UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté des Sciences

Département de mathématiques

MEMOIRE DE MAGISTER

En Mathématiques

Spécialité : Recherche opérationnelle

Thème :

RÉSOLUTION DES PROBLEMES D'ORDONNANCEMENT STOCHASTIQUE

PAR

LES PROCESSUS BANDITS

Par

LEMDANI Rachid

Devant le jury composé de :

F. Hannane	Professeur, U.S.D., Blida	Président
A. Derbala	Maître de conférences, U.S.D., Blida	Promoteur
S. Bouroubi	Maître de conférences, U.S.T.H.B., Alger	Examinateur
O. Tami	Chargé de cours, U.S.D., Blida	Examinateur

Blida, 21 septembre 2008

RÉSUMÉ

« N » tâches sont à exécuter sur une seule machine. Dans beaucoup d'applications industrielles, une incertitude sur les temps d'exécution est à considérer. Ils sont supposés aléatoires de distributions de probabilité connues. Ces problèmes d'ordonnancement sont incertains et sont dits stochastiques. Passer d'une tâche à une autre nécessite en général que la machine soit réglée une seconde fois. Ce temps de réglage est parfois inconnu et non apprécié. L'arrivée de l'information sur les tâches, tel leur temps d'exécution, peut ne pas être disponible. Des généralités et définitions de processus sont énoncées succinctement où les notions de la propriété de Markov et du graphe de transition sont évoquées. Notre fonction objectif est défini par l'espérance du coût prévisionnel. Elle vérifie une équation fonctionnelle. Nous exposons une méthode de détermination d'une politique optimale d'exécution de processus. Nous caractérisons l'existence d'une politique optimale et nous donnons des méthodes récentes de sa détermination. Un exemple d'une politique optimale de remplacement d'une machine est fourni. Les algorithmes de Howard [22] et de l' « itération sur les valeurs », des algorithmes efficaces de détermination de politiques optimales, sont exposés en détail. Ils ont été implémentés en utilisant un langage évolué. Des exemples d'application sont fournis. Une comparaison en temps d'exécution des algorithmes est aussi faite. Un graphe, temps d'exécution en fonction des états du processus, est tracé donnant une allure meilleure de celui de Howard. On a étudié le cas où le processus à décision est non markovien. Durant une transition d'un processus, des actions peuvent être entreprises. Nous présentons une politique optimale de résolution, une politique d'indices. On associe à chaque tâche, une priorité ou un indice d'allocation dynamique. Nous exposons cette méthodologie dûe à Glazebrook succinctement. Nous avons implémenté un algorithme de leur détermination. Un exemple d'application est même fourni. Nous avons fourni un logiciel de gestion Log-version 1.0 de détermination de politique optimale pour un processus à 2-décisions markovien, conçu par nos soins. Dans une partie annexe, des programmes sont insérés.

Mots clés : Ordonnancement stochastique, processus bandit, processus à décision, indice d'allocation dynamique, politique d'indices.

ABSTRACT

"N" tasks are to be executed on only one machine. In a lot of industrial applications, of the uncertainties on the processing times execution, the unavailability of information on the tasks, the time of regulating of the machines at the time of toppling over of a task to another is to consider. They are supposed uncertain of known probability distributions. These uncertain scheduling problems are said stochastic. The generalities and definitions of process are expressed succinctly where the notions of the property of Markov and the graph of transition are evoked. Our objective is the expected discounted cost that verifies a functional equation. We expose a method of determination of an optimal policy of execution of process, a continuation of decisions to undertake. We characterize the existence of an optimal policy and we give recent methods of her determination. An example of an optimal policy of replacement of a machine in breakdown is provided. Howard's algorithms [22] and of the" value iteration ", some algorithms efficient of determination of optimal policy are exposed in detail. They have been implemented while using an advanced language. Some application examples are provided. A comparison in processing time of the algorithms is also made. A graph, processing time according to the states of the process, is drawn giving a pace better of the first algorithm. One studied the case where the process to decision is non Markovien. During a transition of a process, some actions can be undertaken. We present an optimal policy of index. One associates to every task, a priority or a dynamic allocation index. We expose this methodology of Glazebrook succinctly. We have a second time implemented an algorithm of their determination. An application example is even provided. We provided software, conceived by our care, Log-Version 1.0, of determination optimal policy for a process to 2 -decisions Markovien. In a supplementary part, some programs are inserted.

Keywords: Stochastic scheduling, bandit process, decision process, dynamic allocation index, policy of index.

ملخص

"N" مهمة تنفذ على آلة. في كثير من التطبيقات الصناعية، مجاهل على أوقات التنفيذ، عدم توفر المعلومات حول المهمات، وقت المعالجة للآلات خلال الانتقال من مهمة إلى أخرى مهام تؤخذ بعين الاعتبار. تفرض عشوائية بتوزيعات احتمالية معروفة. تسمى مسائل التركيب هذه بالعشوائية. عموميات وتعار يف التطور تقدم بالتفصيل مع مفاهيم فرضية مركوف وتمثيل الانتقال. هدفنا هو أمل الكلفة المقدرة التي تحقق معادلة تابعية. نعرض طريقة لتعيين السياسة العامة لتنفيذ التطور، التي هي عبارة عن متتالية قرارات. نقوم بتحليل وجود سياسة عامة ونعطي طرق حديثة لتعيينها. نعطي مثالا لسياسة عامة لتبديل آلة معطلة. برامج هوورد و «العمليات على القيم » ، هما برنامجان ذو فعالية يقدمان لمعالجة السياسات العامة يعرضان بالتفصيل. تمت برمجتهما باستعمال لغة متطورة. مثالان للتطبيق تم تقديمهما. مقارنة بواسطة وقت التنفيذ بالنسبة للبرنامجين تمت أيضا. البيان لوقت التنفيذ بالنسبة لحالات التطور، تم تمثيله، أعطى الأفضلية للبرنامج الأول. قمنا بدراسة الحالة أين التطور غير مركوفي. خلال انتقال هذا التطور يمكن لقرارات أن تستعمل. نقدم سياسة الدلائل الأحسن. نرفق لكل مهمة أسبقية أو دليل متخصص ديناميكي. نعرض هذه الطريقة لغرازبروك بالتفصيل. عالجنا للمرة الثانية برنامجا لتعيينها. أعطينا مثالا تطبيقيا. قمنا أيضا بانجاز وسيلة برمجيات صبغة 0.1 ، لتعيين السياسة العامة لثنائي القرارات المركوفي. عرضنا في نهاية المذكرة جميع البرامج.

REMERCIEMENTS

Le présent travail est pour moi une occasion agréable de pouvoir exprimer nos reconnaissances et gratitude envers Dieu le tout-puissant.

Je tiens à exprimer ma profonde gratitude à Monsieur **Derbala Ali**, mon promoteur, Maître de conférences à l'université Saad Dahleb Blida pour m'avoir accueilli dans son laboratoire. Je le remercie pour son aide, son encouragement à réaliser ce travail et ses orientations fructueuses.

Je remercie sincèrement le professeur **Hannane Farouk**, de l'université Saad Dahleb Blida, pour avoir accepté d'être président de ce jury.

Tous mes remerciements vont aux professeurs :

Bouroubi Sadek, Maître de conférences à l'université Houari Boumediene d'Alger, USTHB.

et **Tami Omar**, Chargé de cours à l'université Saad Dahleb Blida, pour avoir accepter de juger ce travail.

Je n'oublie pas de remercier aussi mes amis Abdelkader, Rabah, Halim, Mustafa, Ameziane, Hassan, Hamlet et toute personne qui m'a aidé à accomplir correctement mes tâches.

Un grand hommage est exprimé à mes parents pour leurs aides morales et matérielles durant tout mon cursus universitaire.

A tous et à toutes un grand merci.

DEDICACE

Je m'incline devant Dieu le tout puissant qui m'a ouvert la porte du savoir et m'a aidé à la franchir.

Je dédie ce modeste travail :

A ma chère et tendre mère, source d'affectation, de courage et d'inspiration qui a autant sacrifié pour me voir atteindre ce jour de soutenance.

A mon père à qui je dois du respect. Qu'il trouve ici l'expression de ma profonde reconnaissance pour tout l'effort et le soutien incessant qui m'a toujours apporté.

A mes frères.

A mes sœurs.

A toute la promotion de post-graduation recherche opérationnelle 2003.

A touts mes amis de la filière mathématiques.

A tous ceux que je porte dans mon cœur.

Rachid.

TABLE DES MATIÈRES

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

INTRODUCTION

Ordonnancer un ensemble de tâches, c'est programmer leur exécution en allouant les ressources requises et fixant leurs dates de débuts d'exécution [94]. Un ordonnancement consiste en une programmation prévisionnelle détaillée des ressources mobilisées dans l'exécution des opérations nécessaires à la production élémentaire de biens sur un horizon de temps. Il se rencontre dans des domaines variés comme la construction, les projets complexes, l'informatique, la production, l'administration,… etc. Les problèmes d'ordonnancement sont donnés par la liste des tâches à ordonnancer avec leurs contraintes d'enchaînement, la durée d'exécution d'une tâche, les ressources qui sont nécessaires à leur exécution et une fonction à optimiser. D'autres contraintes peuvent être imposées. Lorsque l'ensemble des tâches à ordonnancer, à priorité fixé, est connu à l'avance, on parle d'un problème statique, par opposition à un problème dynamique où l'exécution des tâches évolue dans le temps. Lorsque les données du problème sont connues de façon certaine, on parle d'un problème déterministe, par opposition à un problème stochastique où certaines données du problème sont des variables aléatoires.

Dans un atelier, « N » tâches sont à exécuter sur une seule machine. Nous considérons que certaines données de tâches telle leur temps d'exécution peuvent ne pas être déterministes. Elles ne sont pas connues à priori. Les temps d'exécution de tâches sont supposés aléatoires de distributions de probabilité connues. Les problèmes d'ordonnancement où les temps d'exécution de tâches sont incertains sont dits stochastiques. L'aléatoire ou le stochastique peut représenter, l'incertitude de l'Ordonnanceur sur les temps d'exécution, des erreurs sur les mesures des temps d'exécution, les fluctuations aléatoires dans la fonction objectif ou la vitesse de l'opérateur et / ou de la machine, la non homogénéité des tâches nécessitant de différents temps d'exécution et / ou l'aléatoire en temps requis pour le réglage de la machine pour les différentes tâches à exécuter. Reconnaître l'incertitude peut être une tâche difficile. Passer d'une tâche à une autre nécessite en général que la machine soit réglée une seconde fois. Ce temps

de réglage est parfois inconnu et non apprécié. Dans beaucoup d'applications, une incertitude sur les temps d'exécution est à considérer. L'arrivée de l'information sur les tâches, tel leur temps d'exécution, peut ne pas être disponible. Un exemple est que, si on suppose que les tâches ont des temps d'exécution aléatoires et sont à exécuter sur des machines, les tâches sont délivrées aux clients par un chariot. L'arrivée du chariot dans l'atelier n'est pas sous le contrôle du chef d'atelier est elle lui est inconnue. Une tâche qui a terminé son exécution avant l'arrivée du transporteur cause des coûts de stockage et d'attente élevé. Des politiques d'ordonnancement optimales, de type règles d'indices, sont caractérisées par plusieurs auteurs. A l'instant zéro une tâche est choisie pour exécution. En tout instant, une tâche est soit en exécution ou en attente. Le temps d'exécution ou d'attente d'une tâche à l'instant t, forme un processus markovien à deux décisions. Ce dernier modélise les tâches à exécuter dans un ordonnancement stochastique. Il a été étudié par Miller (1968) [25], Ross (1968) [26], (1970) [13], Lippman (1971) [27] et d'autres auteurs. Les processus à décision markovien sont un outil de résolution des problèmes stochastiques. Notre problème est la détermination de politique optimale d'exécution de tâches, une suite de décisions à prendre. Les critères les plus utilisés sont ceux des coûts prévisionnel et moyen.

Au chapitre un, des généralités et définitions sont énoncées succinctement où les notions de la propriété de Markov et du graphe de transition sont évoquées. Notre intérêt, porté sur la fonction objectif, est défini par l'espérance du coût prévisionnel. Elle vérifie une équation fonctionnelle donnée par Ross [13]. Nous exposons une méthode de détermination d'une politique optimale d'exécution de processus [13]. Nous caractérisons l'existence d'une politique optimale et nous donnons des méthodes récentes de sa détermination. Un exemple d'une politique optimale de remplacement d'une machine est fourni. De nombreux algorithmes de détermination de politiques optimales existent dans la littérature. On peut citer l'algorithme de Howard [22] appelé algorithme de « politique d'itération » (de l'anglais Policy iteration) et l'algorithme de l' « itération sur les valeurs » (value iteration algorithm) qui seront exposés en détail dans la suite.

Au chapitre deux, ces deux algorithmes ont été implémentés en utilisant un langage évolué. Plusieurs exemples de problèmes dont les données sont générées aléatoirement sont fournis. Une comparaison en temps d'exécution des

algorithmes est aussi faite. Un graphe, temps d'exécution en fonction des états du processus, est tracé donnant une allure meilleure du premier algorithme.

Dans le troisième chapitre, on a étudié le cas où le processus à décision est non markovien. Dans les *problèmes classiques*, durant le temps de transition du processus d'un état i vers un état j, aucune action n'est entreprise. Dans notre *problème considéré*, durant une transition d'un processus, des actions peuvent être entreprises. Ce que nous sous-entendant par le cas non markovien. Ce genre de problèmes se rencontrent surtout dans les systèmes informatisés multi-tâches où durant l'impression d'un fichier, par exemple, nous pouvons apporter des modifications dans le texte du dit fichier, le sauvegarder, lancer une seconde impression etc.…Au moins trois actions peuvent être entreprises durant cette impression. Nous présentons une politique optimale de résolution de ces problèmes non markoviens, qui est une politique d'indices. On associe à chaque tâche, une priorité ou un indice d'allocation dynamique. En tout instant on exécute la tâche qui a les plus petit indice. En cas d'égalité des indices, nous arbitrerons selon une règle connue.

De la littérature, Glazebrook [09] a étudié ces problèmes et a pu montrer l'existence et la caractérisation de ces indices. Nous exposons cette méthodologie succinctement. Nous avons implémenté un algorithme de leur détermination en utilisant un langage évolué. Un exemple d'application est même fourni.

Au chapitre quatre, nous avons inséré un logiciel de gestion Log-version 1.0 de détermination de politique optimale pour un processus à 2-décisions markovien, conçu par nos soins. Nous voulons mettre à la disposition des étudiants un logiciel qui leur permet de s'initier à cette approche, ses fonctionnalités, ses tests d'intégration, de validation en utilisant le langage Borland C++, et l'environnement de l'interface Borland C++ Builder. Nous allons détailler les étapes de sa conception, l'interface, le développement, l'utilisation du logiciel. Un exemple illustratif d'utilisation est même fourni. Nous terminons ce mémoire par une conclusion générale. Dans une partie annexe, des programmes sont insérés.

CHAPITRE 1
PROCESSUS A DÉCISIONS MARKOVIENS

1. Introduction :

Notre problème d'ordonnancement dans les ateliers manufacturiers consiste en l'exécution de « N » tâches sur « une » machine en utilisant « s » ressources. Dans la suite, les ressources sont supposées disponibles en un nombre illimité. Dans ce cas, la dite machine est elle-même considérée comme une ressource. En tout instant, on choisit selon une règle établie, quelle tâche est à exécuter sur la machine. Les tâches *non exécutées* sont dites *gelées*. L'exécution d'une tâche est évolutive dans le temps. La suite de variables « temps d'exécution d'une tâche i à un instant t » notée $(t_i^e(t))_{t\in \aleph}$ est un processus stochastique, un phénomène qui évolue dans un intervalle de temps. En tout instant, une décision est à prendre, « exécuter » une tâche ou « ne pas l'exécuter ». Les décisions du type arrêter, transporter, contrôler une tâche etc…peuvent être définies. L'aspect markovien découle du fait qu'une décision en tout instant t ne dépendra que de la dernière décision qui à été entreprise à l'instant t-1 et de l'état du processus. Cette évolution dans l'exécution d'une tâche se modélise donc comme un processus à 2-décision markoviens. Au risque de se répéter, nous considérons que certaines données de tâches telle leur temps d'exécution peuvent ne pas être déterministes. Elles ne sont pas connues à priori. Les temps d'exécution de tâches sont supposés aléatoires de distributions de probabilité connues.

Les problèmes d'ordonnancement où les temps d'exécution de tâches sont incertains sont dits stochastiques. L'aléatoire ou le stochastique peut représenter, l'incertitude de l'Ordonnanceur sur les temps d'exécution, des erreurs sur les mesures des temps d'exécution, les fluctuations aléatoires dans la fonction objectif ou la vitesse de l'opérateur et / ou de la machine, la non homogénéité des tâches nécessitant de différents temps d'exécution et / ou l'aléatoire en temps requis pour le réglage de la machine pour les différentes tâches à exécuter.

Des problèmes d'optimisation dans la gestion, l'économie, l'informatique… etc, peuvent se formuler par ces processus. Une bibliographie abondante existe dont les références consultées sont Ross [13], Gittins [21], Doyon [10] et Derbala [1]. Des généralités et définitions sont énoncées succinctement dans la suite où les notions de la propriété de Markov et du graphe de transition sont évoquées. Notre intérêt, la fonction objectif, est défini par l'espérance du coût prévisionnel. Elle vérifie une équation fonctionnelle donnée par Ross [13].

Nous exposons une méthode de détermination d'une politique optimale d'exécution de processus [13]. Nous caractérisons l'existence d'une politique optimale et nous donnons des méthodes récentes de sa détermination. Un exemple d'une politique optimale de remplacement d'une machine est fourni.
Un processus et une tâche sont synonymes.

2. Généralités et définitions

Un processus est observé dans un état x(t) d'un ensemble Ω à un instant discret t = 0, 1... L'ensemble Ω des états possibles est dénombrable et représentera des nombres entiers positifs 0, 1, 2... .
Après avoir observé l'état du processus x(t)=i, une action « a » est choisie de A(i), l'ensemble de toutes les actions possibles qui est supposé fini à l'état i.

(i) Nous encourons un coût $C(i, a)$.

(ii) les prochains états « j » que le processus atteindra sont choisis selon les probabilités de transitions $P_{ij}(a)$.

Propriété 1. (De Markov) :
Si x_t est l'état du processus à l'instant t et a_t l'action choisie, alors la propriété (ii) est équivalente à : $P\{X_{t+1}=j| X_0, a_0, a_1, \ldots\ldots, X_t=i, a_t=a\}=P_{ij}(a)$ appelé *la propriétée de Markov* ou de « *sans mémoire* ». Nous rappelons encore que les coûts supposés bornés et les probabilités de transition sont fonctions seulement du dernier état du processus et de l'action correspondante.
Pour tout état i et action a, il existe un M réel positif tel que $|C(i,a)| < M$.

2.1. Graphe de transition d'un processus

Un graphe de transition pour un processus à décision Markovien est représenté par un ensemble de sommets et un ensemble d'arcs. Un sommet représente un état « i ». Un arc (i, j) représente la liaison ou la transition de l'état « i » vers l'état « j » selon la probabilité p_{ij}. Le graphe de transition entre les états interprète bien la notion de processus à décision. Il nous permet aussi de connaître l'évolution d'un processus en résumant tous les coûts et les probabilités de transition entre les états. On peut lire directement la politique d'exécution du processus de ce graphe.

Exemple

Soit le processus suivant à deux états « 1 » et « 2 ». L'arc qui lie ces deux états représente la probabilité de transition.

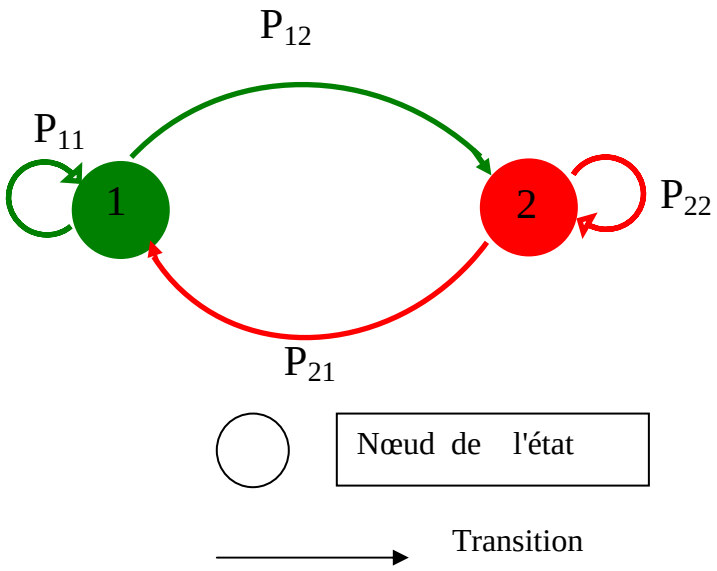


Figure 1.1: Graphe de transition entre 2 états.

Généralisons ce graphe à celui à N_I+1 états. Ce processus à décision qui peut être dans un des états 0, 1, 2,…, | I | à l'instant t=1, 2,…, T successivement tel que I est l'ensemble des états à cardinalité N_I = | I |.

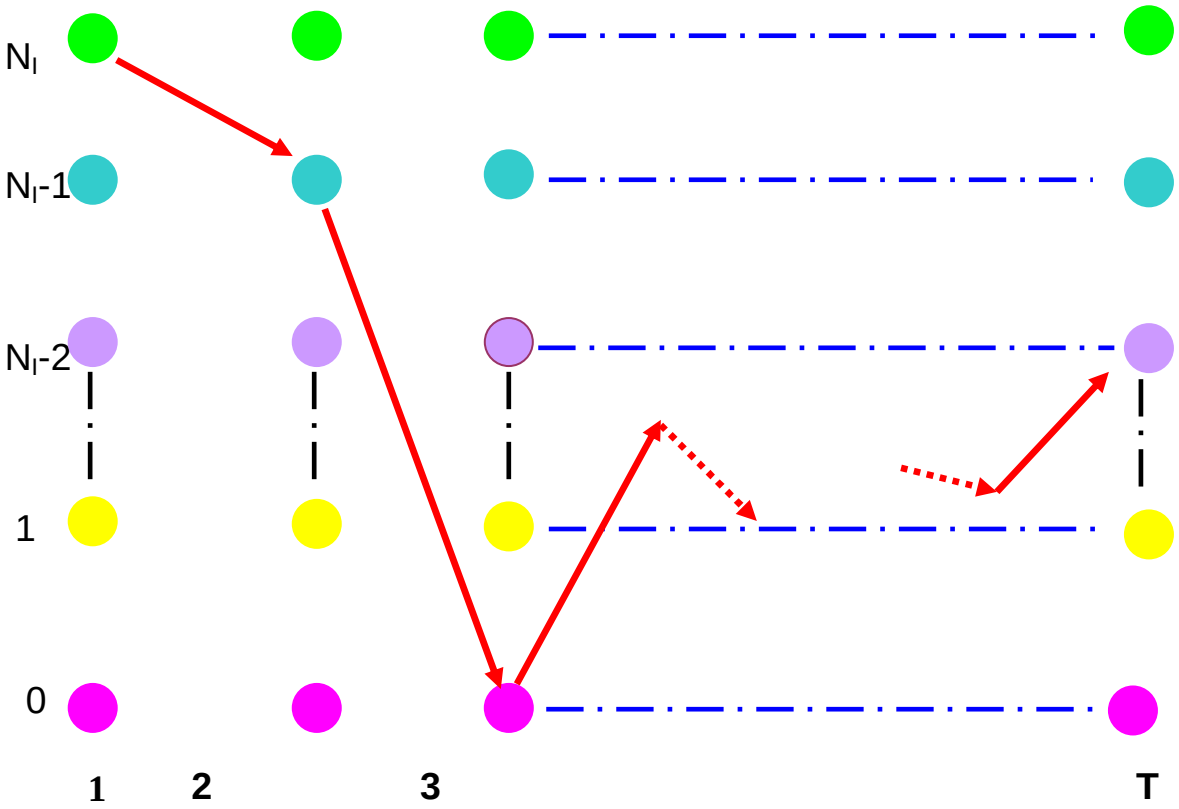


Figure 1.2 : Représentation d'un graphe de transition entre N_l états

Une politique est une suite d'actions à entreprendre, présentée dans ce graphe par la ligne buisée en rouge. Elle se note en général par π, f, etc.

