A Modified Policy Iteration Algorithm for Discounted Reward Markov Decision Processes

Sanaa Chafik
Laboratory of Information Processing and Decision Support. Sultan Moulay Slimane University, BeniMellal, Morocco.

Cherki Daoui
Laboratory of Information Processing and Decision Support. Sultan Moulay Slimane University, BeniMellal, Morocco.

ABSTRACT
The running time of the classical algorithms of the Markov Decision Process (MDP) typically grows linearly with the state space size, which makes them frequently intractable. This paper presents a Modified Policy Iteration algorithm to compute an optimal policy for large Markov decision processes in the discounted reward criteria and under infinite horizon. The idea of this algorithm is based on the topology of the problem; moreover, an Open Multi-Processing (Open-MP) programming model is applied to attain efficient parallel performance in solving the Modified algorithm.

General Terms
Theoretical Informatics, Parallelizing …

Keywords
Markov Decision Processe; Discounted reward criterion; Policy Iteration algorithm; Open Multi-Processing; shared memory; Parallelizing.

1. INTRODUCTION
Markov Decision Process (MDP) is a mathematical framework for modeling sequential decision problems under uncertainty; the applications of this model are numerous. To mention some of them, MDP models have been applied to Inventory Control [4], Queuing Systems [10], Maintenance Management [13], Health Care Management [2] and Transportation Systems [9]. Considering a discrete time MDP with finite state and action spaces under discounted reward optimality criterion. There is a large literature on methods for finding optimal policies for discounted MDP, the classical methods [16;14] when the environment is considered known such as value iteration algorithm (VI), policy iteration (PI) algorithm and linear programming, which find optimal policies in polynomial time.

Our goal is to solve very large MDPs, there are some inherent limitations of this type of statistical model then much research has been devoted to deal with large state problem, the decomposition method is the most widely used for tackling large MDP [6;7]. To this end, a modified Policy Iteration algorithm is introduced that based on the topology of each state in the associated graph.

The role of parallelism in accelerating computing speeds has been recognized for several decades. In this work, this concept is used in our modified algorithm. In general, there are three main models for parallel programming multicore architectures; this paper focuses on one of the main models: the shared memory programming model. In this direction, the Open_MP is employed which is targeted toward use on shared memory systems, it is an Application Program Interface (API), jointly defined by a group of major computer hardware and software vendors.

is devoted to

This paper is organized into 4 sections. After the introduction, some generalities about MDPs is presents by defining the ingredients or input data of the model in mathematical terms, we also define the Policy Iteration algorithm one of the most useful method of resolution of the MDP with a specific technique in the evaluation phase. Section 2, presents a Modified Policy Iteration algorithm to compute an optimal solution based on the topology of the problem, limiting our calculation for each state at the set of her successors. The next section, present in detail our parallel algorithm based on the Open_MP. Section 4 is devoted to discussing the results obtained in terms of execution time.

2. MARKOV DECISION PROCESSES
2.1 Markov chains and stochastic processes
A stochastic process is simply a collection of random variables S_t indexed by time t. It will be useful to consider separately the cases of discrete time and continuous time. a discrete time is considered, which the state changes are preordained to occur only at the integer points 0, 1, 2, …, n. A stochastic process { S_t } (t = 0, 1, …) is a Markov Chain [1] if it has the Markovian property :

$$P(S_{t+1}=s_{t+1}|S_t=s_t,...,S_0=s_0)=P(S_{t+1}=s_{t+1}|S_t=s_t) \quad (1)$$

That is, the conditional probability of any future event t+1 depends only upon the present state t.

2.2 Formalism
MDP are an extension of Markov chains, considering a stochastic dynamic system which is observed at discrete time points t = 1, 2,... . An MDP [16 ;3] is defined by $M=(S,A,T,R)$, where:

- S={1, 2, ..., M} is the set of states of the environment (the state space), let S_t is a random variable which represent the state of the system at time t whose values are in the state space S;
- A is the set of actions (the action space);
- T is the transition function (the probability of transitioning to state s' when action a is executed in state s), where:

$$T : S \times A \times S \rightarrow [0,1]$$

$$T \equiv p(S_t=s'|S_{t-1}=s,a)=p(s'|s,a) \quad (2)$$

Where $p(S_t = s' | S_{t-1} = s, a)$ represents the transition probability from state s to s' that results from taking action a.

