UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté des Sciences

Département de mathématiques

MEMOIRE DE MAGISTER

En Mathématiques

Option : Recherche Opérationnelle

Thème :

LES INDICES D'ALLOCATION DYNAMIQUES DANS L'ORDONNANCEMENT STOCHASTIQUE

Existence, caractérisations et détermination

Par

KALI Abdesselem

Devant le jury composé de :

M. Blidia	Professeur, U.S.D., Blida	Président
A. Derbala	Maître de conférences, U.S.D., Blida	Promoteur
K. Boukhetala	Professeur, U.S.T.H.B., Alger	Examinateur
S. Bouroubi	Maître de conférences, U.S.T.H.B., Alger	Examinateur
H. Ould Rouis	Maître de conférences, U.S.D., Blida	Examinateur

Blida, avril 2007

RÉSUMÉ

Dans un atelier, " N " tâches sont à exécuter sur une machine afin de maximiser l'espérance de la somme des gains linéaires et prévisionnels. Les temps d'exécution des tâches sont supposés aléatoires de lois connues. Les problèmes où les temps d'exécution des tâches sont aléatoires sont dits stochastiques. Une approche théorique pour les résoudre est la théorie des processus bandits, des processus de décision semi-Markoviens. Un exemple de problème d'ordonnancement en temps réel modélisé par ces derniers est exposé. On associe à chaque tâche une priorité dynamique appelée indice d'allocation dynamique et est notée I.A.D. En tout instant, on exécute la tâche qui a le plus grand indice. Ces indices sont calculés en tout instant et durant l'exécution de la tâche. En cas de conflit ou d'égalité entre les plus grands indices, on arbitrera en choisissant une tâche selon une règle connue de type SPT, LPT, FIFO…etc. Si l'objectif du problème d'ordonnancement stochastique est une fonction à coûts séparables, la politique d'indices est montrée optimale. Les preuves de l'existence et de la caractérisation des I.A.D sont données en détail. De la bibliographie, trois algorithmes de détermination des I.A.D existent et ont fait l'objet d'une étude approfondie. Un nouvel algorithme de détermination des I.A.D est proposé et est exposé. Ces quatre algorithmes ont été implémentés en utilisant un langage évolué de programmation. Des expérimentations numériques sur un grand nombre d'exemples de problèmes d'ordonnancement, de l'ordre de mille, ont été effectuées. Un générateur de jeux d'essais est indispensable et est réalisé. Une étude comparative entre ces algorithmes est confectionnée. Si le taux d'actualisation est proche de zéro, notre nouvel algorithme peut prendre en charge des problèmes à cent soixante états. Le temps de calculs est négligeable. Un logiciel I.A.D version 1.0 a été réalisé. Son fonctionnement et aide d'utilisation sont aussi fournis.

Mots-clés : Ordonnancement stochastique, Processus bandits, Indices d'allocation dynamiques, Stratégies préemptive et non préemptive.

ABSTRACT

In a workshop, "N" tasks are to be carried out on a machine in order to maximize the expected sum of the linear and discounted profits. The processing times of the tasks are supposed to be random variables. The problems when the processing times of the tasks are randoms are known as stochastic. A theoretical approach to solve them is the theory of the bandit processes, a semi-Markov decision processes. An example of real-time scheduling problems formulated as a bandit process is presented. At each task we associate a dynamic priority called dynamic allocation index and is noted D.A.I. In any time, we carried out the task which has the greatest index. These indices are calculated in any time and during the continuation of the task. We break the tie by choosing a task according to a known rule of type SPT, LPT, FIFO... etc. If the objective of the stochastic scheduling problem is separable cost function, the index policy is shown to be optimal. The proofs of the existence and the characterization of the D.A.I are given in detail. In references, three algorithms of determination of the D.A.I exist and they are our subject of a thorough study. An improved algorithm of determination of the D.A.I is proposed and is exposed. These four algorithms were implemented by using a high level language of programming. Numerical experiments on a great number of examples of scheduling problems, about thousand, were carried out. A generator of the test is essential and is realized. A comparative study between these algorithms is made. If the discounted factor is near to zero, our new algorithm can deal with problems in a hundred and sixty states. The computing time is negligible. A software tool D.A.I version 1.0 was realized. Its operation and helps of use are also provided.

Key words: Stochastic scheduling, Bandit processes, Dynamic allocation indices, Strategies pre-emptive and non pre-emptive.

[illegible]

[illegible] "N" [illegible]
[illegible]
[illegible]
[illegible]
[illegible] I.A.D [illegible]
[illegible]
[illegible] FIFO [illegible] PT [illegible] PT [illegible]
.I.A.D [illegible]
[illegible] I.A.D [illegible]
[illegible]
[illegible]
[illegible]
.0.1 [illegible] I.A.D [illegible]
[illegible]

REMERCIEMENTS

J'exprime ma gratitude envers mon directeur de recherche **Derbala Ali** maître de conférences à l'université Saad Dahleb de Blida, pour son acceptation de me proposer et de diriger les travaux de cette recherche. Je le remercie également pour ses précieux conseils, sa disponibilité et son aide.

Je remercie le professeur **Blidia Mostafa,** Professeur à l'université Saad Dahleb de Blida, pour l'honneur de présider ce jury.

Mes profonds remerciements vont aux Professeurs qui ont accepté d'examiner ce travail et de participer à ce jury, à savoir

Boukhetala Kamel Professeur à l'université Houari Boumediene d'Alger

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

Ould Rouis Hamid Maître de conférences à l'université Saad Dahleb de Blida.

TABLE DES MATIÈRES

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

INTRODUCTION

Dans un atelier, "N" tâches sont à exécuter sur une machine afin de maximiser l'espérance de la somme des gains linéaires et prévisionnels. Chaque sorte de tâche répond à une demande spécifique qui possède généralement des fluctuations aléatoires autour d'une valeur moyenne. La présence de ces fluctuations ne permet pas de connaître précisément les demandes futures et impose donc une grande réactivité du gestionnaire de l'atelier afin d'assurer la satisfaction de sa clientèle. Les temps d'exécution des tâches sont supposés aléatoires de lois connues. Les problèmes où les temps d'exécution des tâches sont aléatoires sont dits stochastiques. L'aléatoire ou le stochastique peut représenter par exemple, 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-homogéinité 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.

Une approche théorique pour résoudre les problèmes d'ordonnancement stochastique est la théorie des processus bandits, des processus de décision semi-Markoviens. Elle est développée dès 1972 par GITTINS et JONES [05]. Ils modélisent des tâches à exécuter sur une machine. GITTINS et JONES [05] ont étudié les politiques d'ordonnancement de tâches stochastiques de type myope, backwards induction, forwards induction, politique d'indices etc.

Si l'objectif de la résolution d'un problème d'ordonnancement stochastique est une fonction à coûts séparables, ces auteurs ont montré que la politique d'indices est optimale. On associe à chaque tâche une priorité dynamique appelé indice d'allocation dynamique et est noté I.A.D. Ces indices sont calculés en tout instant et durant l'exécution des tâches, permettant à l'ordonnanceur de prendre en considération leurs exécutions et les instants de préemption au fur et à mesure que l'information sur les tâches est recueillie. En tout instant, on exécute la tâche qui a le plus grand indice. 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 ne rapporte plus le maximum de profit, on l'interrompt et on exécute une autre qui rapporte beaucoup plus.

Comparé à la programmation dynamique, l'utilisation de ces indices réduit rigoureusement la complexité en temps et mémoire requise pour déterminer une solution optimale. Cela se fait en réduisant un problème de dimension «p» en «p» problèmes unidimensionnels.

Dans le premier chapitre, nous définissons les processus bandits et nous donnons leurs caractéristiques. Des notions de politiques et de règles sont aussi fournies.

Dans le second chapitre et pour mettre en valeur l'application de ces processus bandits, un exemple de problème d'ordonnancement en temps réel modélisé par ces derniers est exposé.

Dans le chapitre trois, l'existence et la caractérisation des I.A.D données par NASH [12] et GITTINS [06] sont discutées. Des preuves des théorèmes énoncées sont données en détail.

La détermination des I.A.D s'avère d'une importance pratique pour le développement d'une politique optimale d'exécution de tâches. Le domaine n'est pas vierge, de notre recherche bibliographique, au moins trois algorithmes de détermination des I.A.D sont en notre possession ROBINSON [13], VARAIYA et AL. [17] et SONIN [15] et ont fait l'objet d'une étude approfondie au chapitre quatre. Ils ont été implémentés en utilisant un langage évolué de programmation. Les programmes ont été déroulés sur un processeur Intel Pentium III cadencé à 800 MHz et 176 Mo de RAM sous Windows XP.

Le nombre maximum d'états que ces algorithmes peuvent prendre en charge ne dépassait pas respectivement 45, 143 et 113 états.
Dans [11], un nouvel algorithme de détermination des I.A.D est proposé et est exposé en détail au chapitre cinq. Il dérive essentiellement de l'algorithme général de BEALE exposé dans la partie discussion de [06] où des systèmes d'équations linéaires sont à résoudre.

Dans la fonction objectif, l'espérance de la somme des gains linéaires et prévisionnels à maximiser, si le taux d'actualisation ne dépasse pas la valeur 0.5, une méthode itérative est utilisée pour résoudre les systèmes linéaires. Le nouvel algorithme peut prendre en charge des problèmes allant jusqu'à cent soixante états. Le temps de calculs est négligeable.
Si le taux d'actualisation dépasse 0.5, l'algorithme consiste à utiliser une décomposition LU. Il peut prendre en charge des problèmes à cent quarante trois états.

Dans le chapitre six, une étude comparative entre les quatre algorithmes exposés dans ce mémoire est élaborée. Les critères de comparaison sont le temps de calcul et le nombre maximum d'états que ces algorithmes peuvent prendre en charge. Des expérimentations numériques sur un grand nombre d'exemples de problèmes d'ordonnancement, de l'ordre de mille, ont été effectuées. Un générateur de jeux d'essais est indispensable et est réalisé.

Pour évaluer les I.A.D, un logiciel I.A.D version 1.0 a été confectionné dans le dernier chapitre. Son fonctionnement et l'aide d'utilisation sont aussi fournis. Des programmes des différents algorithmes sont annexés.

CHAPITRE 1

LES PROCESSUS BANDITS

1. Introduction

Dans un atelier, "N" tâches sont à exécuter sur une machine afin de maximiser l'espérance de la somme des gains linéaires et prévisionnels. Chaque sorte de tâche répond à une demande spécifique qui possède généralement des fluctuations aléatoires autour d'une valeur moyenne. La présence de ces fluctuations ne permet pas de connaître précisément les demandes futures et impose donc une grande réactivité du gestionnaire de l'atelier afin d'assurer la satisfaction de sa clientèle.
Les temps d'exécution des tâches sont supposés aléatoires de lois connues.
Les problèmes où les temps d'exécution des tâches sont aléatoires sont dits stochastiques.

Dès 1972, la théorie des processus bandits, des processus de décision semi-Markoviens, est développée par GITTINS et JONES [05] pour résoudre les problèmes d'ordonnancement stochastique. Ces processus modélisent des tâches à exécuter sur la machine. A chaque tâche, on associe une priorité dynamique appelée indice d'allocation dynamique et est notée I.A.D. Ces indices sont calculés en tout instant et durant l'exécution des tâches, permettant à l'ordonnanceur de prendre en considération les arrivées de tâches, leurs exécutions et les instants de préemption au fur et à mesure que l'information sur les tâches est recueillie. En tout instant, on exécute la tâche qui a le plus grand indice. 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 ne rapporte plus le maximum de profit, on l'interrompt et on exécute une autre qui rapporte beaucoup plus.

Beaucoup d'auteurs ont contribué à l'étude de ces processus. Nous citons dans l'ordre chronologique [05] [06] [07] [08], [09] [10], [12], [18], [13], [17], [04], [15].

Ils ont été présentés par DERBALA [02] [03]. Il montre qu'une priorité dynamique particulière allouée aux tâches dans un système d'exploitation d'ordinateurs multitâches s'interprète comme deux problèmes d'ordonnancement particuliers, l'ordonnancement de tâches détériorantes à durée opératoires variables et de tâches en retard ou en attente de réparation de la machine. Sous certaines conditions il montre qu'elle est une règle d'indice.

Dans le paragraphe deux, nous définissons les processus bandits et nous donnons leurs caractéristiques. Un exemple simple d'ordonnancement est formulé par ces processus. Des notions de règles, de politiques déterministes, stationnaires, markoviennes et optimales sont définies au paragraphe trois.
Dans le dernier paragraphe, une présentation des *super processus bandits* est faite.

2. Les processus bandits

Un *processus bandit* «x» est défini par le sextuplé $(\Omega, A, P, F, R, \alpha)$ où

- Ω représente l'ensemble d'états du processus qui peut être fini, dénombrable ou continu. Dans notre cas il est supposé fini.
- $A = \{0, 1\}$ l'ensemble des décisions ou d'actions à prendre, où «0» est la décision ou l'action de ne pas exécuter ou geler le processus bandit «x» et «1» est la décision ou l'action d'exécuter le processus «x».
- P, une matrice stochastique de transition entre les états du processus. $P_{ij}(a)$ représente la probabilité de passer de l'état «i» à l'état «j» lorsque la décision ou l'action «a» est appliquée au processus.
- Dans le cas semi-markovien, le temps de séjour dans un état est une variable aléatoire obéissant à une loi de probabilité F, supposé quelconque. Dans le cas markovien, F est exponentielle.
- R, une fonction de coût. R (i, a) est le coût encourut quand le processus bandit est à l'état «i» et la décision ou l'action «a» est appliquée. Il est supposé borner.
- $\alpha \in]0, 1[$ un nombre réel, appelé taux d'actualisation.

Un exemple d'interprétation du taux d'actualisation α est le suivant.
Un capital «T» placé au taux d'intérêt «s» par période (moins, année,…etc.) reçoit au bout de «t» périodes la nouvelle somme d'argent $S = T (1 + s)^t$.

Si on pose $\alpha = \frac{1}{1+s}$, alors $0 < \alpha < 1$ et $T = \alpha^t S$.

D'où si on veut recevoir la somme «S» à un taux d'intérêt constant «s» après «t» périodes il faut déposer la somme initiale «T», dite «S» prévisionnel par α^t.

On impose la condition suivante: Si la décision «0» est appliquée au processus à l'état «i», on ne transite vers aucun état et aucun coût n'est contracté. Se qui se traduit par les écritures $P_{ii}(0) = 1$ et $R(i, 0) = 0$.
Une réalisation $\{x(t), t = 0, 1,\ldots\}$ du processus est appelée trajectoire à qui on associe un coût total donné par $\sum_{t\geq 0}\alpha^t R(x(t),a(t))$. Dans ce cas le processus est dit à fonction *coût séparable.* Pour les automaticiens, ces processus sont de contrôle adaptatifs. Dans la suite, une décision, une action, une commande ou un contrôle sont synonymes.

Une *famille de processus bandits alternatifs* est un ensemble de «N» processus bandits ayant :

- Les mêmes instants de décision.
- En tout instant «t», la décision «1» est appliquée à un processus bandit parmi les «N» et la décision «0» est appliquée au «N - 1» autres processus.
- Le coût encourut est celui contracté par le processus bandit auquel la décision «1» est appliquée.
- Le taux d'actualisation α étant le même pour tous les processus.

Une famille de processus bandits alternatifs est dite *simple* si la décision «1» est appliquée à tous les processus bandits au moins une fois.

En général trois critères sont à considérer. Si un processus suit une trajectoire $\{x(t), t = 0, 1, \ldots\}$, pour toute politique π, on définit respectivement les critères de l'espérance du coût prévisionnel $E_\pi\left\{\sum_{t=0}^{\infty}\alpha^t R(x(t))/x(0)=i\right\}$; $i \geq 0$,

du coût positif non prévisionnel $E_\pi\left\{\sum_{t=0}^{\infty} R(x(t))/x(0)=i\right\}$; $i \geq 0$,

et du coût moyen $\lim_{n\to+\infty}\dfrac{E_\pi\left\{\sum_{t=0}^{n}\alpha^t R(x(t))/x(0)=i\right\}}{n+1}$; $i \geq 0$.

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 fixer les idées, soit "N" tâches à exécuter sur une machine. Les instants de décision sont discrets. Durant l'intervalle de temps [0, t] (t = 0, 1, …), une tâche «x» s'est exécutée pendant $t_x^e(t)$ unités de temps, et elle est en attente pendant $t_x^a(t)$.
Les durées d'exécutions $t_x^e(t)$, t = 0, 1, … (respectivement Les vecteurs des durées d'exécutions $t^e(t)= (t_1^e(t) , t_2^e(t), …, t_N^e(t))$, t = 0, 1, …) peuvent constituer les états pour le processus bandit «x» (respectivement les "N" tâches (x_1, x_2,…, x_N)).
A = {0, 1} est l'ensemble des décisions où «1» est la décision exécuter la tâche «x» et «0» est la décision ne pas exécuter la tâche «x». En tout instant «t», on exécute une et une seule tâche pendant une unité de temps. Si à l'instant «t» on choisit d'exécuter la tâche «x» pendant une unité de temps, alors on a $t_x^e(t + 1) = t_x^e(t) + 1$ et $t_x^a (t + 1) = t_x^a (t)$. Sinon, si la tâche «x» est gelée pendant une unité de temps, alors on a $t_x^e (t + 1) = t_x^e (t)$ et $t_x^a (t + 1) = t_x^a (t) +1$.
Si la tâche «x» est à l'état «i», le prochain état «j» est choisi selon une probabilité de transition P_{ij} dépendant de l'état actuel et de la décision choisie.
La durée de séjour de la tâche «x» dans un état est une variable aléatoire obéissant à une loi de probabilité. $R(t_x^e(t))$ est le coût encourut lorsque la tâche «x» est à l'état $t_x^e(t)$ supposé borné. $\{t_x^e(t), t = 0, 1, …\}$ constitue un processus décisionnel semi-Markovien appelé *processus bandit* et les "N" processus bandits constitueront une famille simple de processus bandit alternative.
Un *ordonnancement* est une suite de décisions à prendre sur un horizon de temps fini ou infini. Un *ordonnancement optimale* π de "N" tâches est celui qui maximise (respectivement minimise) l'espérance du coût prévisionnel:

$$V_\pi(i) = E_\pi \left\{ \sum_{t=0}^{\infty} \alpha^t \, R\left(t^e(t)\right) / \; t^e(0) = i \right\}$$

pour tout état initial «i» si on associe le processus des temps d'exécution au tâches. «i» est l'état du processus bandit qui a été choisi pour exécuter à l'instant zéro et $R(t^e(t))$ le coût encourut par ce processus. $\alpha \in]0, 1[$ étant un même taux d'actualisation.
On doit calculer en tout instant un *indice* ou une *priorité* pour chacune des tâches et exécuter celle qui a l'indice le plus élevé (respectivement le plus petit). L'existence et la caractérisation de cet indice se fait par les théorèmes

de GITTINS et JONES [05] et de NASH [12]. Des preuves sont présentées au chapitre trois. Cet indice se détermine par des algorithmes de ROBINSON [13], VARAIYA et AL. [17], SONIN [15] qui sont présentés au chapitre quatre et un nouvel algorithme qu'on proposera dans cette étude au chapitre cinq.