2.2. Notions de politiques stationnaires :

L'ensemble des actions choisies par une certaine politique à l'état i est noté par $A(i)$.

Ainsi, l'action choisie par une politique peut, par exemple, dépendre de l'histoire du processus jusqu' à cet instant. Elle peut être aléatoire dans le sens où elle choisit une action «a » avec une certaine probabilité P_a, a une action de A (i).

Une sous-classe importante parmi toutes les classes des politiques est celle des politiques *stationnaires*.

Une politique *f* est dite *stationnaire* si elle n'est pas aléatoire et l'action choisie à l'instant t dépend seulement de l'état du processus en cet instant.

Une politique *stationnaire* est une fonction de l'espace d'état dans l'espace d'action.

Il découle facilement que si une politique stationnaire *f* est employée, alors la suite des états $\{X_t, t=0,1,2,.....\}$forme une chaîne de Markov avec des probabilités de transition $P_{ij} = P_{ij}[f(i)]$ et ainsi le processus est appelé *processus Markovien de décision.*

Notre but est de déterminer des politiques optimales d'exécution d'un processus pour critère ou fonction objectif donnée.

Les critères les plus utilisés sont l'espérance du coût prévisionnel total (total expected discounted cost), l'espérance du coût total (total expected cost) et le coût moyen (average cost criterion).

3. <u>Espérance du coût prévisionnel</u>

Pour toute politique arbitraire π on définit:

$$V_\pi(i) = E_\pi\left[\sum_{t=0}^{\infty} \alpha^t C(X_t, a_t) \middle| X_0 = i\right], \quad i \geq 0 \qquad (1)$$

Ce critère suppose un facteur prévisionnel α Є]0, 1[, où on espère le minimiser. Il représente l'espérance du coût prévisionnel total encouru quand la politique π est employée et l'état initial est i.

sachant que $\left|\sum_{t=0} \alpha^t C(X_t, a_t)\right| \leq \sum_{t=0} \alpha^t M = M \frac{1}{1-\alpha}$.

$V_\pi(i)$ est ainsi bien définie. La dépendance de α est omise de V_π (i) L'interprétation du facteur prévisionnel α est que les coûts encourus dans les dates futures seront moins importants que ceux octroyés en date aujourd'hui. On prévisionne ce coût de dévaluation ou de dépréciation par un facteur α par unité de temps.

Notons par:

$$V_\alpha(i) = \inf_\pi V_\pi(i), \quad i \geq 0 . \quad i \text{ fixé.}$$

La politique π^*est dite α - optimal si $V_{\pi^*}(i) = V_\alpha(i)$ pour tout $i \geq 0$

L'espérance du coût prévisionnel est minimale pour chaque état initial.

Cette fonction coût optimale V_α vérifie une équation fonctionnelle soit un processus d'exécution de tâches.

Théorème 1 :[25]

Pour tout état « i » d'un processus, la politique optimale d'exécution

$$V_\alpha\ (i) = \min_a \left\{ C(i,\ a) + \alpha \sum_{j=0}^{\infty} P_{ij}\ (a)\, V_\alpha\ (j) \right\},\ i \geq 0 \qquad (2)$$

où $a \in A(i)$, a une action.

Preuve :

Soit π une politique arbitraire quelconque, et supposons quelle choisit l'action *a* à l'instant 0 avec une probabilité P_a , $a \in A(i)$. Alors,

$V_\pi\ (i) = \sum_{a \in A} P_a \left[C(i,\ a) + \sum_{j=0}^{\infty} P_{ij}\ (a)\, W_\pi\ (j) \right]$ où $W_\pi(j)$ est l'espérance du coût prévisionnel encouru à partir de l'instant 1, sachant que π est utilisée et l'état à l'instant 1 est j. De l'actualisation, cette espérance est au moins égale à « α » fois la valeur optimale de l'état j, état « suivant » de i.

D'où $W_\pi\ (j) \geq \alpha V_\alpha\ (j)$

Il découlera que

$$V_\pi(i) \geq \sum_{a \in A} P_a \left[C(i,a) + \alpha \sum_{j=0}^{\infty} P_{ij}(a) V_\alpha(j) \right]$$

$$\geq \sum_{a \in A} P_a \min_a \left\{ C(i,a) + \alpha \sum_{j=0}^{\infty} P_{ij}(a) V_\alpha(j) \right\} \qquad (3)$$

$$= \min_a \left\{ C(i,a) + \alpha \sum_{j=0}^{\infty} P_{ij}\ (a) V_\alpha\ (j) \right\}$$

La politique π étant arbitraire, l'inégalité (3) implique que :

$$V_\alpha(i) \geq \min_a \left\{ C(i,a) + \alpha \sum_{j=0}^{\infty} P_{ij}(a) V_\alpha(j) \right\} \qquad (4)$$

Pour montrer l'autre sens de l'inégalité, soit a_0 une action qui réalise l'égalité.

Soit $C(i,a_0) + \alpha \sum_{j=0}^{\infty} P_{ij}(a_0) V_\alpha(j) = \min_a \left\{ C(i,a) + \alpha \sum_{j=0}^{\infty} P_{ij}(a) V_\alpha(j) \right\}$ (5).

soit π la politique qui choisit a_0 à l'instant t; et si le prochain état est j, alors le processus est vu comme son origine est j suivant la politique π_j qui est tel que :

$V_{\pi j}(j) \leq V_\alpha(j) + \varepsilon$. Par conséquent,

$$V_\pi(i) = C(i,a_0) + \alpha \sum_{j=0}^{\infty} P_{ij}(a_0) V_{\pi j}(j)$$

$$\leq C(i,a_0) + \alpha \sum_{j=0}^{\infty} P_{ij}(a_0) V_\alpha(j) + \alpha\varepsilon$$

Sachant que $V_\alpha(i) \leq V_\pi(i)$, il découle que :

$$V_\alpha(i) \leq C(i,a_0) + \alpha \sum_{j=0}^{\infty} P_{ij}(a_0) V_\alpha(j) + \alpha\varepsilon$$

et, de (5), on obtient :

$$V_\alpha(i) \leq \min_a \left\{ C(i,a) + \alpha \sum_{j=0}^{\infty} P_{ij}(a) V_\alpha(j) \right\} + \alpha\varepsilon \qquad (6).$$

D'où le résultat énoncé.

Soient un processus à l'état « i » et $B(i)$ l'ensemble de toutes les fonctions bornées à valeurs réelles sur l'espace d'état d'un processus.

Sachant que les coûts sont bornés alors V_π est aussi une fonction bornée pour toute politique π, d'où $V_\pi(i) \in B(i)$.

Pour toute politique stationnaire *f*, on définit une application transformation T_f de toute application bornées en une application bornée par :

$$T_f : B(I) \to B(I)$$
$$u \mapsto T_f u$$

tel que $$(T_f u)(i) = C[i, f(i)] + \alpha \sum_{j=0}^{\infty} P_{ij}[f(i)]u(j) \qquad (7).$$

De sa définition, $T_f u$ est ainsi bornée et donc $T_f u \in B(I)$.

L'interprétation de $T_f u$ est que sa valeur en i représentant l'espérance du coût si on utilise *f* mais on terminera après une période et on encourt un coût final $\alpha u(j)$ quand l'état final est j.

Notons T_f par T_f^1 et pour tout n > 1 $T_f^n = T(T_f^{n-1})$.

Définition 3

Pour tout couple de fonctions (u, v) de $B(I)$, on dit que :

- $u \le v$ si $u(i) \le v(i)$, pour $i \ge 0$. Similairement, on définit l'égalité de deux fonctions.
- Pour $u_n, u \in B(I)$, on dira que $(u_n)_{n\in \mathbb{N}} \to u$ si $u_n(i) \to u(i)$ uniformément en i, pour $i \ge 0$.

Le lemme suivant établi quelques propriétés importantes de la transformée T_f.

Lemme 1 :[25]

Pour $u, v \in B(I)$ et f une politique stationnaire on a les trois résultats suivants :

(i) $u \le v \Rightarrow T_f u \le T_f v$

(ii) $T_f V_f = V_f$

(iii) $T_f^n u \to V_f$ pour tout $u \in B(I)$.

Preuve:

- La partie (i) découle directement de la définition de T_f .
- La partie (ii) est l'écriture de l'expression:

$$\left(T_f V_f\right)(i)=V_f(i)=C[i,f(i)]+\alpha\sum_{j=0}^{\infty}P_{ij}[f(i)]V_f(j)$$

ce qui est vu comme vraie en conditionnant sur l'état à l'instant 1 que l'action est $f(i)$.

- Pour (iii) nous noterons que :

$$\left(T_f^2 u\right)(i)=C[i,f(i)]+\alpha\sum_{j=0}^{\infty}P_{ij}[f(i)]\left(T_f u\right)(j)$$

$$=C[i,f(i)]+\alpha\sum_{j=0}^{\infty}P_{ij}[f(i)]\left[C[j,f(j)]+\alpha\sum_{k=0}^{\infty}P_{jk}[f(j)]u(k)\right]=$$

$$C[i,f(i)]+\alpha\sum_{j=0}^{\infty}P_{ij}[f(i)]c[j,f(j)]+\alpha^2\sum_{j=0}^{\infty}\sum_{k=0}^{\infty}P_{ij}[f(i)]P_{jk}[f(j)]u(k)$$

T_f^2 représente l'espérance du coût si on utilise la politique f mais on terminera après deux périodes et on encourt un coût final $\alpha^2 u$. Par une récurrence on montre alors que $T_f^n u$ représente l'espérance du coût si on utilise f pour n étapes et on encourt un coût final $\alpha^n u$. Sachant que $\alpha\prec 1$ et u est bornée, le résultat en découle facilement.

Nous sommes maintenant en mesure de démontrer le théorème important caractérisant de la politique optimale.

Théorème 2 : [25]

Soit f_α une politique stationnaire, laquelle, quand le processus est dans l'état i, sélectionne l'action (ou une action) minimisant le coté droit de l'égalité (2), soit $f_\alpha(i)$ est tel que

$$C[i, f_\alpha(i)] + \alpha \sum_{j=0}^{\infty} P_{ij}[f_\alpha(i)] V_\alpha(j) = \min_a \left\{ C(i,a) + \alpha \sum_{j=0}^{\infty} P_{ij}(a) V_\alpha(j) \right\}, i \geq 0$$

alors $V_{f_\alpha}(i) = V_\alpha(i)$ pour tout $i \geq 0$.

Par conséquent, f_α est α-optimal.

Preuve:

En appliquant l'application T_{f_α} à V_α, on obtient :

$$\begin{aligned}
\left(T_{f_\alpha} V_\alpha\right)(i) &= C[i; f_\alpha(i)] + \alpha \sum_{j=0}^{\infty} P_{ij}[f_\alpha(i)] V_\alpha(j) \\
&= \min_a \left\{ C(i,a) + \alpha \sum_{j=0}^{\infty} P_{ij}(a) V_\alpha(j) \right\} \\
&= V_\alpha(i)
\end{aligned}$$

où la dernière équation suit du théorème (1). Par conséquent: $T_{f_\alpha} V_\alpha = V_\alpha$

ce qui implique que :

$$T_{f_\alpha}^2 V_\alpha = T_{f_\alpha}\left(T_{f_\alpha} V_\alpha\right) = T_{f_\alpha} V_\alpha = V_\alpha$$

Par récurrence, on peut établir que $T_{f_\alpha}^n V_\alpha = V_\alpha$ pour tout n.

Lorsque n tend vers l'infini on trouve que $V_{f_\alpha} = V_\alpha$

Ainsi, une politique α-optimale existe, elle peut être stationnaire et est déterminée par l'équation fonctionnelle (2). Si on peut déterminer la fonction espérance de coût total V_α, alors la politique stationnaire laquelle, quand à l'état i, sélectionne l'action (ou une action s'il y en a plusieurs) minimisant

$$C(i,a) + \alpha \sum_{j=0}^{\infty} P_{ij}(a) V_\alpha(j) \text{ est } \alpha\text{- optimal}$$

Supposons qu'on a évalué la fonction espérance de coût V_α d'une politique stationnaire f et soit f^* la politique stationnaire qui dans l'état i sélectionne l'action minimisant :

$$C(i,a)+\alpha\sum_{j=0}^{\infty}P_{ij}(a)V_f(j) \text{ i.e. } f^*(i) \text{ est tel que}$$

$$C\left[i, f^{\bullet}(i)\right]+\alpha\sum_{j=0}^{\infty}P_{ij}\left[f^{\bullet}(i)\right]V_f(j)=\min_a\left\{C(i,a)+\alpha\sum_{j=0}^{\infty}P_{ij}(a)V_f(j)\right\}, i\geq 0 \quad (8)$$

A combien f^* est comparativement meilleure que f ? Le corollaire ci-dessus montre que f^* est au plus aussi bonne que f . On montrera par la suite que f^* est strictement meilleure que f pour au moins un état initial, ou aussi f est α-optimale.

Corollaire 1 :[25]

Pour toute politique stationnaire f^* strictement meilleure que f et pour tout état $i\in\Omega$ on a :

$$V_{f^*}(i) \leq V_f(i)$$

Preuve:

Soit la transformée

$$\left(T_{f^{\bullet}}V_f\right)(i)=C\left[i,f^{\bullet}(i)\right]+\alpha\sum_{j=0}^{\infty}P_{ij}\left[f^{\bullet}(i)\right]V_f(j)$$

$$\leq C[i,f(i)]+\alpha\sum_{j=0}^{\infty}P_{ij}[f(i)]V_f(j) \quad (9)$$

$$=V_f(i) \quad (10)$$

où (9) découle de la définition de $f^{\bullet}$, et (10) découle du théorème (1), d'où $T_{f^{\bullet}}V_f \leq V_f$

et appliquant $T_{f^{\bullet}}$ aux deux côtés et donne par la monotonie de $T_{f^{\bullet}}$

et utilisant le lemme (1)(i) on obtient $T_{f^{\bullet}}^2 V_f \leq T_{f^{\bullet}}V_f \leq V_f$ et par induction

$$T_{f^{\bullet}}^n V_f \leq V_f$$

A la limite et en utilisant, et en employant le lemme (2) (iii), on obtient le résultat désiré.

La technique de démarrer avec une politique initiale et d'utiliser le corollaire (1) pour l'améliorer, et reaméliorer la nouvelle politique, etc., est connu comme « l'algorithme de l'amélioration de la politique ».

3.2. Les fonctions contractions

comment détermine t'on V_α ?

On a besoin des résultats préliminaires.

Pour toute fonction $u \in B(I)$, définissons la norme de u notée $\|u\| = \sup_{i \geq 0} |u(i)|$

Définition 4

Une application $T: B(I) \to B(I)$ est appelée une « contraction » si :

$\exists\ \beta \prec 1$ et $\forall\ u, v \in B(I)$ on a $\|T_u - T_v\| \leq \beta \|u - v\|$, [$u - v$ est la fonction à valeur en i de $u(i) - v(i)$]

Nous rappelons à dessous le théorème du point fixe.

3.3. Théorème 2 [25]

Si $T: B(I) \to B(I)$ est une contraction, alors il existe une unique fonction

$g \in B(I)$ telle que $T_g = g$.

$\forall\ u \in B(I)$, $T^n u \underset{n \to \infty}{\to} g$ c-à-d : la fonction de contraction T_α converge vers la fonction unique g. Dans l'ordre d'appliquer ce théorème, définissons l'application ;

$$T_\alpha : B(I) \to B(I)$$

par : $$(T_\alpha u)(i) = \min_a \left\{ C(i,a) + \alpha \sum_{j=0}^{\infty} P_{ij}(a)\, u(j) \right\} \quad i \geq 0 \qquad (11).$$

Par le théorème (1), il suit que : $T_\alpha V_\alpha = V_\alpha$

Si on peut montrer que T_α est une contraction, alors V_α est l'unique solution de (2) est que $V\alpha$ est obtenue (au moins à la limite) par l'application successive de T_α à n'importe quelle fonction $u \in B(I)$.

Elle est connue comme la méthode des « approximations successives ».

Théorème 3 : [25]

L'application T_α définie par l'équation (11) est une « contraction ».

Preuve :

pour tout u, $v \in B(I)$, calculons la différence

$$\begin{aligned}(T_\alpha u)(i)-(T_\alpha v)(i) &= \min_a \left\{ C(i,a)+\alpha \sum_{j=0}^{\infty} P_{ij}(a)\, u(j) \right\} \\ &\quad - \min_a \left\{ C(i,a)+\alpha \sum_{j=0}^{\infty} P_{ij}(a)\, v(j) \right\} \\ &= \min_a \left\{ C(i,a)+\alpha \sum_{j=0}^{\infty} P_{ij}(a)\, u(j) \right\} \\ &\quad - C\left(i, \bar{a}\right)-\alpha \sum_{j=0}^{\infty} P_{ij}\left(\bar{a}\right) v(j)\end{aligned} \qquad (12).$$

où $\bar{a}$ est l'action qui vérifie

$$C\left(i, \bar{a}\right)+\alpha \sum_{j=0}^{\infty} P_{ij}\left(\bar{a}\right) v(j) = \min_a \left\{ C(i, a)+\alpha \sum_{j=0}^{\infty} P_{ij}(a)\, v(j) \right\}$$

et par conséquence, par (12), on obtient :

$$\begin{aligned}(T_\alpha u)(i)-(T_\alpha v)(i) &\le \alpha \sum_{j=0}^{\infty} P_{ij}\left(\bar{a}\right) u(j)-\alpha \sum_{j=0}^{\infty} P_{ij}\left(\bar{a}\right) v(j) \\ &= \alpha \sum_{j=0}^{\infty} P_{ij}\left(\bar{a}\right) [u(j)-v(j)] \\ &\le \alpha \sum_{j=0}^{\infty} P_{ij}\left(\bar{a}\right) \sup_j [u(j)-v(j)] \\ &\le \alpha \| u-v \|\end{aligned} \qquad (13).$$

par l'égalité (13) on obtient que:

$$\sup_{i\geq 0}\{(T_\alpha u)(i)-(T_\alpha v)(i)\}\leq \alpha\|u-v\| \qquad (14).$$

En inversant l'ordre de u et v dans l'inégalité (14),

$$\sup_{i\geq 0}\{(T_\alpha v)(i)-(T_\alpha u)(i)\}\leq \alpha\|u-v\| \qquad (15).$$

De (14) et (15),

$$\sup_{i\geq 0}|(T_\alpha u)(i)-(T_\alpha v)(i)|\leq \alpha\|u-v\|$$

ou : $\|T_\alpha u-T_\alpha v\|\leq\alpha\|u-v\|$

D'où T est une contraction car α≤1.

Corollaire 2 :[25]

La fonction coût optimale V_α est l'unique solution de

$$V_\alpha(i)=\min_a\left\{C(i,a)+\alpha\sum_{j=0}^{\infty}P_{ij}(a)V_\alpha(j)\right\},\quad i\geq 0$$

et $\forall\, u\in B(I)$, $T_\alpha^n u\to V_\alpha$ quand $n\to\infty$.

On utilisera le théorème 3 et le théorème du point fixe pour le montrer.

Remarque

a) Un choix usuel particulier est de considérer que *u*=0, la fonction nulle, définie par $u(i)=0$ pour tout état i.

si $V_\alpha(i,n)=\left(T_\alpha^n\ 0\right)(i)$

alors $V_\alpha(i,n)$ est égale à l'espérance du coût prévisionnel minimale pour un problème de « n » périodes. Souvent des résultats importants concernant $V_\alpha(i)$ peut être prouvés par les prouver premièrement pour $V_\alpha(i,n)$ et alors faire tendre n à l'infini. On peut montrer que $V_\alpha(i)$ est monotone en i en montrant que c'est vrai pour $V_\alpha(i,n)$.

La similarité de la structure de $V_\alpha(i)$ et $V_\alpha(i,n)$ sera indiquée dans le prochain paragraphe.

b) Le corollaire (2) nous permet de montrer que pour la politique de technique d'amélioration, une nouvelle politique est strictement meilleure que l'ancienne, ou sinon elles sont les deux optimales. Si $V_{f^\bullet} = V_f$, alors par (8) et le lemme (1)(ii) V_f satisfait l'équation d'optimalité (2) et par l'unicité $V_f = V_\alpha$.

c) Il est aussi facile de prouver que T_f est une contraction par l'utilisation du lemme (2) (ii), V_f est l'unique solution de

$$V_f(i) = C(i, f(i)) + \alpha \sum_{j=0}^{\infty} P_{ij}[f(i)] V_f(j)$$

d) Les résultats établis pour un espace d'états dénombrable, s'étendent à un ensemble non dénombrable. La supposition de l'espace d'action fini est essentielle. Aussi $C(i, a)$ n'est pas forcément un coût actuel, mais peut représenter l'espérance du coût si l'action a est prise à l'état i.

4. Exemple du **« Modèle de remplacement de machine ››**

Une machine peut être dans un des états 0, 1, 2, …

Supposons qu'au début de chaque jour, l'état de la machine est noté et une décision est prise si ou non son remplacement sera fait. Si oui on suppose que la machine est instantanément remplacée par une nouvelle machine dont son état est 0.

Le coût de remplacement d'une machine est noté par R, et un coût de maintenance C (i) est encouru chaque jour que la machine est à l'état *i*.

P_{ij} : représente la probabilité que la machine à l'état *i* au début d'un jour sera à l'état *j* au début du lendemain.

C'est un processus Markovien à deux actions « remplacement » et « non remplacement » de la machine.

Les coûts à une étape et les probabilités de transition sont donnés par :

$C(i, 1) = R + C(0), \quad C(i, 2) = C(i), \quad i \geq 0.$

$P_{ij}(1) = P_{0j}, \quad P_{ij}(2) = P_{ij}, \quad i \geq 0.$

On imposera les suppositions suivantes sur les coûts et les probabilités de transition :

(i) $\{C(i),\ i \geq 0\}$ est une suite croissante de coûts bornés.