- $R(s,a)$ is the reward function (the immediate utility of executing the action a in state s), where :

$$R : S \times A \to \square \quad (3)$$

A strategy π is defined by a sequence $\pi = (\pi^1, \pi^2, ...)$ where π^t is a decision rule, that is a function $\pi^t : H_t \to \Psi$ where $H_t = (S \times A)^{t-1} \times S$ the set of all histories up to time t, and $\Psi = \{(q_1, q_2, ..., q_{|A|}) \in R^{|A|} : \sum_{i=1}^{|A|} q_i = 1, q_i \geq 0, 1 \leq i \leq |A|\}$ the set of probability distributions over $A = \bigcup_{i \in S} A(i)$. A Markov strategy is a strategy π in which π^t depends only on the current state at time t, a stationary strategy is a Markov strategy with identical decision rules, and a deterministic or pure strategy is a stationary strategy whose single decision rule is nonrandomized. That the stationary process is assumed in the following. For there to exist an optimal policy, there must exist an optimality criterion that places some kind of ordering on different policies. The most studied criteria in the theory of MDP :

- The finite reward MDP: the total expected reward is used over the planning horizon n, the feature of this criteria is using a fixed finite number of stages:

$$E\left[r_0 + r_1 + r_2 + ... + r_{n-1} | s_0\right] \quad (4)$$

- The discounted reward MDP: In this criterion, an infinite planning horizon is considered, i.e an infinite trajectory:

$$E\left[r_0 + \gamma\, r_1 + \gamma^2 r_2 + ... + \gamma^t r_t + ... | s_0\right] \quad (5)$$

  Where $\gamma \in [0,1]$ is the discount factor. When there is no discounting, i.e. the discount factor equals 1.

- The total reward MDP : also it is working in an infinite horizon:

$$E\left[r_0 + r_1 + r_2 + ... + r_t + ... | s_0\right] \quad (6)$$

- The average reward MDP:

$$\lim_{n \to \infty} \frac{1}{n} E\left[r_0 + r_1 + r_2 + ... + r_{n-1} | s_0\right] \quad (7)$$

The usual goal in MDPs is to find a policy that yields the maximum expected return over time. In this paper, we consider the discounted reward MDP. We define V_γ^π the value function which, for each policy π associate and each initial state s ∈ S:

$$V_\gamma^\pi(s) = E^\pi[r_0 + \gamma r_1 + \gamma^2 r_2 + ... + \gamma^t r_t + ... | s_0 = s]$$

$$V_\gamma^\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right] \quad (8)$$

The coefficient γ is named the discount factor wich γ∈[0, 1] smaller than 1 in order to ensures convergence of the sum. And the optimal policies π^* verifies the condition :

$$\forall \pi, \forall s \in S \; V_\gamma^\pi(s) \leq V_\gamma^{\pi^*}(s) \text{ or } \pi^* \in \arg\max_\pi V_\gamma^\pi \quad (9)$$

3. POLICY ITERATION ALGORITHM

The PI algorithm is another approach to solve discounted reward MDPs such as the Value Iteration algorithm, it aims to calculate successively policies increasingly well-behaved for MDPs :

- Starting with a random policy ;
- Calculate its value ;
- Build a better policy than the previous one ;
- Return to step, as far as we can produce a policy strictly better off under the last policy.

