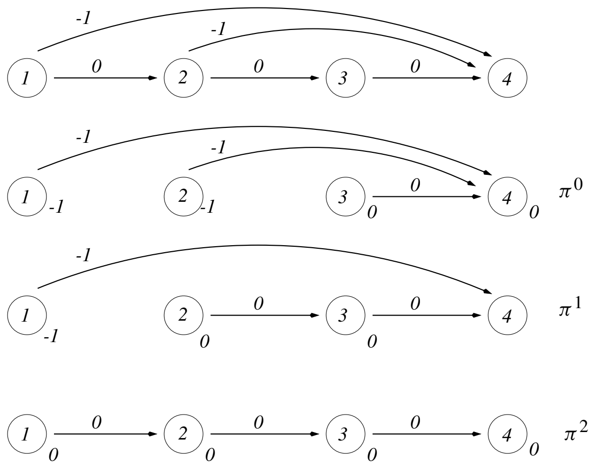


FIGURE 3. The policy iteration algorithm may stop at step $n-1$

case, policy iteration and value iteration produce the same sequence of vectors: $x^k = y^k$ for all k.

What makes the number of policy iterations smaller than the number of iterations of the Bellman algorithm is the global nature of the policy improvement step (as opposed to fixed point iterations, which have a local character). This is illustrated by the following example.

Example 5.2. For the graph of Fig. 4, with the initial policy π_0 shown on the figure, the policy improvement algorithm stops at step 1, whereas the Bellman algorithm stops at step 4.

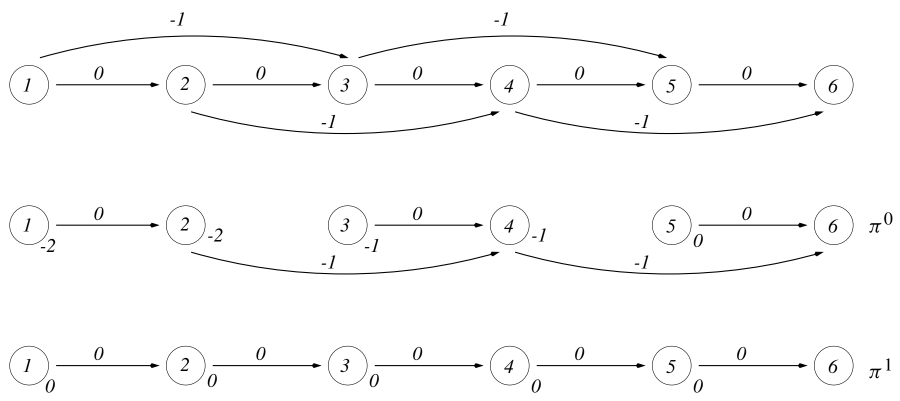


FIGURE 4. Policy improvement is parallel.

Numerical tests indicate that the average number of policy iterations grows very slowly with the size of the graph, as do the number of iterations of the Bellman and Ford-Bellman algorithm. For instance, Fig. 5 shows the number of iterations[1] of the three algorithms for full graphs (i.e. for graphs in which any two nodes are connected by an arc), as a function of the dimension. We drew randomly the n^2 entries of the matrix A, independently, with the uniform distribution in $[-1, 0]$, and we drew the n entries of the vector b, independently with a uniform distribution on $[-2n, 0]$. (Taking entries of b in $[-1, 0]$ would make short optimal paths, the interval $[-2n, 0]$ rules out this too simple case.) We started with the initial policy $\pi_0(i) = n + 1$, for all i. We drew one random pair (A, b) for each $n \in \{3, 13, 23 \dots, 2993\}$, which accounts for the random shape of the graph.

Fig. 6 shows the number of iterations for sparse graphs. The matrix A has three finite entries per row, i.e., for all i, we drew three distinct indices j in $\{1, \dots, n\}$ with the uniform distribution, and we drew the corresponding entries A_{ij} independently with the uniform

[1]The number of iterations is an implementation independent quantity, unlike the execution times that we did not display. With our current implementation (a simple program in C), one policy iteration is between 1 and 6 times slower than one iteration of the Ford-Bellman algorithm. Each policy improvement operation takes essentially the same time as a step of the Ford-Bellman algorithm. The added cost comes from the value determination operation, which requires the reverse depth first search of Lemma 2.2. For full graphs, value determination takes a negligible time, and the execution time of one iteration is the same for both algorithms.