Ce qui est traduit par le coût de la maintenance est une fonction croissante de l'état de la machine.

(ii) $\sum_{j=k}^{\infty} P_{ij}$ est croissante en *i*, pour chaque k ≥ 0.

La probabilité de transition dans n'importe quel block d'états {k, k+1, …} est une fonction croissante du présent état.

Dans l'ordre de déterminer la structure de la politique optimale, on a besoin des lemmes suivants.

Lemme 2 : [25]

La condition ii) implique que pour toute fonction croissante $h(i)$, la fonction $\sum_{j=0}^{\infty} P_{ij}\, h(j)$ est aussi croissante en i.

Lemme 3 :[25]

Sous les conditions i) et ii), $V_\alpha(i)$ est croissante en i.

Preuve :

Soit $V_\alpha(i,1) = \min\{R + C(0);\ C(i)\}$ et pour tout $n \succ 1$,

$$V_\alpha(i,\ n) = \min\left\{R + C(0) + \alpha \sum_{j=0}^{\infty} P_{0j} V_\alpha(j,\ n-1);\quad C(i) + \alpha \sum_{j=0}^{\infty} P_{ij} V_\alpha(j,\ n-1)\right\}$$

de i) $V_\alpha(i,1)$ est croissante en i, et si on suppose que $V_\alpha(i, n-1)$ est croissante en i, alors $V_\alpha(i, n)$ est aussi croissante en i par le lemme (7).

Par récurrence, on a que $V_\alpha(i, n)$ est croissante en i pour tout n, et que $V_\alpha(i) = \lim_{n\to\infty} V_\alpha(i, n)$ est que croissante.

La politique optimale est déterminée par le théorème 4 ci dessous.

Théorème 4 [25]

Sous les suppositions i) et ii), il existe un entier i^*, i^* fini, tel qu'une politique α-optimale pour $i \succ i^*$ remplace et ne remplace pas pour $i \leq i^*$.

Preuve :

Par le théorème (1), on a :

$$V_\alpha(i) = \min\left\{ R + C(0) + \alpha \sum_{j=0}^{\infty} P_{0j} V_\alpha(j);\ C(i) + \alpha \sum_{j=0}^{\infty} P_{ij} V_\alpha(j) \right\}. \quad (16)$$

soit : $i^* = \max\left\{ i \ : \ C(i) + \alpha \sum_{j=0}^{\infty} P_{ij} V_\alpha(j) \ \leq \ R + C(0) + \alpha \sum_{j=0}^{\infty} P_{ij} V_\alpha(j) \right\}$

Maintenant, par les deux derniers lemmes, il découle que $C(i) + \alpha \sum_{j=0}^{\infty} P_{ij} V_\alpha(j)$ est croissante en i, et donc par (16),

$$V_\alpha(i) = \begin{cases} C(i) + \alpha \sum_{j=0}^{\infty} P_{ij} V_\alpha(j) & pour \quad i \leq i^* \\ R + C(0) + \alpha \sum_{j=0}^{\infty} P_{0j} V_\alpha(j) & pour \quad i \succ i^* \end{cases}$$

Le résultat découle du théorème (3).

6. Conclusion :

Un processus à 2-décisions markoviens, l'exécution d'une tâche à un instant t, sont défini. L'existence et la caractérisation d'une politique optimale est fournie par des équations fonctionnelles ci-dessus. Des algorithmes de détermination d'une politique optimale tels que celui de Howard existent dans la littérature. Ils seront l'objet du chapitre 2 qu'on développera et où 2 algorithmes seront donnés.

CHAPITRE 2
DEUX ALGORITHMES DE DÉTERMINATION DE POLITIQUES OPTIMALES POUR UN PROCESSUS A DÉCISIONS MARKOVIEN

1. Introduction

L'étude des processus à décisions Markoviens qu'on notera dans la suite PDM a intéressé pendant longtemps beaucoup d'auteurs tels que Howard [23], Philippe [22], Alani [1], …

Le problème consiste en la détermination d'une politique optimale d'exécution d'un tel processus. De nombreux algorithmes de détermination de politiques optimales existent dans la littérature. On peut citer l'algorithme de Howard [23] appelé algorithme de « politique d'itération » (de l'anglais Policy iteration) et l'algorithme de l' « itération sur les valeurs » (value iteration algorithm) qui seront exposés en détail dans la suite. Ils ont été implémentés en utilisant un langage évolué. Plusieurs exemples de problèmes dont les données sont générées aléatoirement sont fournis. Une comparaison en temps d'exécution des algorithmes est aussi faite. Un graphe, temps d'exécution en fonction des états du processus, est tracé donnant une allure meilleure du premier algorithme.

La résolution d'un problème d'ordonnancement modélisé par un PDM consiste à optimiser une fonction objectif, espérance du coût $E\left(\sum_{t=0}^{T} \alpha^t \times C_t(x_t, a_t)\right)$, où c_t représente à un instant t, le coût d'exécution du processus à l'état x_t sachant que l'action a_t a été entreprise. α est un facteur prévisionnel, une valeur réelle, choisie dans l'intervalle]0,1[.

Un processus est dit à *horizon infini* si le paramètre temps est infini, sinon il est dit à *horizon fini*.

2. Algorithme de Howard [23]

Soient un processus à 2-décisions qui est observé à l'instant t = 0, 1, 2, … à un état i, i = 1, 2, 3,….

Ω l'ensemble des états possibles du processus qui est supposé fini ou dénombrable. Pour l'exécution de ce processus, une politique π définie par une suite d'actions est employée.

Le coût total d'exécution d'un processus sous une politique π est l'expression :

$$V_\pi(i) = c(i, \pi(i)) + \alpha \sum_{j \in I} P(i, \pi(i), j)\, V_\pi(j)$$

Notre objectif est de déterminer une politique d'exécution de tel processus notée π^* qui minimise cette fonction de coût.

L'idée de cet algorithme est de considérer à l'étape initiale, une politique arbitraire dont on détermine sa valeur. Par une procédure, de permutation d'actions, on construit une nouvelle politique strictement meilleure en coût que la précédente. Tant que l'on arrive à produire une politique strictement meilleure, on répétera cette procédure d'amélioration.

Si au bout d'un nombre d'itérations que l'on fixe au départ, aucune politique ultérieure n'améliore la fonction coût, la politique actuelle est déclarée optimale.

Dans la suite, la politique d'exécution du processus est supposée déterministe. Une difficulté dans la détermination de la politique optimale peut survenir dans le cas où les actions sont aléatoires.

Algorithme 1

Entrée :
Déclarer :
- un PDM : (Ω, A, P, C, α)
- un seuil de précision ε

Initialiser une politique π_0 aléatoirement
k ← 0
Répéter
Initialiser V_0^{π} aléatoirement
it←0
Répéter
pour tout état $i \in I$ faire

$$V_{it+1}^{\pi k} \leftarrow \sum_{j \in I} P\left(i, \pi_k(i), j\right)\left[C\left(i, \pi_k(i), j\right) + \alpha V_{it}^{\pi k}(j)\right]$$

fin pour
it← it+1
Jusque $\left\| V_{it}^{\pi k} - V_{it-1}^{\pi k} \right\| \leq \varepsilon$
pour tout état $i \in I$ faire

$$\pi_{k+1}(i, a) \leftarrow \operatorname{argmin}_{a \in A(i)} \sum_{j \in I} P(i, a, j)\left[C(i, a, j) + \alpha V^{\pi_k}(j)\right]$$

Fin pour
k←k+1
Jusque $\pi_k = \pi_{k-1}$
Sortie : la politique optimale π^*

Figure 2.1 : Schéma de l'algorithme d'itération de la politique.

2.1. Justification de l'algorithme 1

Nous rappelons qu'une politique stationnaire (markovienne) f est une fonction associant à chaque état i une action $a_i \in A(i)$ à entreprendre chaque fois que le processus se trouve dans l'état i. Par la suite, on la désignera tout simplement par politique. Etant donné une politique f, on cherchera à construire une nouvelle politique f^* apportant éventuellement, une amélioration quel que soit l'état initial i. Une manière de déterminer la politique optimale π^* consiste à chercher, pour chaque $i \in I$, l'action a^* correspondant à l'argument de la solution du problème.

L'idée fondamentale de cet algorithme est qu'il choisit au début une politique quelconque π_0 et qu'on va l'améliore par une autre π_k en changeant les actions de cette dernière ainsi de suite en passant toute les actions puis; on calcule à chaque fois la valeur de coût correspondante $V_{\pi(k)}$, on calcule la déférence $\| V_{\pi(k)} - V_{\pi(k-1)} \|$. Si cette valeur est inférieure ou égale à ε, alors la valeur minimale est $V_{\pi(k)}$ et on prend la politique optimale $\pi(k)$ sinon on prend $\pi(k-1)$ comme une politique optimale.

L'évaluation de la politique se fait avec une précision fixée ϵ. Le choix de ϵ est prépondérant dans la convergence afin d'éviter un phénomène de cyclage ou de non-amélioration de la politique [3].

L'algorithme d'itération de la politique converge vers π^* au bout d'un nombre fini d'itérations.

Exemple 1

Soit un processus à 3 états et à 3 actions. A l'instant t = 1, 2, …, le processus est supposé successivement à l'état 1, 2, 3. On encourt un coût c (i, a) pour toute action a, tel que : a = 1, 2, 3. On calcule la valeur du coût *v* (i, a) à chaque état i et pour toute actions a=1, 2, 3 par l'algorithme d'itération de la politique. A chaque fois, on détermine l'action qui minimise la fonction du coût *v* à chaque état. Les coûts et les probabilités induits à chaque état sous les différentes actions sont récapitulés ci-dessous.

Tableau 2.1 : Valeurs des coûts en fonction des actions en chaque état et des probabilités de transitions.

i	a_i	$P_{i1}(a_i)$	$P_{i2}(a_i)$	$P_{i3}(a_i)$	$C(i, a_i)$
1	1	0.5	0.3	0.2	5
	2	0.4	0.6	0	-1
	3	0	0.5	0.5	5
2	1	0.5	0.5	0	10
	2	0.5	0.1	0.4	0
	3	0.3	0.3	0.4	-1
3	1	0	0.3	0.7	0
	2	0.8	0.1	0.1	2
	3	0.2	0.5	0.3	10

L'exemple s'est déroulé sur un programme élaboré par nos soins. Par une saisie à la main, on initialise toutes les valeurs des coûts et des probabilités de transitions sous les différentes actions.

Les coûts et les probabilités de transitions sont donnés respectivement par un tableau à deux dimensions de cardinalité ׀Ω׀×׀A׀ et ׀Ω׀×׀A׀×׀Ω׀.
Un exemple de fenêtre du programme de cette procédure est insérée ci -dessous.

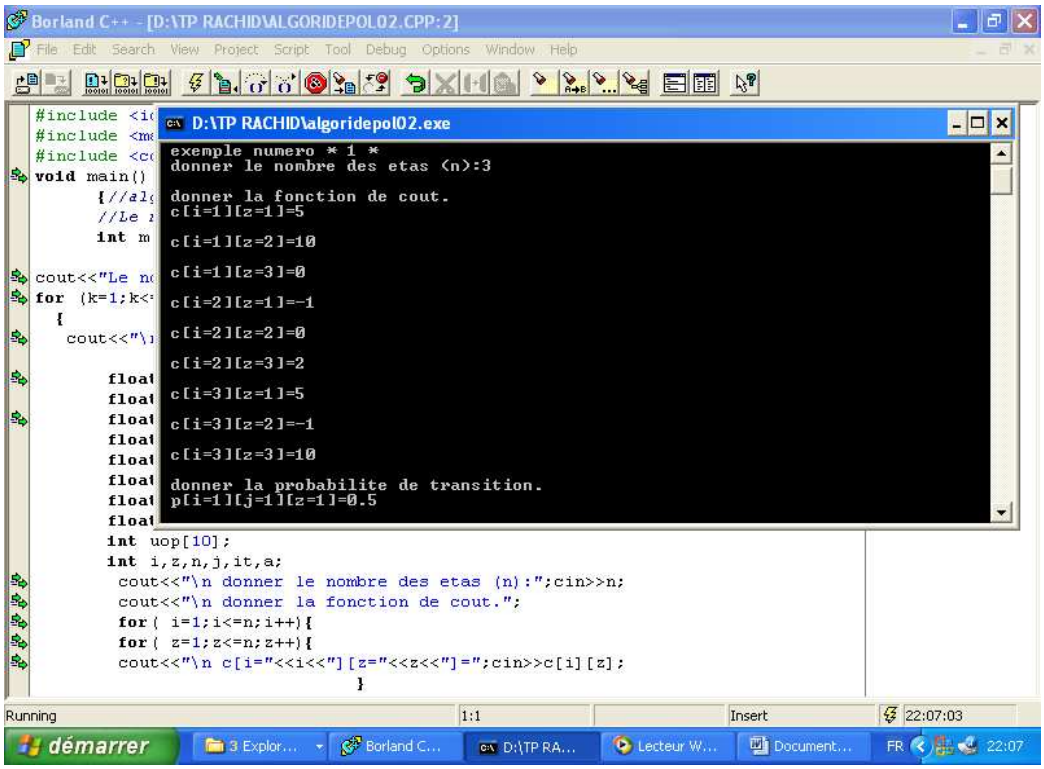


Figure 2.2 : Exécution de l'algorithme 1.

Après 10 secondes, une politique optimale s'affiche. La fenêtre suivante montre le fonctionnement de cette opération.

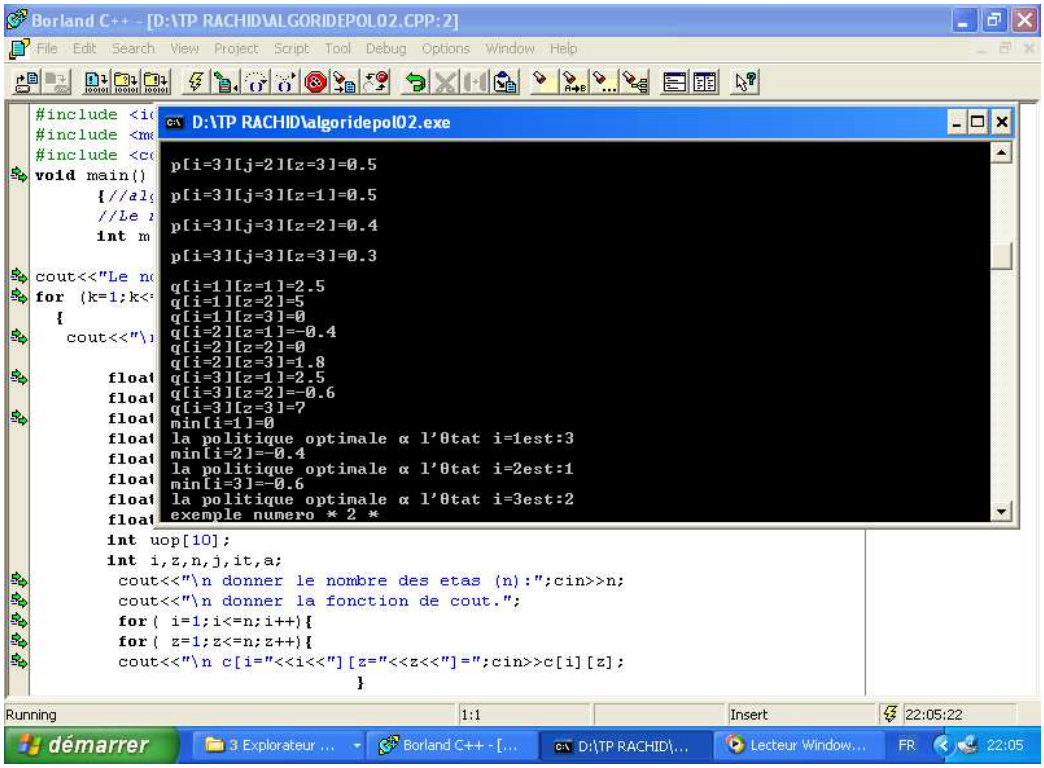


Figure 2.3 : L'affichage de la politique optimale.

Nous commentons le déroulement de l'exécution de l'algorithme en dressons un tableau des valeurs obtenues de *V*, la fonction coût, à chaque état sous les différentes actions.

Tableau 2.2 : La valeur V à chaque état sous différentes actions.

i	a_i	$V_i(a_i)$
1	1	2.5
	2	5
	3	0
2	1	-0.4
	2	0
	3	1.8
3	1	2.5
	2	-0.6
	3	7

La politique optimale choisit à chaque état l'action correspondante à la valeur minimale de *V* comme suit :

A l'état 1, en calculant le min (V[i=1][a_1], V[i=1][a_2], V[i=1][a_3])= V[i=1][a_3]=0, l'action optimale choisie est: uop[1]=a_3. De même à l'état 2, le min (V[i=2][a_1], V[i=2][a_2], V[i=2][a_3])= V[i=2][a_1]=-0.4. uop[i=2]=a_1 sera l'action optimale. Il découlera qu'à l'état 3, min (V[i=3][a_1], V[i=3][a_2], V[i=3][a_3])= V[i=3][a_2]=-0.6 et l'action choisie à l'état 3 est uop[i=3]=a_2. D'où la politique optimale du processus est π^*= (a_3, a_1, a_2).

Pour tracer des graphes, un logiciel pédagogique gratuit, scilab, est disponible à notre département. Son utilisation est facile. Il suffit d'introduire les fonctions de coûts, les 3 actions pour qu'une allure graphique se représente. Cette procédure est imagée dans la fenêtre du logiciel.

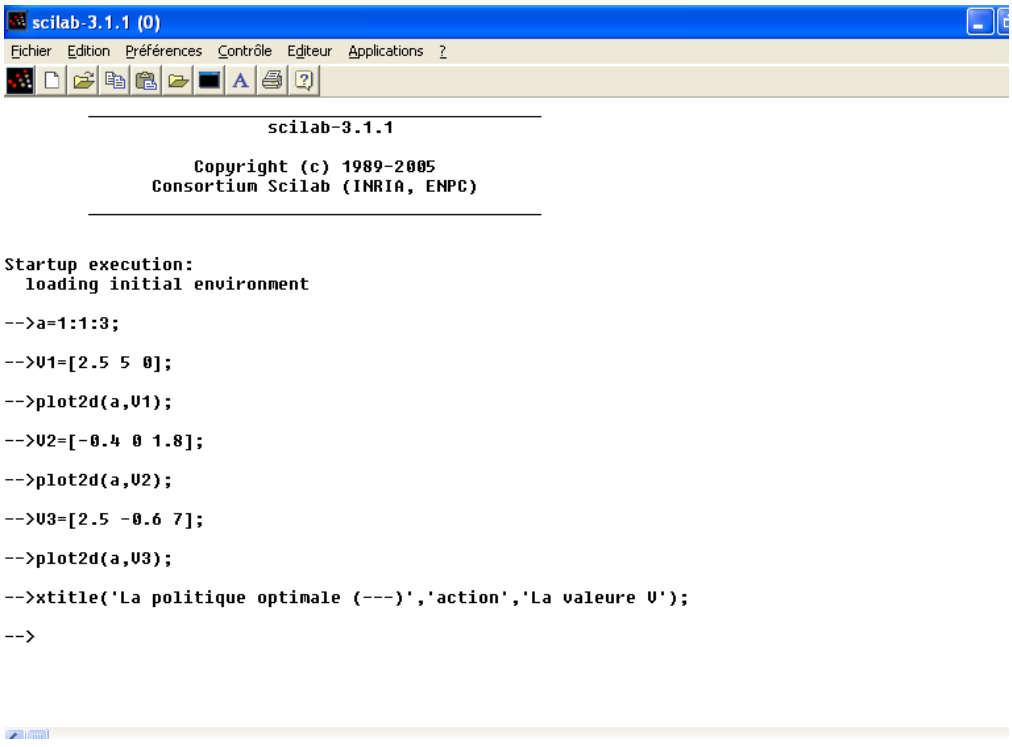


Figure 2.4 : Initialisation des données de la valeur *V* à chaque état sous différentes actions.

L'évolution de la fonction de valeurs *V* du processus par rapport aux différentes actions est représentée par le graphe ci-dessous.

Le « trait noir », le « rouge » et le « vert » sont utilisés pour représenter respectivement la variation de *V* en fonction des actions à l'état 1, à l'état 2 et à l'état 3.

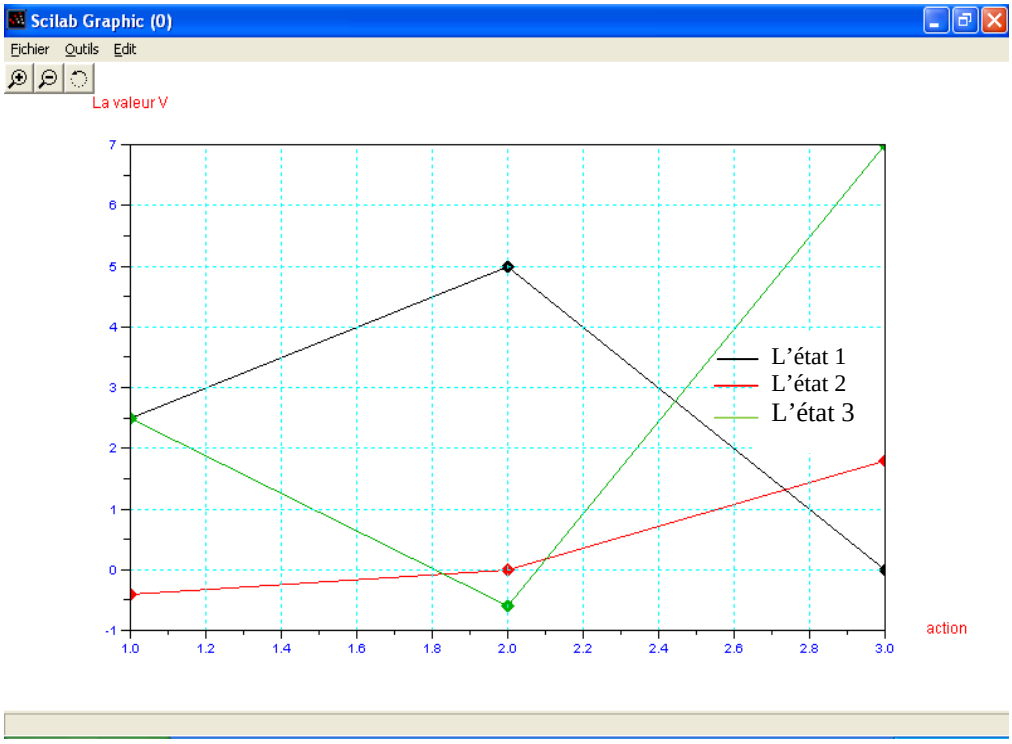


Figure 2.5 : La valeur *V* à chaque état en fonction des actions.

Pour cet exemple, la politique optimale $\pi^* = (a_3, a_1, a_2)$ d'exécution du processus en tout état est donnée par la suite d'actions à entreprendre et est représentée par une allure graphique simple.