The algorithm starts with an arbitrary stationary policy and iteratively improves until there is no changes are made to the policy value. In general, the PI algorithm consisting of two interleaved steps (let k=0, 1, 2…):

- Evaluation of the policy π_k : calculate the V^{π_k}
- Improvement of the policy: calculate the π_{k+1} better than π_k , deduced from V^{π_k}

$$\pi_{k+1}(s) = \arg\max_{a \in A(s)} [R(s,a) + \gamma \sum_{s' \in S} p(s' | s,a) V_i^{\pi_k}(s')], s \in S \quad (10)$$

Concerning the evaluation step, they exist several methods of resolution:

- The first method aims to solve a set of linear equations with |S| unknown and |S| equations :

$$V^\gamma(\pi_k) = \sum_{t=1}^{\infty} \gamma^{t-1} P^{t-1}(\pi) r(\pi) = [I - \gamma P(\pi_k)]^{-1} r(\pi_k)$$

Then:

$$[I - \gamma [P(\pi_k)]_{ss'}] V_{\pi_k}{}^\gamma(s') = [r(\pi_k)]_s, s \in S \quad (11)$$

The $P(\pi_k)$ (which is marked in Eq.11) is the stochastic transition matrix, following the stationary policy π_k :

$$P_{s,s'}(\pi_k) = \sum_{a \in A(s)} \pi_k(a,s) p(s' | s,a) \quad (12)$$

And the vector $r(\pi_k)$ represent the gain vector, following the policy π_k :

$$[r(\pi_k)]_s = \sum_{a \in A(s)} \pi_k(a,s) r(s,a) \quad (13)$$

The term I stands for the M × M identity matrix and the exponent denotes a full matrix inversion.

Algorithm .1: PI algorithm (version 1)

$k \leftarrow 0$

Randomly initialize the policy π_k

Repeat

$\pi_k \leftarrow \pi_k'$

For $s \in S$ do

calculate $V_{\pi_k}(s)$ by solving a set of linear equations

End for

For each $s \in S$ do

If $\exists a \in A(s)$ such as:

$$R(s,a)+\gamma\sum_{s'\in S} p(s'|s,a)V_{\pi_k}(s') > V_{\pi_k}(s)$$

Then

$$\pi_k'(s) \leftarrow a$$

Else if

$$\pi_k'(s) \leftarrow \pi_k(s)$$

End for

k ← k+1

While $\pi_k = \pi_k'$

Return V_k, π_k

The number of operations required to solve the system of equations (Eq.11), for example, by Gaussian elimination is on the order of $O(|S|^3)$ [8].

- The second method is based on the iteration of values for a fixed policy [11;15]. The basic idea is to iteratively update the value functions $V_{i+1}^{\pi_k}$ (Eq. 14) of every state s using the value functions $V_i^{\pi_k}$ of the others state s' and according to the strategy π_k until the value function calculated on two successive steps are close enough. The number of operations (Eq. 14) required is on the order of $O(|S|^2)$.

Algorithm .2: PI algorithm (version 2)

Specified a threshold ε

$k \leftarrow 0$

Randomly initialize the policy π_k

Repeat

i ← 0

For all $s \in S$

$$V_i^{\pi_k}(s) = 0$$

Repeat

For all $s \in S$ do

$$V_{i+1}^{\pi_k}(s) = R(s,\pi_k(s)) + \gamma\sum_{s'\in S} P(s,\pi_k(s),s')V_i^{\pi_k}(s') \quad (14)$$

End for

i ← i+1

While $\left|V_i^{\pi_k}(s) - V_{i-1}^{\pi_k}(s)\right| < \varepsilon$

For all $s \in S$ do

$$\pi_{k+1}(s) = \arg\max_{a\in A(s)}[R(s,a)+\gamma\sum_{s'\in S} p(s,a,s')V_i^{\pi_k}(s')] \quad (15)$$

End for

k ← k+1

While $\pi_k = \pi_{k-1}$

Remark: For each iteration, the equation (Eq. 15) requires approximately $A|S|^2$ [12] multiplications and divisions. Thereafter, the segond version is best appropriate, but it is always computationally impractical in the case of very large MDP. Therefore, in the next section, a Modified Policy Iteration algorithm to alleviate the burden of dimentionality.