Si on associe le processus des temps d'exécution au tâches, l'I.A.D est obtenu en résolvant le problème d'arrêt optimal: trouver l'instant d'arrêt optimal τ^* qui assure un coût maximal (respectivement minimal) pour $V_\pi(t_x^e(0))$ où $V_\pi(t_x^e(0))$ est obtenu en exécutant la tâche « x » pendant le temps τ^*.

Soit $V(t_x^e(0)) = E\{\sum_{t=0}^{\tau^*-1} \alpha^t R(t_x^e(t))) + M\alpha^{\tau^*} / t_x^e(0)\}$. Où $M\alpha^{\tau^*}$ est le coût d'arrêt à l'instant τ^*.

3. Notions de politiques et de règles

Une *politique* est une suite de décisions ou d'actions à entreprendre.

Pour n'importe quel état initial, une politique qui maximise (respectivement minimise) l'espérance du coût prévisionnel sur l'ensemble des politiques est appelée *politique optimale.*

Les politiques *déterministes*, *stationnaires* et *markoviennes* sont respectivement celles qui ne contiennent aucune décision aléatoire, n'ont aucune dépendance explicite du temps et celles où la décision choisie à l'instant «t» dépend seulement de «t» et de l'état à l'instant «t».

Pour un processus bandit, toute politique est appelée *règle de gèle.*

Une règle de gèle qui applique la décision «1» jusqu'à l'*instant d'arrêt* «τ», puis la décision «0» à partir de cet instant, est appelée *règle d'arrêt*. Quand la décision «0» est appliquée, on dira que le processus va à l'état «∞».

Une règle d'indice est une politique qui exécute en tout instant une tâche d'indice le plus élevé. Une règle d'indice est une politique déterministe, stationnaire et de Markov.

Si l'ensemble des décisions d'un processus markovien est fini et si les coûts encourus sont bornés, BLACKWELL (1965) [05] a prouvé que quelque soit l'état du processus, il existe une politique déterministe, stationnaire et de Markov pour laquelle l'espérance du coût prévisionnel est optimale. *Une règle d'indice est ainsi optimale.*

Il est important de signaler qu'une règle d'indices n'est optimale que si la fonction coût est séparable. Sinon, elle peut ne pas être optimale.

On appelle *politique myope*, une règle d'exécution des tâches qui en tout instant de décision maximise la fonction objectif jusqu'au prochain instant de décision supposé connu.

Pour un processus de décision, une *politique d'induction en avant* (forward induction) est celle qui maximise en tout instant l'espérance du coût prévisionnel jusqu'à un instant d'arrêt arbitraire.

En générale, la politique myope et d'induction en avant ne sont pas optimales.

Si on considère une famille simple de processus bandits alternatifs, GITTINS [08] a montré que la politique des indices d'allocation dynamiques est optimale si elle coïncide avec celle d'induction en avant.

4. Les super processus bandits

La définition des super processus bandits est due à NASH [12]. Il les utilisent dans sa preuve qu'une politique d'indice de priorité peut être étendue au cas des arrivées de nouveaux processus bandits suivant une loi de Poisson.

Soit «M» un processus décisionnel semi-Markovien, ainsi qu'une politique déterministe stationnaire et de Markov «g».

La donnée du couple (M, g) s'appelle super processus bandit.

A l'instant «t», l'application de la décision gèle «0» au super processus bandit (M, g) signifie que le processus «M» est gelé quand il est à l'état $i = x(t)$ et que $P_{ii}(0) = 1$ et $R(i, i) = 0$.

L'application de la décision exécuter «1» au super processus bandit (M, g) est équivalente à appliquer la décision $g(i)$ à M s'il est à l'état $i = x(t)$.

De même, la notion de *famille de super processus bandits alternatifs* ainsi que *famille simple de super processus bandits alternatifs* peut être définie.

CHAPITRE 2
MODÈLE SEMI-MARKOVIEN POUR UN PROBLÈME D'ORDONNANCEMENT EN TEMPS RÉEL ET LE SUPER-PROCESSUS BANDIT ASSOCIÉ

1. Introduction

La tâche répétitive d'identifier la prochaine décision à prendre en tout instant discret dans un système de fabrication est souvent désigné par *ordonnancement en temps réel* [19].

En temps réel, un grand nombre de règles d'ordonnancement déterministes existent tel que la règle SPT (short processing time first), LPT (longest processing time first), FCFS (first come, first served) ou EDF (earliest due-date first). Elles sont généralement efficaces et faciles à implémenter.

Un problème d'ordonnancement stochastique est souvent modélisé comme celui d'un processus de décision semi-markovien. Son intérêt est qu'une théorie bien établie existe [14]. Une politique optimale d'exécution des tâches peut être développée. En effet, si la fonction objectif est à coûts séparables, une politique optimale n'est qu'une politique d'indices [06] [08]. Le modèle obtenu peut nécessiter des données de type coûts, probabilités de transition etc., qui ne peuvent être obtenues facilement. Il est susceptible d'inclure une matrice de transition entre les états de grande taille, qu'il est difficile de lui appliquer des opérations telle l'inversion. Dans notre cas d'étude, c'est la dimension de l'espace d'états du processus qui est démesurée. Généralement, l'espace d'état peut être réduit en l'observant et en l'analysant. Les états peu probables sont éliminés et un nouveau processus lui sera associé.

Dans le second paragraphe, nous traitons un problème simple d'ordonnancement en temps réel, nous définissons les états du processus exécution des tâches et les différentes transitions entre ces états sont indiquées dans la figure 2.2.

Ce modèle semi-markovien et son super-processus associé sont proposés. Dans un cas plus général de problème d'ordonnancement en temps réel, un processus de décision lui est attribué au paragraphe trois.

Un exemple de problème d'ordonnancement *à un seul robot* dans une chaîne de production illustre le processus et est exposé en détail au dernier paragraphe.

2. Exemple d'ordonnancement

Un système de production traite séquentiellement des pièces par deux machines différentes. La durée de traitement d'une pièce par chaque machine est supposée constante. A l'entrée de l'atelier, les pièces sont disponibles pour être charger sur la première machine. Chaque fois une pièce est traitée, un chariot suffisamment rapide la transporte et la déplace pour la mettre sur la seconde machine qui de nouveau sera exécutée. Une fois son exécution terminée sur la seconde machine, la dite pièce est déchargée dans une zone de sortie de l'atelier. On suppose qu'aucun espace tampon ou de déchargement appelé aussi « Buffer » entre les deux machines n'existe. Il est représenté par la figure ci-dessous.

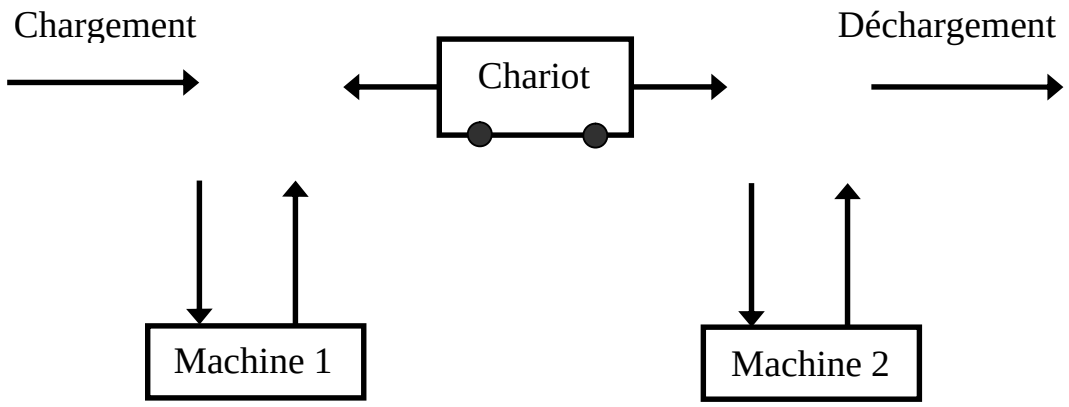


Figure 2.1: Un système de production.
Cas de deux machines en série.

On désire développer un ordre de mouvements de pièces sur les deux machines afin de maximiser le nombre de pièces traitées.

Nous supposons que le temps de transport des pièces par le chariot et le temps de réglage des machines est négligeable. Ils sont inclus dans le temps d'exécution des pièces. Le chariot est toujours disponible et que le cas de panne n'est pas considéré.

L'état de ce système de production est décrit par les trois modes: la machine est vide, la machine est occupée par une pièce dont le traitement est terminé et la machine est occupée par une pièce dont le traitement n'est pas encore terminé. Visualisons le système quand le mode de la machine change. La relation entre les états du système est décrite par un diagramme de transition d'états représenté

par la figure 2.2. Nous supposons que l'état 6 du système: la machine 1 est occupée par une pièce dont le traitement est terminé et la machine 2 est occupée par une pièce dont le traitement n'est pas terminé est impossible à ce réaliser.

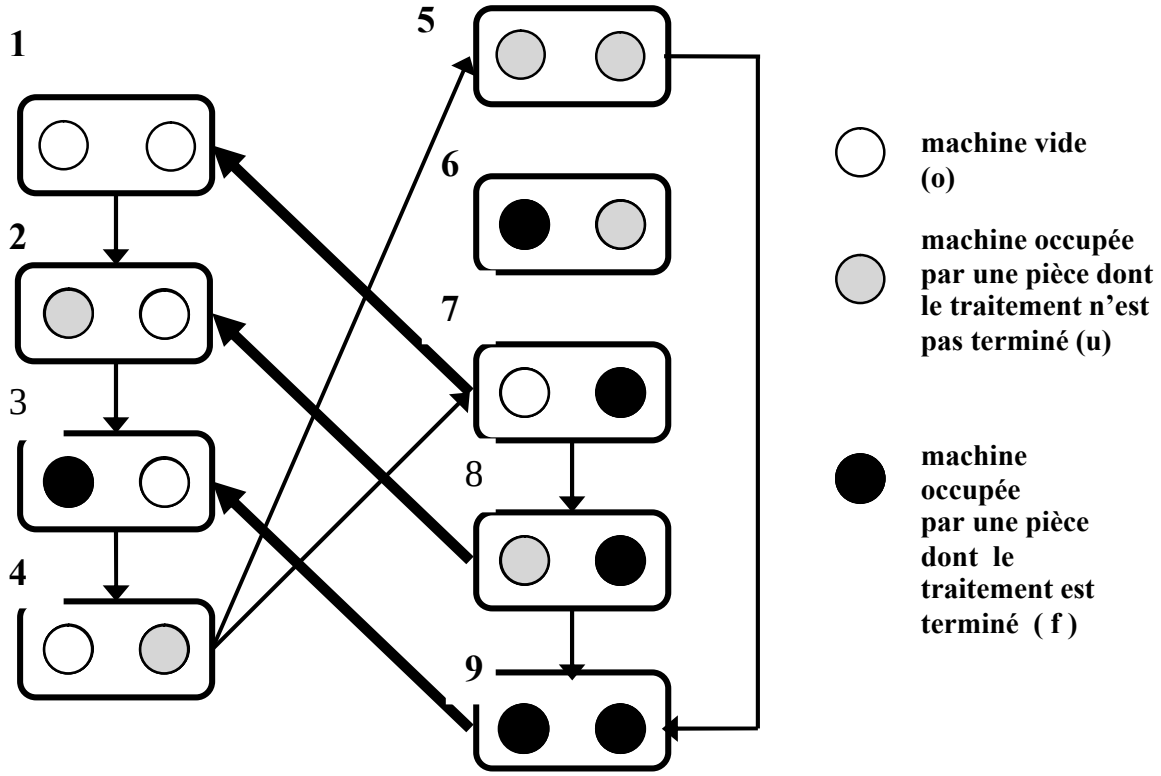


Figure 2.2 : Diagramme de transition d'états à deux machines.

Les flèches en gras signifient des transitions des pièces vers la sortie de l'atelier. Les autres flèches indiquent que la pièce est toujours en exécution sur l'une des machines.

Une interprétation de la figure 2.2 est que: si on est à l'état 1, une seule décision est disponible, charger la machine 1, on transite à l'état 2, où aussi une seule décision est possible, exécuter la pièce chargée sur la machine 1, on transite une nouvelle fois à l'état 3 où la décision disponible est charger la machine 2. Dans l'état 4, deux décisions sont disponibles, charger la machine 1 ou exécuter la pièce chargée sur la machine 2.

La figure 2.2 indique tous les ordres possibles des états de 1 à 9.

Le temps de séjours dans un état n'est pas associé aux arcs. Le tableau 2.1 beaucoup plus explicite où les décisions à entreprendre, les durées de séjours 0 ou C, où C

est la durée de traitement d'une pièce par chaque machine, les états ultérieurs sont donnés. Sans confusion, un numéro est associé à chaque état des deux machines.

Tableau 2.1: Transitions d'états pour le problème à deux machines.

Numéro d'état	État des deux machines	Décision	La durée de séjour dans l'état actuel	L'état suivant
1	[oo]	Charger la machine 1	0	2 [uo]
2	[uo]	Traiter la pièce chargée dans la machine 1	C	3 [fo]
3	[fo]	Charger la machine 2	0	4 [ou]
4	[ou]	Traiter la pièce chargée dans la machine 2	C	7 [of]
		Charger la machine 1	0	5 [uu]
5	[uu]	Traiter la pièce chargée dans la machine 2	C	9 [ff]
7	[of]	Charger la machine 1	0	8 [uf]
		Décharger la machine 2	0	1 [oo]
8	[uf]	Traiter la pièce chargée dans la machine 1	C	9 [ff]
		Décharger la machine 2	0	2 [uo]
9	[ff]	Décharger la machine 2	0	3 [fo]

Dans le tableau 1 l'état [ou] signifie que la machine 1 est vide et que la machine 2 est occupée par une pièce dont le traitement n'est pas terminé. [ff] signifie que les deux machines sont occupées par des pièces dont le traitement est terminé.

De notre lecture, deux mécanismes différents d'identification du prochain état dans le tableau 2.1 sont remarqués. *Un seul successeur* est associé si le système est dans un des états 1, 2, 3, 5 ou 9. L'exemple de l'état possible qui suit [uo] est [fo]. *Plusieurs états successeurs* sont associés pour les états 4, 7 et 8. Dans ce cas, l'état de successeur dépend d'une décision prise par l'opérateur. Quand le système est par exemple dans l'état [ou], l'opérateur peut attendre la deuxième machine termine le traitement de la pièce, ou il peut charger une nouvelle pièce sur la première machine.

Notre problème, en contrôlant ce système, est celui de la détermination des décisions appropriées à prendre quand le système est dans l'un des états 4, 7 et 8.

Le modèle décisionnel semi-markovien est défini par son espace d'état
Ω = {1, 2, 3, 4, 5, 7, 8, 9} et son ensemble de décision A = {charger la machine 1, charger la machine 2, traiter la pièce chargée dans la machine 1, traiter la pièce chargée dans la machine 2, décharger la machine 2}.
Si le processus est à l'état «i» et la décision «a» est choisie alors le prochain état «j» est atteint avec une probabilité de transition $P_{ij}(a)$. Les transitions d'état indiquées par les flèches gras dans la figure 2.2 sont d'intérêts. Elles reflètent la finalisation du traitement d'une pièce dans l'atelier. La matrice des probabilités de transition est récapitulée dans le tableau suivant.

Tableau 2.2: Matrice des transitions d'états.

Etat	Décision	1	2	3	4	5	7	8	9
1	Charger la machine 1	0	1	0	0	0	0	0	0
2	Traiter la pièce chargée dans la machine 1	0	0	1	0	0	0	0	0
3	Charger la machine 2	0	0	0	1	0	0	0	0
4	Charger la machine 1	0	0	0	0	1	0	0	0
	Traiter la pièce chargée dans la machine 2	0	0	0	0	0	1	0	0
5	Traiter la pièce chargée dans la machine2	0	0	0	0	0	0	0	1
7	Charger la machine 1	0	0	0	0	0	0	1	0
	Décharger la machine2	1	0	0	0	0	0	0	0
8	Traiter la pièce chargée dans la machine1	0	0	0	0	0	0	0	1
	Décharger la machine2	0	1	0	0	0	0	0	0
9	Décharger la machine2	0	0	1	0	0	0	0	0

La durée de séjour dans un des états indiqués dans le tableau 2.1 est décrite par une variable aléatoire «Z» de loi de probabilité P[Z = 0] = 7/11 et P[Z = c] = 4/11. Pour maximiser le nombre des pièces traitées par les deux machines, nous attribuons un indicateur de profit «+1» à la transition d'état dont l'exécution de la pièce est terminé sur la machine 2 et «0» sinon. De notre figure 2.1, si l'état initial est 1, les états 2, 3 et 4 sont obtenus successivement puisque une seule décision est possible dans chacun de ses états. Dans l'état 4, le meilleur choix est de charger la machine 1 car dans l'état 5 le processus d'usinage pour les deux machines aura lieu en même

temps. Dans les états 5 et 9 une seule décision est possible. A partir de n'importe quel état le système fonctionnera dans une transition cyclique d'état: 3, 4, 5, 9 puis 3 et la politique optimale obtenue est déterministe stationnaire et de Markov.

Soit «M» le processus décisionnel semi–Markovien déjà défini, ainsi qu'une politique déterministe stationnaire et de Markov «g». La donnée du couple (M, g) s'appelle super-processus bandit. A l'instant «t», l'application de la décision gèle «0» au super processus bandit (M, g) signifie que le processus «M» est gelé quand il est à l'état i = x(t) et que $P_{ii}(0) = 1$ et $R(i, i) = 0$. L'application de la décision exécuter «1» au super processus bandit (M, g) est équivalente à appliquer la décision g(i) à «M» s'il est à l'état i = x(t).

Dans un problème d'ordonnancement en temps réel modélisé comme un processus décisionnel semi-Markovien, tout les «N» états doivent être énumérés, leur définition n'est ni toujours immédiate, ni toujours triviale. Le nombre d'état tend à se développer exponentiellement ou plus rapidement quand le nombre de composants dans le modèle augmente. Le nombre d'état dans notre exemple augmente de huit à quarante si les modes du chariot sont inclus dans la variable d'état. Tous les «N^2» probabilités de transition doivent être déterminées.

Dans la suite, nous discutons la formulation d'un problème d'ordonnancement en temps réel en un processus à décision semi-Markovien.

3. Formulation d'un problème d'ordonnancement en temps réel

Dans un atelier, un manipulateur peut prendre les décisions suivantes: charger une pièce sur une machine, décharger une pièce d'une machine, déplacer une pièce d'une machine à une autre, exécuter la pièce chargée sur une machine et attendre la prochaine affectation.

Pour un problème d'ordonnancement en temps réel modélisé par un processus à décision semi-Markovien, un point de régénération de l'axe des temps se produit au moment où une décision doit être prise par le manipulateur. Il se produit à l'instant où une machine est déchargée et devienne libre. Dans la figure 2.3, sur un axe de temps, les décisions et les différents points de régénération sont représentés.