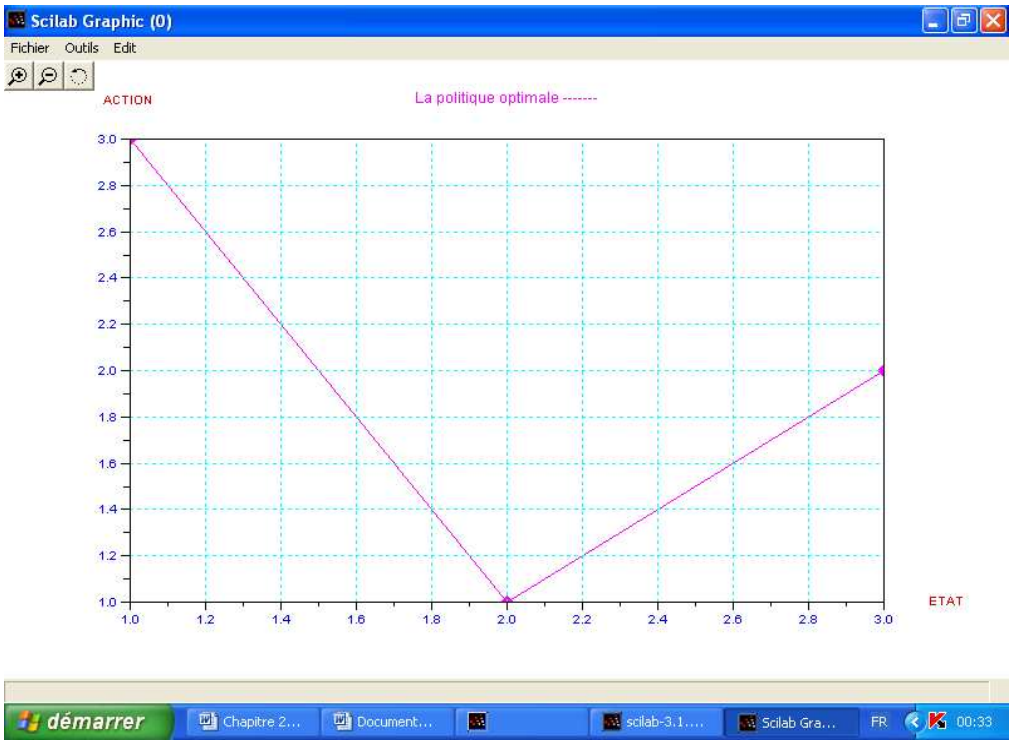


Figure 2.6 : Présentation de la politique optimale.

2.2. Complexité de l'algorithme d'itération de la politique

Chaque itération de l'algorithme consiste en deux opérations : la résolution du système d'équations, nécessite un peu moins de $| \Omega |^3$ opérations, et la phase d'amélioration, qui est effectuée en $| A | \times | \Omega |^2$ opérations. Comme pour l'algorithme d'itération de la valeur, le nombre d'itérations nécessaires à la convergence de l'algorithme est dépendant du problème ou de la cardinalité de l'ensemble des états et d'actions.

3. Algorithme d'itération sur les valeurs pour un processus à décision Markovien (value iteration).

Définissons la fonction coût optimale $v_{i,\alpha}^{\pi}$ du théorème (1) comme suit :

$v_{i,\alpha}^{\pi}(t)$= le coût minimal réduit à partir de l'instant t étant donné que le processus commence à l'état i, la solution d'un processus discret de décision déterministe est un problème d'optimisation sur l'ensemble de politiques. Le nombre de politiques peut être assez grand. Par exemple si |A(i)| ne dépend pas de l'état

courant, alors il y a $|A(i)|^{|\Omega|}$ politiques. Ainsi, explorer cet ensemble pour trouver la politique optimale peut être coûteux [06].

Afin d'éviter cette difficulté, l'idée fondamentale de cet algorithme consiste à évaluer d'abord la valeur optimale da la fonction v, i.e. v^*. Une fois que cette valeur a été calculée, il est possible d'obtenir une politique optimale en prenant une mesure gloutonne qui prend l'action correspondante à la valeur optimale v^*. Ainsi le problème est ramené à estimer la valeur optimale v^*. Ceci peut être effectué grâce à l'équation (2) qui donne la valeur d'un état i en fonction des valeurs de l'état successeur possible j.

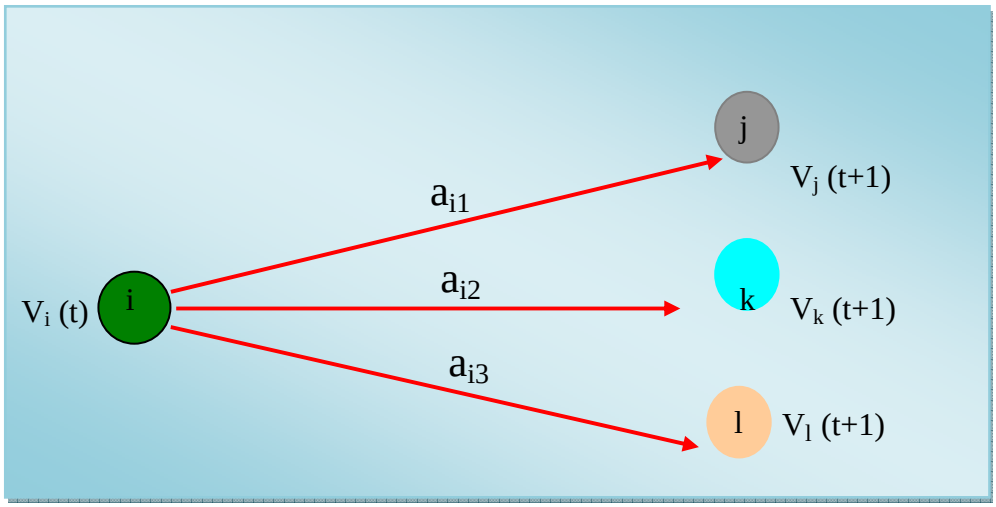


Figure 2.7 : La valeur de *V* à l'état i en fonction des valeurs de leurs états successeurs.

Les actions possibles à un état i sont a_{i1}, a_{i2}, a_{i3}. Si les valeurs optimales des états successeurs correspondants j, k et l sont connues, alors donne la valeur optimale de l'état i.

3.1 Q – Valeurs [06]

Soit π^{It} une politique existante obtenue après un nombre It itérations, pour laquelle la fonction valeur $v_i^*(It)$ est connue pour tous les états i. La Q-valeurs $Q(i, a_{iz})$ pour chaque état $i \in \Omega$ et l'action $a_{iz} \in A(i)$ est définie comme le coût immédiat $c_t(i, a_{iz})$ plus la somme dégressive de tous les états subséquents selon la politique π:

$$Q(i, a_{iz}) = c(i, a_{iz}) + \alpha \sum_{j=0}^{\infty} P_{ij}(a_{iz}) V^*_{j,\alpha}$$

La figure (2.8) montre le fonctionnement de cet algorithme.

Algorithme 2

1- Initialiser It, toutes $v_i^{\pi}(It)$ pour tout état iϵΩ, ϵ>0 arbitraire. 0<α≤1;

2- Itérer pour chaque valeur d'état i $\in \Omega$, calculer $v_i^{\pi}(It+1)$ à partir de

$$V_i^{\pi}(It+1) = \min_{a_{iz}} \left[c(i, a_{iz}) + \alpha \sum_{j=0}^{\infty} P_{ij}(a_{iz}) V_i^{\pi}(It) \right]$$

3- Si pour tous les états i, $\left\| V_i^{\pi}(It+1) - V_i^{\pi}(It) \right\| \prec \frac{\varepsilon . (1-\alpha)}{2.\alpha}$

alors aller à l'étape (4) sinon It=It+1 et retourner à l'étape (2) avec les valeurs ε-optimales de v_i^* ;

4-Calculer les Q-valeurs pour tous les états iϵΩ et toutes les actions a_{iz}ϵA(i),

$$Q(i, a_{iz}) = c(i, a_{iz}) + \alpha \sum_{j=0}^{\infty} P_{ij}(a_{iz}) V_j^*$$

5- Déterminer la politique optimale comme la mesure gloutonne pour $v_i^{\pi*}$,

$u^*(i) = \arg \min_{a_{iz}} Q(i, iz)$ Pour tout état iϵΩ.

Figure 2.8 : Algorithme d'itération sur les valeurs (VI).

L'algorithme d'itération sur les valeurs est l'algorithme le plus utilisé pour résoudre des processus décisionnels Markovien, il permet de résoudre l'équation d'optimalité en plusieurs itérations successives.

Exemple 2 : Utilisation de l'algorithme d'itération sur les valeurs.

Pour comparer l'algorithme d'itération sur les valeurs avec l'algorithme d'itération de la politique on utilise les mêmes valeurs de probabilité de transition que dans l'exemple 1.

Les coûts induits et les probabilités de transition entre les états sous différentes actions sont donnés dans le tableau suivant :

Tableau 2.3 : La probabilité de transition et les coûts induits à chaque état sous différentes actions.

i	a_i	$P_{i1}(a_i)$	$P_{i2}(a_i)$	$P_{i3}(a_i)$	$C(i,a_i)$
1	1	0.5	0.3	0.2	5
	2	0.4	0.6	0	10
	3	0	0.5	0.5	0
2	1	0.5	0.5	0	-1
	2	0.5	0.1	0.4	0
	3	0.3	0.3	0.4	2
3	1	0	0.3	0.7	5
	2	0.8	0.1	0.1	-1
	3	0.2	0.5	0.3	10

3.2. Détermination de la politique optimale

On initialise toutes les valeurs de probabilités de transition et les coûts encourus à chaque état : (figure (2.9)).

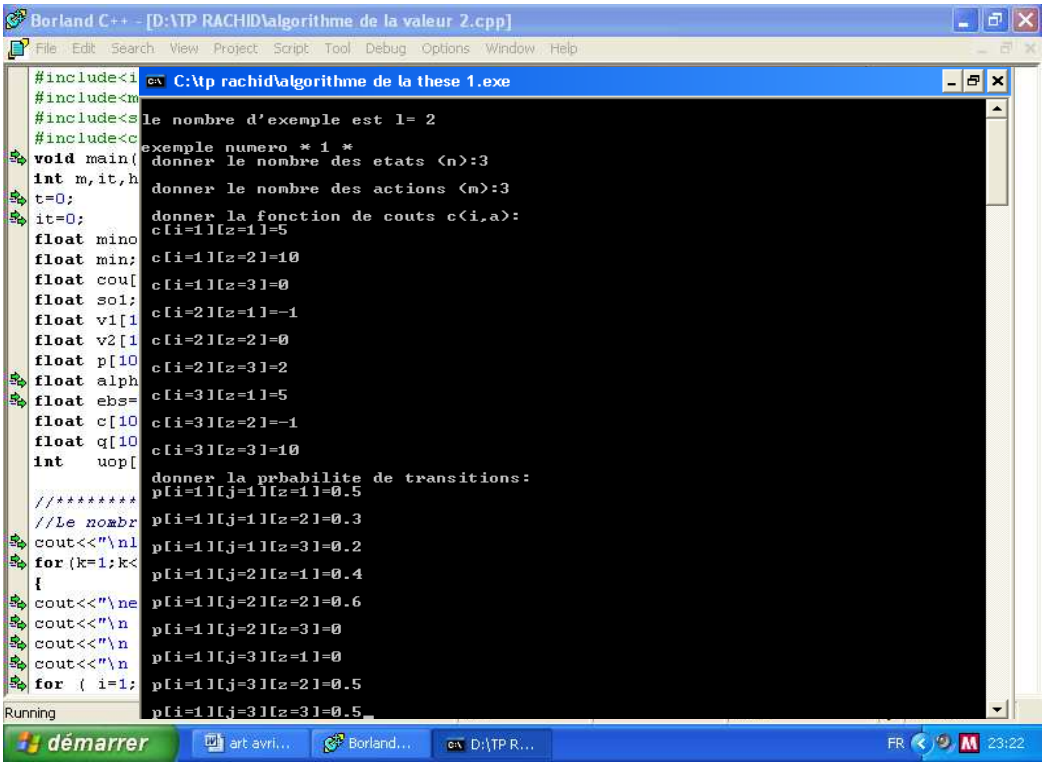


Figure 2.9 : L'exécution du programme (VI).

Résultat :

Les valeurs de Q à chaque état sous différentes actions sont résumées dans le tableau suivant :

Tableau 2.4 : La Q-valeurs à chaque état sous différentes actions.

i	a_i	$Q_i(a_i)$
1	1	5
	2	10
	3	0
2	1	-1
	2	-0.38
	3	5
3	1	4.23392
	2	-1.61286
	3	9.54035

A l'état 1, en calculant le min (Q[i=1][a_1], Q[i=1][a_2], Q[i=1][a_3])= Q[i=1][a_3]=0.

Alors l'action optimale choisie à l'état 1 par l'algorithme d'itération sur les valeurs est: uop[1]=a_3. A l'état 2, le min (Q[i=2][a_1], Q[i=2][a_2], Q[i=2][a_3])= Q[i=2][a_1]=-1. L'action choisie à l'état 2 est : uop[i=2]=a_1. De même à l'état 3 le min (Q[i=3][a_1], Q[i=3][a_2], Q[i=3][a_3])= Q[i=3][a_2]=-1.61286. Il découlera que l'action choisie à cet état est : Uop[i=3]=a_2. D'où la politique optimale du processus est π^*= (a_3, a_2, a_1). Tous ces résultats sont affichés dans la figure suivante :

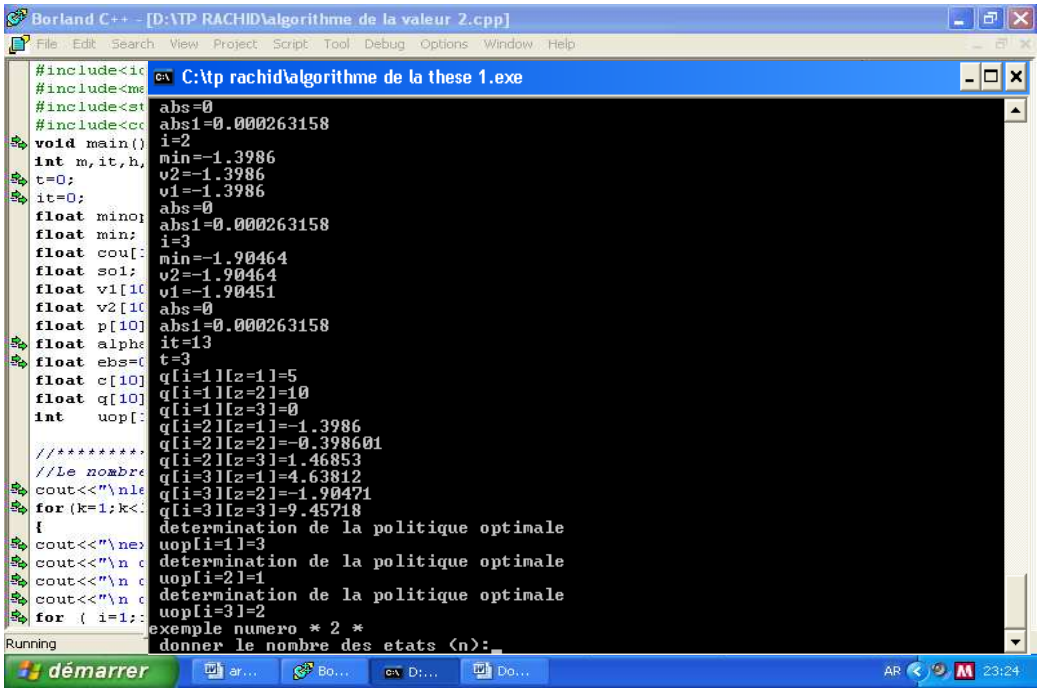


Figure 2.10 : L'affichage de la politique optimale pour l'algorithme VI.

Maintenant on trace le graphe qui représente la politique optimale du processus pour l'exemple 2 on suivant les étapes suivantes :

A l'étape 1, pour tracer ce graphe, il faut d'abord lancer logiciel Scilab, puis on va écrire les données indiquées dans la figure (2.11) suivante :

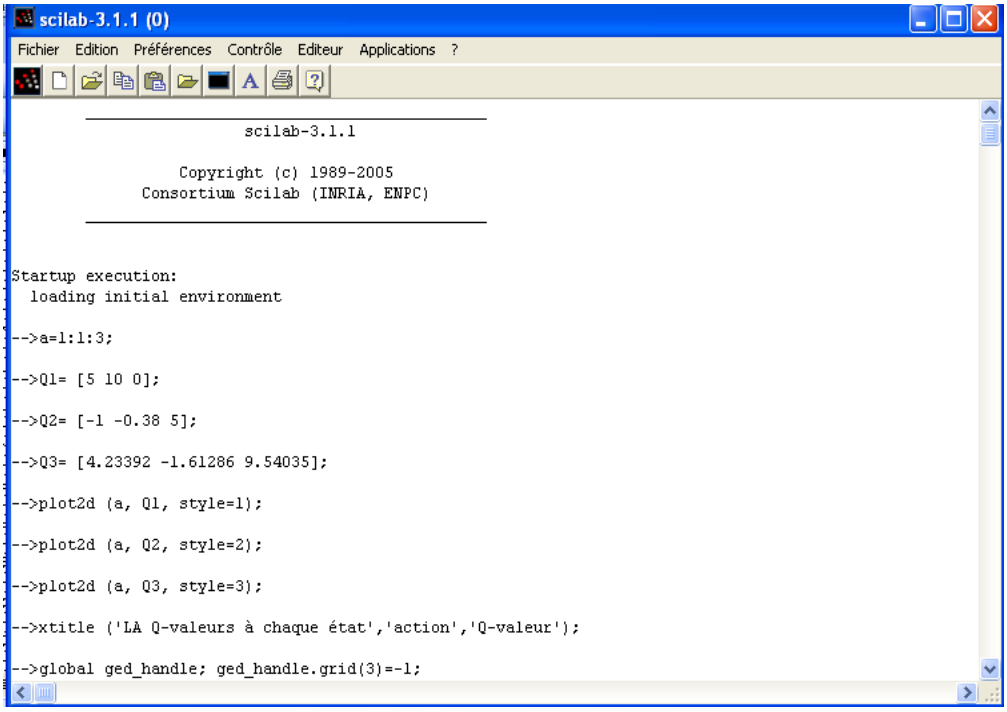


Figure 2.11 : Les données de l'exemple 3 pour tracer le graphe représentant la politique optimale.

La figure (2.12) est le graphe représentant la valeur Q à chaque état en fonction des actions : Le « trait noir », le « rouge » et le « vert » sont utilisés pour représenter respectivement la variation de Q en fonction des actions à l'état 1, à l'état 2 et à l'état 3.

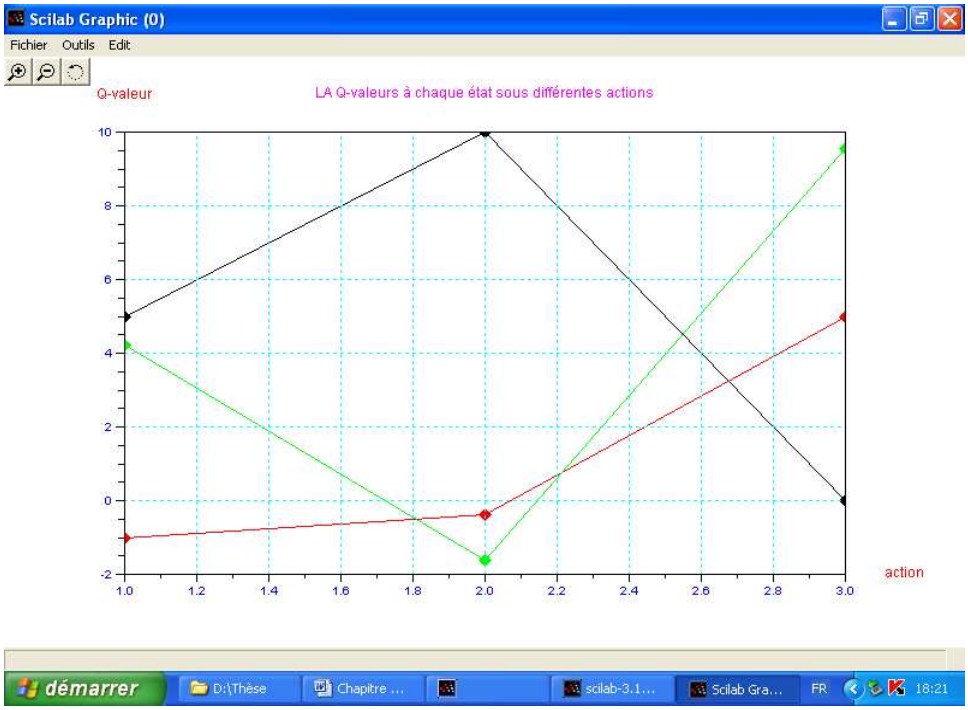


Figure 2.12 : Représentation de la Q-valeur en fonction des actions.

La politique optimale est représentée par le graphe dans la figure (2.13) suivante :

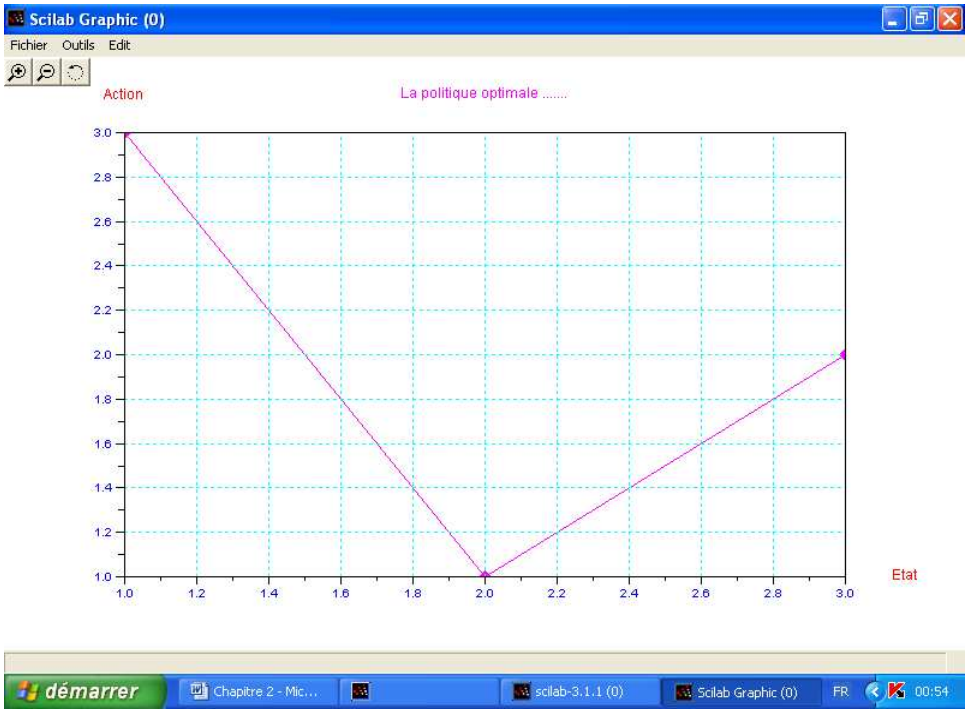


Figure 2.13 : Présentation de la politique optimale.