4. THE MODIFIED POLICY ITERATION ALGORITHM

We introduce in this section a Modified PI algorithm (Algorithm 2). Let $s \in S$, the state space S should be seen as a union of two sets: Γ(s) a set of successors according to the state s and the rest of states {S \ Γ(s)}:

$$S = \{\Gamma(s) \cup \{S \setminus \Gamma(s)\} \quad (16)$$

Where a set $\Gamma(s) = \{s' \in S / \exists a \in A(s);\ p(s'|s,a) > 0\}$.

For example in Figure 1, the state space $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$.

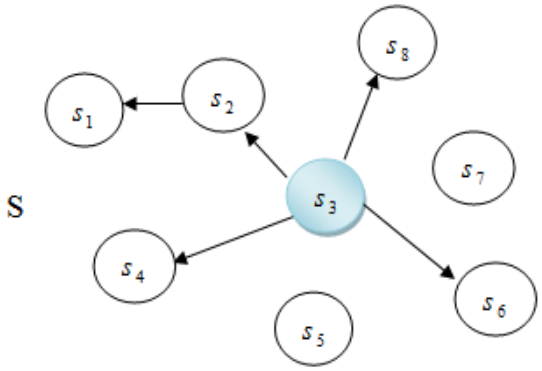

Figure 1: example of state space with 8 states

Then the state space $S = \{\Gamma(s_3) \cup (S \mid \Gamma(s_3))\}$ (Figure 2), such as $\Gamma(s_3) = \{s_2, s_3, s_4, s_6, s_8\}$ and $S \mid \Gamma(s_3) = \{s_1, s_5, s_7\}$

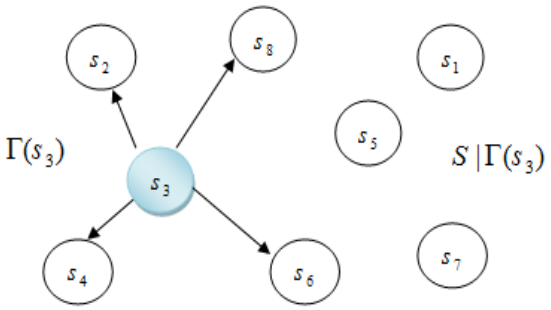

Figure 2: The decomposition of the state space S according to the successors of state s3

Therefore, we can rephrase respectively the equations (Eq. 14) and (Eq. 15) as follows:

$$V_{i+1}^{\pi_k}(s) = R(s,\pi_k(s)) + \gamma \sum_{s' \in S} p(s' \mid s,\pi_k(s)) V_i^{\pi_k}(s')$$

$$V_{i+1}^{\pi_k}(s) = R(s,\pi_k(s)) + \gamma \left[\sum_{s' \in \Gamma(s)} p(s' \mid s,\pi_k(s)) V_i^{\pi_k}(s') + \sum_{s' \in \{S \mid \Gamma(s)\}} p(s' \mid s,\pi_k(s)) V_i^{\pi_k}(s') \right]$$

$$V_{i+1}^{\pi_k}(s) = R(s,\pi_k(s)) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,\pi_k(s)) V_i^{\pi_k}(s') \quad (17)$$

$$\pi_{k+1}(s) = \arg\max_{a \in A(s)} [R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V_i^{\pi_k}(s')]$$

$$\pi_{k+1}(s) = \arg\max_{a \in A(s)} [R(s,a) + \gamma \left[\sum_{s' \in \Gamma(s)} p(s' \mid s,a) V_i^{\pi_k}(s') + \sum_{s' \in \{S \mid \Gamma(s)\}} p(s' \mid s,a) V_i^{\pi_k}(s') \right]]$$

$$\pi_{k+1}(s) = \arg\max_{a \in A(s)} [R(s,a) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,a) V_i^{\pi_k}(s')] \quad (18)$$

Next, we present the proposed Modified PI algorithm.