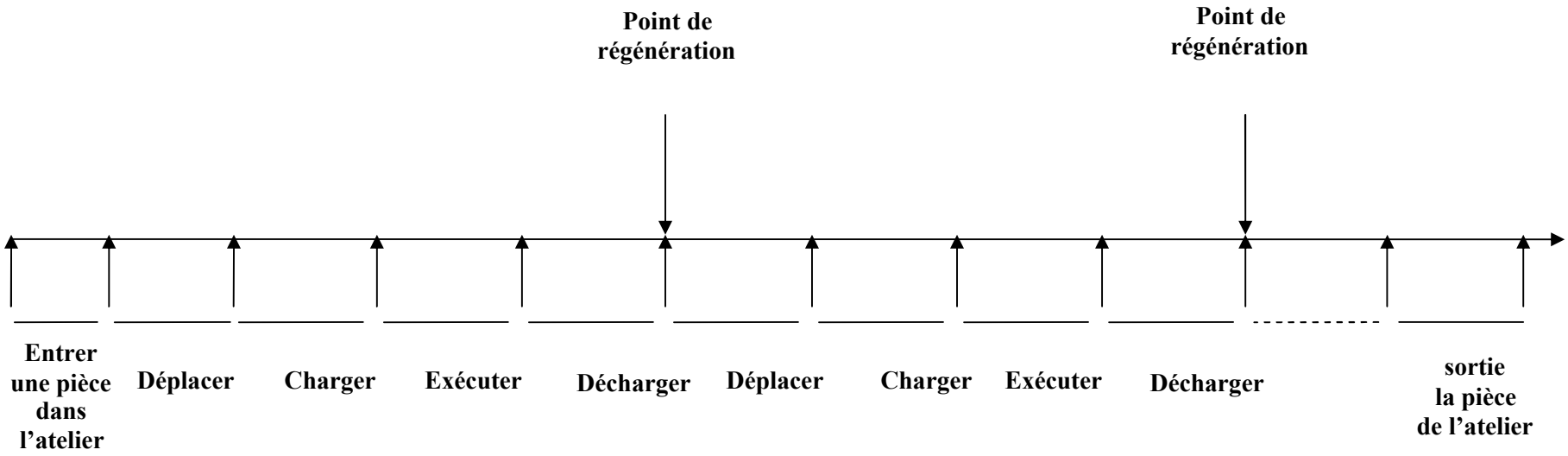


Figure 2.3 : Décisions séquentiels du manipulateur
et les différents points de régénération.

Les caractéristiques des objets d'un atelier sont appelées *configurations*.

Un état du processus d'ordonnancement est une configuration au point de régénération. Il est identifie par un indice de temps.

La durée de séjours dans un état est une variable aléatoire de loi de probabilité connue.

A chaque état une ou plusieurs décisions possibles peuvent être prise indépendamment.

A l'état «i», si une décision particulière «k» est prise, un coût borné «r_i^k» est attribué. Une transition à un des états «j» est obtenue avec une probabilité de transition P_{ij}^k. Elle vérifie $P_{ij}^k \geq 0$ et $\sum_j P_{ij}^k = 1$.

Illustrons par un exemple cette formulation.

Un atelier est composé de cinq machines différentes, d'un espace tampon d'entrée, d'un espace tampon de sortie et d'un robot manipulateur. Un espace tampon est une zone de stockage ou de déchargement d'une pièce. Il est supposé qu'aucun espace tampon n'existe entre les machines. C'est le problème dénoté *no-wait.*

Il est schématisé par la figure suivante.

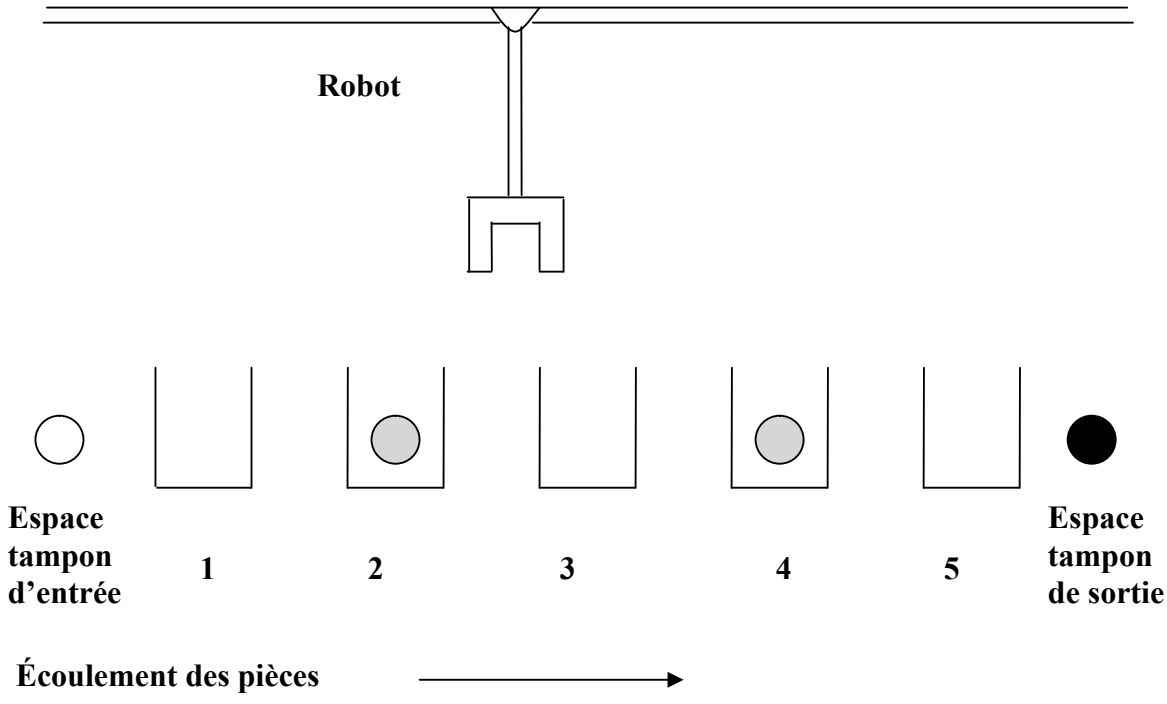


Figure 2.4 : Disposition d'une chaîne de production

Les machines sont indépendantes dans leur fonctionnement. Le robot manipulateur a une vitesse de déplacement constante, et ne peut prendre en charge qu'une seule pièce à la fois. Selon une demande établie, on fabrique un article qui est composé de plusieurs pièces. Elles doivent être produites par lots. L'ordre de passage de chaque pièce sur les machines est unidirectionnel (de la machine 1 à la cinquième machine). Les temps d'exécutions des tâches sur les machines sont connus.
Une pièce est imparfaite si elle est exécutée par un ordre ou dans des périodes de temps non prescrits par l'organigramme de commande. Sinon elle est dite parfaite. Le système est contrôlé par un opérateur externe. Il doit prendre des décisions. A un instant «t», il doit prendre au plus une des décisions suivantes: choisir une pièce à déplacer, faire entrer une nouvelle pièce dans l'atelier, déplacer une pièce d'une machine à une autre et déplacer le robot.
Un point de régénération est défini quand le robot a finit de déplacer une pièce et devient libre. Pour le robot, la décision à prendre dans la suite, est qu'elle pièce devait être déplacer.
Un état «S» peut être défini en tout instant de régénération noté par «t» comme suit:
S = (nouvelle_entrée, machine[1], machine[2], machine[3], machine[4], machine[5]).
Où une nouvelle_entrée est faisable s'il n'y a aucun conflit pour l'utilisation d'une machine entre la nouvelle pièce entrante et les pièces précédentes. On lui associé un indicateur nouvelle_entrer qui vaut 1 sinon l'indicateur vaut zéro.
En tout instant «t», on associe à chaque machine «m» occupée par une pièce parfaite «i» le paramètre $P^{r}_{m,i}(t)$ qui représente le temps restant pour la fin d'exécution de la pièce «i» sur la machine «m» à partir de l'instant «t» appelé aussi temps résiduel. Ces machines sont ordonnées selon l'ordre croissant des $P^{r}_{m,i}(t)$.
En sous entendant par rang d'une machine la place qu'elle occupe dans cette ordre.
Nous définissons « machine [m] » par le rang de la machine « m » si elle est occupée par une pièce parfaite, par l'indicateur 6 si elle est occupée par une pièce imparfaite et si la machine est libre l'indicateur vaut zéro.
L'état (1, 0, 6, 2, 0, 1) signifie les situations suivantes: la nouvelle entrée est faisable, la machine 2 est occupée par une pièce imparfaite, les machines 1 et 4 sont vides et les machines 3 et 5 sont occupées par des pièces parfaites. Le plus court temps

d'exécution restant et pour la machine 5 et le deuxième plus court temps d'exécution est pour la machine 3.

La décision à prendre est de choisir une pièce à retirer ou à faire entrer une nouvelle pièce dans le système de production.
Dans l'état (1, 0, 0, 1, 0, 0), il faut faire entrer une nouvelle pièce, une transition vers un des états est faite à savoir (0, 2, 0, 1, 0, 0) ou (0, 1, 0, 1, 0, 0) ou (0, 1, 0, 2, 0, 0). Ou retirer la pièce de la machine 3. Le prochain état sera (1, 0, 0, 0, 1, 0).
La probabilité de transition P_{ij}^{k} de l'état «i» à l'état «j» sous la décision «k» peut être estimée par : $P_{ij}^{k} \cong \frac{\text{fréquence de transition de l'état i à l'état j sous l'alternative k}}{\text{fréquence total de visiter l'état i sous l'alternative k}}$.

Le coût de transition r_i^k à l'état «i» sous la décision «k» est défini par un indicateur «+1» si cette transition ajoute immédiatement une nouvelle pièce parfaite dans l'espace tampon de sortie, «-1» si cette transition ajoute immédiatement une nouvelle pièce imparfaite. Sinon l'indicateur vaut zéro.
Dans l'état (1, 0, 6, 1, 0, 1), Les décisions respectives retirer la pièce du machine 5, retirer la pièce de la machine 2 et entrer une nouvelle pièce encoure des coûts respectifs +1, -1 et 0.

CHAPITRE 3
LES INDICES D'ALLOCATION DYNAMIQUES: EXISTENCE ET CARACTÉRISATION

1. Introduction

Les processus bandits sont montrés dans le chapitre deux être un modèle adéquat à la résolution des problèmes d'ordonnancement stochastique dans un atelier. GITTINS et JONES [05] ont étudié les politiques d'ordonnancement de tâches stochastiques de type myope, backwards induction, forwards induction, politique d'indices etc.

Si l'objectif de la résolution d'un problème d'ordonnancement stochastique est une fonction à coûts séparables, ces auteurs ont montré que la politique d'indices est optimale. Ils ont donné une preuve de l'existence et de la caractérisation de ces indices. GLAZEBROOK [10] qualifie la preuve de GITTINS [06] comme étant obscure et difficile à suivre. WHITTLE [18] a donné une autre preuve beaucoup plus élégante et facile à assimiler.

Plusieurs auteurs ont contribué à l'étude de l'existence et de la caractérisation de ces I.A.D à savoir NASH [12], GITTINS [06] [07] [08], GLAZEBROOK [09] [10], WHITTLE [18], FROSTIG et WEISS [04] et SONIN [15].

Dans le paragraphe deux, nous énoncerons l'existence et la caractérisation des I.A.D données par GITTINS [06] et NASH [12].

Pour plus de clarté, nous avons préféré regrouper les preuves de ces théorèmes dans un paragraphe final et distinct. De ce fait, le lecteur ne sera pas ennuyé par les formules mathématiques qui demandent une certaine attention pour la compréhension.

2. Existence et caractérisation des indices d'allocation dynamiques

Un ordonnancement de "N" tâches sur une machine est un processus séquentiel en M étapes. Il est formé d'un système dynamique à temps discret évoluant pendant un nombre fini de périodes dont l'objectif est une fonction coût additif au fil des périodes. Il peut se résoudre par la programmation dynamique. C'est une technique mathématique à implémenter qui a pour objet d'aider à prendre des décisions séquentielles indépendantes les unes des autres. Le problème peut être décomposé

en étapes et une décision doit être prise en chaque étape. En chaque étape, la décision prise transforme l'état actuel en un état associé à l'étape suivante avec une distribution de probabilité.

Etant donné un état, une stratégie optimale pour les étapes restantes est indépendante des décisions prises aux étapes précédentes. L'état actuel contient toute l'information nécessaire aux décisions futures. Cette propriété est dite principe d'optimalité ou de BELLMAN (1957) [06]. L'algorithme de recherche de la solution optimale commence par trouver la stratégie optimale pour tous les états de la dernière étape. Une relation de récurrence identifie la stratégie optimale dans chaque état de l'étape k à partir de la stratégie optimale dans chaque étape de l'étape k + 1. En utilisant cette relation de récurrence, l'algorithme procède en reculant étape par étape. Il détermine la stratégie optimale pour chaque état de chaque étape.

Si l'espace d'état du processus est de grande dimension, la programmation dynamique est rendue inefficace. Le temps d'exécution du programme est démesuré.

L'utilisation des indices d'allocation dynamique réduit rigoureusement la complexité en temps et mémoire requise pour déterminer une solution optimale. Cela se fait en réduisant un problème de dimension «N» en «N» problèmes unidimensionnels. L'indice d'allocation dynamique d'un processus bandit dépend seulement des caractéristiques de ce processus et il est indépendant de tout autre processus bandit.

Soit $F = \{ (\Omega_n, A_n, P_n, F_n, R_n, \alpha) ; 1 \leq n \leq N \}$ une famille de "N" processus bandits alternatifs. En tout instant t, l'état du système est donné par le vecteur $X(t) = (x_1(t), x_2(t),\ldots, x_N(t))$ des états $x_n(t)$ des processus bandits x_n, $n = 1, 2,\ldots,N$.

Nous considérons l'objectif celui de maximiser l'espérance des coûts linéaires et prévisionnels $V_\pi(i) = E_\pi\{ \sum_{t=0}^{\infty} \alpha^t R(X(t)) / X(0) = i \}$.

Les coûts sont supposés bornés, la fonction objective sera ainsi bien définie.

Le théorème de BLACKWELL (1965) [14] [06] assure l'existence d'une politique optimale qui est déterministe, stationnaire et de Markov. GITTINS et JONES [05] ont énoncé le premier théorème d'existence de ces indices. Ce théorème se formule comme suit:

Théorème 3.1 [06]

Il existe N fonctions $\nu_n : \Omega_n \rightarrow R$ (n = 1, 2, …, N) tel que la politique π^* qui exécute le processus bandit x_n à l'instant t est optimale si et seulement si $\nu_n\{ x_n(t) \} = \underset{1 \le i \le N}{\text{Max}} \nu_i\{ x_i(t) \}$.

La fonction ν_n est appelée *indice d'allocation dynamique* notée en abrégé I.A.D. Elle est aussi appelée par WHITTLE [18] *indice de GITTINS.*

La politique π^* qui en tout instant t, exécute le processus bandit x_n tel que $\nu_n\{ x_n(t) \} = \underset{1 \le i \le N}{\text{Max}} \nu_i\{ x_i(t) \}$ est appelée *règle d'indices*.

NASH [12] a donné la première caractérisation de l'I.A.D.

Soit x un processus bandit et μ un réel positif. Le processus bandit stoppable (x, μ) est défini en excluant l'utilisation de la décision gèle et en ajoutant à l'ensemble des décisions la notion d'arrêt du processus. En tout instant t, le processus bandit peut être soit exécuté ou arrêté. L'arrêt encourt un gain de valeur $\mu\, \alpha^t$.

Si un processus x suit un trajectoire $\{x(t), t = 0, 1, \ldots\}$ et s'il s'arrête à l'instant T, le profit total est $C = \sum_{t=0}^{T-1} \alpha^t R(x(t)) + \alpha^T \mu$.

τ est l'instant d'arrêt optimal pour le processus bandit x pour lequel les décisions optimales sont exécuter ou arrêter.

L'indice d'allocation dynamique, noté par γ, est équivalent à un coût d'arrêt. On arrête d'exécuter une tâche qui ne rapporte plus le maximum de profit.

On peut écrire pour tout état « i », $\gamma(i) = E_\pi \{ \sum_{t=0}^{\tau(i)-1} \alpha^t R(i) + \alpha^{\tau(i)} \gamma(i) \ / \ x(0) = i \}$.

L'espérance est prise sur l'ensemble des réalisations π du processus jusqu'à l'instant d'arrêt aléatoire $\tau(i)$ défini par $\tau(i) = \min\{ t / \gamma(x(t)) < \gamma(i) \}$. Ce nombre peut être infini.

Théorème 3.2 [12]

Pour un processus bandit x, si $\gamma(i)$ est défini et est fini pour tout état «i», alors $\gamma(i)$ est l'unique solution finie des deux équations:

$$\gamma(i) = \frac{E_\pi\left[\sum_{t=0}^{\tau(i)-1} \alpha^t R(x(t))/x(o)=i\right]}{1-E_\pi\left[\alpha^{\tau(i)}\right]} \quad \text{et} \quad \tau(i) = \min\{\, t / \gamma(x(t)) < \gamma(i)\,\}.$$

Comme corollaire, GITTINS et GLAZEBROOK [09] ont obtenu une formule de l'I.A.D.

Corollaire 3.1 :

$$\gamma(i) = \sup_{\tau>0} \frac{E\left[\sum_{t=0}^{\tau-1} \alpha^t R(x(t))/x(o)=i\right]}{1-E\left[\alpha^{\tau}\right]}$$ où le supremum est pris sur l'ensemble des instants d'arrêt positifs τ tel que le rapport $\frac{E\left[\sum_{t=0}^{\tau-1} \alpha^t R(x(t))/x(o)=i\right]}{1-E\left[\alpha^{\tau}\right]}$ soit fini.

GITTINS [06] propose deux caractérisations de l'I.A.D notées respectivement par ν et ν'.

Pour un processus bandit x à l'état «i», $\nu(i)$ est donné par l'expression

$$\nu(i) = \sup_{\tau>0} \frac{E\left[\sum_{t=0}^{\tau-1} \alpha^t R(x(t))/x(0)=i\right]}{E\left[\sum_{t=0}^{\tau-1} \alpha^t /x(0)=i\right]}.$$

La deuxième caractérisation est $\nu'(i) = \sup_{\{f\,:\,f_0=0\}} \frac{E\left[\sum_{t=0}^{\infty} \alpha^{t+f_t} R(x(t))/x(0)=i\right]}{E\left[\sum_{t=0}^{\infty} \alpha^{t+f_t} /x(0)=i\right]}$

où f_t est le nombre de fois que la décision «gèle» à été employée avant la $(t+1)^{ième}$ application de la décision «exécuter».

GITTINS [08] montre que $\nu' = \nu$ et que les instants d'arrêt existent quand les supremums respectifs sont atteints.