3.3. Justification

On considère un processus à décision Markovien à 3 états et 3 actions, on associe les mêmes coûts et les actions que dans l'exemple (3) à ces états. Ce graphe représente l'évolution de la fonction de la valeur Q à chaque état i en fonction des actions. On calcule la valeur Q pour toutes les actions a_i (i=1, 2, 3) par l'algorithme d'itération sur les valeurs et on prend l'action qui minimise la valeur de Q à chaque état. Enfin on choisit l'action a_3 à l'état 1, l'action a_1 à l'état 2, et l'action a_2 à l'état 3, et donc la politique optimale du processus est $\pi^* = (a_3, a_1, a_2)$.

3.4. Complexité de l'algorithme d'itération sur les valeurs

L'efficacité de l'algorithme dépend de deux facteurs : la complexité d'une itération, et le nombre d'itérations nécessaires pour converger. Chaque itération consiste à calculer la valeur d'une transition entre les états pour chaque action. Cela nécessite $A\times|\Omega|^2$ opérations. Le nombre d'itération nécessaire est plus difficile à calculer car il dépend du problème étudié. Littman et al.1995b montrent que l'algorithme est polynomial selon $|\Omega|$ |A|, α et B où B est le nombre d'octets nécessaires à la représentation des données du problème.

4. La résolution

Les algorithmes "Value iteration" et "Policy iteration" n'étant pas meilleurs l'un que l'autre, ils ont été implémentés tous les deux [3]. Les comparaisons des résultats sont consignées dans le tableau 2.5. Il faut noter que l'algorithme « Policy iteration » est initialisé avec une politique aléatoire.

Tableau 2.5 : Comparaison des algorithmes " itération de la politique" et " itération de la valeur" (Le temps est en millisecondes).

Taille de l'espace	***Temps policy itération (en millisecondes)***	***Temps value iteration (en millisecondes)***
2×2×2	15	15
3×3×3	16	31
4×4×4	78	78
5×5×5	94	31
6×6×6	719	732

Un graphe représentant le temps d'exécution des deux algorithmes en fonction des actions peut être obtenu. La figure (2.14) représente la différence entre les deux algorithmes en fonctions du temps d'exécution :

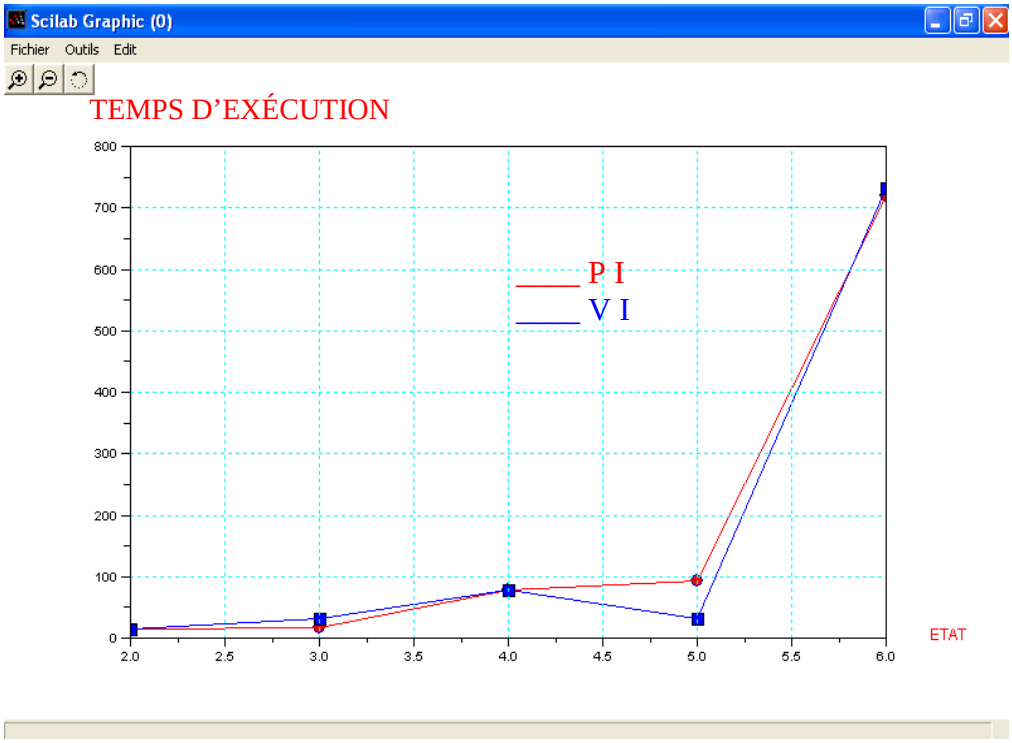


Figure 2.14 : Comparaison de l'algorithme (P I) avec (V I).

Pour des raisons de lisibilité, nous n'avons pas pu représenter un graphe à un grand nombre d'états. Mais nous concluons que l'algorithme VI est asymptotiquement meilleur.

5. Comparaison des algorithmes

Les deux algorithmes ont un fonctionnement différent :

L'algorithme d'itération sur les valeurs fonctionne par petite amélioration de la fonction de la valeur, ce qui donne un grand nombre d'itérations rapides pour que l'algorithme converge.

L'algorithme d'itération de la politique au contraire procède par de grandes améliorations de la fonction de la valeur, donc la convergence de l'algorithme se fait en un petit nombre d'itérations très coûteuses.

6. Remarques sur l'implantation des algorithmes

La grande difficulté pour la résolution d'un problème de décision Markovien concerne la forme du problème que l'on veut résoudre [22]. Pour le reste, chaque état est numéroté entre 1 et | Ω |, chaque action également, entre 1 et |A|. La fonction de valeur est donc simplement un tableau de | Ω | réels. Une politique déterministe est un tableau de | Ω | entiers compris entre 1 et |A|. La fonction de transition est un tableau tridimensionnel de dimensions respectivement | Ω | × |A| × | Ω | et contenant des réels compris entre 0 et 1. La fonction retour est un tableau tridimensionnel de dimensions respectivement | Ω | × |A| × | Ω | et contenant des réels. Dans l'algorithme d'itération de la valeur, on peut, au choix, utiliser deux tableaux distincts pour Vi et Vi+1 ou confondre les deux. L'algorithme converge dans les deux cas.

7. Conclusion

Pour évaluer les performances de l'algorithme d'itération de la politique (PI) avec l'algorithme d'itération sur les valeurs (VI), on les a programmés tous les deux. L'efficacité de l'algorithme d'itération de la politique n'a été examinée qu'après plusieurs essais possibles. Nous augmentons le nombre de l'espace (état-action) et nous calculons à chaque fois le temps d'exécution.

En pratique [22] : on peut appliquer ces algorithmes à des PDM ayant jusque de l'ordre de 10^6 états. La complexité en temps de calcul est O (| Ω ||A|) pour l'algorithme d'itération de la valeur ; et pour calculer la fonction valeur, les méthodes de programmation linéaire sont moins efficaces que l'algorithme d'itération de la valeur et ne permettent de traiter que des problèmes de taille 100 fois plus petite que ceux traités par les algorithmes vus ici.

La recherche directe de la politique optimale est sans intérêt : la complexité est non polynomiale en O (| Ω | $^{|A|}$) (c'est un problème NP- difficile). De nombreuses heuristiques sont possibles pour accélérer l'itération de la valeur. On peut ne pas parcourir systématiquement tous les états à chaque balayage ; il faut cependant garantir que tous les états seront visités suffisamment de fois ; voir en particulier la programmation dynamique temps-réel (cf. Barto et al. [1995]).

CHAPITRE 3
ORDONNANCEMENT DE TACHES STOCHASTIQUES NON MARKOVIENNES

1. Introduction

Dans un travail récent, Kali [17] a étudié le modèle markovien. Soit « N » tâches à exécuter sur une machine. Leur temps d'exécution sont supposés aléatoires de distribution connue. A une tâche on peut associer un processus à deux décisions, « exécuter » la tâche ou la « geler ». Dans les *problèmes classiques*, durant le temps de transition du processus d'un état i vers un état j, aucune action n'est entreprise. Dans notre *problème considéré*, durant une transition d'un processus, des actions peuvent être entreprises. Ce que nous sous-entendons par le cas non markovien. Ce genre de problèmes se rencontre surtout dans les systèmes informatisés multi-tâches où durant l'impression d'un fichier, par exemple, nous pouvons apporter des modifications dans le texte du dit fichier, le sauvegarder, lancer une seconde impression etc.…Au moins trois actions peuvent être entreprises durant cette impression. Nous présentons une politique optimale de résolution de ces problèmes non markoviens, qui est une politique d'indices. On associe à chaque tâche, une priorité ou un indice d'allocation dynamique. En tout instant on exécute la tâche qui a les plus petit indice. En cas d'égalité des indices, nous arbitrerons selon une règle connue.

De la littérature, Glazebrook [10] a étudié ces problèmes et a pu montrer l'existence et la caractérisation de ces indices. Nous exposons cette méthodologie succinctement. Nous avons implémenté un algorithme de leur détermination en utilisant un langage évolué. Un exemple d'application est même fourni.

2. Formulation d'un problème d'ordonnancement stochastique :

Nous considérons un problème d'ordonnancement stochastique de « n » tâches sur une machine dans un atelier. Durant un intervalle de temps de longueur t, une tâche « i » s'exécute pendant $t_i^e(t)$ et est en attente pendant $t_i^a(t)$. L'ensemble

$\{t_i^e(t), t = 0, 1,\ldots\}$ peut représenter un processus de décision qui modélise une tâche où $t_i^e(t)$ représente l'état de la tâche i à l'instant t.

Dans les *problèmes classiques*, à un état « i », si une décision a_i, par exemple « exécuter » une tâche est choisie, une transition selon une probabilité p_{ij} s'effectuera à un état « j ». Aucune action n'est faite durant la transition du processus entre les états. La stratégie optimale est déterminée par la théorie des processus bandits, des processus à 2-décisions. A chaque processus, un indice d'allocation dynamique ou priorité lui est attribué.

Les processus sont exécutés dans l'ordre croissant de ces indices. En cas de conflit ou d'égalité entre les plus grands indices, on arbitrera en choisissant une tâche selon généralement une règle connue de type SPT, LPT, FIFO…etc. L'indice d'allocation dynamique s'interprète comme un coût d'arrêt d'exécution de la tâche. Dés qu'une tâche nous coûte le plus, on l'interrompe et on exécute une autre qui nous coûte moins. L'existence, la caractérisation et leur détermination sont faites dans un cas d'étude à notre département par Kali [03]. Dans notre *problème considéré*, *durant une transition d'un processus, des actions peuvent être entreprises.* Ce que nous sous-entendons par le cas non markovien. Les décisions « exécuter » et « geler » une tâche sont faites d'une façon non markovienne. Ces problèmes se rencontrent surtout en informatique dans un système multi-tâches. Par exemple, nous lançons l'impression d'un fichier, entre temps nous pouvons apporter des modifications dans le texte du dit fichier, le sauvegarder, lancer une seconde impression etc.…Au moins trois actions peuvent être entreprises durant cette impression. Nous présentons une politique optimale de résolution de ces problèmes non markoviens, qui est une politique d'indices inspirée de celle des indices de Gittins. Glazebrook [10] a étudié ces problèmes et a pu caractériser ces indices. Nous exposons cette méthodologie succinctement. Nous avons implémenté l'algorithme de leur détermination en utilisant un langage évolué. Un exemple d'application est même fourni.

3. Processus et Famille alternative de processus bandit

Un processus bandit est un processus semi Markovien à deux décisions. Il est défini par la donnée d'un sextuplé (Ω, U, P, F, C, α) où Ω est l'espace d'états du processus et peut être fini, dénombrable ou continu.

U est un ensemble de décisions consistant en deux éléments « 1 » (exécuter le processus) et « 0 » (geler le processus). P est la loi de transition entre les états du processus,

F est la fonction de répartition de la durée de transition τ de loi quelconque. Dans le cas Markovien, F suit une loi exponentielle.

C est une fonction coût et «α » est un réel ($0 < \alpha < 1$) appelé facteur de prévision.

Une transition au temps t de l'état x(t) à l'état x(t+1) implique une augmentation du gain de α^t c[x (t), x (t+1), u(t), t].

Les réalisations {x(t): t = 0, 1, ...} du processus sont appelées trajectoires auxquelles on associe avec chaque ensemble de décisions et d'états, un gain. Le gain total qui s'accroît si le processus décrit une trajectoire { x(t): t = 0, 1,…} est

$\sum_{t \geq 0} \alpha^t$ c[x(t),x(t+1), u(t), t], où c[x, y, u, t] est une fonction coût. Dans ce cas, le processus est dit à fonction « coût séparable ». Pour toute politique π, le critère de l'espérance du coût prévisionnel est définit par $V_\pi(i) = E_\pi\left[\sum_{t=0}^{\infty} \alpha^t c(x_t, a_t) / x_0 = i\right]$, $i \geq 0$.

Les coûts $c(x_t, a_t)$ sont supposés bornés et $\alpha < 1$. E_π désigne l'espérance conditionnelle sachant que la politique π est employée et que l'état du processus à l'instant zéro est i.

Pour les automaticiens, ces processus sont de contrôle adaptatif.

Dans la suite, une décision, une action, une commande ou un contrôle sont synonymes.

Un processus et une tâche sont aussi synonymes.

On appelle *famille de processus bandits alternatifs* une suite de processus

{ (Ω_i,U, P_i, F_i, C_i, α): $1 < i < n$ } tel qu'en chaque instant t (t = 0,1,2,..), les processus évoluent simultanément. Ils ont les mêmes instants de décision. En tout instant la décision « 1 » est appliquée à un processus " i " et « 0 » est appliquée à tous les autres. Le facteur prévisionnel " α " étant le même pour tous les processus.

Le problème est de trouver une stratégie d'exécution de tâches d'espérance de la somme des coûts prévisionnels optimale.

Gittins et Jones [07] ont énoncé le premier théorème d'existence de ces IAD. Ils représentent un rapport de moyenne de coût sur une moyenne de temps utilisé. C'est aussi un coût d'arrêt du processus si on introduit la notion d'arrêt. Nash [21] et Glazebrook [08] ont pu donner la première caractérisation de ces indices notés ν_i ($1 \le i \le n$). Sachant que ces IAD existent et reconnaissant leur forme, au moins trois algorithmes de détermination des I.A.D sont cités dans la litérature. Il s'agit de ceux de Robinson [26], de Varaiya et al [29] et de Sonin [28].

3.1 Théorème du IAD (the Dynamics allocation indices)

La caractérisation de la stratégie optimale π^* est le résultat fait par Gittins et Jones (1972), appelé le théorème du DAI. Ce résultat caractérise la stratégie optimale pour un type particulier de processus à décision qui est appelé par Gittins et Jones *une famille alternative du processus bandit*.

4 Processus en temps discret à coûts prévisionnels

Considérons un processus discret qui peut être à un instant t dans un des états 0, 1, 2,… de l'espace des états Ω supposé être fini (il peut être aussi dénombrable ou continu).

A chaque état i, une action a_i doit être choisit d'un ensemble A(i) supposé fini.

A un instant t, une transition de l'état i à l'état j surviendra avec une probabilité f(j, q / i, a) sachant que l'action « a » est exécutée q fois.

Notons par : $$F(j, q \mid i, a) = \sum_{r=1}^{q} f(j, r \mid i, a) \qquad (3.1)$$

La probabilité d'atteindre au moins une fois l'état j en utilisant l'action « a » une fois ou deux fois ou au plus q fois.

$G(q \mid i, a) = 1 - \sum_{j \neq i} F(j, q \mid i, a)$ (3.2), représentera la probabilité de n'atteindre aucun état j partant de l'état i.

On suppose aussi que $\lim_{q \to \infty} G(q \mid i, a) = 0,\ a \in A(i),\ i \in \Omega$ (3.3)

Cette limite représente l'événement de certitude d'atteindre au moins un état j.

Il découlera que :

(i) un processus est observé à l'état i à un instant t, Il est resté dans l'état « i », « s »unités de temps. une action « a » est utilisée t_a fois.

(ii) si une action a est prise pour (t_a + 1) fois alors la probabilité que le processus est observé dans l'état j ≠ i à l'instant (t+1) est:

$$f\left(j,\quad t_a+1 \mid i,\quad a\right)\ \left\{G\left(t_a \mid i,\quad a\right)\right\}^{-1}.$$

(iii) Le coût encouru est c(i, a) α^t où 0 < α < 1. Ces coûts sont aussi supposés positifs et bornés.

Si une stratégie π (une suite d'actions) est adoptée durant l'évolution du processus, alors l'espérance du coût total est définie par

$$v_\pi(i) = E_\pi\left\{ \sum_{t=0} c(x_t a_t)\alpha^t \middle| x_0 = i \right\} \qquad (3.4)$$

Notre but est de déterminer une stratégie qui minimise cette expression pour tout état i. Notons par $v_\alpha(i) = \inf_\pi v_\pi(i)$.

<u>Théorème 1</u>(Gittins et Jones 1972)

Étant donné une famille alternative de processus bandit {(Ω_j, P_j, c_j, α), j=1, …,N}. Il existe des fonctions {y_j, $\Omega_j \rightarrow$ R, j=1, …, N} tel que l'action a_i est optimale quand l'état du processus est X_t si et seulement si :

$$y_i\{x_i(t)\} \quad = \min_{j=1,..,\ N} \left[y_j\{x_j(t)\} \right] \qquad (3.5)$$

Si le minimum dans (3.5) est atteint à plus d'une valeur, il est peu important que les actions correspondantes soient choisies. Les fonctions y_j (et ses valeurs) sont connus par IADs ou Indices d'Allocation dynamique. Le théorème 2 qui est donné par Gittins et Glazebrook [09] sera utilisé dans le paragraphe 4 pour obtenir les IADs.

<u>Théorème 2</u> : (Gittins et Glazebrook 1977)

$$y_i\ (x_i\ ) = \inf_s \frac{E\left[\sum_{t=0}^{s-1} c_i\ \{x_i\ (t\ ), x_i\ (t+1)\}\alpha^t \right]}{1-E\left(\alpha^s\ \right)} \qquad (3.6)$$

Où $x_i = x_i(0)$ et l'inférieur est pris sur tout le temps d'arrêt s.

Le résultat fournit est finit pour tout état $x_i \in \Omega_i$.

Notons que le temps d'arrêt s pour le processus bandit (Ω_j, P_j, c_j, α) est un entier évalué, c'est un variable aléatoire.

Concernant les conditions du théorème 2, la supposition des coûts bornés assure cela l'inférieur dans cette expression et est toujours fini.

5. <u>Caractérisation de la stratégie optimale</u>

Considérons un processus de décision D(i, u) à un état initial $i \in \Omega$. On doit lui associer une famille de processus bandits alternatifs $\{(\Omega_{ia}, P_{ia}, c_{ia}, \alpha), \quad a = 1,....., |A(i)|\}$ à fonction coûts séparables construite de la façon suivante. Supposons qu'une transition à un état j aura lieu à un instant t. Un coût u(j) α^t est encouru.

Aucun coût additionnel n'est encouru après la première transition. Les ensembles d'actions « a » sont notés par A(i) où

a = 1, 2, …, $|A(i)|$. La famille est définit comme suit :

i. à un instant t, $X_t = \{x_{i1}(t), x_{i2}(t),...., x_{i|A(i)|}(t)\}$ représente un vecteur d'états où chacun des états est formé d'un couple $x_{ia}(t) = \{ x_{ia}^1(t), x_{ia}^2(t) \}$.

La première composante $x_{ia}^1(t)$ représente le nombre de fois que l'action a a été choisie jusqu'à l'instant t et :

$$x_{ia}^2(t) = \begin{cases} k \quad si \ la \ première \ transition \ était \ dans \ l'état \ k \ avant \ l'instant \ t \ en \ actionnant \ "a" \\ 0 \qquad\qquad si \ aucune \ transition \ ne \ s'est \ produite \end{cases}$$

ii. l'espace d'action est {i_1, i_2,…, $i_{|A(i)|}$ } où l'action « a » exécute le processus bandit i_a.

iii. si i_a est choisi à l'instant t quand la famille est à l'état X_t alors une transition à X_{t+1} se produira où

$x_{ib}(t+1) = x_{ib}(t)$ pour tout $b \neq a$. $x_{ia}(t+1)$ est déterminé par la loi de transition :

- Si $x_{ia}(t) = (s, 0)$ alors :

$$x_{ia}(t+1) = \begin{cases} (s+1, k) \ avec \quad une \ probabilité \quad f(k, \ s+1 | \ i, \ a)\{G(s | \ i, \ a)\}^{-1} & (4.1) \\ (s+1, 0) \ avec \quad une \ probabilité \quad G(s+1 | \ i, \ a) \ \{G(s | \ i, \ a)\}^{-1} & (4.2) \end{cases}$$

- Si $x_{ia}(t) = (s, \ k), k \in \Omega$ (4.3)

alors $x_{ia}(t+1) = x_{ia}(t)$.

iv. la transition décrite dans (4.1) est de coût $c(i,a)\ \alpha^t + u(k)\ \alpha^{t+1}$, dans (4.2) de coûts c(i, a) α^t et de coûts nul dans (4.3).

La stratégie optimale pour la famille $\{(\Omega_{ia}, P_{ia}, c_{ia}, \alpha), a = 1,...,|A(i)|\}$, pour D(i, u), est donné par une suite des IAD caractérisés par l'énoncé ci-dessous.

Théorème de Glazebrook (1978)

Supposons que à l'instant t, D(i, u) est tel que :

(a) aucune transition n'est encourue, et

(b) l'action « a » était toujours employée s_{ia} fois, a ∈ A(i).

Dans la stratégie optimale, une action b doit être employée ultérieurement si et seulement si : $y_{ib}(s_{ib}; u) = \min_{a=1,...,|A(i)|} \{y_{ia}(s_{ia}; u)\}$, (4.4)

où : $$y_{ia}(s_{ia}, u) = \inf_{r>0} \frac{\left\{ \sum_{s=0}^{r-1} \alpha^s C(i,a)\ \ G(s_{ia}+s|i,a)\ \ + \sum_{s=1}^{r} \sum_{j\neq i} \alpha^s u(j) f(j, s_{ia}+s|i,a) \right\}}{\left\{ G(s_{ia}|i,a) - \alpha^r G(s_{ia}+r|i,a) \right\}} \quad (4.5)$$

Preuve :

Les conditions (a) et (b) dans le théorème sont équivalent du rapport que le processus bandit i_a est à l'état (s_{ia}, 0), a∈A(i) et qui donne un coût c(i, a) α^t + u(k) α^{t+1}. Les indices d'allocation dynamique (IAD) pour ce problème sont calculés par la formule (3.2).

Il n'est pas difficile pour prouver qu'il est suffisant de prendre l'inférieure dans (3.2) sur les temps d'arrêt $t_{ia}(r)$, r∈Z, qui sont définit comme suit :

D(i ; u) est à l'état écrit dans le rapport du théorème à l'instant t.

Une action *a* est choisie pour le prendre sans interruption jusqu'à l'instant t.