Algorithm .3: Modified PI algorithm

Specified a threshold ε

$k \leftarrow 0$

Randomly initialize the policy π_k

Repeat

i ← 0

For all s ∈ S

$V_i^{\pi_k}(s) = 0$

Repeat

For all s ∈ S do

$$V_{i+1}^{\pi_k}(s) = R(s,\pi_k(s)) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,\pi_k(s)) V_i^{\pi_k}(s') \quad (19)$$

End for

i ← i+1

While $\left| V_i^{\pi_k}(s) - V_{i-1}^{\pi_k}(s) \right| < \varepsilon$

For all s ∈ S do

$$\pi_{k+1}(s) = \arg\max_{a \in A(s)} [R(s,a) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,a) V_i^{\pi_k}(s')] \quad (20)$$

End for

k ← k+1

While $\pi_k = \pi_{k-1}$

Remark:

- It is easy to establish the convergence and correctness of Algorithm.3.
- There is a transition from state s to state s' if the transition probability $a_{ss'}$ is greater than zero. In general, in each system (Eq. 21) they are $K \times |S|$ transitions where K is the average number of successors to a state.

$$\sum_{s=1}^{|S|} \sum_{\substack{s'=1 \\ a_{ss'}>0}}^{|S|} 1 = K \mid S \mid \quad (21)$$

Then the modified PI algorithm [5] takes, for each iteration, in the order of $K \times |S|$ computations in Eq. 19 and $A \times K \times |S|$ computations in Eq. 20, which represents a reduction in the complexity of the Version 2 of the PI algorithm (Algorithm.2).

5. PARALLELIZING CONCEPTS

Defining the dynamic programming table $H = \{h_{ts}\}_{t=1,2,\ldots}^{s=1,2,\ldots,M}$, which h_{ts} (Table 1) represents the elements of the table and the index t and j the s^{th} entry of the t^{th} row respectively.

Table 1: Dynamic programming table

$h_{t=1,s=1}$	$h_{t=1,s=2}$	…	$h_{t=1,s=M}$
$h_{t=2,s=1}$	$h_{t=2,s=2}$	…	$h_{t=2,s=M}$
…	…	…	…

This table is used to store the value of the value function (Eq. 22 and Table 2)

$$h_{t=i+1,s} = V_{i+1}^{\pi_k}(s) = R(s,\pi_k(s)) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,\pi_k(s)) V_i^{\pi_k}(s')$$

$$h_{t=i+1,s} = R(s,\pi_k(s)) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,\pi_k(s)) V_i^{\pi_k}(s') \quad (22)$$

Table 2: Value function

$h_{t=1,s=1} = V_1^{\pi_k}(1)$	…	$h_{t=1,s=M} = V_1^{\pi_k}(M)$
$h_{t=2,s=1} = V_2^{\pi_k}(1)$	…	$h_{t=2,s=M} = V_2^{\pi_k}(M)$
…	…	…

Afterward, to calculate an optimal strategy (Eq. 23 and Table 3)

$$h_{t=k+1,s} = \pi_{k+1}(s)$$

$$h_{t=k+1,s} = \arg\max_{a \in A(s)} [R(s,a) + \gamma \sum_{s' \in \Gamma(s)} p(s'|s,a) V_i^{\pi_k}(s')] \qquad (23)$$

Table 3: Optimal strategy

$h_{t=1,s=1} = \pi_1(1)$	⋯	$h_{t=1,s=M} = \pi_1(M)$

Noting that the entries of each row in H depend only on the entries in the previous row, which implies that each entry in the t^{th} row of H can be computed in parallel (Table 2 and Table 3).