L'égalité $\nu' = \nu$ montre qu'au moins un instant d'arrêt existe. Le théorème suivant donnés par FROSTIG et WEISS [04] caractérisent tous les instants d'arrêts.

Théorème 3.3 [04]

Les seuls instants d'arrêt pour lesquels le supremum

$$\nu(i) = \sup_{\tau>0} \frac{E\left[\sum_{t=0}^{\tau-1} \alpha^t R(x(t)) / x(0) = i\right]}{E\left[\sum_{t=0}^{\tau-1} \alpha^t / x(0) = i\right]}, i \geq 0 \qquad (3.1)$$

est atteint sont $\tau(i) = \min\{ t / \nu\{ x(t) \} < \nu(i) \}$ (3.2)

et tous les instants σ vérifiant $\sigma \leq \tau$ et $\nu\{ x(\sigma) \} \leq \nu(i)$ (3.3)

Finalement, un point essentiel doit être souligné. Pour tout état «i» du processus bandit, la relation entre la caractérisation de GITTINS "$\nu(i)$" et celle de NASH "$\gamma(i)$" est la suivante: $\nu(i) = (1 - \alpha)\, \gamma(i)$.

3. Preuves des théorèmes et corollaire énoncés

a) Preuve du théorème 3.1 [04]

La démonstration est basée sur un principe dit d'échange. La politique d'indice π^* peut être changé en une autre politique $\pi^{(0)}$ en exécutant à un des instants arbitraires et pendant une période de temps négligeable un processus autre que celui qu'on doit exécuter suivant π^*.

La fonction objective suivant $\pi^{(0)}$, V_π, ne sera pas maximale. Une fois que le théorème est prouvé pour tout échange possible, l'optimalité de π^* est obtenue.

Dans la suite, chaque processus bandit exécuté par la politique π^* est noté x^*.

ν^* et τ^* sont respectivement l'I.A.D et l'instant d'arrêt défini par $\tau^* = \inf \{ t / \nu^*\{x^*(t)\} < \nu^*\{x^*(0)\}\}$ pour le processus x^*.

Soit $\pi^{(0)}$ une politique qui exécute à l'instant $t = 0$ un processus bandit x_n tel que $\nu_n\{x_n(0)\} < \nu^*\{x^*(0)\}$, et à partir de $t = 1$ elle suit la politique π^*.

Sous la politique $\pi^{(0)}$, le processus bandit x^* est exécuté après une période d'exécution du processus bandit x_n.

Dans la figure 3.1, les deux politiques $\pi^{(0)}$ et π^* sont schématiquement représentées.

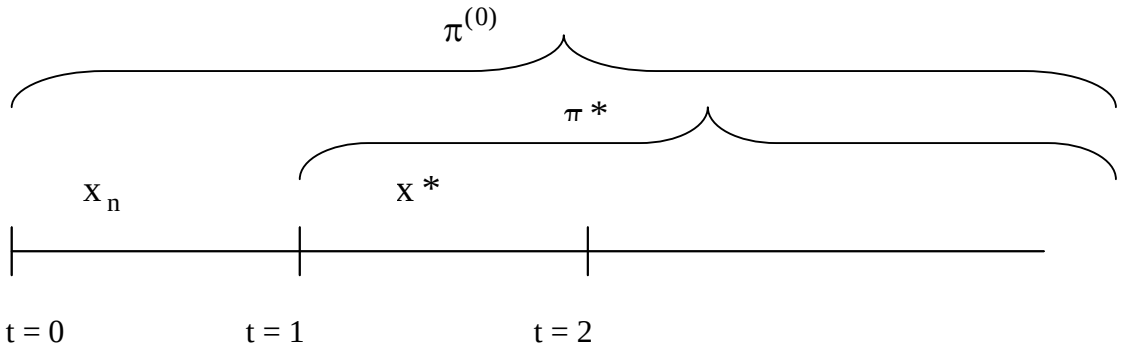


Figure 3.1: Représentation de la politique $\pi^{(0)}$ et π^*

Montrons que pour tout état «i» on a $V_{\pi^*}(i) \geq V_{\pi}^{(0)}(i)$.

Définissons par induction une suite récurrente et convergente de politiques $\{\pi^{(s)}, s = 1, 2, \ldots\}$ telle que pour tout état «i», $V_{\pi}^{(s)}(i)$ converge vers $V_{\pi^*}(i)$ quand s tend vers $+\infty$, et $V_{\pi}^{(s)}(i) \geq V_{\pi}^{(s-1)}(i)$.

$\pi^{(s)}$ est la politique qui exécute le processus bandit x* durant les instants $t = 0, 1, \ldots, \tau^*-1$. A partir de $t = \tau^*$ et l'état $x^*(\tau^*)$, $\pi^{(s)}$ procède dans son exécution comme $\pi^{(s-1)}$ à l'instant $t = 0$ et l'état $x^*(\tau^*)$ comme indiqué sur la figure 3.2.

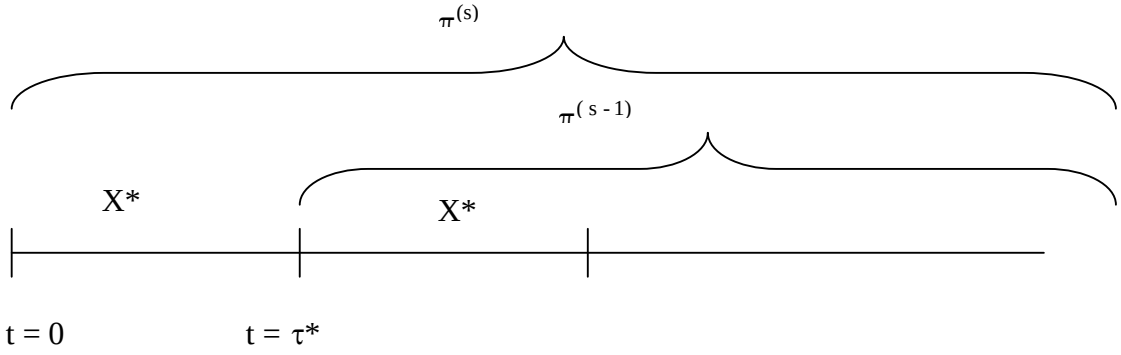


Figure 3.2: Représentation de la suite des politiques $\pi^{(s)}$

Ainsi construite, $\pi^{(s)}$ est identique à π^* durant une période de temps de longueur $\tau^* \geq 1$. Comme $\pi^{(s)}$ suit $\pi^{(s-1)}$ qui suit π^* après τ^*, on en déduit que $\pi^{(s)}$ suit π^* pour une période au-delà de τ^*. Par induction $\pi^{(s)}$ suit π^* durant les s premiers périodes de temps. Par conséquent, si la valeur de s augmente indéfiniment, on obtient la politique π^*.

Soit $\pi^{(s)}$ converge vers π^* quand s tend vers $+\infty$.

D'où $V_{\pi}^{(s)}(i)$ converge vers $V_{\pi^*}(i)$ quand s tend vers $+\infty$.

Etablissons l'inégalité $V_\pi^{(s)}(i) \geq V_\pi^{(s-1)}(i)$ par induction sur s.

Pour s > 1 la différence des gains est:

$$V_\pi^{(s)}(i) - V_\pi^{(s-1)}(i) = E_\pi^{(s)}\{\sum_{t=0}^{\infty} \alpha^t R(X(t)) / X(0) = i\} - E_\pi^{(s-1)}\{\sum_{t=0}^{\infty} \alpha^t R(X(t)) / X(0) = i\}.$$

De la définition des politiques $\pi^{(s)}$ et $\pi^{(s-1)}$ on obtient que:

$$V_\pi^{(s)}(i) - V_\pi^{(s-1)}(i) = E_\pi^{(s)}\{\sum_{t=0}^{\tau^*-1} \alpha^t R(x^*(t)) + \sum_{t=\tau^*}^{\infty} \alpha^t R(X(t)) / x^*(0) = i\}$$

$$- E_\pi^{(s-1)}\{\sum_{t=0}^{\tau^*-1} \alpha^t R(x^*(t)) + \sum_{t=\tau^*}^{\infty} \alpha^t R(X(t)) / x^*(0) = i\}.$$

Comme $\pi^{(s)}$ et $\pi^{(s-1)}$ exécutent le même processus bandit x^* pendant les instants $t = 0, 1, \ldots, \tau^* - 1$ et d'après la linéarité de l'espérance conditionnelle $E(a + b) = E(a) + E(b)$, pour toutes variables aléatoires a et b on obtient:

$$V_\pi^{(s)}(i) - V_\pi^{(s-1)}(i) = E\{\sum_{t=0}^{\tau^*-1} \alpha^t R(x^*(t)) / x^*(0) = i\} + E_\pi^{(s-1)}\{\sum_{t=\tau^*}^{\infty} \alpha^t R(X(t))\}$$

$$- E\{\sum_{t=0}^{\tau^*-1} \alpha^t R(x^*(t)) / x^*(0) = i\} - E_\pi^{(s-2)}\{\sum_{t=\tau^*}^{\infty} \alpha^t R(X(t))\}$$

$$= E_\pi^{(s-1)}\{\sum_{t=\tau^*}^{\infty} \alpha^t R(X(t))\} - E_\pi^{(s-2)}\{\sum_{t=\tau^*}^{\infty} \alpha^t R(X(t))\}.$$

De la propriété $E(\lambda a) = \lambda E(a)$, λ un réel et «a» une variable aléatoire, l'expression**:**

$$V_\pi^{(s)}(i) - V_\pi^{(s-1)}(i) = E_\pi^{(s-1)}\{\alpha^{\tau^*} \sum_{t=\tau^*}^{\infty} \alpha^{t-\tau^*} R(X(t))\} - E_\pi^{(s-2)}\{\alpha^{\tau^*} \sum_{t=\tau^*}^{\infty} \alpha^{t-\tau^*} R(X(t))\}$$

$$= \alpha^{\tau^*} E_\pi^{(s-1)}\{\sum_{t=\tau^*}^{\infty} \alpha^{t-\tau^*} R(X(t))\} - \alpha^{\tau^*} E_\pi^{(s-2)}\{\sum_{t=\tau^*}^{\infty} \alpha^{t-\tau^*} R(X(t))\}$$

$$= \alpha^{\tau^*}[E_\pi^{(s-1)}\{\sum_{t=\tau^*}^{\infty} \alpha^{t-\tau^*} R(X(t))\} - E_\pi^{(s-2)}\{\sum_{t=\tau^*}^{\infty} \alpha^{t-\tau^*} R(X(t))\}].$$

Se rappelant que $E(a) = E(E(a / b))$ pour toutes variables aléatoires a et b:

$$V_\pi^{(s)}(i) - V_\pi^{(s-1)}(i) = \alpha^{\tau^*}[E(E_\pi^{(s-1)}\{\sum_{t=\tau^*}^{\infty} \alpha^{t-\tau^*} R(X(t))\} / X(\tau^*))$$

$$- E(E_\pi^{(s-2)}\{\sum_{t=\tau^*}^{\infty} \alpha^{t-\tau^*} R(X(t))\}] / X(\tau^*))].$$

Et de la définition de $V_\pi^{(s)}$ et $V_\pi^{(s-1)}$**:**

$$V_\pi^{(s)}(I) - V_\pi^{(s-1)}(I) = \alpha^{\tau^*}[E(V_\pi^{(s-1)}(X(\tau^*))) - E(V_\pi^{(s-2)}(X(\tau^*)))]$$

$$= \alpha^{\tau^*} E[(V_\pi^{(s-1)}(X(\tau^*)) - V_\pi^{(s-2)}(X(\tau^*))]$$

$$= E\{\alpha^{\tau^*} E[(V_\pi^{(s-1)}(X(\tau^*)) - V_\pi^{(s-2)}(X(\tau^*))] / X(\tau^*)\}.$$

Pour que la relation $V_\pi^{(s)}(i) \geq V_\pi^{(s-1)}(i)$ résulte ainsi, on doit vérifier que $V_\pi^{(1)}(i) \geq V_\pi^{(0)}(i)$.

En effet, si $x_n = x^*$ alors il n'existe pas x_n tel que $\nu_n\{x_n(0)\} < \nu^*\{x^*(0)\}$ alors $\pi^{(1)} = \pi^{(0)} = \pi^*$. Le résultat est établi. Si $x_n \neq x^*$, on définit l'instant d'arrêt pour le processus bandit x_n par $\sigma = \min\{t \geq 1 / \nu_n\{x_n(t)\} < \nu^*\}$. La politique $\pi^{(0)}$ exécute le processus bandit x_n pendant les instants t = 0, 1, …,σ - 1. A l'instant t = σ l'indice le plus élevé est ν* et $\pi^{(0)}$ suit π* en exécutant le processus bandit x* pendant les instants t = σ, σ + 1, …,σ + τ* - 1. A l'instant t = σ + τ* l'état de x* est x*(τ*), celui de x_n est $x_n(\sigma)$. Si $x_m \neq x_n$ et $x_m \neq x^*$, l'état de x_m est $x_m(0)$.

La politique $\pi^{(1)}$ exécute x* pendant les instants t = 0, 1, …, τ* - 1. A l'instant t = τ*, $\pi^{(1)}$ exécute x_n pendant les instants t = τ*, τ* + 1, …, τ* + σ - 1 ensuite elle suit π*. A l'instant t = σ + τ*, l'état de x* est x*(τ*) et celui de x_n est $x_n(\sigma)$ et si $x_m \neq x_n$ et $x_m \neq x^*$, l'état de x_m est $x_m(0)$.

Les processus bandits x_n et x* étant indépendants, les instants d'arrêt σ et τ* sont aussi indépendants.

De la définition des deux politiques $\pi^{(0)}$ et $\pi^{(1)}$ on obtient:

$$V_\pi^{(0)}(i) - V_\pi^{(1)}(i) = E\{\sum_{t=0}^{\sigma-1} \alpha^t R(x_n(t)) + \alpha^\sigma \sum_{t=0}^{\tau^*-1} \alpha^t R(x^*(t)) + \alpha^{\sigma+\tau^*} \sum_{t=0}^{\infty} \alpha^t R(X(t)) / X(0) = I\} - E\{\sum_{t=0}^{\tau^*-1} \alpha^t R(x^*(t)) + \alpha^{\tau^*} \sum_{t=0}^{\sigma-1} \alpha^t R(x_n(t)) + \alpha^{\sigma+\tau^*} \sum_{t=0}^{\infty} \alpha^t R(X(t)) / X(0) = I\}$$

$$= E(1 - \alpha^{\tau^*}) E\{\sum_{t=0}^{\sigma-1} \alpha^t R(x_n(t))\} - E(1-\alpha^\sigma) E\{\sum_{t=0}^{\tau^*-1} \alpha^t R(x^*(t))\}$$

$$= E(1 - \alpha^{\tau^*}) E(1 - \alpha^\sigma) \frac{E\left\{\sum_{t=0}^{\sigma-1} \alpha^t R(x_n(t))\right\}}{E(1-\alpha^\sigma)} - \frac{E\left\{\sum_{t=0}^{\tau^*-1} \alpha^t R(x^*(t))\right\}}{E(1-\alpha^{\tau^*})}$$

$$= E(1 - \alpha^{\tau^*}) E(1 - \alpha^\sigma) \frac{E\left\{\sum_{t=0}^{\sigma-1} \alpha^t R(x_n(t))\right\}}{(1-\alpha) E\left(\sum_{t=0}^{\sigma-1} \alpha^t\right)} - \frac{E\left\{\sum_{t=0}^{\tau^*-1} \alpha^t R(x^*(t))\right\}}{(1-\alpha) E\left(\sum_{t=0}^{\tau^*-1} \alpha^t\right)}$$

$$= \frac{1}{1-\alpha} E(1 - \alpha^{\tau^*}) E(1 - \alpha^\sigma) [\nu_n(x_n, \sigma) - \nu^*],$$

de sorte que $\nu_n(x_n,\sigma)=\dfrac{E\left\{\sum_{t=0}^{\sigma-1}\alpha^t\ R(x_n(t))\right\}}{E\left(\sum_{t=0}^{\sigma-1}\alpha^t\right)}$ et $\nu^*=\dfrac{E\left\{\sum_{t=0}^{\tau^*-1}\alpha^t\ R(x^*(t))\right\}}{E\left(\sum_{t=0}^{\tau^*-1}\alpha^t\right)}$.

Il découle que $V_\pi^{(0)}(I) - V_\pi^{(1)}(I) \le \dfrac{1}{1-\alpha}\ E(1 - \alpha^{\tau^*})\ E(1 - \alpha^{\sigma})\ [\nu_n - \nu^*]$

si $\nu_n = \sup_{\sigma>0} \nu_n(x_n,\sigma) = \sup_{\sigma>0} \dfrac{E\left\{\sum_{t=0}^{\sigma-1}\alpha^t\ R(x_n(t))\right\}}{E\left(\sum_{t=0}^{\sigma-1}\alpha^t\right)}$.

Il vient de l'inégalité $\nu_n - \nu^* \le 0$ que $V_\pi^{(1)}(i) \ge V_\pi^{(0)}(i)$.

Par conséquent $V_\pi^{(0)}(i) - V_\pi^{(1)}(i) \le 0$.

La relation $V_\pi^{(s)}(i) \ge V_\pi^{(s-1)}(i)$ en découle aussitôt.

D'une manière analogue, on peut valider l'inégalité $V_\pi^{(s)}(i) \ge V_\pi^{(s-1)}(i)$ par les échanges possibles de π^*. D'où l'optimalité de la politique π^*. Ce qui termine la preuve □.

b) <u>Preuve du théorème 3.2.</u> [12]

La preuve qui suit est du à FROSTIG et WEISS [04].

Pour un processus décisionnel de Markov, la fonction de gain maximal $V_\alpha(i) = \sup_\pi V_\pi(i)$ où $V_\pi(i) = E_\pi[\sum_{t=0}^{\infty} \alpha^t R(x(t), a(t)) / x(0) = i]$ vérifie une équation fonctionnelle donnée par $V_\alpha(i) = \max_a \{R(i, a) + \alpha \sum_{j\ge0} P_{ij}(a)V_\alpha(j)\}$ [14]

où R(i, a) et $P_{ij}(a)$ sont respectivement le gain reçu à l'état «i» et la probabilité de transition de l'état «i» à un état «j» si l'action «a» est utilisé.

Pour un processus bandit stoppable (x, μ), la fonction gain maximal $V_\alpha(i)$, qu'on notera $V_\alpha(i, \mu)$, s'écrit donc $V_\alpha(i, \mu) = \max\{R(i) + \alpha \sum_{j\ge0} P(i, j)\ V_\alpha(j, \mu)\ ;\ \mu\}$.

Considérons les ensembles suivants:

$C_\mu=\{i / R(i) + \alpha \sum_{j\ge0} P(i, j)V_\alpha(j, \mu) > \mu\}$ qui représente un ensemble d'états du processus appelé d'exécution où l'action entreprise « exécuter » est optimale.

$S_\mu = \{i / R(i) + \alpha \sum_{j\geq 0} P(i, j)V_\alpha(j, \mu) < \mu\}$ un ensemble d'états du processus appelé d'arrêt ou l'action entreprise «arrêter» est optimale.