On définit $t_{ia}(r)$, r∈Z, pour que r si une transition du processus n'est produit sous l'action a par le temps t+r est ∞ autrement.

Donc on trouve :

$\forall r \in Z^+$,

$$\frac{E\left[\sum_{s=0}^{t_{ia}(r)-1} c_{ia}\left\{x_{ia}(t+s), x_{ia}(t+s+1)\right\}\alpha^{s}\right]}{1-E\left(\alpha^{t_{ia}(r)}\right)} =$$

$$\frac{\left\{\sum_{s=0}^{r-1} C(i,a)\, G(s_{ia}+s \mid i,a)\,\alpha^{s} + \sum_{s=1}^{r}\sum_{j\neq i} u(j)\, f(j, s_{ia}+s \mid i,a)\,\alpha^{s}\right\}}{\left\{G(s_{ia} \mid i,a) - \alpha^{r} G(s_{ia}+r \mid i,a)\right\}} \qquad (4.6)$$

Théorème de Glazebrook suit maintenant de (4.6) et les théorèmes (1) et (2).

Dans le corollaire (1) nous obtenons une caractérisation de la stratégie optimale pour la classe des processus de décision intéressé.

Corollaire 1[10]

Supposons que à un certains instant le processus de décision définie dans le paragraphe 2 est à l'état *i* et cela pendant sa résidence présente dans *i* l'action *a* a été toujours employé s_{ia} fois, $a \in A(i)$.

Dans la stratégie optimale, action b doit être employé après si et seulement si

$$y_{ib}(s_{ib}\ ; V_{\alpha}) = \min_{a=1,..,|A(i)|}\{y_{ia}(s_{ia};\ V_{\alpha})\} \qquad (4.7)$$

Si le minimum dans (4.7) est atteint à plus d'une valeur, il est peu important que des actions correspondantes sont choisies.

Preuve :

Notons que l'existence d'une stratégie optimale suit de la théorie générale des processus à décision Markovien, par exemple de Ross (1970) (Voir chapitre 1)

Suivant à son processus une transition dans son état présent *i* il évolue sous actions choisies de A(i) jusqu'à ce que son état change encore. Le prochain état est clair qu'une stratégie optimale sera poursuivie après cette prochaine transition.

Par conséquent, si la transition produit à l'instant t et est dans j, l'espérance du coût total de t est $V_{\alpha}(j)\ \alpha^{t}$. Clairement alors que le problème de choisir de façon optimale des actions de A(i) pendant la résidence du processus à l'état *i* est

équivalent au problème de choisir une stratégie optimale pour D(i ; V_α). Le résultat suit alors du théorème 3.

Section 5 contient une exposition d'une méthode itérative de calculer une stratégie optimale pour le processus bandit D(i ; V_α).

6. Une méthode itérative pour déterminer la stratégie optimale

P(j, t | i, π) représente la probabilité que la première transition de i à j dans D(i, u) sous la stratégie π se produit au temps t. B(Ω) représente l'ensemble des fonctions bornée sur Ω .

Pour tout $u \in B(\Omega)$, la norme de u est définie par $\|u\| = \sup_{i\in\Omega} |u(i)|$.

Définissons l'application $T_\alpha : B(\Omega) \rightarrow B(\Omega)$ par:

$$(T_\alpha u)(i) = \inf_{\pi} \left\{ c(i, \pi) + \sum_{t=1}^{\infty} \sum_{j\neq i} P(j, t|i,\pi)\, u(j)\, \alpha^t \right\} \qquad (5.1)$$

L'inférieur dans le coté droit de (5.1) est atteint par la stratégie optimale de D(i ; u), cette stratégie est donnée par le théorème 3, un fait qui nous permet de calculer la fonction $T_\alpha u$, qui est important dans la procédure itérative.

La suite des lemmes et corollaire caractérisent la politique optimale. Ils sont donnés sans preuves.

Lemme 1 [10]

L'application T_α est une contraction involutive où $T_\alpha V_\alpha = V_\alpha$.

Récursivement l'application $T_\alpha^n : B(\Omega) \rightarrow B(\Omega)$ est définie par :

$$T_\alpha^1 u = T_\alpha u \text{ et } T_\alpha^n = T_\alpha \left\{ T_\alpha^{n-1} \right\}, \quad u \in B(\Omega).$$

Lemme 2[10]

$$T_\alpha V_\alpha = V_\alpha .$$

C'est un rapport du fait que V_α satisfont les équations d'optimalités de programmation dynamiques. Plusieurs preuves standard de ceci sont disponibles.

On définie maintenant l'application $T_\alpha^n : B(\Omega) \to B(\Omega)$ inductivement, comme suit :

$$T_\alpha^1 u = T_\alpha u \text{ et } T_\alpha^n = T_\alpha \left\{ T_\alpha^{n-1} \right\}, u \in B(\Omega).$$

Comme immédiatement conséquence du lemme 1 et 2 et la contraction traçant le théorème de point fixe.

On obtient les corollaires suivants.

Corollaire 2 [10]

V_α est l'unique solution de l'équation

$$V_\alpha(i) = \inf_{\pi} \left\{ c(i, \pi) + \sum_{t=1}^{\infty} \sum_{j \neq i} P(j, t | i, \pi) \; V_\alpha(j) \; \alpha^t \right\}, \quad i \in \Omega$$

En outre pour tout $u \in B(\Omega)$, $\lim_{n \to \infty} T_\alpha^n u = V_\alpha$ (5.2)

Le corollaire 2 rapporte un type d'itération de valeur, est une méthode pour calculer la fonction V_α itérativement. Nous avons également un résultat de convergence pour DAI, celle-ci lemme 3.

En définissant les applications $y_{ia}; \; B(\Omega) \to B[Z^+ \cup \{0\}]$ par:

$$(y_{ia} u)(s) = y_{ia}(s; u), \quad a \in A(i), \quad i \in \Omega .$$

Un résultat s'obtient.

Lemme 3 [10]

Pour tout $u \in B(\Omega)$, $\lim_{n \to \infty} y_{ia}\left(T_\alpha^n u\right) = y_{ia}(V_\alpha)$. (5.3)

5.1. Algorithme de déterminer une politique optimale pour un processus à décision non Markovien

Nécessaire un processus à décision non-Markovien : $(\Omega,\ A,\ P,\ C, \alpha)$, $0 \prec \alpha \prec 1$,

Nécessaire un seuil ε.

Initialiser la fonction bornée $u(i) = 0$ pour tout état $i \in \Omega$.

Répéter

Pour tout état iЄΩ faire

Pour toute action aЄA(i) faire

$$y_{ia}(s_{ia}, u_k) \leftarrow \inf_{r \succ 0} \left[\left\{ \sum_{s=0}^{r-1} \alpha^s C(i,a) G(s_{ia} + s | i, a) + \sum_{s=1}^{r} \sum_{j \neq i} \alpha^s u(j)\, f(j, s_{ia} + s | i, a) \right\} / \left\{ G(s_{ia} | i, a) - \alpha^r G(s_{ia} + r | i, a) \right\} \right]$$

Fin pour.

$$\pi_k(i) = Arg\ \min_{a \in A(i)} \{ y_{ia}(s_{ia}; u_k) \}$$

$$T_\alpha^k u_k(i) \leftarrow \inf_{\pi_k(i)} \left\{ c(i,\ \pi_k(i)) + \sum_{t=1}^{\infty} \sum_{j \neq i} f(j,\ t | i,\ \pi_k(i))\ u(j)\ \alpha^t \right\}$$

Fin pour

Jusque $\left\| T_\alpha^{k+1} u(i) - T_\alpha^k u(i) \right\| \leq \varepsilon$

Figure 3.1 : Algorithme d'itération sur les valeurs pour un processus à décison non- markovien

7. Exemple

Soit un processus à 2-états et à 2-actions où l'espace des états est Ω={1, 2} et les espaces des actions sont A(1)={a_1, a_2} et A(2)={a_3, a_4}.

Les coûts induits à chaque états sous différent actions sont c(1, a_1)=4, c(1, a_2)=4, c(2, a_3)=3,

c(2, a_4)=7. Le facteur prévisionnel α=0.5 et l'erreur d'appréciation est ε=0.001. La transition de l'état 1 à l'état 2 se fait par une suite d'actions identiques à entreprendre selon les probabilités v_f(2, .|1, a_1)={0.2, 0.2, 0.3, 0.3} et v_f(2, .|1, a_2)={0.3, 0.4, 0.1, 0.2}.

Les transitions réciproques de l'état 2 à l'état 1 se font selon les probabilités

$v_f(1, .|2, a_3)=\{0.1, 0.2, 0.3, 0.4\}$ et $v_f(1, .|2, a_4)=\{0.1, 0.4, 0.2, 0.3\}$.

1ère itération

- Déclarons tous les coûts induits et les probabilités de transition entre les états.

La figure 1 montre le fonctionnement de l'exécution du programme :

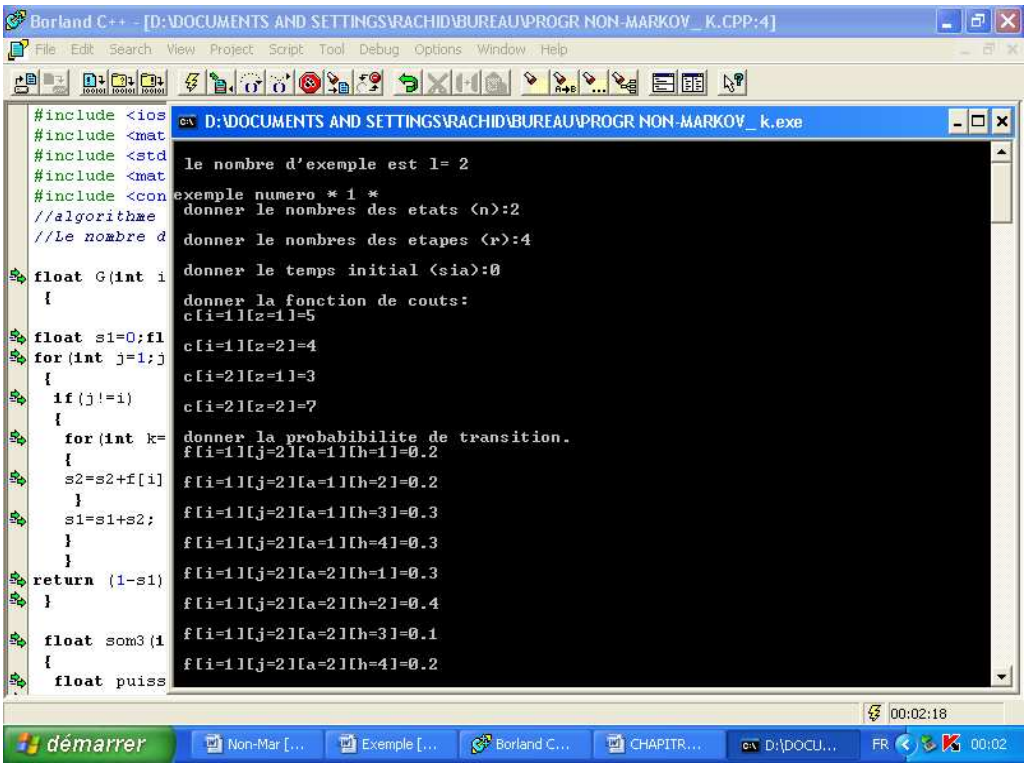


Figure 3.2 : Fenêtre du programme d'initialisation.

- Initialisons la fonction bornée « u » qu'on appliquera à la contraction T_α par u(1)=u(2)=0. Nous rappelons que s_{ia} représente le temps que le processus à passé dans l'état i sous l'action a. Soit $s_{11}=s_{12}=0$ et $s_{23}=s_{24}=0$.

Calculons les indices du processus D (i ; u) à chaque état « i » sous les différentes actions « a » par la formule (4.5).

Prenons le minimum de ces indices en chaque état et lui associons l'action qui le réalise.

Les valeurs obtenues sont :

A l'état 1 : y_{11} (s_{11}, u) = 3.9675, y_{12} (s_{12}, u) = 2.9. Le min [y_{11} (s_{11}, u) , y_{12} (s_{12}, u)]=2.9.

L'action optimale est $a^*=a_2$. ;

A l'état 2 : $y_{23}(s_{23}, u) = 2.5125$, $y_{24}(s_{24}, u) = 5.64375$.

Le min $[y_{23}(s_{23}, u), y_{24}(s_{24}, u)]=2.5125$. D'où $a^*=a_3$.

A l'état 1, il est optimal d'actionner la suite (a_2, a_2, a_2, a_2) pour atteindre l'état 2. Après une transition dans l'état 2, employons les actions (a_3, a_3, a_3, a_3) jusqu'à ce qu'une transition dans l'état 1 se produira. Calculons la valeur de l'application contraction $(T_{0.5}u)$ par (5.1).

Les valeurs obtenues sont $(T_{0.5}u)(1)=5.09375$, $(T_{0.5}u)(2)=3.075$.

On utilise ces valeurs pour calculer les IAD. Deux autres itérations sont indispensables et qu'omettrons de reproduire. Les valeurs dans le tableau ci-dessous récapitulent les valeurs des indices d'allocation dynamiques en tout état.

Tableau 3.1 : Valeurs itératives des IADs.

	Itération 1	Itération 2	Itération 3	Itération 4
y_1	$y_1(a_2)= 2.9$	$y_1(a_2)= 2.921$	$y_1(a_2)= 2.9501$	$y_1(a_2)= 2.9761$
y_2	$y_2(a_3)= 2.5125$	$y_2(a_3)=2.5314$	$y_2(a_3)= 2.5350$	$y_2(a_3)= 2.71352$

Prenons les valeurs minimales des IAD en tout état.

Les figures 3.3 et 3.4 représentent l'évolution de la fonction d'indice à chaque état en fonction des itérations de l'algorithme. Une croissance est remarquée.

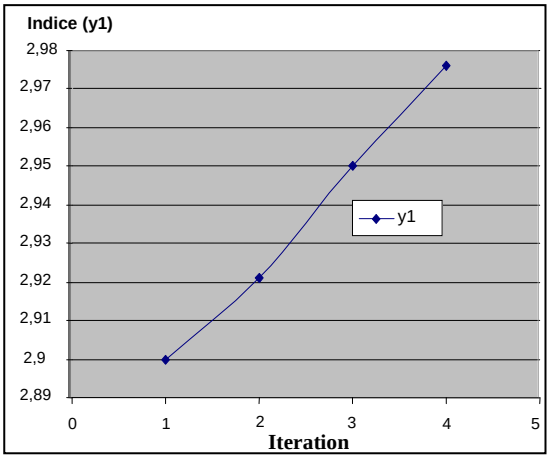


Figure 3.3 : IAD à l'état 1

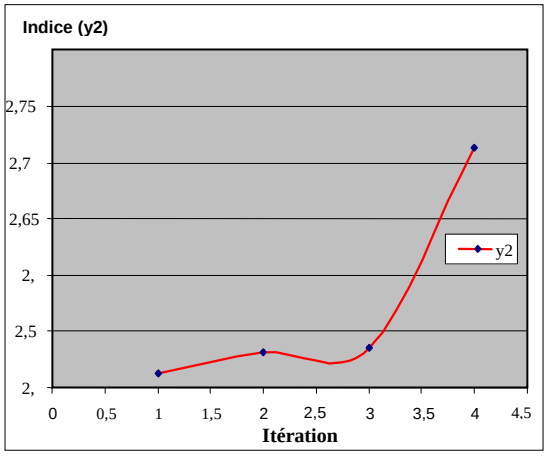


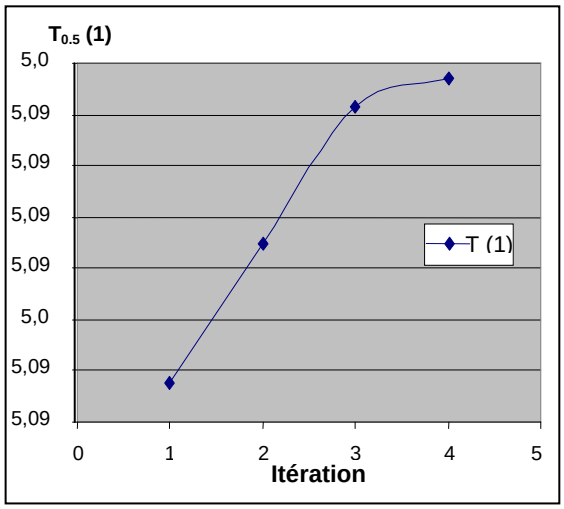
Figure 3.4 : IAD à l'état 2

Un second tableau résumera les valeurs de l'application T_α à chaque état en fonction des itérations successives.

Tableau 3.2 : Valeurs itératives de la contraction T_α

	Itération (1)	Itération (2)	Itération (3)	Itération (4)
$T_\alpha(1)$	5.09375	5.0943	5.09483	5.09494
$T_\alpha(2)$	3.075	3.07692	3.07818	3.07823

Une représentation graphique peut s'obtenir dans les figures 3.5 et 3.6. Elle représente la variation des valeurs $T_{0.5}(i)$ pour chaque état et à chaque itération :



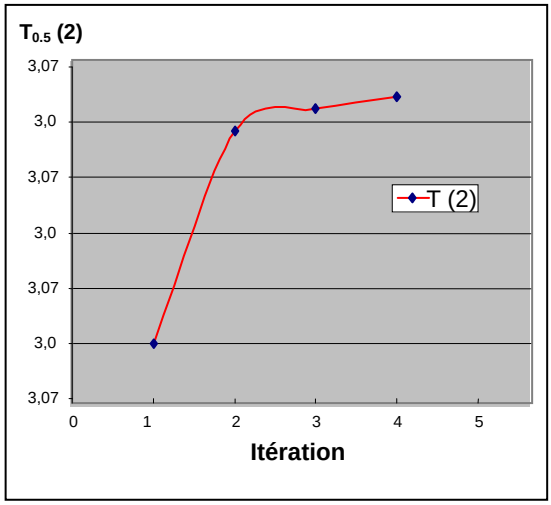


Figure 3.5: Variation de $T_{0.5}(1)$ à l'état 1 **Figure 3.6:** Variation de $T_{0.5}(2)$ à l'état 2.

Graphiquement, la valeur de la fonction coût $T_{0.5}$ augmente à chaque itération est converge dés l'itération 2 vers la fonction optimale. La stratégie optimale pour D (i ; $T_{0.5}^n$) est déduite par l'application du théorème 3.

Pour n ≥ 2, il faut toujours employer l'action a_2 à l'état 1. Après une transition à l'état 2, employer la suite d'actions (a_3, a_3, a_3, a_3) successivement jusqu'à ce qu'une transition à l'état 1 se produit. Nous trouvons que $(T_{0.5}^3u)(1)=5.09483$. $(T_{0.5}^3u)(2)=3.07713$.

Evidemment en calculant, les deux écarts

$| (T_{0.5}^3u)(1)- (T_{0.5}^2u)(1) | = 0.00053<0.001$ et

$| (T_{0.5}^3u)(2)- (T_{0.5}^2u)(2) | = 0.00005<0.001$, l'algorithme s'arrêtera.

La figure 2 nous montre la convergence vers la solution optimale.

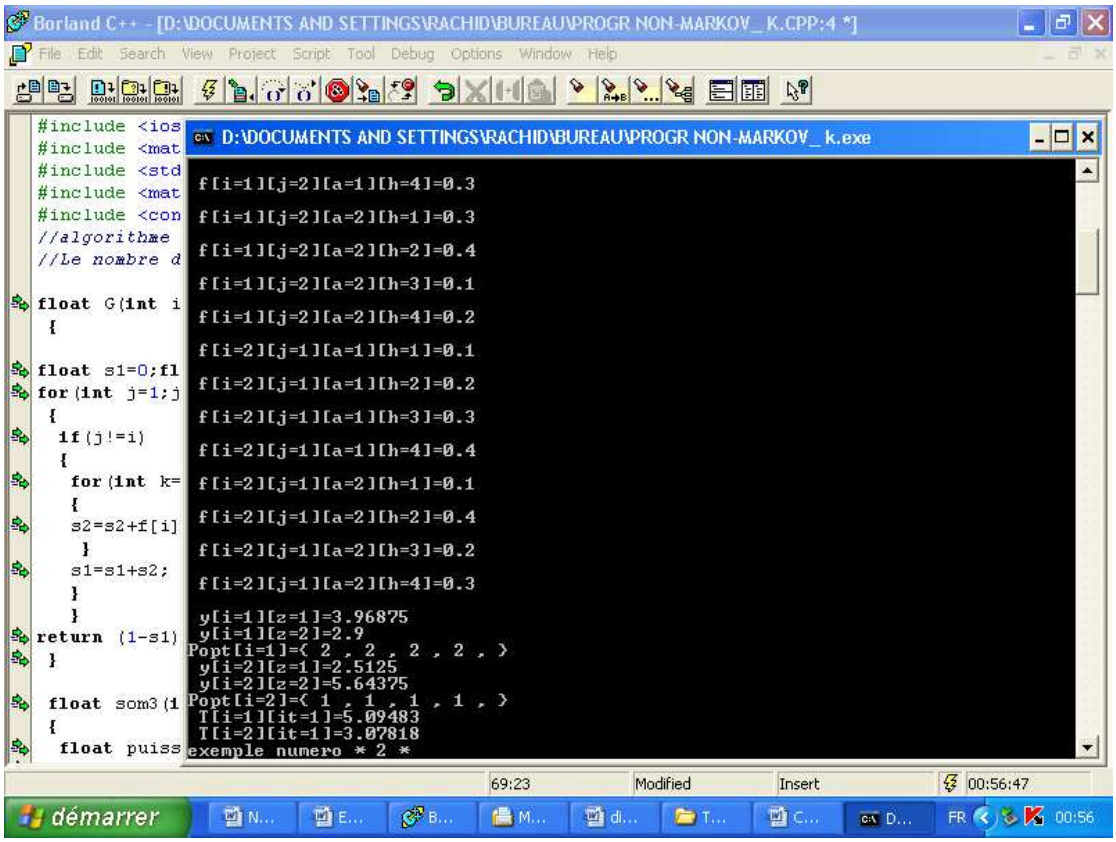


Figure 3.7 : Fenêtre de l'exécution du programme optimalement.

La stratégie optimale est d'employé l'action a_2 à l'état 1 et l'action a_3 à l'état 2..

8. Conclusion

Ce travail peut être étendu au cas où des coûts d'interruption du processus sont contractés.

Un travail ultérieur considérera le cas où la machine peut être soumise à la panne. La politique d'indice persistera t-elle à être optimale ? si oui qu'elle est la caractérisation des indices et comment les déterminer.

CHAPITRE 4

UN LOGICIEL DE DETERMINATION DE LA POLITIQUE OPTIMALE POUR L'EXECUTION D'UN PROCESSUS A 2-DECISIONS MARKOVIEN

1. Introduction

Nous avons choisi pour notre projet de concevoir un logiciel de gestion Log-version 1.0 de détermination de politique optimale pour un processus à 2-décisions markovien. Nous voulons de mettre à la disposition des étudiants un logiciel qui leur permet de s'initier à cette approche, ses fonctionnalités, ses tests d'intégration, de validation en utilisant le langage Borland C++, et l'environnement de l'interface Borland C++ Builder.