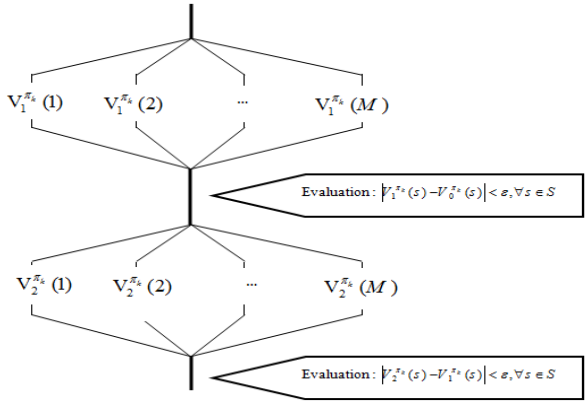

Figure 3: Parallel computing for the value function

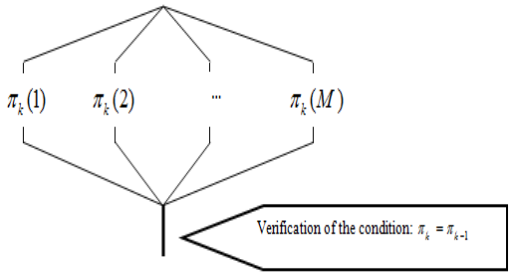

Figure 4: Parallel computing for the optimal policy

The Open Multi-Processing (OpenMP) is an Application Program Interface (API). In comparison to other types of parallelism, the OpenMP can be used to specify shared memory (Figure 5) parallelism in Fortran, C and C++ on most processor architectures and operating systems, Linux, Windows… .

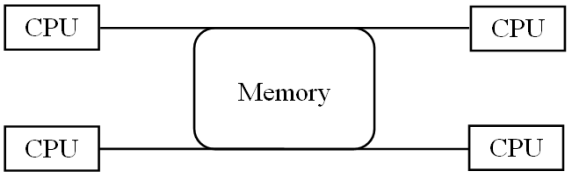

Figure 5: Shared memory

The OpenMP [17] is not a language but it consists of a set of compiler directives which are based on the #pragma compiler directives in C languge. In this programming model, tasks share a common address space, which they read and write asynchronously.

The parallel program begins with a single thread exists called the master thread that will continue to process serially until it encounters a parallel region which is a block of code that must be executed by a team of threads in parallel.

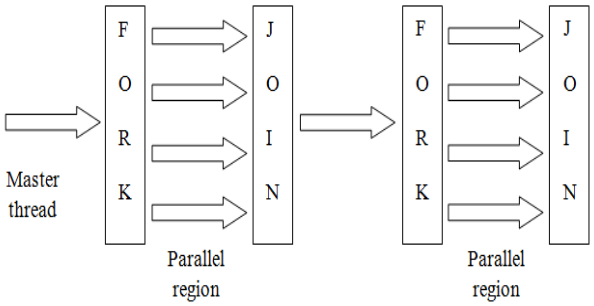

Figure 6: The parallel concept

Thereby, an implementation of the parallel modfied PI algorithm based on OpenMP is proposed, the main idea in this algorithm is to specify a value function and the policy defined as (Eq. 19 Eq. 20) of each state of the problem:

- Input: shared date N = {the set of state S, the set of action A, the transition function T, the reward function R}, the factor γ , .
- Output:
  - The threads create the private variables and coordinate betwen us for access on the shared memory;
  - Calculate the value function and the value of policy in parallel;
  - Repeat the previous step until the stabilization of the strategies.

The Fig. 7 shows the performance results by using the parallelization technique on the Modified PI algorithm. Therefore, noting that the Modified PI algorithm (Algorithm.3), accomplish in high speed up when the number of states is increased in comparison to the classical version (Algorithm.2), and the same effect when the parallelization technique is used on the Modified PI algorithm which is shown in the 4th column.