On appelle *état indifférent* un état du processus où une action arbitraire, exécuter ou arrêter, est optimale.

$I_\mu = \{i / R(i) + \alpha \sum_{j\geq 0} P(i, j)V_\alpha(j, \mu) = \mu\}$ l'ensemble des états indifférents.

A un instant «t», si le processus bandit «x» est respectivement dans un des états suivants, état $x(t)\in C_\mu$, $x(t)\in S_\mu$, $x(t)\in I_\mu$., il est respectivement optimal de l'exécuter, de l'arrêter ou d'utiliser une action arbitraire.

Notons par $\gamma(i)$ la petite valeur de μ pour laquelle $V_\alpha(i, \mu) = \mu$. Soit $\gamma(i) = \inf\{\mu / V_\alpha(i, \mu) = \mu\}$.

Et par $\tau(i, \gamma(i))$ l'instant du premier passage du processus bandit x à partir de l'état «i» dans l'ensemble $S_{\gamma(i)} = \{i / R(i) + \alpha \sum_{j\geq 0} P(i, j)V_\alpha(j, \gamma(i)) < \gamma(i)\}$.

Montrons que l'I.A.D $\nu(i) = \sup_{\tau>0} \frac{E\left[\sum_{t=0}^{\tau-1} \alpha^t R(x(t)) / x(0) = i\right]}{E\left[\sum_{t=0}^{\tau-1} \alpha^t / x(0) = i\right]}$ défini par GITTINS au deuxième paragraphe peut être évalué comme celui de $(1-\alpha)\gamma(i)$ et l'instant d'arrêt optimal $\tau(i)$ réalisant le supremum dans $\nu(i)$ est caractérisé comme celui de $\tau(i, \gamma(i))$.

La preuve se fait en trois étapes.

<u>Etape 1</u>

Montrons que pour un gain final μ vérifiant la relation $(1 - \alpha)\mu < \nu(i)$, il est optimal d'exécuter le processus x à l'état «i» que de l'arrêter.

Soit une politique π à un état initial «i» qui exécute le processus bandit x jusqu'à un instant d'arrêt $\tau(i)$ et reçoit un gain final $\alpha^\tau \mu$.

La fonction objective $V_\pi(i, \mu)$ s'écrit $E_\pi\{\sum_{t=0}^{\tau(i)-1} \alpha^t R(x(t)) + \alpha^\tau \mu / x(0) = i\}$.

Par une opération élémentaire, on peut représenter $\alpha^\tau \mu$ par une série géométrique

$(1-\alpha)\mu \sum_{t=\tau(i)}^{\infty} \alpha^t$.

D'où $V_\pi(i, \mu) = E_\pi\{ \sum_{t=0}^{\tau(i)-1} \alpha^t R(x(t)) + (1 - \alpha)\, \mu \sum_{t=\tau(i)}^{\infty} \alpha^t \ / \ x(0) = i\}$.

Sachant que $\nu(i) > (1 - \alpha)\, \mu$, $\dfrac{E_\pi\left[\sum_{t=0}^{\tau(i)-1} \alpha^t R(x(t)) / x(o) = i\right]}{E_\pi\left[\sum_{t=0}^{\tau(i)-1} \alpha^t / x(o) = i\right]} > (1 - \alpha)\, \mu$ en découle.

Une inégalité s'obtient. $E_\pi\{ \sum_{t=0}^{\tau(i)-1} \alpha^t R(x(t)) / x(0) = i\} > E_\pi\{(1 - \alpha)\, \mu \sum_{t=0}^{\tau(i)-1} \alpha^t / x(0) = i\}$.

Par suite,

$$V_\pi(i, \mu) = E_\pi\{ \sum_{t=0}^{\tau(i)-1} \alpha^t R(x(t)) + (1 - \alpha)\, \mu \sum_{t=\tau(i)}^{\infty} \alpha^t / x(0) = i\}$$

$$> E_\pi\{(1 - \alpha)\, \mu \sum_{t=0}^{\tau(i)-1} \alpha^t + (1 - \alpha)\, \mu \sum_{t=\tau(i)}^{\infty} \alpha^t \ / \ x(0) = i\} = (1 - \alpha)\, \mu\, E_\pi\{ \sum_{t=0}^{\tau(i)-1} \alpha^t\}$$

$$= (1 - \alpha)\, \mu \frac{1}{1-\alpha} = \mu.$$

L'inégalité $V_\pi(i, \mu) > \mu$ signifie que $i \in C_\mu$.

Il s'ensuit qu'il est préférable d'exécuter un processus à l'état «i» pendant $\tau(i)$ unités de temps puis l'arrêter et on reçoit un gain final $\alpha^\tau\, \mu$ que de l'arrêter immédiatement avec un gain μ. De la définition de $\gamma(i)$, il découle que $\gamma(i) \geq \mu$. En multiplions cette inégalité par un facteur $(1 - \alpha)$, on obtiendra l'inégalité souhaité $(1 - \alpha)\, \gamma(i) \leq \nu(i)$.

Etape 2:

Nous établissons que si le gain final μ est tel que $(1 - \alpha)\, \mu < \nu(i)$, il est optimal d'arrêter l'exécution du processus x à l'état «i» et obtenir un gain μ que de poursuivre son exécution.

En effet, en suivant les mêmes démarches que dans l'étape 1 on peut montrer que $V_\pi(i, \mu) > \mu$, et que $i \in S_\mu$.

Dans ce cas, il est optimal d'arrêter l'exécution du processus à l'état «i» avec un gain terminal μ que de continuer l'exécution.

Et par conséquent que $\gamma(i) \leq \mu$ puis que $(1 - \alpha)\gamma(i) \geq \nu(i)$.

Etape 3:

Comme μ est une variable, nous montrons que le gain final μ est tel que $(1 - \alpha)\ \mu = \nu(i)$, alors $\nu(i)$ est la petite valeur de $(1 - \alpha)\ \mu$ pour laquelle il est optimal d'arrêter immédiatement l'exécution du processus.

Dans ce cas le temps d'arrêt $\tau(i, \gamma(i))$ est nul.

En effet, d'une manière analogue que dans les étapes 1 et 2 la relation $V_\pi(i, \mu) = \mu$ en découle aussitôt.

Il s'ensuit que $i \in I_M$.

Dans ce cas, il est indifférent d'arrêter immédiatement l'exécution du processus bandit x ou de continuer son exécution. D'où $\gamma(i) = \mu$. En multiplions les deux membres de cette égalité par le paramètre $(1 - \alpha)$ et en se rappelant la définition de $\nu(i)$, l'égalité $(1-\alpha)\ \gamma(i) = \nu(i)$ en découlera.

A partir d'un état «i» et un instant d'arrêt $\tau(i, \gamma(i))$, pour tout les états «j» tel que $\gamma(j) \geq \gamma(i)$, le processus bandit x s'exécutera dans $C_{\gamma(i)}$.

Sachant que $(1 - \alpha)\ \gamma(i) = \nu(i)$, on obtient que $\nu(j) \geq \nu(i)$.

Se rappelant que $\tau(i)$ est défini par $\min\{t / \nu(x(t)) < \nu(i)\}$et de la définition de $\tau(i, \gamma(i))$ qui représente le premier passage du processus bandit dans un état «j» à partir de l'état «i» dans l'ensemble $M_{\gamma(i),}$ on en déduit que en tout état «i», $\tau(i, \gamma(i))$ est identique à $\tau(i)$.

Ce qui termine la preuve □

Finalement, on peut écrire pour tout état «i»:

$$\gamma(i) = E_\pi\{ \sum_{t=0}^{\tau(i)-1} \alpha^t R(i) + \alpha^{\tau(i)} \gamma(i) /\ x(0) = i \}.$$

Ce qui donne $\gamma(i) = \dfrac{E_\pi\left[\sum_{t=0}^{\tau(i)-1} \alpha^t R(x(t)) / x(o) = i \right]}{1 - E_\pi\left[\alpha^{\tau(i)}\right]}$.

c) Preuve du corollaire 3.1 [09]

Soient x_1, x_2,…une suit de variables aléatoires indépendantes. Une variable aléatoire positive à valeur entière N est appelée « temps d'arrêt ou instant d'arrêt » (stopping time) pour la suite x_1, x_2,…si l'événement $\{N = n\}$ est indépendant de x_{n+1}, x_{n+2},… pour tout n, n = 1, 2, …

Intuitivement, on observe x_n en un instant et N n'est autre que l'instant où en s'arrête.

Si N = n, alors on s'arrête après avoir observer x_1, x_2,… x_n et avant d'observer x_{n+1}, x_{n+2},…

Soient x_n n = 1, 2,… des variables aléatoires indépendantes tel que :

P{ x_n = 0} = P{ x_n = 1}=1 /2. N = min { n : $x_1 + x_2 +\ldots + x_n = 10$ } est un instant d'arrêt.

Il s'ensuit que $\tau(i) = \min\{ t / \nu(x(t)) < \nu(i)\}$est un instant d'arrêt.

D'où $\gamma(i) = \sup_{\tau>0} \dfrac{E\left[\sum_{t=0}^{\tau-1} \alpha^t R(x(t)) / x(o) = i\right]}{1 - E\left[\alpha^\tau\right]}$, où τ sont des instants d'arrêts.

d) <u>Preuve du théorème 3.3</u> [04]

Elle se fait en cinq étapes:

Dans la première et la seconde, nous montrons que pour tout instant d'arrêt σ ne vérifiant pas l'inégalité (3.3), le supremum n'est pas atteint. Dans la troisième, nous caractérisons des instants d'arrêt vérifiant (3.3) et pour lesquels le supremum est atteint. Dans la quatrième, nous montrons que le supremum est atteint par l'instant d'arrêt $\tau(i)$ de la formule (3.2).

Dans la cinquième, nous montrons que le supremum est atteint par tout instant d'arrêt vérifiant (3.3).

Rappelons que pour quatre réels a, b, c et d strictement positifs, l'inégalité $\dfrac{a}{c} < \dfrac{b}{d}$ est équivalente à la double inégalité $\dfrac{a}{b} < \dfrac{a+b}{c+d} < \dfrac{b}{d}$ (3.4).

<u>Etape 1:</u>

Pour tout instant d'arrêt σ vérifiant l'inégalité $\nu\{x(\sigma)\} > \nu(i)$, le supremum n'est pas atteint.

En effet, pour un état «j» vérifiant $\nu(j) > \nu(i)$, considérons un instant d'arrêt σ tel que $x(\sigma) = j$ (3.5).

Et par la définition (3.1), il suit qu'il existe un instant d'arrêt σ' tel que $\nu(j, \sigma') = \nu(j)$.

$\nu(i, \sigma + \sigma)$ s'écrit sous forme

$$\nu(i, \sigma+\sigma') = \frac{E\left[\sum_{t=0}^{\sigma-1} \alpha^t R(x(t)) / x(0) = i\right] + E\left[\sum_{t=\sigma}^{\sigma+\sigma'-1} \alpha^t R(x(t)) / x(\sigma) = j\right]}{E\left[\sum_{t=0}^{\sigma-1} \alpha^t / x(0) = i\right] + E\left[\sum_{t=\sigma}^{\sigma+\sigma'-1} \alpha^t / x(\sigma) = j\right]}$$

Il vient des deux inéquations (3.4) et (3.5), que

$$\nu(i, \sigma + \sigma) > \frac{E\left[\sum_{t=0}^{\sigma-1} \alpha^t R(x(t)) / x(0) = i\right]}{E\left[\sum_{t=0}^{\sigma-1} \alpha^t / x(0) = i\right]} = \nu(i, \sigma).$$

L'inégalité obtenue $\nu(i, \sigma + \sigma) > \nu(i, \sigma)$ montre que le maximum n'est pas atteint par σ.

Etape 2:

Pour tout instant d'arrêt σ vérifiant $\sigma > \tau(i)$ et $\nu\{x(\sigma)\} < \nu(i)$, le supremum n'est pas atteint.

En effet, soit σ un instant d'arrêt tel que $\sigma > \tau(i)$ et $\nu\{x(\sigma)\} < \nu(i)$ et posons «j» l'état du processus tel que $x(\sigma) = j$. Considérons un instant d'arrêt σ' défini par $\sigma' = \min\{t / x(t) = j\}$.

De la définition de σ', on a $\sigma \geq \sigma'$.

Distinguons deux cas. Si $\nu(i, \sigma) \leq \nu(j)$ alors le supremum n'est pas atteint. Puisque par hypothèse $\nu(j) < \nu(i)$. Supposons que $\nu(j) < \nu(i, \sigma)$ (3.6).

Puisque $\sigma \geq \sigma'$, $\nu(i, \sigma)$ s'écrit

$$\nu(i, \sigma) = \frac{E\left[\sum_{t=0}^{\sigma'-1} \alpha^t R(x(t)) / x(0) = i\right] + E\left[\sum_{t=\sigma'}^{\sigma-1} \alpha^t R(x(t)) / x(\sigma') = j\right]}{E\left[\sum_{t=0}^{\sigma'-1} \alpha^t / x(0) = i\right] + E\left[\sum_{t=\sigma'}^{\sigma-1} \alpha^t / x(\sigma') = j\right]}$$

Il découle des deux formules (3.4) et (3.6) que :

$$\nu(i, \sigma) < \frac{E\left[\sum_{t=0}^{\sigma'-1} \alpha^t R(x(t)) / x(0) = i\right]}{E\left[\sum_{t=0}^{\sigma'-1} \alpha^t / x(0) = i\right]} = \nu(i, \sigma').$$

Par conséquent le supremum n'est pas atteint par σ.

De ces deux étapes, le supprémum est pris sur l'ensemble des instants d'arrêt $\sigma > 0$ qui satisfont (3.3), et on va restreindre dans la troisième étape l'attention sur des instants d'arrêts particuliers.

Etape 3:

Etablissons que le supremum est atteint par quelques instants d'arrêt vérifiant (3.3).

Raisonnons par l'absurde. Supposons que le supremum n'est pas atteint par aucun instant d'arrêt vérifiant (3.3), instant d'arrêt σ tel que $0 < \sigma \leq \tau(i)$ et $\nu\{x(\sigma)\} \leq \nu(i)$.

On a supposé que le supremum dans $\nu(i)$ n'est pas atteint par σ. Il suit que $\nu(i, \sigma) = \nu_0 < \nu(i)$.

Soit «j» un état du processus bandit tel que $x(\sigma) = j$. De la définition de $\tau(i)$, le minimum des instants « t » tel que $\nu(x(t)) < \nu(i)$, il s'ensuit que $\nu(j) = \nu(i)$.

On peut trouver un instant d'arrêt σ' tel que $\sigma + \sigma' \leq \tau(i)$.

Dans le cas où $\sigma = \tau(i)$, on aura $\sigma' = 0$.

Prenons $\sigma_1 = \sigma + \sigma'$. En utilisant la même démarche que dans l'étape 1, on obtient que $\sigma \leq \sigma_1 \leq \tau(i)$, $\nu(i, \sigma) \leq \nu(i, \sigma_1) = \nu_1 < \nu(i)$.

On peut construire une suite des instants d'arrêt $(\sigma_n)_{n\in N}$ et une suite $(\nu_n)_{n\in N}$, avec $\sigma_{n-1} \leq \sigma_n \leq \tau(i)$ et $\nu_{n-1} = \nu(i, \sigma_{n-1}) \leq \nu(i, \sigma_n) = \nu_n < \nu(i)$.

La suite $(\sigma_n)_{n\in N}$ est croissante et majorée par $\nu(i)$. Elle est donc convergente. σ_n et $\tau(i)$ étant des entiers naturels, alors il existe un rang n_0 tel que $\sigma_{n0} = \tau(i)$. Pour tout $n \geq n_0$, définissons σ_n par $\tau(i)$. De même, la suite $(\nu_n)_{n\in N}$ est croissante et majorée par $\nu(i)$, elle est convergente vers $\nu_{n0.}$

On a $\nu_{n0} = \nu(i, \sigma_{n0}) = \nu(i, \tau(i))$. D'où la contradiction $\nu(i, \tau(i)) < \nu(i, \tau(i))$ est obtenue.

Pour tout état initial $x(0) = i$, le suprémum est atteint pour les instants d'arrêt σ_n, $\forall n \geq n_0$.

<u>Etape 4:</u>

Montrons que le supremum est atteint par $\tau(i)$.

Supposons que $x(0) = i$ et soit σ un instant d'arrêt qui satisfait (3.3) et pour lequel le suprémum est atteint. C'est-à-dire $\sigma \leq \tau(i)$ et $\nu(i, \sigma) = \nu(i)$.

Posons «j» l'état du processus bandit tel que $x(\sigma) = j$. Si $\sigma = \tau(i)$, c'est terminé, le résultat est établi. Si $\sigma < \tau(i)$ alors d'après (3.3) il s'ensuit que $\nu(j) = \nu(i)$ puisque $\tau(i)$ est le minimum des instants «t» tel que $\nu(x(t)) = \nu(i)$. On peut trouver un instant d'arrêt σ' tel que $\sigma + \sigma' = \tau(i)$ et si $\sigma' = \tau(i)$ alors $\sigma' = 0$.

Prenons $\sigma_1 = \sigma + \sigma'$. On a $\sigma \leq \sigma_1 \leq \tau(i)$ et on procède de la même manière que dans l'étape 1 on obtient l'inégalité $\nu(i, \sigma) \leq \nu(i, \sigma_1)$.

Nous rappelons que si $\frac{a}{b} = \frac{c}{d}$ alors $\frac{a+c}{b+d} = \frac{a}{b}$, il vient que $\nu(i) = \nu(i, \sigma) \leq \nu(i, \sigma_1) \leq \nu(i)$ puis que $\nu(i, \sigma) = \nu(i, \sigma_1) = \nu(i)$.

Soit la suite des instants d'arrêt $(\sigma_n)_{n\in N}$, avec $\sigma_{n-1} \leq \sigma_n \leq \tau(i)$ et $\nu(i, \sigma_{n-1}) = \nu(i, \sigma_n) = \nu_n$.

La suite $(\sigma_n)_{n\in N}$ est croissante et majorée par $\nu(i)$ elle est donc convergente. Et puisque σ_n et $\tau(i)$ sont des entiers naturels alors $\exists\ n_0 \in N$ / $\sigma_{n0} = \tau(i)$ et le supremum est atteint par $\tau(i)$.

Etape 5:

Montrons que le supremum est atteint par tout instant d'arrêt vérifiant (3.3).

Soit σ un instant d'arrêt tel que $\sigma \le \tau(i)$. Si $\sigma = \tau(i)$, c'est terminé. Le résultat est établi. Supposons que $\sigma < \tau(i)$ et soit «j» l'état du processus bandit tel que $x(\sigma) = j$. $\tau(i)$ étant le minimum des instants « t » tel que $\nu(x(t)) < \nu(i)$, on aura donc $\nu(j) = \nu(i)$.