Nous allons détailler les étapes de conception, l'interface, le développement, l'utilisation du logiciel. Un exemple illustratif d'utilisation est même fourni.

2. L'interface du logiciel

Elle est claire, sobre et surtout très facile à utiliser. Une organisation des différentes fonctionnalités du logiciel sous forme d'une barre de menu déroulante et de boutons est proposée. L'interface graphique est formée d'une fenêtre principale et de plusieurs pages à l'intérieur. Les interfaces sous forme de multiples documents, sont confuses et difficiles à manipuler. L'utilisation des tableaux pour organiser les données n'a pas été omise. Chaque fonctionnalité est accessible par le menu, est représentée par une page armature dans la fenêtre principale qui doit s'afficher plein écran. Une couleur de fond simple et claire bleue est inséré. Les pages comportent des labels pour indiquer leur rôle.

3. Développement du logiciel

3.1. Borland C++ Builder

C++Builder est un langage de programmation, il regroupe tout un ensemble d'outils permettant d'effectuer un maximum de tâches de développement au sein du même environnement de travail. C++ Builder est de plus un environnement de développement visuel Borland C++, il permet de construire rapidement des applications et interagissant avec une base de données en utilisant des composants et simplifie au maximum l'écriture du code et la réalisation de l'interface.

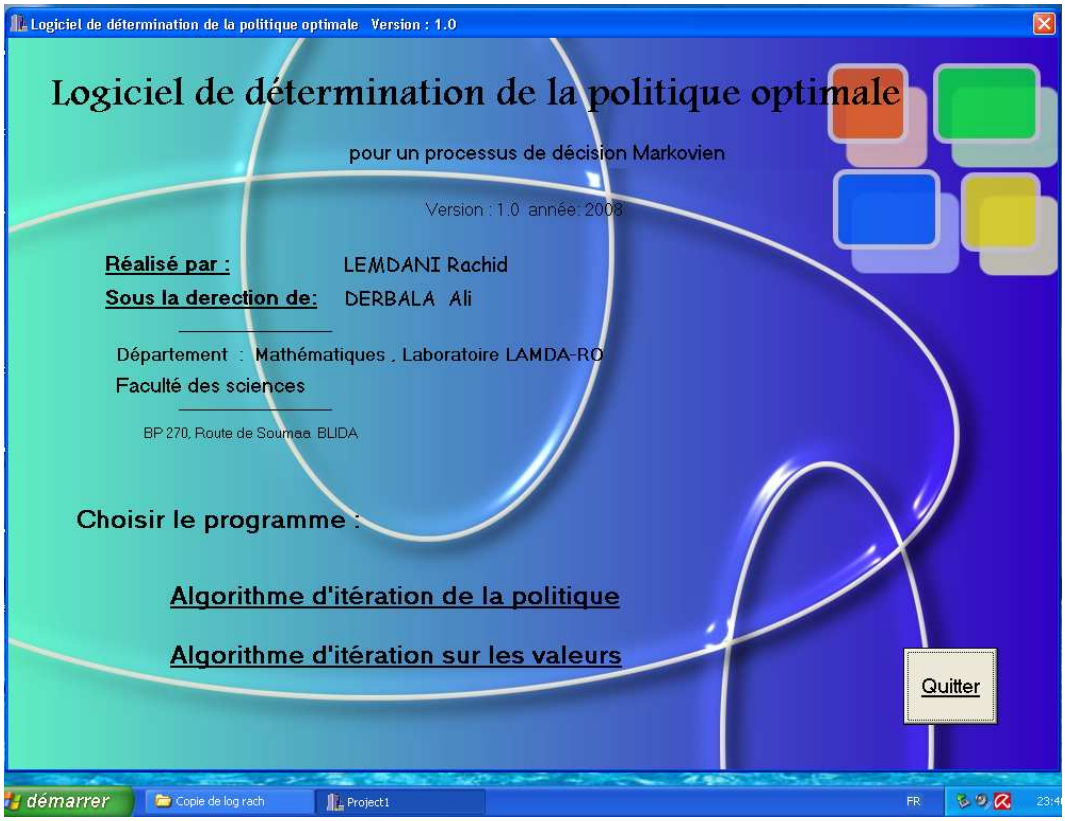
3.2. Analyse et Modélisation

Toutes les fonctionnalités sont intégrées à l'aide de composants visuels intégrés à l'interface et interagissant avec les deux programmes (PI) et (VI). La modélisation du logiciel suit le modèle de son interface. Cette interface comporte d'une manière simple une fenêtre principale et deux à l'intérieur accessibles grâce à une barre de menu. Les fenêtres sont composées de plusieurs composants visuels et de composants orientés qui permettent de faciliter le déplacement entre les fenêtres et l'affichage du résultat. Certains composants sont considérés comme d'événements et exécutent des méthodes en réponse.

3.3. Conception du logiciel

3.3.1. La fenêtre principale

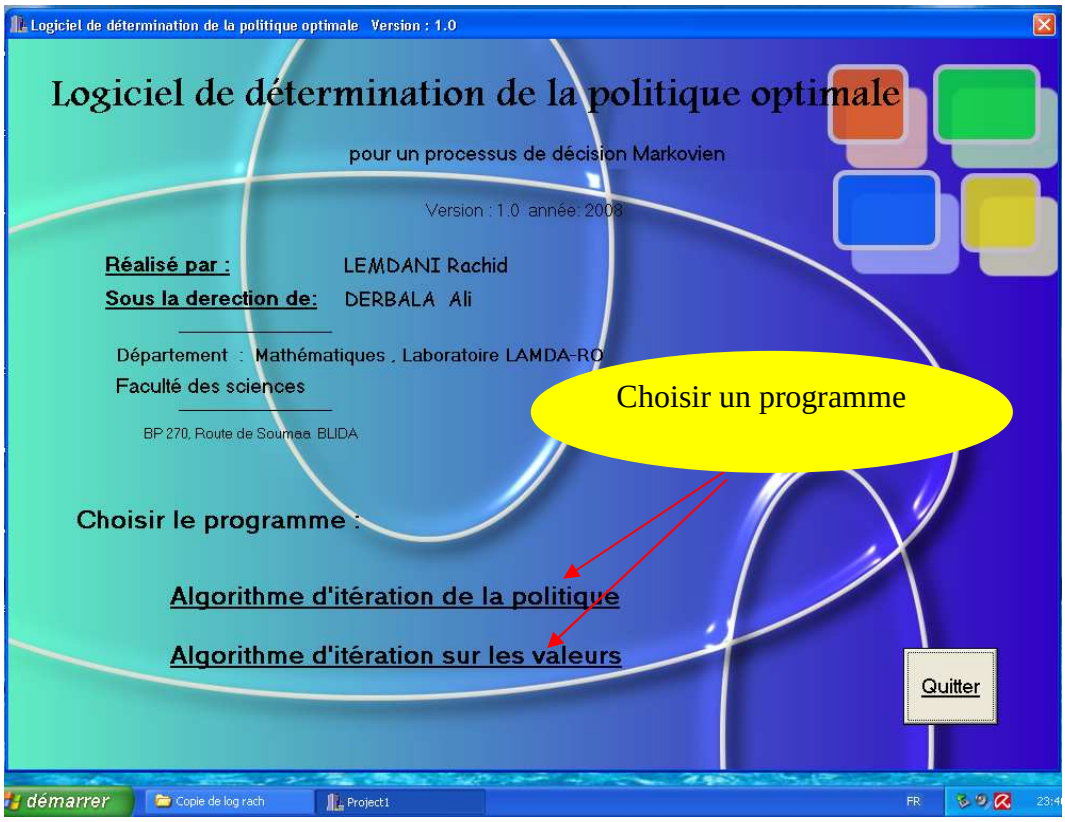
La fenêtre principale (écran 4.1) est un objet héritant. Elle est standard et est affichée en plein écran. Elle se compose de plusieurs boutons. Le bouton permettra de naviguer à travers les pages en utilisant une méthode qu'on désire afficher avec des opérations de mise à jour des composants de la page. La fenêtre principale facilite le déplacement aux autres fenêtres.



Ecran 4.1 : La fenêtre principale du logiciel

4. Utilisation du logiciel

☞ Lorsqu'on démarre le logiciel, la fenêtre principale apparaît. Elle comporte un choix de deux programmes, l'algorithme d'itération de la politique et l'algorithme d'itération sur les valeurs. On choisit celui qu'on désire utiliser comme il est indiqué dans l'écran suivant (4.2). en cliquant dessus.



Ecran 4.2 : Fenêtre du chois d'un programme.

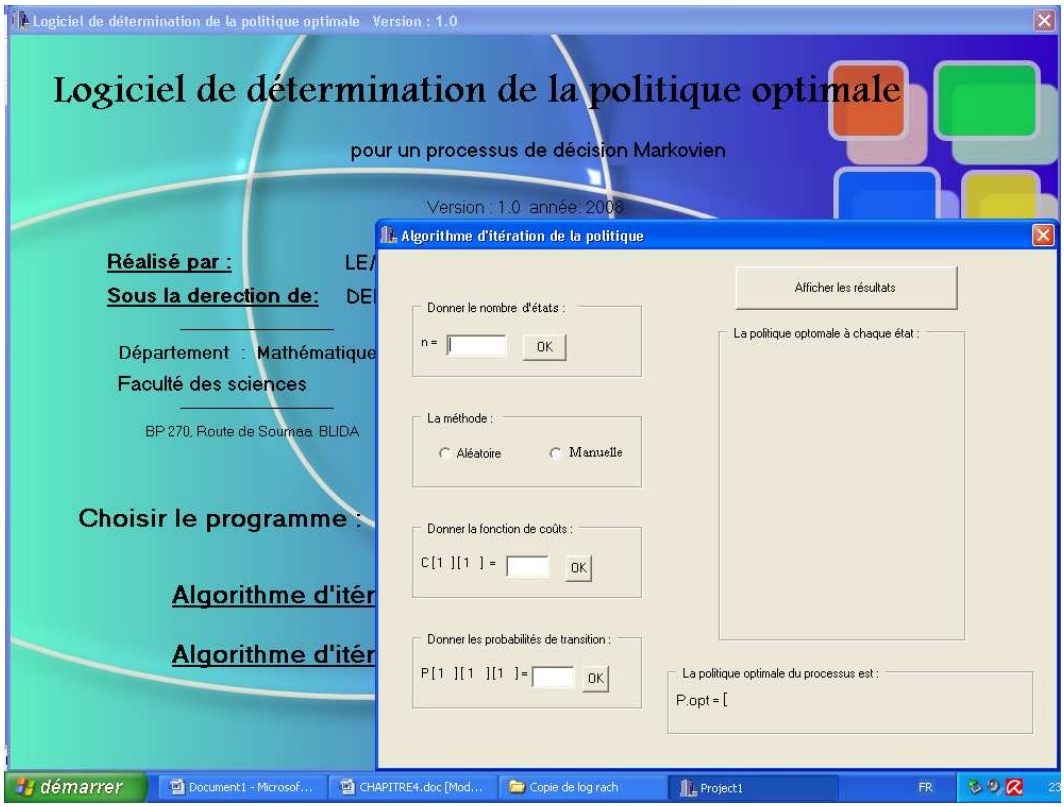
☞ si on veut quitter le logiciel on clique sur **Quitter** du même écran 4.2.

4.1. Utilisation de l'algorithme d'itération de la politique

Cliquer sur l'icône **Algorithme d'itération de la politique.**

Ecran 4.3 : Icône permettant d'ouvrir la fenêtre de l'algorithme d'itération de la politique.

Une fenêtre apparaît (écran 4.4) :



Ecran 4.4 : Fenêtre de l'algorithme d'itération de la politique.

Cette fenêtre contient plusieurs tableaux et boutons :

Dans le tableau 1, une case pour saisir le nombre d'états **(n)** du processus est disponible.

- Entrez le nombre d'états (n) dans la case correspondante.
- Cliquez sur le bouton **OK.** Puis on passe à l'étape suivante.

2. Un menu pour choisir la méthode d'entrer des coûts et des probabilités de transition **aléatoire** ou **manuelle**, est offert.
   - Sélectionnez la méthode désirée en cliquant dessus.

3. Une case pour entrer le coût à chaque état et pour toute action. La fonction de coûts est une matrice carrée de dimension ($|\Omega| \times |A|$), où Ω : l'ensemble des états et A : l'ensemble des actions. C[i][a] représente le coût encouru à l'état i quand l'action *a* est choisie . Les éléments de cette matrice sont de type **float** (réel).

- Si on choisit la méthode **Aléatoire** la fonction de coûts sera affichée avec une faible intensité de couleur ce qui indique que les valeurs des coûts sont données d'une manière aléatoire par le logiciel et on ne peut pas les remplir.

- si on choisit la méthode manuelle on va suivre les étapes suivantes :

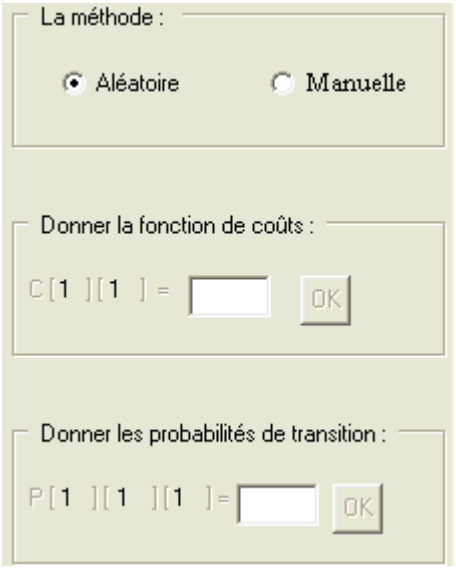
☞ Cliquez sur **Manuelle**, et noter les valeurs de coûts de notre exemple dans la case adéquate.

Commençons par le coût C[1][1] ; le coût encouru à l'état 1 lorsque l'action 1 est choisie, puis cliquez sur le bouton **OK** pour confirmer et aller à l'action suivante C[1][2]. Le numéro d'action augmente chaque fois quand on clique sur **OK** jusqu'au le dernier ; puis on fait la même chose pour l'état 2, et ainsi de suite pour tous les états. Afin de finir de remplir les valeurs de la matrice de coût le bouton **OK** est éteint; ce qui indique que le remplissage de la matrice de coût est fini.

4. Un tableau réservé pour entrer les valeurs de la matrice de probabilité de transition c'est une matrice tridimensionnelle de dimension (| Ω | × | A |× |Ω|), où Ω : l'ensemble des états et A : l'ensemble des actions. P[i][j][a] est la probabilité de transition de l'état i à l'état j quand l'action ***a*** est appliquée. Cette matrice est une matrice stochastique ; c'est-à-dire la somme de chaque ligne pour cette matrice est inférieure ou égale à 1. Les éléments de cette matrice sont de type **float** (réel).

Même chose pour la fonction de probabilité, comme on l'a déjà vu dans le paragraphe 3 par rapport à la fonction du coût.

- Si on choisit la méthode **Aléatoire** au début, alors la fonction du **coût** et la **probabilité de transition** sont donnés aléatoirement en même temps (écran 4.5).



Ecran 4.5 : La méthode **Aléatoire** : Le coût et la probabilité de transition sont affichés avec une faible intensité.

- Si on choisit l'autre méthode (**Manuelle**), on suivra les étapes suivantes :

☞ Cliquez sur **Manuelle**.

☞ Remplissage de la matrice de la probabilité de transition en suivant les étapes suivantes :

- la numérotation des actions et des états se fait automatiquement par le logiciel, ce que nous allons faire est de donner seulement les valeurs. On commence par la première valeur P[1][1][1] ; c'est la probabilité de transition de l'état 1 vers lui même 1, quand l'action 1 est choisie, on donne la valeur puis ; on clique sur **OK** pour passer à la valeur suivante P[1][1][2], et ainsi de suite. On introduit toutes les valeurs de probabilité de transition de l'état **i** vers l'état **j** quand l'action **a** est choisie. Une fois les valeurs de la matrice de probabilité de transition initialisées, le bouton **OK** est affiché avec une faible intensité.

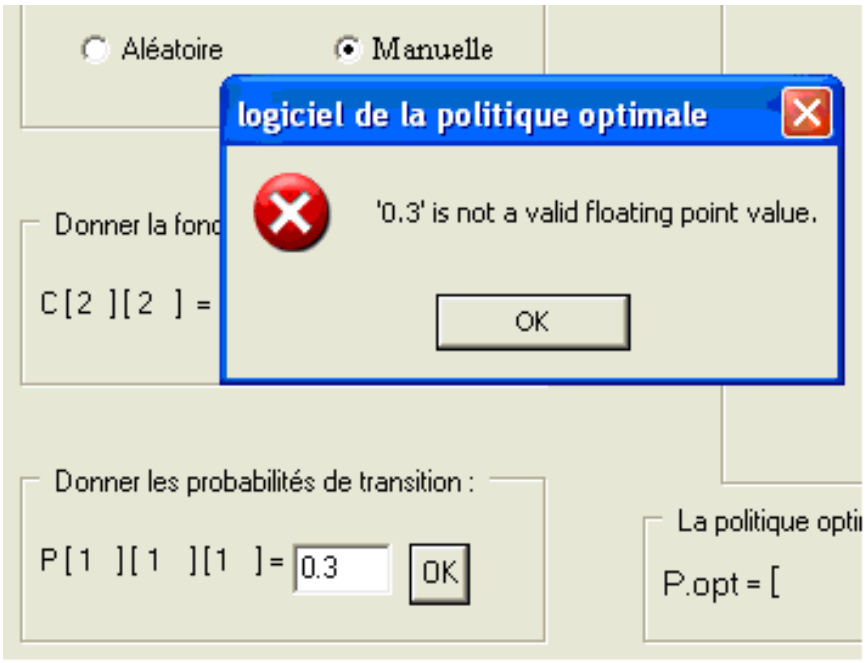
Après avoir entré le nombre des états, le coût encouru à chaque état et la probabilité de transition entre les états on vient à l'étape 5.

5. Un espace est réservé à l'affichage de la politique optimale pour le processus de décision Markovien (écran 4.6). Il est constitué d'un bouton **Afficher les résultats**, qui permet de donner le résultat si on clique dessus. Un cadre pour afficher l'action qui minimise le coût à chaque état et un autre pour la politique optimale du processus.

Ecran 4.6 : Tableau de l'affichage de la politique optimale.

Les mêmes démarches se feront si on choisit d'utiliser l'algorithme d'utilisation sur les valeurs.

En cas où on entrerait une valeur erronée, négative ou supérieure à 1 pour la matrice de transition, un message s'affichera nous indiquant que cette valeur est fausse. Par exemple : si on entre une valeur « 0.3 » au lieu de « 0,3 », le logiciel affichera le message suivant dans l'écran ci-dessous.



Ecran 4.7 : Message affiché par le logiciel s'il rencontre une erreur.

4.3. Exemple illustratif d'utilisation du logiciel

Soit un exemple d'un processus de décision Markovien à 3 états et 3 actions. Supposons que le processus soit aux états 1, 2 et 3 aux instants 1, 2, 3 successivement. Les coûts induits à chaque état, sous les différentes actions sont, donnés dans le tableau suivant :

Tableau 4.1 : Les valeurs des coûts en fonction des actions à chaque état.

	Action 1	Action 2	Action 3
Etat 1	4	5	2
Etat 2	1	0	2
Etat 3	7	3	5

Les probabilités de transition entre les états sous différentes actions sont résumées dans le tableau suivant :

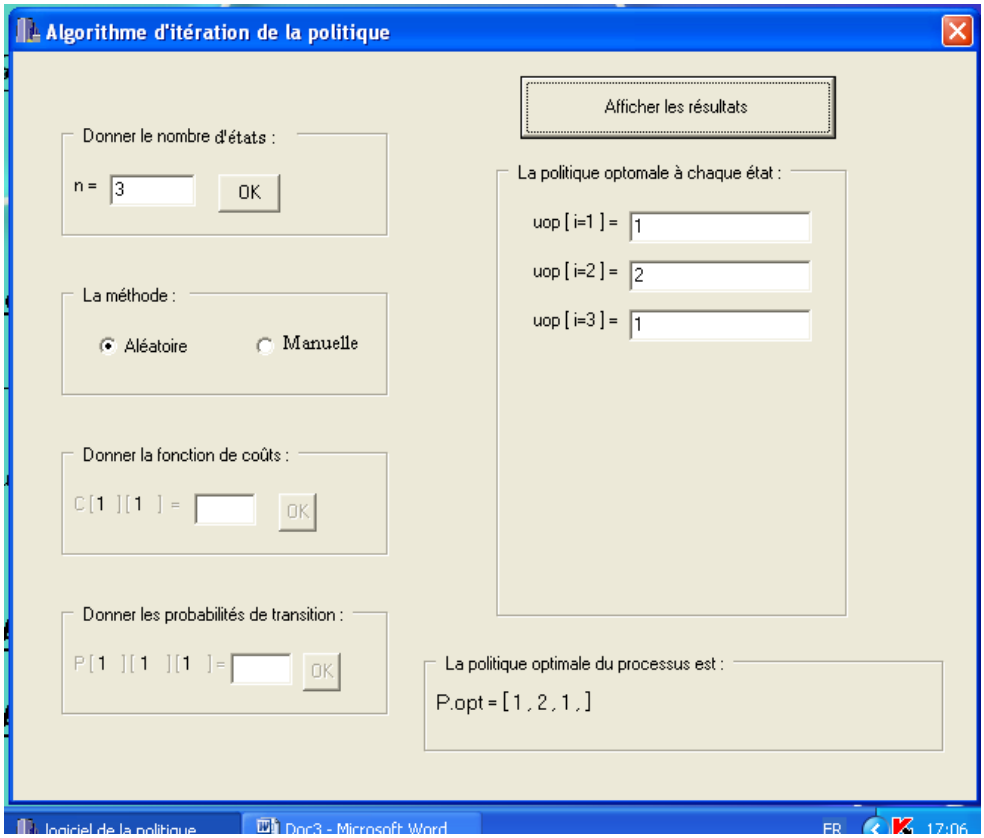
Tableau 4.2: La probabilité de transition entre les états sous différentes actions.

i	a_i	$P_{i1}(a_i)$	$P_{i2}(a_i)$	$P_{i3}(a_i)$
1	1	0.3	0.4	0.3
	2	0.2	0.5	0.3
	3	0	0.3	0.7
2	1	0.4	0.5	0.1
	2	0.6	0.1	0.3
	3	0.4	0.2	0.4
3	1	0.1	0.6	0.3
	2	0.4	0.2	0.4
	3	0.8	0.1	0.1

Algorithme d'itération de la politique

Après avoir loncé le logiciel et choisi l'algorithme d'itération de la politique, on va entrer le nombre des états (n) puis, on a le choix d'utiliser deux méthode (aléatoire ou manuelle).