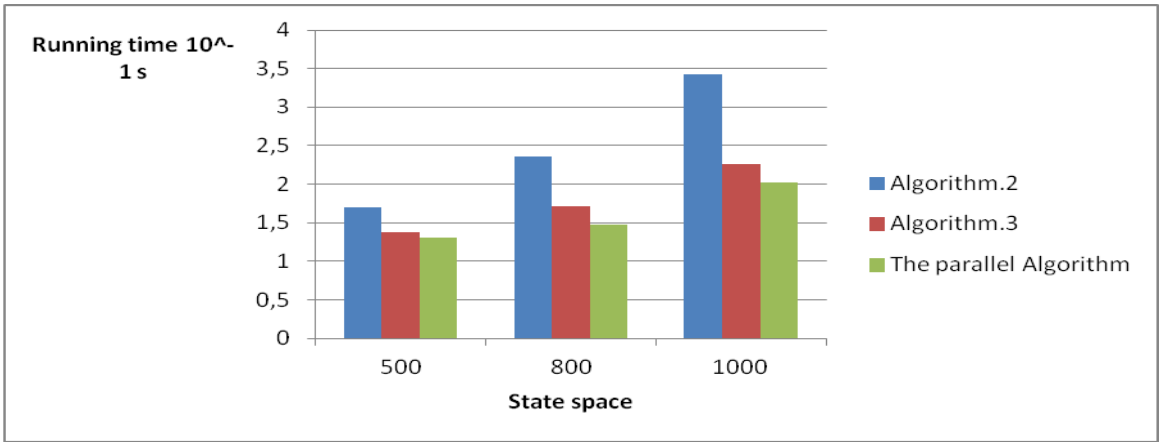

Figure 7: Running time a function of number of state

6. CONCLUSION

The PI algorithm is one of the several standard methods for finding the optimal policies, but it remains intractable for solving large MDP. The research presented in this work is an effort toward developing some modified approaches to solve this class of MDPs. We based on the topology of each state in the associated graph and the parallelization technique with an application program interface named Open-MP.

Moreover, we will hope later on to merge the decomposition method with the techniques of parallelism for more efficiency.

7. ACKNOWLEDGEMENT

In conclusion we are very grateful to my professor Mister DAOUI Cherki for their encouragement, their cooperation, their advices and their guidance in the realization of this work.

8. REFERENCES

[1] Aurélie, B. 2006. Une contribution à la résolution des processus décisionnels de Markov décentralisés avec contraintes temporelles. Caen university. French. <tel-00112014>

[2] Alagoz, O., Maillart, L. M., Schaefer, A. J., & Roberts, M. S. 2007. Determining the acceptance of cadaveric livers using an implicit model of the waiting list. Operations Research, 55(1), 24-36.

[3] Bellman, R. 1957. Dynamic Programming. Princeton University Press.

[4] Benjaafar, S., & ElHafsi, M. 2006. Production and inventory control of a single product assemble-to-order system with multiple customer classes.Management Science, 52(12), 1896-1912.

[5] Chafik, S., & Daoui, C. 2015. A Modified Value Iteration Algorithm for Discounted Markov Decision Processes. Journal of Electronic Commerce in Organizations (JECO), 13(3), 47-57.

[6] Daoui, C., Abbad, M., & Tkiouat, M. 2010. Exact decomposition approaches for Markov decision processes: A survey. Advances in Operations Research,2010.

[7] Dean, T., & Lin, S. H. 1995. Decomposition techniques for planning in stochastic domains. In IJCAI (Vol. 2, p. 3).

[8] Galoppo, N., Govindaraju, N. K., Henson, M., & Manocha, D. 2005. LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware. In Proceedings of the 2005 ACM/IEEE conference on Supercomputing (p. 3). IEEE Computer Society.

[9] Goto, J. H. 1999. A Markov decision process model for airline meal provisioning (Doctoral dissertation, University of British Columbia).

[10] Lewis, M. E., Ayhan, H., & Foley, R. D. 2002. Bias optimal admission control policies for a multiclass nonstationary queueing system. Journal of Applied Probability, 20-37.

[11] Munos, R. Introduction à l'apprentissage par renforcement, from http://researchers.lille.inria.fr/ ~ munos/master-mva/.

[12] Papadimitriou, C. H., & Tsitsiklis, J. N. 1987. The complexity of Markov decision processes. Mathematics of operations research, 12(3), 441-450.

[13] Pavitsos, A., & Kyriakidis, E. G. 2009. Markov decision models for the optimal maintenance of a production unit with an upstream buffer. Computers & Operations Research, 36(6), 1993-2006.

[14] PDMIA, G. (2008). Processus décisionnels de Markov en intelligence artificielle. Edité par Olivier Buffet et Olivier Sigaud, 1.

[15] Preux, P. (2008). Apprentissage par renforcement, GRAPA.

[16] Puterman, M. L. (2014). Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons.

[17] Sato, M. (2002, October). OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors. In Proceedings of the 15th international symposium on System Synthesis (pp. 109-111). ACM.