De $\tau(i) - \sigma = \tau(j)$, $\nu(j, \tau(i) - \sigma) = \nu(i)$ et en utilisons la remarque

$$\frac{a}{b} = \frac{c}{d} \text{ implique } \frac{a+c}{b+d} = \frac{a}{b}$$

on obtient:

$$\nu(i) = \nu(i, \tau(i)) = \frac{E\left[\sum_{t=0}^{\sigma-1} \alpha^t R(x(t)) / x(0) = i\right] + E\left[\sum_{t=\sigma}^{\tau(i)-1} \alpha^t R(x(t)) / x(\sigma) = j\right]}{E\left[\sum_{t=0}^{\sigma-1} \alpha^t / x(0) = i\right] + E\left[\sum_{t=\sigma}^{\tau(i)-1} \alpha^t / x(\sigma) = j\right]} = \nu(i, \sigma).$$

Ce qui achève la preuve □.

CHAPITRE 4
DÉTERMINATION DES INDICES D'ALLOCATION DYNAMIQUES

1. Introduction

Dans un problème d'ordonnancement stochastique, la détermination des I.A.D s'avère d'une importance pratique pour le développement d'une politique optimale d'exécution de tâches. Le domaine n'est pas vierge, de notre recherche bibliographique, avec l'aide de l'Internet et dans la littérature, au moins trois articles sur la détermination des I.A.D sont en notre possession [13], [17] et [15] et ont fait l'objet de ce chapitre. Une tâche et un processus sont synonymes. Ces algorithmes nous fournissent en tout état d'un processus un I.A.D. Ils déterminent dans l'ordre le plus grand indice, le second plus grand indice etc.

Initialement, l'algorithme de ROBINSON [13] détermine des états à gain maximum. On attribue à l'I.A.D en cet état cette valeur maximale. Dans une seconde étape, à chaque itération, d'une formule itérative, nous déterminons des matrices et des vecteurs nécessaires aux calculs d'un autre I.A.D en un autre état. Un exemple de son application est fourni. Nous avons implémenté l'algorithme en utilisant un langage évolué de programmation. Le programme réalisé est exécuté sur un processeur Intel Pentium III cadencé à 800 MHz et 176 Mo de RAM, sous Windows XP. Il est annexé par une image écran. Une représentation graphique des I.A.D de cet exemple en tout état d'un processus est réalisée. Sa justification, sa finitude, son implémentation et des expérimentations sont faites. A partir d'un générateur de données qu'on a confectionné, nous avons mené une compagne de tests où les données sont fournies aléatoirement. Les temps d'exécution en fonction du nombre des états d'un processus sont dressés dans un tableau. Un graphe de leur comportement est représenté. Un exemple de détermination d'une politique optimale d'exécution de trois tâches est aussi traité. Un écran de fenêtre d'exécution du programme est imprimé. D'un histogramme, la politique optimale d'exécution des tâches est déterminée. Les algorithmes de VARAIYA et AL. [17] et de SONIN [15] sont exposés respectivement en détail aux paragraphes trois et quatre.

2. Algorithme de ROBINSON [13]

GITTINS [06] a montré que pour une famille des processus bandits alternatifs et si l'objectif est une fonction à coûts séparables, les politiques optimales sont d'indices et peuvent être déterminées. Un algorithme efficace est présenté par ROBINSON [13]. Il se formule par une formule de récursivité.

Un processus bandit à espace d'état fini Ω évolue dans le temps selon une loi de transition entre les états donnée par une matrice P. Un profit R(i) est octroyé en tout état «i» et il sera dévalué à un instant t à un taux $\alpha^t R(i)$ où a, $0< \alpha <1$ est appelé taux d'actualisation. Si l'objectif est de maximiser l'espérance des gains prévisionnels, l'indice d'allocation dynamique pour le processus bandit x à l'état «i» est défini par

$$\nu(i) = \max_{B' \in B} \frac{E\left[\sum_{t=0}^{\tau(B')-1} \alpha^t R(x(t)) / x(0) = i\right]}{E\left[\sum_{t=0}^{\tau(B')-1} \alpha^t / x(0) = i\right]} .$$

B représente la famille de sous-ensembles de Ω.

Pour tout B' un élément de B, $\tau(B')$ est le nombre de transitions entre les états du processus considéré jusqu'à l'accession à un état du l'ensemble B'.

Dans son article, ROBINSON fournit un algorithme sous une forme non explicite mais indique la façon de déterminer les indices. Nous le formulons dans une version universelle [16].

Dans l'algorithme suivant A_k est une matrice carrée d'ordre k. S_k et T_k des vecteurs de dimension k.

Algorithme 1

- **Input**

  L'espace d'état Ω et sa dimension N;

  La valeur du taux d'actualisation α dans l'intervalle]0, 1[;

  Les valeurs des probabilités de transition P(i, j) $\forall$ i, j $\in \Omega$;

  Les valeurs des gains R(i) $\forall$ i $\in \Omega$;

- **Output** L'indice d'allocation dynamique $\nu(i)$ pour tout état i $\in \Omega$.

Etape 1 :

- ***Trouver*** un état x satisfaisant $R(x) = \max_{j \in \Omega} R(j)$;
- ***Faire*** $\nu(x) := R(x)$;
- ***Supprimer*** l'état x de Ω ;
- ***Créer*** un ensemble vide Y ;

Etape 2:

Pour k = 2 **à** N **faire**

- ***Pour*** tout état i de Y ***faire***

  $Z_{k-1}(i) := \sum_{l \in Y} A_{k-1}(i, l)\, P(l, x)$;

- ***Faire*** $A_k(x, x) := \{1-\alpha\, P(x, x)-\alpha^2 \sum_{l \in Y} P(x, l)\, Z_{k-1}(l)\}^{-1}$;

- ***Pour*** tout état i de Y ***faire***

  $A_k(i, x) := \alpha\, A_k(x, x)\, Z_{k-1}(i)$;

  $A_k(x, i) := \alpha\, A_k(x, x) \sum_{l \in Y} P(x, l)\, A_{k-1}(l, i)$;

- ***Pour*** tous les états i et j de Y ***faire***

  $A_k(i, j) := A_{k-1}(i, j) + \alpha\, Z_{k-1}(i)\, A_k(x, j)$;

- ***Insérer*** l'état x dans l'ensemble Y;
- ***Pour*** tout état i de Y ***faire***

  $S_k(l) := \sum_{m \in Y} A_k(l, m)\, R(m)$;

  $T_k(l) := \sum_{m \in Y} A_k(l, m)$;

- ***Pour*** tout état j de Ω ***faire***

  $V(j) := [\, R(j) + \alpha \sum_{l \in Y} P(j, l)\, S_k(l)\,] / [\, 1+ \alpha \sum_{l \in Y} P(j, l)\, T_k(l)\,]$;

- ***Trouver*** un état x satisfaisant $V(x) := \max_{j \in \Omega} V(j)$;
- ***Faire*** $\nu(x) := V(x)$;
- ***Supprimer*** l'état x de Ω ;

Ce premier exemple illustre les différentes étapes de l'algorithme.

2.2. Exemple 1

Soit un processus bandit d'espace d'états $\Omega = \{A, B, C, D\}$. Les gains en tout état sont donnés par R(A) = 2, R(B) = R(D) = 3 et R(C) = 5. Les probabilités de transition P(i, j) sont tel que $P(i, j) = 0.25 \quad \forall\, i, j \in \Omega$. Le taux d'actualisation α étant de 0.8.

- Première étape

Déterminons le coût maximal. Il est de 5 et l'état qui lui est associé est «C».

Posons $\nu(C) = R(C) = 5$.

Eliminons l'état «C» de l'espace Ω qui devient $\Omega = \{A, B, D\}$.

Soit Y l'ensemble vide.

- Second étape

☞ A l'itération 2 (k = 2):

Calculons $A_2(C, C)$. $A_2(C, C) = (1 - \alpha P (C, C))^{-1} = 1.25$.

Insérons l'état «C» dans l'ensemble Y qui devient Y = {C}.

Déterminons les valeurs de $S_2(C)$ et $T_2(C)$.

$S_2(C) = A_2(C, C) \times R(C) = 6.25$; $T_2(C) = A_2(C, C) = 1.25$.

Pour tout état j de $\Omega = \{A, B, D\}$, la valeur de V(j).

$V(A) = [R(A) + \alpha \times P(A, C) \times S_2(C)] / [1 + \alpha \times P(A, C) \times T_2(C)] = 2.6$.

$V(B) = [R(B) + \alpha \times P(B, C) \times S_2(C)] / [1 + \alpha \times P(B , C) \times T_2(C)] = 3.4$.

$V(D) = [R(D) + \alpha \times P(D, C) \times S_2(C)] / [1 + \alpha \times P(D, C) \times T_2(C)] = 3.4$.

Le V(j) maximal est 3.4 et il est atteint en deux états à savoir «B» et «D».

Posons $\nu(B) = \nu(D) = 3.4$.

Choisissons un état noté x parmi «B» ou «D». x = B.

Eliminons l'état «B» de l'espace Ω qui deviendra $\Omega = \{A, D\}$.

☞ A l'itération 3 (k = 3):

Les valeurs des expression $Z_2(C)$, $A_3(B, B)$, $A_3(C, B)$, $A_3(B, C)$ et $A_3(C, C)$.

$Z_2(C) = A_2(C, C) \times P(C, B) = 0.3125$;

$A_3(B, B) = \{1-\alpha \times P(B, B) - \alpha^2 \times P(B, C) \times Z_2(C)\}^{-1} = 1.33333$;

$A_3(C, B) = \alpha \times A_3(B, B) \times Z_2(C) = 0.333333$;

$A_3(B, C) = \alpha \times A_3(B, B) \times P(B, C) \times A_2(C, C) = 0.333333$;

$A_3(C, C) = A_2(C, C) + \alpha \times Z_2(C) \times A_3(B, C) = 1.333333$.

Insérons une nouvelle fois l'état «B» dans l'ensemble Y qui deviendra Y = {C, B}.

Pour tout état « l »de l'ensemble Y = {C, B}, les valeurs des expression $S_3(l)$ et $T_3(l)$.

$S_3(B) = A_3(B, B)\times R(B) + A_3(B, C)\times R(C) = 5.66667$;

$T_3(B) = A_3(B, B) + A_3(B, C) = 1.66667$;

$S_3(C) = A_3(C, B)\times R(B) + A_3(C, C)\times R(C) = 7.66667$;

$T_3(C) = A_3(C, B) + A_3(C, C) = 1.66667$;

Pour tout état j de $\Omega = \{A, D\}$, la valeur de V(j).

$V(A) = [\ R(A) + \alpha\times(\ P(A, B)\times S_3(B) + P(A, C)\times S_3(C))\ ] / [\ 1 + \alpha\times(\ P(A, B)\times T_3(B) + P(A, C)\times T_3(\ C)\ )] = 2.8$;

$V(D) = [\ R(D) + \alpha\times(\ P(D, B)\times S_3(B) + P(D, C)\times S_3(C))\ ] / [1+\alpha\times(\ P(D, B)\times T_3(B) + P(D, C)\times T_3(C)\ )] = 3.4.$

3.4 est la valeur du V(j) maximal atteint à l'état «D».

Posons une deuxième fois $\nu(D) = V(D) = 3.4$.

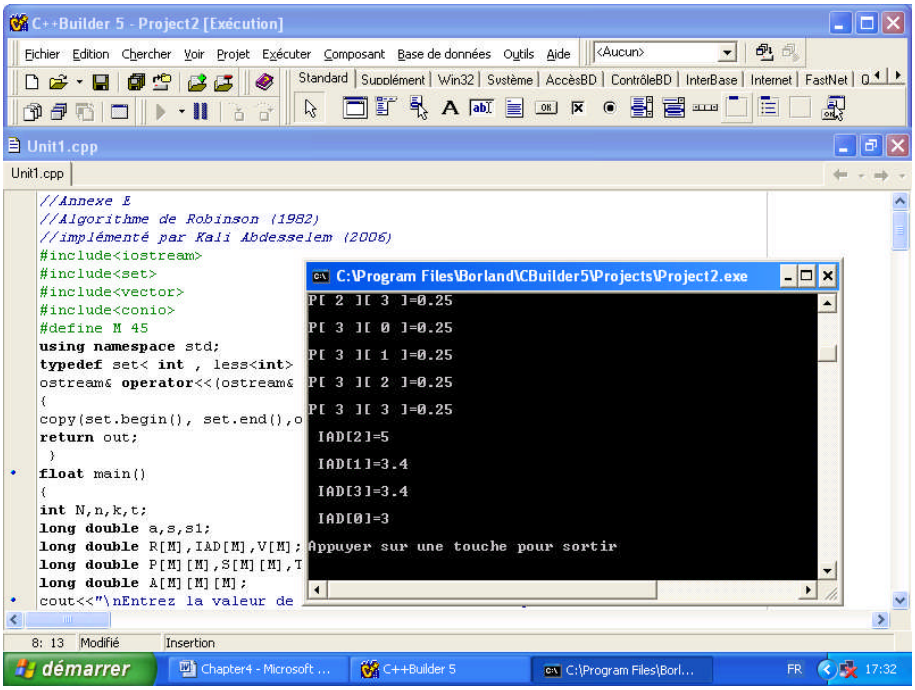
Nous constatons que l'état «D» a été visité deux fois. Même si les calculs semblent inutiles, ils doivent être faits.

Eliminons l'état «D» de l'espace Ω qui devient $\Omega = \{A\}$.

A l'itération 4 (k = 4), des calcules lourds sont faits qu'on va pas reproduire ici,

on obtiendra $\nu(A) = 3$.

Une image de l'écran du programme qu'on a implémenté est ci-dessous reproduite.



Ecran 4.1: Fenêtres du programme et de son exécution.

Les I.A.D en tout état du processus sont graphiquement représentés.

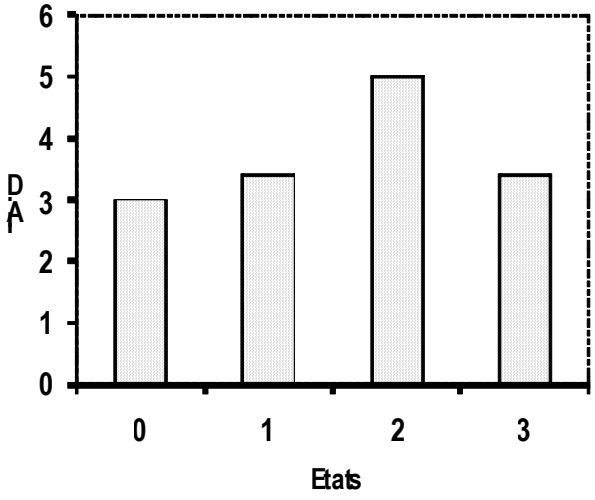


Figure 4.1: Représentation des I.A.D
en tout état du processus.

2.3. Justification

Considérons une famille de processus bandits alternatifs à espace d'état fini $\Omega = \{1,2,\ldots, n\}$. x(t) est l'état du processus bandit x après son exécution durant «t» unités de temps.

B représente la famille de sous-ensembles de Ω. Pour tout B' un élément de B, $\tau(B')$ est le nombre de transitions entre les états du processus considéré jusqu'à l'accession à un état du l'ensemble B'.

Posons $\nu(i, B') = \dfrac{E\left[\sum_{t=0}^{\tau(B')-1} \alpha^t R(x(t))/x(0)=i\right]}{E\left[\sum_{t=0}^{\tau(B')-1} \alpha^t /x(0)=i\right]}$, un pré-indice. Il représente un rapport de l'espérance du gain prévisionnel sur l'espérance du temps prévisionnel utilisé à un état initial «i» jusqu'à un instant d'arrêt $\tau(B')$.

Pour que ce pré-indice soit bien défini, l'état initial n'est pas à considérer dans cette formule, soit $\tau(B') \geq 1$. L'indice d'allocation dynamique $\nu(i)$ pour le processus bandit x à l'état «i» est défini par $\nu(i) = \max_{B' \in B} \nu(i, B') = \max_{B' \in B} \dfrac{E\left[\sum_{t=0}^{\tau(B')-1} \alpha^t R(x(t))/x(0)=i\right]}{E\left[\sum_{t=0}^{\tau(B')-1} \alpha^t /x(0)=i\right]}$.

Notons maintenant par k' l'état du processus pour lequel $\nu(k')$ occupe la $k^{ième}$ place dans l'ordre décroissant des indices. 1' représente l'état dont l'indice est le plus grand. 2' est l'état dont l'indice est le second plus grand etc.

A_k est une matrice d'ordre k. S_k et T_k sont des vecteurs de dimension k.

ROBINSON [13] a proposé une méthode itérative pour évaluer les I.A.D.

Soit l'état 1' un état «i» satisfaisant $R(i) = \max_{j \in \Omega} R(j)$.

$\nu(1') = R(1')$ et $A_1(1', 1') = (1-\alpha P(1', 1'))^{-1}$.

Pour k = 2, 3, …, n, l'état k' est un certain état «i» non inclus dans l'ensemble {1', 2',…, (k - 1)'} vérifiant la formule suivante:

$$\frac{R(i) + \alpha \sum_{l<k} P(i,l') S_{k-1}(l)}{1 + \alpha \sum_{l<k} P(i,l') T_{k-1}(l)} = \max_j \frac{R(j) + \alpha \sum_{l<k} P(j,l') S_{k-1}(l)}{1 + \alpha \sum_{l<k} p(j,l') T_{k-1}(l)} .$$

Où le maximum est pris sur les états «j» non inclus dans l'ensemble {1', 2',…, (k - 1)'}. $\nu(k')$ est calculé selon la formule établie. Les formules, S_k et T_k sont donnés par

$S_{k-1}(l) = \sum_{m=1}^{k-1} A_{k-1}(l, m) R(m')$ et $T_{k-1}(l) = \sum_{m=1}^{k-1} A_{k-1}(l, m)$.

La matrice A_k est donnée par

$A_k(k, k) = \{1-\alpha P(k', k') - \alpha^2 \sum_{l<k} P(k', l') Z_{k-1}(l)\}^{-1}$,

$A_k(i, k) = \alpha A_k(k, k) Z_{k-1}(i)$; i = 1, 2,…, k - 1,

$$A_k(k, j) = \alpha\, A_k(k, k) \sum_{l<k} P(k', l')\, A_{k-1}(l, j) ; \quad j = 1, 2, \ldots, k\text{-}1,$$

et $A_k(i, j) = A_{k-1}(i, j) + \alpha\, Z_{k-1}(i)\, A_k(k, j) ; \quad i , j = 1, 2, \ldots, k\text{-}1.$

Où $Z_{k-1} = \sum_{l<k} A_{k-1}(i, l)\, P(l', k') ; \quad i = 1 , 2 ,\ldots, k - 1.$

Sa justification n'est pas indiquée dans son article. Nous omettons de la produire. L'algorithme a été implémenté sans difficulté. Plusieurs exemples de son application ont été déroulés. Les résultats fournis sont exacts.

Si «N» est la dimension d'espace d'état du processus, de nos calculs, pour déterminer les «N» indices, il faut au plus $\frac{1}{3}(19N^3 + 24N^2 - 19N - 9)$ opérations élémentaires. Il est même considéré comme un algorithme polynomial.