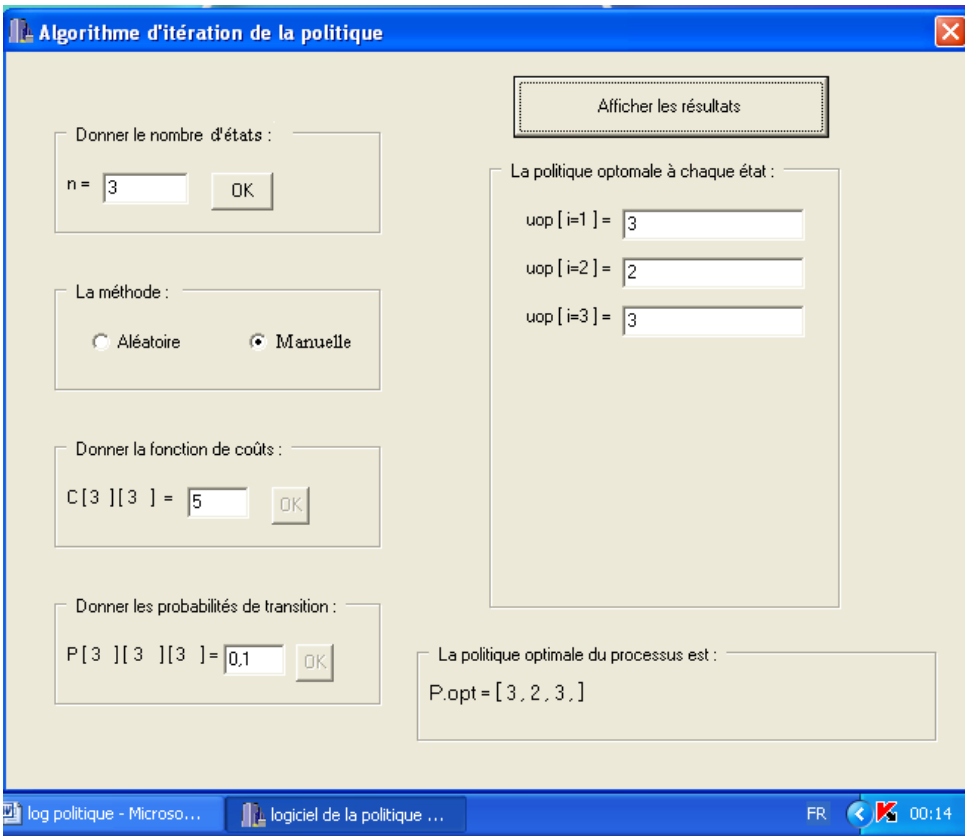
a) La méthode aléatoire : le coût et la probabilité de transition sont donnés par le logiciel, il suffit de cliquer sur **Afficher le résultat** pour trouver la politique optimale (écran 4.8) :



Ecran 4.8 : La politique optimale du processus en utilisant l'algorithme **PI** (Méthode aléatoire).

La politique optimale du processus est : P.opt = (1, 2, 1).

b) <u>La méthode manuelle</u> : Si on veut utiliser cette méthode, il faut d'abord cliquer sur le bouton **<u>Manuelle</u>** puis, on va donner toutes les valeurs de coûts et les probabilités de transition à chaque état sous différentes actions, et on va cliquer sur **<u>Afficher le résultat</u>** pour obtenir la politique optimale (écran 4.9).



Ecran 4.9: La politique optimale du processus en utilisant l'algorithme **PI** (Méthode manuelle).

La politique optimale du processus est : P.opt = (3, 2, 3).

La politique optimale d'un processus à décision Markovien pour les deux algorithmes si on utilise la méthode aléatoire est différente de celle de la méthode manuelle; car la matrice de probabilité et le coût ne sont pas les mêmes dans les deux cas.

- Si on veut utiliser l'algorithme d'itération sur les valeurs on suivi les mêmes étapes de l'algorithme d'itération de la politique dans le paragraphe précédente.

Dans ce logiciel on considère que le nombre des états est égal ou inférieur au nombre d'actions, car en général si le nombre des actions est plus grand que le nombre des états, la détermination d'une politique optimale est NP-complet.

CONCLUSION ET PERSPECTIVES

Les processus à 2-décisions Markoviens ont été notre outil de résolution des problèmes d'ordonnancement stochastiques. Les algorithmes d'itération de la politique (PI) et d'itération sur les valeurs (VI) ont été introduits, exposé et programmés en utilisant un langage évolué. L'implémentation a été réalisée sur un PC Intel Pentium 4 doté d'un processeur 2.26 GHz et de 352 Mo de Ram. Le premier algorithme est itératif qui utilise deux étapes, une d'*´evaluation* de la politique courante et l'autre d'*amélioration* de cette politique. Si la politique courante ne peut plus être améliorée au bout d'un nombre d'itérations, son optimalité est garantie. Lorsque les espaces d'états et d'actions sont trop grands, seule une évaluation approchée de la politique courante est possible. La phase d'amélioration peut être simplifiée aussi, afin de gagner en temps et en espace. Nous avons testé sur des exemples l'efficacité de ces algorithmes pour de différentes valeurs d'états et d'actions. Sur chaque exemple, la politique optimale trouvée par les algorithmes PI et VI s'est révélée être une politique presque optimale. Nous avons effectué une comparaison entre les solutions obtenues par l'algorithme PI et VI. La complexité des deux algorithmes a été calculée sur la taille de l'espace $|\Omega|\times|A|\times|\Omega|$. On a réussi à tester ces algorithmes sur des processus à huit états et huit actions à entreprendre. Les solutions obtenues par l'algorithme PI et VI sont de bonnes qualités. L'algorithme d'itération sur les valeurs est moins efficace que l'algorithme d'itération de la politique. Des difficultés surviennent lorsque le nombre d'actions est plus grand que le nombre d'états. Dans ce cas, la politique courante ne peut pas être optimale. Le problème est NP-difficile.

Le cas non Markovien est résolu par une règle d'indice. Nous avons étudié cette classe du processus en programmant un algorithme itératif de détermination d'une politique optimale. Cet algorithme fonctionne en deux étapes. On calcule les IADs à chaque état et en tout instant pour déterminer la stratégie optimale et on assure l'optimalité de cette politique par la convergence de la fonction de contraction T_α définie dans le mémoire.

Des difficultés dans le calcul des IADs et la fonction T_α lors de l'exécution de ce programme sont remarquées. Une difficulté est apparue lors de changement des actions entre les états pour déterminer la stratégie optimale. Pour que l'algorithme converge vers la politique optimale, nous engendrons une augmentation de la taille de l'espace mémoire. Nous rappelons que la résolution d'un processus à décision markovien ou non-markovien se trouve facilement confrontée à des difficultés dûes aux nombres d'états et d'actions qui explosent. Dans la littérature, très peu de publications sont dédiées aux cas non-markoviens. Beaucoup de problèmes restent ouverts.

ANNEXE

Vous trouvez ci-dessous les programmes élaborés en utilisant le langage C++ Borland.

A. ALGORITHME D'ITERATION SUR LES VALEURS

```
#include<iostream.h>
#include<math.h>
#include<conio.h>
void main(){
//algorithme d'itération sur les valeurs
int m, it, h, n, i, j, z, k, l, t ;
int uop[10];
t=0;
it=0;
float  minop[10], min, cou[10], so1, v1[10], v2[10], p[10][10][10];
float alpha=0.95;
float ebs=0.01;
float c[10][10];
float q[10][10];
//*****************************************************************
//Le nombre d'exemple
cout<<"\nle nombre d'exemple est l= ";cin>>l;
for(k=1;k<l+1;k++)
{
cout<<"\nexemple numero * "<<k<<" *";
cout<<"\n donner le nombre des etats (n):";cin>>n;

cout<<"\n donner le nombre des actions (m):";cin>>m;
cout<<"\n donner la fonction de couts c(i,a):";
for ( i=1;i<=n;i++)
{
for (int z=1;z<=m;z++)
{
cout<<"\n c[i="<<i<<"][z="<<z<<"]=";
cin>>c[i][z];
}
}
cout<<"\n donner la prbabilite de transitions:";
for( i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
```

```
for(int z=1;z<=m;z++)
{
 cout<<"\n p[i="<<i<<"][j="<<j<<"][z="<<z<<"]=";
 cin>>p[i][j][z];
}
}
}
//le programme principale************************************************
cout<<"\n initialiser les couts:";
for(i=1;i<=n;i++)
{cout<<"\n v1[i="<<i<<"]=";
cin>>v1[i];}
while(t<n)
{
it=it+1;
t=0;
for(i=1;i<=n;i++)
{
cout<<"\n i="<<i;
for(z=1;z<=n;z++)
{
for(j=1;j<=n;j++)
{
so1=0;
so1=so1+alpha*p[i][j][z]*v1[i];
}
cou [z]=c[i][z]+so1;
}
 min=cou[1];
 for(h=2;h<=n;h++)
{
if(cou [h]<min )
{
min =cou [h];
}
}
v2[i]=min;
cout<<"\n min="<<min;
if( v2[i]>v1[i])
{
if( (v2[i]-v1[i])<((ebs*(1-alpha))/(2*alpha)))
{
t=t+1;
}
}
 else
{
 if( (v1[i]-v2[i])<((ebs*(1-alpha))/(2*alpha)))
{
 t=t+1;
```

```
}
}
 cout<<"\n v2="<<v2[i];
 cout<<"\n v1="<<v1[i];
 cout<<"\n abs="<<abs(v2[i]-v1[i]);
 cout<<"\n abs1="<<((ebs*(1-alpha))/(2*alpha));
v1[i]=v2[i];
}
}
cout<<"\n it="<<it;
cout<<"\n t="<<t;
for(i=1;i<=n;i++)
{
for(z=1;z<=n;z++)
{
for(j=1;j<=n;j++)
{
so1=0;
so1=so1+alpha*p[i][j][z]*v2[i];
}
q[i][z]=c[i][z]+so1;
cout<<"\n q[i="<<i<<"][z="<<z<<"]="<<q[i][z];
}
}
 for(i=1;i<=n;i++)
{
minop[i]=q[i][1];
uop[i]=1;
for(z=2;z<=n;z++)
{
if(q[i][z]<minop[i])
{
minop[i]=q[i][z];
uop[i]=z;
}
}
cout<<"\n determination de la politique optimale";
cout<<"\n uop[i="<<i<<"]="<<uop[i];
}
}
```

B. ALGORITHME D'ITERATION DE LA POLITIQUE

```
#include <iostream.h>
#include <time.h>
#include <math.h>
#include <conio.h>
void main(){
//algorithme d'itération de la politique.
//Le nombre d'exemple
```

```
int m ,k;
cout<<"Le nombre des exemples a faire est ";cin>>m;
for (k=1;k<=m;k++)
{
cout<<"\n exemple numero * "<<k<<" *";
float alpha=0.5;
float min[10];
float ebs=0.01;
float s1,s2;
float v[10][10][1000];
float p[10][10][10];
float q[10][10];
float c[10][10];
int uop[10];
 int i,z,n,j,it,a;
cout<<"\n donner le nombre des etas (n):";cin>>n;
cout<<"\n donner la fonction de cout.";
for( i=1;i<=n;i++)
{
for( z=1;z<=n;z++)
{
cout<<"\n c[i="<<i<<"][z="<<z<<"]=";cin>>c[i][z];
}
}
cout<<"\n donner la probabilite de transition.";
for (i=1;i<=n;i++)
{
for (j=1;j<=n;j++)
{
for (z=1;z<=n;z++)
{
cout<<"\n p[i="<<i<<"][j="<<j<<"][z="<<z<<"]=";cin>>p[i][j][z];
}
}
}
a=0;
do
{
it=1;
for ( i=1;i<=n;i++)
{
v[i][z][it]=0;
}
do
{
for ( i=1;i<=n;i++)
{
s1=0;
for ( j=1;j<=n;j++)
{
```

```
if(j!=i)
{
s1=s1+p[i][j][z]*(c[i][z]+alpha*v[i][z][it]);
}
}
v[i][z][it+1]=s1;
}
it=it+1;
}
while(abs(v[i][z][it+1]-v[i][z][it])>=ebs);
for ( i=1;i<=n;i++)
{
for ( z=1;z<=n;z++)
{   s2=0;
for ( j=1;j<=n;j++)
{
 if (j!=i)
{
s2=s2+p[i][j][z]*(c[i][z]+alpha*v[j][z][it+1]);
}
}
q[i][z]=s2;
cout<<"\n q[i="<<i<<"][z="<<z<<"]="<<q[i][z];
}
}
min[i]=q[i][1];
uop[i]=1;
for(i=1;i<=n;i++)
{
for (z=1;z<=n;z++)
{
if(q[i][z]<=q[i][1])
{
min[i]=q[i][z];
uop[i]=z;
}
}
cout<<"\n min[i="<<i<<"]="<<min[i];
cout<<"\n la politique optimale à l'état i="<<i<<"est:"<<uop[i]<<endl;
}
a=a+1;
}
while(a>n);
}
getch();
}
```

C. ALGORITHME D'ITERATION DE LA VALEUR POUR UN PROCESSUS A DECISION NON-MARKOVIEN

```
#include <iostream.h>
#include <math.h>
#include <conio.h>
//algorithme d'itération pour les procéssus à décision non-markovien
//Le nombre d'exemple
float G(int i,int z,int s,int sia,int n,float f[10][10][10][10])
 {
float s1=0;float s2=0;
for(int j=1;j<=n;j++)
{
if(j!=i)
{
for(int k=1;k<=sia+s;k++)
{
 s2=s2+f[i][j][z][k];
}
s1=s1+s2;
}
}
return (1-s1);
 }
 float som3(int r,int i,int z,int sia,int n,float alpha,float c[10][10],float
f[10][10][10][10])
{
float puiss=1;float s3=0;
for (int s=0;s<=r-1;s++)
{
puiss=puiss*alpha;
s3=s3+puiss*c[i][z]*G(i,z,s,sia,n,f);
}
return s3;
}
float som4(int i,int z,int s,int sia,int n,int r,float alpha,float u1[10],float
f[10][10][10][10])
{
float s5=0,s4=0;float puiss=1;
for (int j=1;j<=n;j++)
{
if (j!=i)
 {
puiss=puiss*alpha;
 s5=s5+puiss*u1[j]*f[i][j][z][s];
}
 s4=s4+s5;
}
return s4;
```

```
}
 float yia(int i,float z,int s,int r,float sia,int n,float alpha,float c[10][10],
float f[10][10][10][10],float u1[10])
{
float puiss=1;
for (int g=1;g<r;g++)
puiss=puiss*alpha;
return (som3(r,i,z,sia,n,alpha,c,f)+som4(i,z,s,sia,n,r,alpha,u1,f))/(G(i,z,0,sia,n,f)-
puiss*G(i,z,r,sia,n,f));
}
void main(){int l,k;
cout<<"\n le nombre d'exemple est l= ";cin>>l;
for(k=1;k<l+1;k++)
{
cout<<"\nexemple numero * "<<k<<" *";
float alpha=0.5, u1[10],  c[10][10];
int  r, i, z, s, sia, n, t, it, j, indice;
float puiss, ebs=0.01;
float so1[10][10], so2[10][10],  so3[10][10], u[10][10000], f[10][10][10][10];
cout<<"\n donner le nombres des etats (n):";cin>>n;
cout<<"\n donner le nombres des etapes (r):";cin>>r;
cout<<"\n donner le temps initial (sia):";cin>>sia;
cout<<"\n donner la fonction de couts:";
for(int i=1;i<=n;i++){
for(int z=1;z<=n;z++){
cout<<"\n c[i="<<i<<"][z="<<z<<"]=";cin>>c[i][z];
}
}
cout<<"\n donner la probabibilite de transition.";
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
if(i!=j)
{
for(int z=1;z<=n;z++)
{
for(int h=1;h<=r;h++)
{
cout<<"\n f[i="<<i<<"][j="<<j<<"][a="<<z<<"][h="<<h<<"]=";
cin>>f[i][j][z][h];
}
}
}
}
}
float y[10][10];
for(int d=0;d<=10;d++)u1[d]=0;
for( i=1;i<=n;i++)
{
```

```
for(z=1;z<=n;z++)
{
y[i][z]=yia(i,z, s,r,sia,n,alpha,c,f,u1);
cout<<"\n y[i="<<i<<"][z="<<z<<"]="<<y[i][z];
}
 indice=1;
int popti[10];
float minyia=y[i][1];
for(int hh=1;hh<=n;hh++)
{
 if(minyia>y[i][hh])
{
 minyia=y[i][hh];
 indice=hh;
}
}
cout<<endl<<"Popt[i="<<i<<"]=
{ ";
for(int dd=1;dd<=r;dd++)
{
 popti[dd]=indice;
cout<<indice<<" , ";
}
cout<<"
}";
}
it=1;
do
{
for ( i=1;i<=n;i++)
{
for (z=1;z<=n;z++)
{
puiss=1;
for ( t=1;t<=r;t++)
{
 puiss=puiss*alpha;
for ( j=1;j<=n;j++)
{
if(j!=i)
{
u[j][it]=0;
so1[j][z]=0;
so2[j][z]=0;
so1[j][z]=so1[j][z]+(f[i][j][indice][t]*u[j][it]*puiss);
so2[i][z]=so1[j][z]+c[i][indice];
so3[i][z]=c[i][indice]+(f[i][j][indice][t]*so2[i][z]*puiss);
}
}
}
```

```
}
u[i][it+1]=so3[i][1];
for(z=2;z<=n;z++)
{
if(so3[i][z]<=so3[i][1])
{
u[i][it+1]=so3[i][z];
cout<<"\n T[i="<<i<<"][it="<<it<<"]="<<u[i][it+1];
}
}
}
it=it+1;
}
while((u[i][it+1]-u[i][it])>=ebs);
}
getch();
}
```

D. ALGORITHME PERMET DE DONNER UNE MATRICE STOCHASTIQUE

```
#include <dos.h>
void main()
{
double p[10][10][10];
int n=7;
for(int z=0;z<n;z++)
{
for(int i=0;i<n;i++)
{
double a=0;
for(int j=0;j<n-1;j++)
{
p[i][j][z]=(rand() %(1000/n));
a=a+p[i][j][z];
}
p[i][n-1][z]=1000-a;
}
}
for(int z=0;z<n;z++)
{
for(int i=0;i<n;i++)
{
for(int j=0;j<n;j++)
{
p[i][j][z]=p[i][j][z]/1000;
}
}
}
//////////////////////////////////////////////////////
```

```
for(int z=0;z<n;z++)
{
for(int i=0;i<n;i++)
{
for(int j=0;j<n;j++)
{
cout<<p[i][j][z]<<"  ";
}
cout<<endl;
}
}
 getch();
}
```

REFERENCES

1. ALANI. T, "Introduction à la programmation dynamique stochastique (PDS), processus de décision de Markov (PDM) " Laboratoire A²SI-ESIEE-Paris , (Janvier 2005), e-mail: t.alani@esiee.fr, 1-55.

2. Blackwell, D (1965). "Discounted dynamic programming," Annals of Mathematical Statistics, 36, pp. 226-235.

3. CALENGE. N, "Processus décisionnels de Markov Multi-agents" DEA Informatique-Université de CAEN, 14000 Caen Cedex – France. e-mail : ncalenge@etu.info.unicaen.fr , (septembre 2002), 11-23.

4. Derbala, A., (2002), "Un ordonnancement dynamique de tâches stochastiques sur un seul processeur", *RAIRO, Operations Research*, N°6, 365-373.

5. Derbala, A (2004). Problématique d'un ordonnanceur dans un système d'exploitation des ordinateurs multitâches monoprocesseurs. Thèse de " Doctorat d'Etat " en mathématiques option Recherche opérationnelle, USTHB, Alger.

6. FRODUALD. K, "Planification en IA -Planification non déterministe (Chapitres 16 & 17)" IFT 702, Eté 04, pp. 13-30.

7. Gittins, J.C et D.M. Jones (1972). A dynamic index for the sequential Design of experiments. *Colloquia Mathematica JANES BOLAI*, 9, European meeting of statisticians, Budapest, Hongrie, pp.241-266.

8. Glazebrook, K.D (1976). Stochastic scheduling. Phd Thesis. Downing College, Cambridge.

9. Gittins. J. C. et K.D. Glazebrook (1977). On Bayesian models in stochastic scheduling. Appl. Prob. 14., pp. 556-565.

10. Glazebrook, K.D (1978). On a class of non-markov decision processes. J. Appl. Prob. 15, pp.689-698.

11. Gittins, J. C. (1979), "Bandit processes and dynamic allocation indices (with discussion)", J. *Roy. Statist. Soc. Ser. B* 41, 2, 148-177.

12. Gittins, J.C., (1982), "Forwards induction and dynamic allocation indices", M. A. H.Dempster et al(eds.), Deterministic and Stochastic Scheduling, 125-156. Nato advanced Study Institutes Series, Reidel.

13. Glazebrook K.D, (1983), "Methods for the evaluation of permutations as strategies in stochastic scheduling problems ".

14. Gittins, J.C, (1989), "Multi-armed Bandit Allocation Indices", John Wiley & Sons.

15. Gérald D,"Système et réseaux de télécommunication en régime stochastique", Masson, Paris Milan Barcelone Mexico 1989.

16. Gotha. Les problèmes d'ordonnancement. Recherche opérationnelle/Operations, Research. Vol.27, n°1, 1993, pp. 77- 150.

17. Kali, A (2007). Les indices de Gittins dans les ordonnancements stochastiques : Existence, caractérisation et détermination. Thèse de Magister, Département de mathématiques, USDBlida, 15 Mars 2007.

18. Lippman, S, A (1971). Maximad average-reward policies for semi-Markov decision processes with arbitrary state and action space. Ann. Math. Statist, 42. 1717-1726.

19. Lemdani Rachid et Derbala Ali. Ordonnancement de tâches stochastiques non markoviennes. Actes du Colloque international MOAD 2007, Méthodes et outils d'aide à la décision. Session : ordonnancement et gestion de production, Béjaia, Algérie, 18, 19 et 20 Novembre 2007, pp. 775-781.

20. Miller, B. L. (1968) Finite state continuous time Markov decision processes with an infinite planning horizon. J. Math. Anal. Appl, 22, 552-569.

21. Nash, P (1973). Optimal allocation of resources between research projects, *Ph.D. Thesis*, Cambridge University.

22. PHILIPPE. P, "Apprentissage par renforcement. Notes de cours" philippe.preux@univ-lille3.fr GRAPPA, (Novembre 2004), p. 7-9.

23. R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.

24. Ross. S. M, (1968) Arbitrary state Markovien decision processes. Ann. Math.

Statist 39. 2118-2122.

25. Ross. S.M, (1970) Applied Probability Models with Optimization Applications. Holden-Day. San Francisco.

26. Robinson D.R.(1982). Algorithms for evaluating the dynamic allocation index. *Oper. Res. Lett.* **1** (1982) 72-74.

27. RUHLMANN. C, "L'apprentissage par renforcement (The reinforcement learning)- Rapport 3", (Février 2004), p. 1-13.

28. Sonin, I. M, (2005), "The optimal Stopping of a Markov Chain, the Generalized Gittins Index, and Recursive Solution of Poisson and Bellman Equations", http://www.math.uncc.edu/~imsonin.

29. Varaiya, P, Walrand, J, Buyukkoc, C., (1985), "Extensions of the Multiarmed Bandit Problem: The Discounted Case". *IEEE Trans.Autom. Control,* AC-30, 5, 426-439.