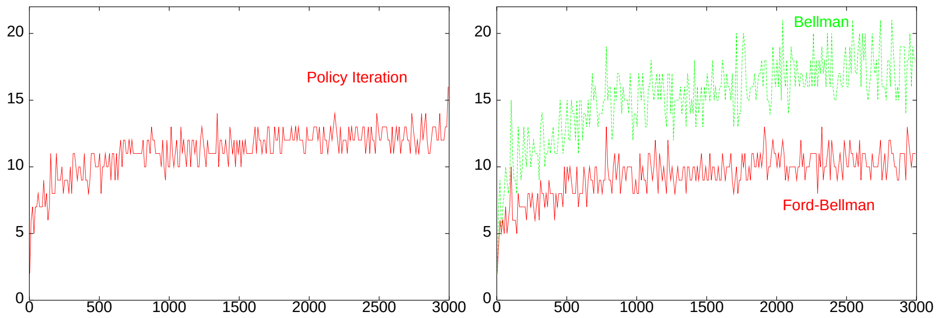


FIGURE 5. Number of iterations of the policy iteration, Bellman, and Ford-Bellman algorithms, for full random graphs, as a function of the number of nodes.

distribution on $[-1, 0]$. We took the vector b and the initial policy as in the previous test. We drew one random pair (A, b) for each $n \in \{10000, 11000, 12000, \dots, 500000\}$.

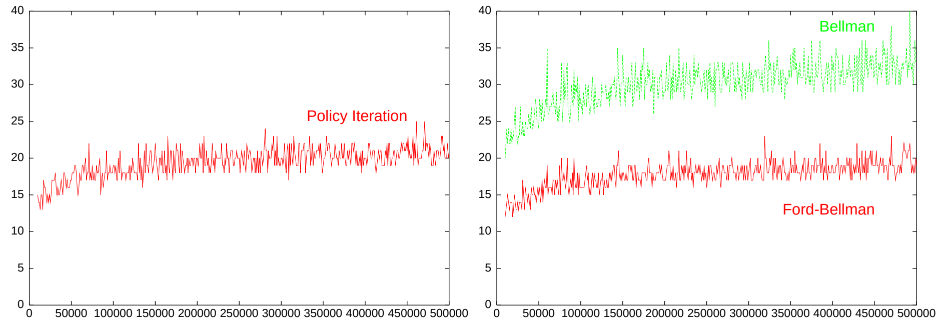


FIGURE 6. Number of iterations of the policy iteration and Bellman algorithms, for sparse random graphs with 3 arcs per node, as a function of the number of nodes.

Finally, we ran the algorithms for graphs with positive circuits. We drew the matrices as in the previous tests, but with $A_{ij} \in [-0.5, 0.5]$, which creates circuits with positive weight. The number of iterations remained of the same order (typically ≤ 20) as in the above examples. Therefore, policy iteration seems to be a good method to find a circuit of positive weight.

The authors thank Jean-Pierre Quadrat, Benoît Rottembourg, and Marianne Akian for useful suggestions and comments.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, 1993.

[2] F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronization and Linearity*. Wiley, 1992.

[3] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995. Volume one.

[4] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, June 1996.

[5] J. Cochet-Terrasson. Phd thesis. In preparation.

[6] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. Mc Gettrick, and J-P. Quadrat. Numerical computation of spectral elements in max-plus algebra. In *Proc of the IFAC Conference on System Structure and Control*, Nantes, July 1998.

[7] J. Cochet-Terrasson, S. Gaubert, and J. Gunawardena. A constructive fixed point theorem for min-max functions. *Dynamics and Stability of Systems*, 14(4), 1999.

[8] A. Dasdan, S. S. Irani, and R. K. Guptka. An experimental study of minimum mean cycle algorithms. Technical Report 98-32, Dpt. of Computer Science, University of California, Irvine, 1998.

[9] E.B. Dynkin and A.A. Yushkevich. *Controlled Markov Processes*. Springer, 1979.

[10] G. Gallo and S. Pallotino. Shortest path algorithms. *Annals of Operations Research*, 13:3–79, 1988.

[11] S. Gaubert and J. Gunawardena. A non-linear hierarchy for discrete event dynamical systems. In *Proc. of the Fourth Workshop on Discrete Event Systems (WODES98)*, Cagliari, Italy, 1998. IEE.

[12] M. Gondran and M. Minoux. *Graphes et algorithmes*. Eyrolles, Paris, 1979. Engl. transl. *Graphs and Algorithms*, Wiley, 1984.

[13] R.A. Howard. *Dynamic programming and Markov processes*. The Technology Press of M.I.T., Cambridge, Mass., 1960.

[14] M.L. Puterman. Markov decision processes. In *Handbook in Operations Research and management science*, volume 2, pages 331–434. North-Holland, Amsterdam, 1990.

[15] M.L Puterman. *Markov Decision Processes*. Wiley, 1994.

[16] G. Soto y Koelemeijer. The power and Howard algorithms in the (max,+) semiring. In *Proc. of the Fifth Workshop on Discrete Event Systems (WODES00)*, Ghent, Belgium, 2000. IEE.

[17] R.E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput*, 1:146–160, 1972.

[18] P. Whittle. *Optimization over Time*, volume II. Wiley, 1986.

October 6, 2000.