2.4. Implémentation et expérimentation

L'algorithme permet de résoudre des exemples de taille réduite allant à au plus 45 états. Il représente une performance au niveau de temps de calcul. Pour un problème à 45 états, seulement 0.07 secondes sont nécessaires à sa résolution. Pour un nombre d'états fixé, on exécute un nombre déterminé de processus et on chronomètre par l'horloge du système le temps de calcul de chaque processus. La moyenne de ces temps d'exécution d'un processus en fonction du nombre d'états récapitulés ci-dessous, représente le temps de calcul d'un processus. Il est donné en seconde.

Tableau 4.1: Temps d'exécutions d'un processus
en fonction du nombre de ses états.

Nombre d'états	5	10	15	20	25	30	35	40	45
Temps d'exécution (seconde)	0.003	0.0038	0.005	0.0065	0.0098	0.018	0.03	0.05	0.075

Deux critères sont à considérer. Le temps de calcul que fournit l'application de l'algorithme et le nombre maximum d'états que l'algorithme de ROBINSON peut prendre en charge. Il s'est avéré efficace pour le premier critère et limité à 45 états pour le second. Le comportement de ces instants d'exécution en fonction des nombres d'états est une *branche parabolique.* Il est représenté dans la figure suivante.

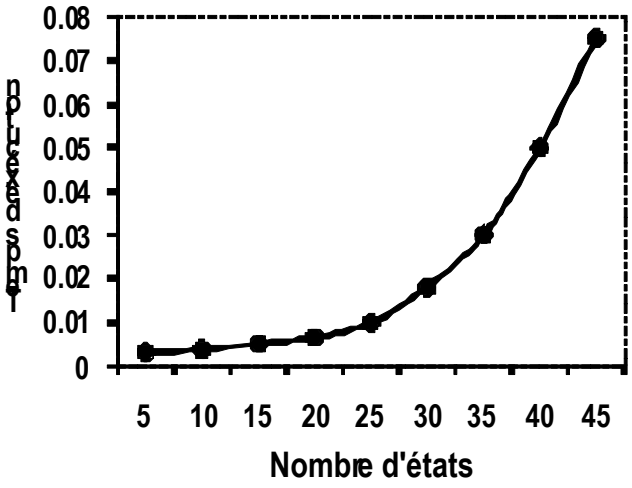


Figure 4.2: Comportement des temps d'exécution en fonction du nombre des états du processus.

Nous rappelons que cet algorithme nous fournit un I.A.D en tout état. Dans le cas des problèmes d'ordonnancement stochastiques, chaque tâche est modélisée par un processus bandit. Pour déterminer un ordonnancement de «N» tâches, il faut associer en tout instant qui représente un état un I.A.D à chaque tâche et exécuter celle qui a l'indice le plus grand. De ce fait, en tout état représenté par le temps, nous appliquons l'algorithme de ROBINSON « N » fois. Une politique optimale appelée aussi ordonnancement est ainsi construite. Nous la relatons dans l'exemple 2.

2.5. Exemple 2 (Politique optimale d'exécution de trois tâches).

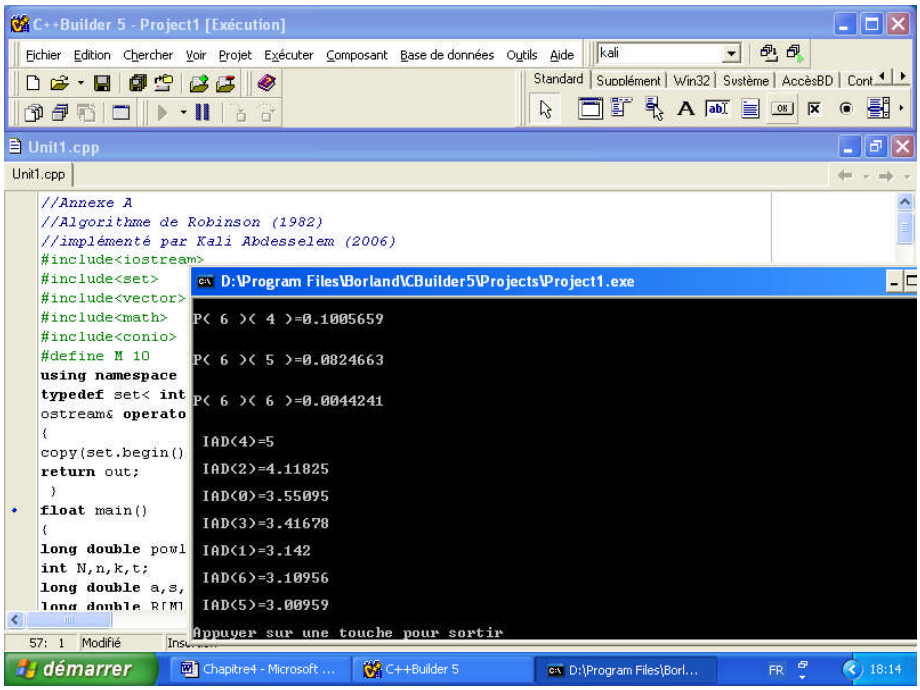
Trois tâches A, B et C à qui on associe des processus bandit à sept états 0, 1, 2, 3, 4, 5, 6 et un taux d'actualisation α égal à 0.95 sont à exécuter sur une machine non soumise à la panne. Le vecteur des gains R en tout état du processus, la matrice des probabilités de transition entre ces états et les I.A.D déterminés par l'algorithme de ROBINSON sont donnés dans le tableau ci-dessous. La procédure genmarkov du logiciel scilab-3.1.1 de calcul numérique, Copyright (c) 1989-2005 Consortium Scilab (INRIA, ENPC), disponible à l'université

Saad Dahleb de Blida, est utilisée pour générer les probabilités de transition entre les états d'un processus.

Tableau 4.2: Récapitulatif des données des trois tâches A, B et C.

Tâche	Etat	R	P							IAD
A	0	3	0.2667087	0.0699819	0.0407185	0.1963534	0.2459537	0.0474827	0.1328012	3.55095
	1	2	0.0319435	0.2157729	0.2097555	0.0706610	0.2280316	0.1253448	0.1184907	3.142
	2	4	0.2029819	0.1736524	0.0813866	0.2457533	0.1016728	0.1080034	0.0865495	**4.11825**
	3	3	0.0740988	0.1964113	0.1138440	0.1946577	0.1213544	0.1851099	0.1145239	3.41678
	4	5	0.1109286	0.0118800	0.23883497	0.1401341	0.2944974	0.1191716	0.0850386	**5**
	5	1	0.3943675	0.0961707	0.0194148	0.0477479	0.27569	0.1020312	0.0645780	3.00959
	6	2	0.2159596	0.1707030	0.2007120	0.2251690	0.1005659	0.0824663	0.0044241	3.10956
B	0	2	0.2111982	0.1423628	0.2204329	0.0391286	0.1093977	0.0015030	0.2759768	2.97273
	1	5	0.0041931	0.1166616	0.1261725	0.1014020	0.1244218	0.3563191	0.1708298	**5**
	2	1	0.0538872	0.1567848	0.1579864	0.0183878	0.2144625	0.2299900	0.1685014	2.86992
	3	1	0.0881583	0.0692747	0.1376232	0.2474729	0.0174963	0.1404969	0.2994776	2.77265
	4	3	0.0893725	0.1792689	0.2537929	0.0108506	0.2389279	0.2019500	0.0258374	3.53409
	5	4	0.0842953	0.2427088	0.1044089	0.1425136	0.1908120	0.0578348	0.1774266	4.20592
	6	3	0.1109269	0.1548336	0.1617209	0.0725593	0.1102176	0.0423768	0.3473649	**3.41457**
C	0	4	0.0479931	0.1557336	0.1236041	0.2118812	0.0754382	0.2085902	0.1767595	**4.32069**
	1	5	0.2580282	0.29972442	0.0792042	0.0732405	0.2025573	0.0149257	0.0723198	**5**
	2	3	0.0001295	0.0400321	0.1353784	0.1830477	0.2936415	0.2821172	0.0656538	3.65367
	3	3	0.1156697	0.1963908	0.0757983	0.1266331	0.1529736	0.0924286	0.2401060	**3.70073**
	4	2	0.1991802	0.1982750	0.2644403	0.0874774	0.0806180	0.1241725	0.0458366	3.43292
	5	5	0.1501896	0.1736024	0.1559548	0.1353791	0.1511893	0.0670771	0.1666077	**5**
	6	2	0.2644488	0.0617796	0.0957309	0.1502044	0.1261007	0.0398366	0.2618990	3.32705

Pour la tâche A, du clavier de l'ordinateur et de la touche impr écran / Syst, une image écran 4.2. défile l'exécution du programme de l'algorithme de ROBINSON qu'on a réalisé.



Ecran 4.2: Fenêtres du programme et de son exécution de la tâche A.

De la lecture de cette image écran, nous remarquons que les I.A.D en tout état de la tâche A sont fournis dans un ordre décroissant.

De même pour les tâches B et C, des images écrans peuvent s'obtenir et qu'on a omis de reproduire.

Comme le coût maximal associé aux deux processus A et B aux états 4 et 1 est identique et est égal à 5, nous remarquons que les I.A.D en ces états sont identiques et sont égaux à ce même nombre.

Pour la détermination d'une politique optimale d'exécution des trois tâches A, B et C nous reproduisons dans un même graphe les histogrammes I.A.D en tout état pour ces tâches.

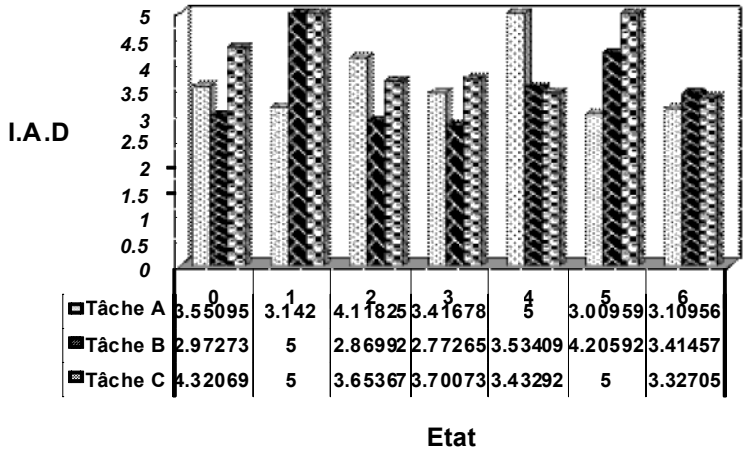


Figure 4.3: Détermination de la politique optimale.

A l'instant 0, on doit exécuter la tâche C durant une unité de temps. A l'instant 1 on continue l'exécution de la tâche C ou on fera son interruption et on basculera sur la tâche B. Ce dernier choix entraîne une perte de temps et un coût de changement de contexte est encourut. Il est préférable de continuer l'exécution de la tâche C. La sous-politique optimale exécute respectivement les tâches A, C, A, C et B durant une unité de temps. On obtiendra que la politique optimale est (C, C, A, C, A, C, B) ou (C, C, A, C, A, C, B).

3. Algorithme de VARAIYA et AL. [17]

Soit "N" machines indépendantes les unes des autres à qui on associe des couples de paramètres $\{R_i(t), F_i(t)\}$ où $R_i(t)$ est le gain octroyé quand la machine «i» est utilisée à l'instant t et $F_i(t)$ est l'information disponible sur cette machine après son utilisation durant les instants 1, 2, …, t - 1. A chaque instant t, on utilise une seule machine et on gèle les autres. A un instant t, si on choisit d'utiliser la machine «i» le coût octroyé est $R_{i(t)}$. Pour le reste des machines non utilisées «j» le coût associé est nul. L'objectif est de déterminer quel type de machines est à utiliser afin de maximiser l'espérance des gains prévisionnels $E\sum_{t=1}^{\infty} a^t R_{i(t)}$ où a, $0 < a < 1$ est un taux d'actualisation.

VARAIYA et AL. [17] montrent que la politique optimale est celle d'indice. Elle associe à chaque machine «i» à l'état x_i un indice $\nu_i(x_i)$ défini par l'expression

$$\max_{\tau>1} \frac{E\left\{\sum_{t=1}^{\tau-1} a^t R_i(x_i(t)) / x_i(I) = x_i\right\}}{E\left\{\sum_{t=1}^{\tau-1} a^t / x_i(I) = x_i\right\}}.$$

En tout instant, on utilise la machine à qui on a associée le plus grand indice. En cas d'égalité des plus grands indices, le choix de la machine est arbitraire entre celles qui réalisent ce maximum.

Dans leur article, VARAIYA et AL. fournissent un algorithme sous une forme non explicite mais indique pour tout état «i» la façon de déterminer l'indice ν_i. Nous le formulons dans une version universelle [16].

Algorithme 2

- **Input** : L'espace d'état Ω ;
  La valeur du taux d'actualisation α dans l'intervalle $]0, 1[$;
  Les valeurs des probabilités de transition P_{ij} $\forall$ i, j $\in \Omega$;
  Les valeurs des gains R_i $\forall$ i $\in \Omega$;
- **Output** L'indice d'allocation dynamique ν_i pour tout état i de Ω.

Etape 1:

- $Y := \{ i \in \Omega \ / \ R_i = \max_{j\in\Omega} R_j \}$

- ***Pour*** tout état i de Y ***faire*** $\nu_i := R_i$;

Etape 2 : **Tant que** $Y \neq \Omega$ **faire**

- $Z := \Omega \backslash Y$**;**
- ***Pour*** tout état i de l'ensemble Y ***calculer*** V_i (resp. W_i) à partir du système linéaire

$$\begin{cases} V_i = R_i + \alpha \sum_{j \in Y} p_{ij} V_j \\ i \in Y \end{cases} \quad (\text{resp.} \begin{cases} W_i = 1 + \alpha \sum_{j \in Y} p_{ij} W_j \\ i \in Y \end{cases}) ;$$

- ***Pour*** tout état i de l'ensemble Z calculer V_i (resp. W_i) puis le rapport $\frac{V_i}{W_i}$

$V_i := R_i + \alpha \sum_{j \in y} P_{ij} V_j$ (resp. $W_i := 1 + \alpha \sum_{j \in y} P_{ij} W_j$);

- $T := \{ i \in Z / \frac{V_i}{W_i} = \max_{j \in Z} \frac{V_j}{W_j} \}$;
- ***Pour*** tout état i de T, $\nu_i := \frac{V_i}{W_i}$;
- $Y := Y \cup T$**;**

Fin tant que

Initialement l'algorithme détermine les états dont le gain est maximum. L'I.A.D en ces états sont définis par cette valeur. Dans sa seconde étape, à chaque itération k, une résolution successive de deux systèmes linéaires est indispensable à la détermination des états qui ont le même indice. L'ensemble des états à considérer dans la résolution se réduira en éliminant ces derniers à la prochaine itération. Un I.A.D en d'autres états se déterminera.

L'exemple suivant illustre les étapes de cet algorithme.

3.2. Exemple 3

A, B et C représentent les états d'un processus bandit et formeront l'ensemble Ω.

Les gains en tout état sont donnés par R(A) = 5, R(B) = R(C) = 3. Les probabilités de transition entre les états sont tel que $P_{ij} = 1/3$, $\forall$ i, j $\in \Omega$. Le taux d'actualisation α étant de 0.9.

- Etape 1

Le gain maximal est 5 et l'état qui lui est associé est «A».

Initialisons l'ensemble Y à { A } et posons $\nu_A = R(A) = 5$.

Y est différent de Ω. La condition d'arrêt de l'algorithme n'est pas vérifiée. Passons à la seconde étape:

- Etape 2

Le nouvel ensemble d'exécution est $Z := \Omega \setminus Y = \{B, C\}$.

Comme Y ne contient qu'un seul état «A», on doit calculer respectivement V_A et W_A.

V_A est solution de l'équation $V_A = R_A + \alpha P_{AA} V_A$. Sa valeur est de 50/7.

De même W_A est solution de l'équation $W_A = 1 + \alpha P_{AA} W_A$. Sa valeur est de 10/7.

Pour tout état «i» de Z = {B, C}, déterminons les valeurs V_i, W_i et du rapport $\frac{V_i}{W_i}$.

$V_B = R_B + \alpha P_{BA} V_A = 36/7$;

$W_B = 1 + \alpha P_{BA} W_A = 10/7$; $\frac{V_B}{W_B} = 18/5$.

Analogiquement, on obtient $\frac{V_C}{W_C} = 18/5$.

D'où $\frac{V_B}{W_B} = \frac{V_C}{W_C} = \max_{j \in Z} \frac{V_j}{W_j}$.

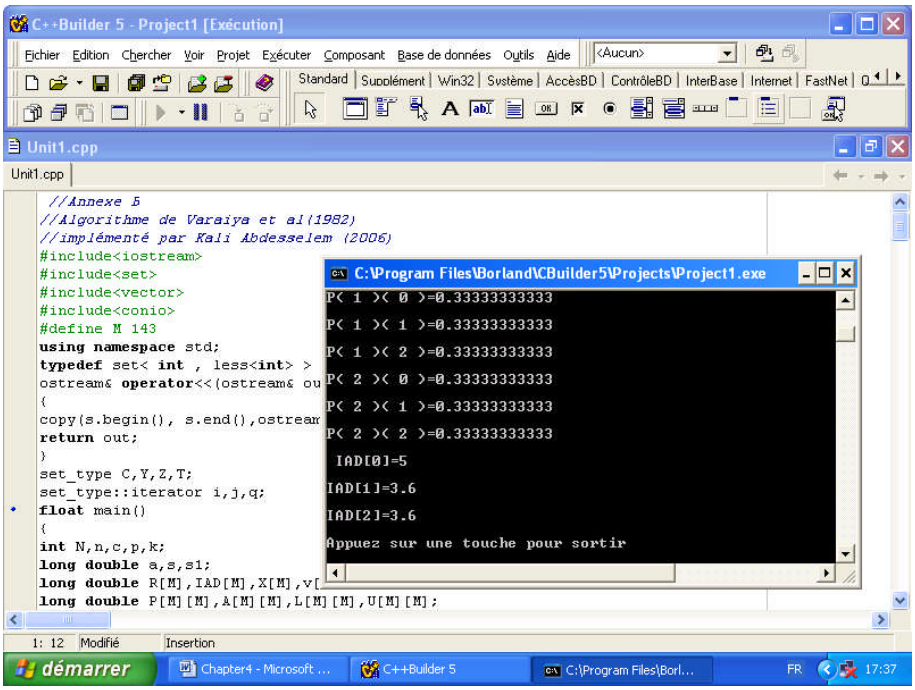
L'ensemble T donné dans l'algorithme se composera des états B et C.

Posons alors que $\nu_B = \nu_C = 18/5$.

Déterminons une deuxième fois l'ensemble Y défini par

$Y := Y \cup T = \{A, B, C\} = \Omega$. Ainsi la condition d'arrêt de l'exécution de l'algorithme est vérifiée.

Une image de l'écran du programme qu'on a implémenté et de son exécution est ci-dessous reproduite.



Ecran 4.3: Fenêtres du programme et de son exécution.

Les I.A.D en tout état du processus et leur comportement sont illustrés dans la figure suivante.

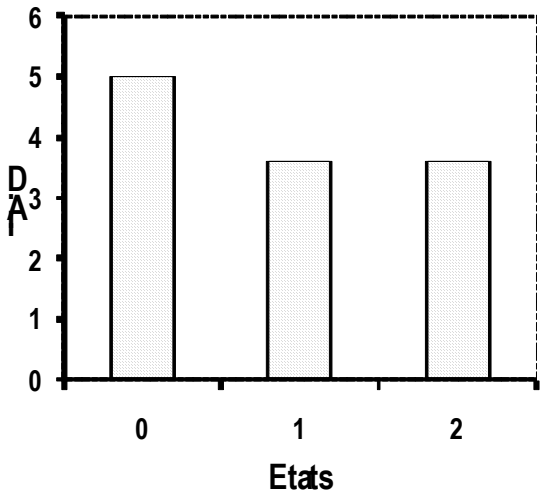


Figure 4.4: Représentation des I.A.D en tout état du processus.

3.3. Justification

Soit x un processus bandit d'espace d'état $\Omega = \{ 0, 1,\ldots, N - 1 \}$. Dans le cas où les I.A.D ν_i sont tel que $\nu_0 \geq \nu_1 \geq \ldots \geq \nu_{N-1}$, l'instant optimal τ_i pour ν_i est donné par $\tau_i = \min\{ t > 0 / x(t) \notin \{ 0, 1,\ldots, i - 1\} \}$.

En effet, du théorème 3.3 chapitre trois, pour tout état «i»; i = 0, 1, … , N-1 l'instant optimal pour ν_i est défini par $\tau_i = \min\{ t > 0 / \nu\{ x(t) \} < \nu_i \}$ et par les inégalités $\nu_0 \geq \nu_1 \geq \ldots \geq \nu_{N-1}$, il découle immédiatement que

$$\tau_i = \min\{t > 0 / x(t) \notin \{0, 1,\ldots, i - 1\}\}.$$

Nous utilisons ce résultat pour déterminer l'état à indice le plus élevé, le second plus élevé indice, le troisième plus élevé indice, etc.

Supposons maintenant qu'il existe «m» tel que $\nu_0 \geq \nu_1 \geq \ldots \geq \nu_{m-1}$ et par un instant d'arrêt τ défini par $\tau = \min\{ t > 0 / x(t) \notin \{ 0, 1, \ldots, m - 1 \} \}$, l'I.A.D ν_m qui occupe la m^ième place est donné par $\nu_m = \dfrac{E\left[\sum_{t=0}^{\tau-1} \alpha^t R(x(t)) / x(0) = m\right]}{E\left[\sum_{t=0}^{\tau-1} \alpha^t / x(0) = m\right]}$.

Il s'ensuit que ν_m est le $\sup_{i \geq m} \nu_i = \sup_{i \geq m}\left\{ \dfrac{E\left[\sum_{t=0}^{\tau-1} \alpha^t R(x(t)) / x(0) = i\right]}{E\left[\sum_{t=0}^{\tau-1} \alpha^t / x(0) = i\right]} \right\}$.

Evaluons méthodiquement la dernière expression de ν_m.

Pour tout état «i», i = 0, 1, …, N – 1, posons

$V_i^m = E[\sum_{t=0}^{\tau-1} \alpha^t R(x(t)) / x(0) = i]$ et $W_i^m = E[\sum_{t=0}^{\tau-1} \alpha^t / x(0) = i]$.

V_i^m et W_i^m sont respectivement solutions des équations

$V_i^m = R(i) + \alpha \sum_{j<m} P_{ij} V_j^m$ et $W_i^m = 1 + \alpha \sum_{j<m} P_{ij} W_j^m$ [14].

Pour les N états, deux systèmes d'équations s'obtiennent:

$$\begin{cases} V_i^m = R(i) + \alpha \sum_{j<m} p_{ij} V_j^m \\ i \in \{0,1,\ldots,N-1\} \end{cases} \quad \text{et} \quad \begin{cases} W_i^m = 1 + \alpha \sum_{j<m} p_{ij} W_j^m \\ i \in \{0,1,\ldots,N-1\} \end{cases}$$

Sous une forme matricielle, les deux systèmes d'équations linéaires s'écrivent respectivement $V^m = (I - \alpha P^m)^{-1} R$ et $W^m = (I - \alpha P^m)^{-1} 1$.

$V^m = (V_0^m, \dots , V_{N-1}^m)^T$, $W^m = (W_0^m, \dots, W_{N-1}^m)^T$ et les probabilités de transition P_{ij}^m sont définies par $P_{ij}^m = \begin{cases} p_{ij} & j < m \\ 0 & j \geq m \end{cases}$. $R = (R(0),\dots, R(N-1))^T$ et $1 = (1, \dots,1)^T$.

L'algorithme a été implémenté sans difficulté. Plus de mille exemples de son application ont été déroulés. Les résultats fournis sont exactes. Il s'exécute polynomialement en $O(N^3)$ où «N» est la dimension d'espace d'états.

3.4. Implémentation et expérimentation

En escomptant d'obtenir une prise en charge des problèmes où les processus sont à un grand nombre d'état et ou le temps de calcul des I.A.D est acceptable, une attention particulière doit être accordée aux choix de la méthode de résolution des systèmes linéaires.

En général, pour les problèmes concrets, les matrices des systèmes linéaires qui leur sont associées présentent des structures peu pratiques. Rarement les matrices sont unitaire, triangulaire, symétrique définie positif…etc. Les méthodes de résolution de ces dits systèmes ne sont pas en un grand nombre. Un choix s'impose.

Nous rappelons que toutes les sous matrices carrées et diagonales d'une matrice quelconque sont inversibles. De ce fait, la décomposition L U s'impose.

Nous avons réalisé des tests de différentes instances, de tailles assez variées générées aléatoirement à partir d'un générateur de données qu'on a réalisé. L'algorithme permet de résoudre des exemples de tailles industrielles, un nombre maximum d'états égal à 143 et avec un temps de calcul très acceptable de l'ordre de 14.7 secondes.

Pour un nombre d'états fixé auparavant, le résultat et les performances indiqués dans le tableau ci-dessous correspondent à la moyenne des temps d'exécution de 100 essais.

Tableau 4.3. Temps d’exécutions d’un processus en fonction du nombre de ses états.

Nombre d'états	Temps d'exécution (seconde)
5	0.005
10	0.007
15	0.008
20	0.01
25	0.02
30	0.04
35	0.07
40	0.11
45	0.17
50	0.25
55	0.35
60	0.49
65	0.70
70	0.87
75	1.13
80	1.45
85	1.84
90	2.28
95	2.81
100	3.43
105	4.15
110	5
115	5.95
120	7
125	8.30
130	9.60
135	11.16
140	12.90
143	14.7

Le comportement des temps d'exécution en fonction du nombre des états du processus qui s'avère d'une allure *branche parabolique* est représenté graphiquement.

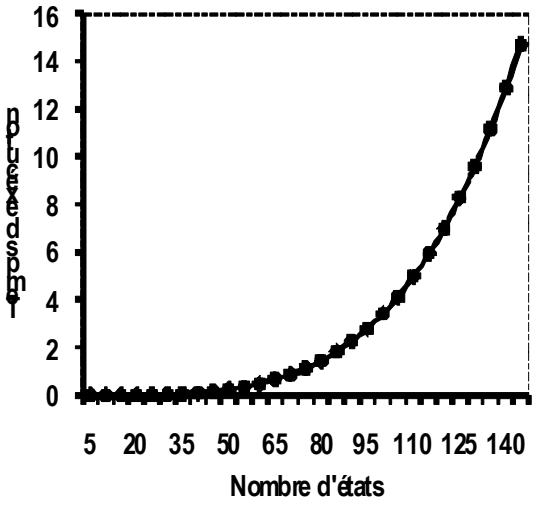


Figure 4.5. Comportement des temps d'exécution
en fonction du nombre des états du processus

L'algorithme de VARAIYA et AL. a été appliqué à un exemple de trois taches A, B et C et une politique optimale est prescrite.

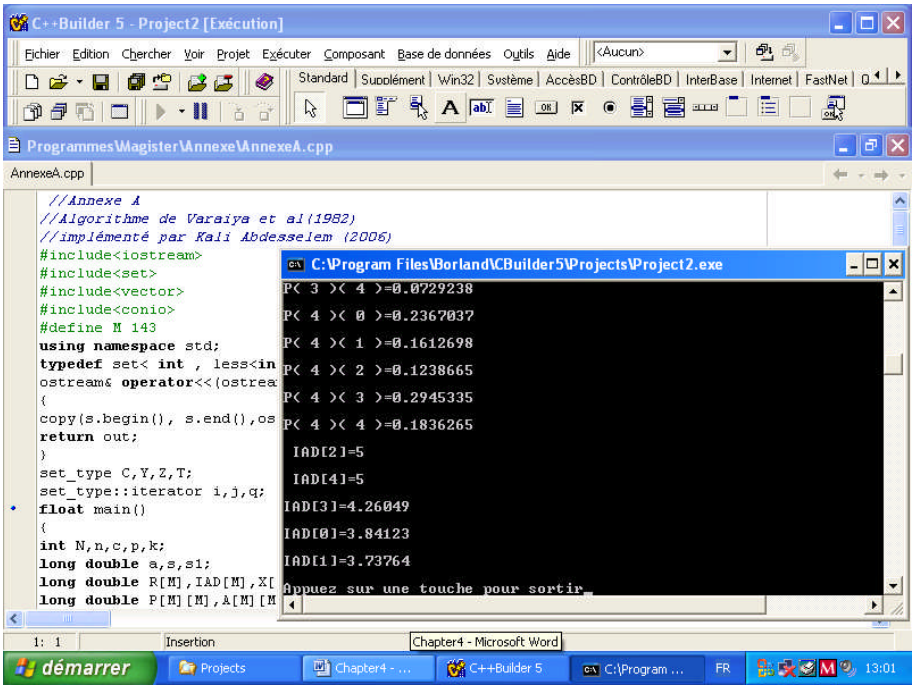
3.5. Exemple 4

Soient trois tâches A, B, C à qui on associe des processus bandit à cinq états 0, 1, 2, 3, 4 et un taux d'actualisation α égal à 0.95 sont exécutées sur une machine non soumise à la panne. Le vecteur des gains R en tout état du processus, la matrice des probabilités de transition entre ces états et les I.A.D déterminés par l'algorithme de VARAIYA et AL. sont récapitulés dans le tableau suivant. Les probabilités de transition sont générées par la procédure genmarkov du logiciel scilab-3.1.1.

Tableau 4.4: Récapitulatif des données des trois tâches.

Tâche	Etat	R	P					I.A.D
A	0	3	0.2183328	0.1357567	0.1461126	0.2041456	0.2956523	**3.84123**
	1	3	0.2566603	0.1686254	0.3195507	0.2340270	0.0211366	3.73764
	2	5	0.2575085	0.3465104	0.0982759	0.0450820	0.2526232	**5**
	3	4	0.3219733	0.3430120	0.1805222	0.0815687	0.0729238	**4.26049**
	4	5	0.2367037	0.1612698	0.1238665	0.2945335	0.1836265	**5**
B	0	1	0.3072229	0.2097444	0.2147215	0.0580455	0.2102657	2.52134
	1	6	0.1793866	0.3285149	0.1003817	0.2452547	0.1464621	**6**
	2	2	0.2663558	0.0266646	0.0780461	0.0933593	0.5355742	2.75278
	3	2	0.1150713	0.2058842	0.3325559	0.2978490	0.0486396	2.92993
	4	3	0.2942455	0.1227804	0.0985678	0.3914525	0.0929538	3.43493
C	0	2	0.2902854	0.0298295	0.2374428	0.2846716	0.1577707	2.65299
	1	5	0.3109079	0.2774897	0.1148231	0.152195	0.1445843	5
	2	2	0.2620476	0.2042602	0.0713056	0.2877103	0.1746763	2.89905
	3	3	0.2948375	0.3123024	0.0930707	0.0922700	0.2075194	3.57438
	4	2	0.3371358	0.0762071	0.3391214	0.1200192	0.1275165	2.63499

Pour la tâche A, l'image écran 4.4 ci-dessous défile l'exécution du programme de l'algorithme de VARAIYA et AL. qu'on a réalisé.



Ecran 4.4: Fenêtres du programme et de son exécution de la tâche A.

De la lecture de cette image écran, nous remarquons que les I.A.D en tout état de la tâche A sont fournis dans un ordre décroissant. Le coût maximal associé au processus aux états 2 et 4 est égal à 5 et constitue la valeur de l'I.A.D en ces états.
Identiquement, des images écran pour les tâches B et C peuvent être reproduits.
Il est optimal d'exécuter la tâche B à l'état «1».
La politique optimale gèle toujours la tâche C dans tous ses états.

Nous reproduisons dans un même graphe les histogrammes des I.A.D en tout état pour ces tâches A, B et C et à partir duquel une politique optimale d'exécution est indiquée.

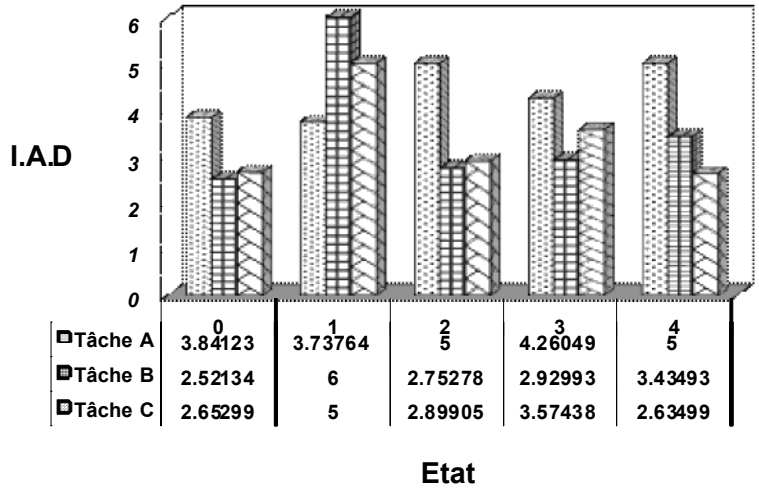


Figure 4.6. Détermination de la politique optimale.

Il s'agit d'exécuter dans cet ordre (A, B, A, A, A). La politique optimale gèle toujours la tâche C.

4. Algorithme de SONIN [15]

Cet algorithme est présenté pour déterminer l'I.A.D noté γ, un coût d'arrêt optimal et caractérisé par NASH [12].

ν, la caractérisation donnée par GITTINS est liée à celle de NASH par la formule $(1 - \alpha) \gamma$.

Dans son article, SONIN fournit un algorithme sous une forme non explicite mais indique pour tout état «i» la façon de déterminer l'indice γ_i. Nous le formulons dans une version universelle [16].

4.1. Algorithme 3

- **Input** :

  L'espace d'état Ω ;

  La valeur du taux d'actualisation α dans l'intervalle $]0, 1[$;

  Les valeurs des probabilités de transition $P(i, j)$ $\forall i, j \in \Omega$;

  Les valeurs des gains $R(i)$ $\forall i \in \Omega$;

- **Output** L'indice d'allocation dynamique ν_i pour tout état i de Ω.

Etape 1: ***Ajouter*** à l'espace d'état Ω un état absorbant x* et ***recalculer*** les probabilités de transition $P(i, j)$ $\forall i , j \in \Omega \cup \{x^*\}$:

$P(i, j) := \alpha P(i, j)$ $\forall i , j \in \Omega$;

$P(i, x^*) := 1- \alpha$ $\forall i \in \Omega$;

$P(x^*, i) := 0$;

$P(x^*, x^*) := 1$;

Etape 2: Tant que $\Omega \neq \phi$ **faire**

1) $\mu (i) := R(i) / P(i, x^*)$ $\forall i \in \Omega$;

2) $\mu := \max_{i \in \Omega} \mu (i)$;

3) $D := \{ i \in \Omega / \mu (i) = \mu \}$;

4) $\nu(i) := (1 - \alpha)\times \mu$ $\forall i \in D$;

5) $\Omega := \Omega \setminus D$**;**

6) ***Si*** $D = \{ k \}$ (i.e. D contient un seul élément noté k) ***faire***

$P(i, j) := P(i, j) + P(i, k)(1- P(k, k))^{-1} P(k, j)$ $\forall i , j \in \Omega \cup \{x^*\}$;

$R(i) := R(i) + P(i, k)(1- P(k, k))^{-1} R(k)$ $\forall i \in \Omega$;

Sinon (i.e. D contient plus d'un élément) ***faire***

P': = {P(i, j)} i , j $\in \Omega \cup \{x^*\}$;

H := {P(i, j)} i $\in \Omega \cup \{x^*\}$ et j $\in$ D;

Q : = {P(i, j)} i , j $\in$ D ;

T : = {P(i, j)} i $\in$ D et j $\in \Omega \cup \{x^*\}$;

$N : = (I - Q_1)^{-1}$;

P : = P'+ H N T ;

R_Ω : = { R(i)} i $\in \Omega$;

R_D : = { R(i)} i $\in$ D;

R : = R_Ω + H N R_D ;

Fin tant que

Dans sa première étape, un état absorbant noté x* est ajouté à l'espace d'état Ω et la matrice des probabilités de transition entre les états est étendue à l'espace $\Omega \cup \{x^*\}$. A la deuxième étape et à chaque itération, on détermine les états dont le rapport du gain par la probabilité de transition à l'état x* est maximal. L'I.A.D "ν" en ces états est initialisé à cette valeur obtenue multipliée par (1 - α). La matrice des probabilités de transition entre les états de l'espace ($\Omega \cup \{x^*\}$)\ D et le vecteur des gains en tout état de l'espace Ω \ D sont déterminés par les formules explicitées dans l'algorithme. Dans le modèle réduit Ω représente Ω \ D. Le principe de la seconde étape peut être répété.

Pour les preuves du théorème 4.1 et des lemmes 4.1, 4.2, 4.3, nous référons le lecteur à l'article [15]. Cet exemple illustre les différentes étapes de l'algorithme.

4.2. Exemple 5

Pour des raisons de simplicité de déroulement et de calculs, nous considérons toujours un processus bandit d'espace d'état Ω = {A, B, C}. L'algorithme prend en charge des processus à cent treize états. Le vecteur des gains R en tout état du processus et la matrice P des probabilités de transition entre ces états sont tel que $R = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$ et $P = \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/2 & 1/3 & 1/6 \\ 1/9 & 5/9 & 1/3 \end{bmatrix}$. Le taux d'actualisation α étant de 0.9.

Etape 1

Ajoutons à l'espace d'états Ω un état absorbant x*. En appliquant le lemme 4.3, la matrice des probabilités de transition pour le nouveau espace d'état $\Omega \cup \{x^*\}$ est $P = \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.1 \\ 0.45 & 0.3 & 0.15 & 0.1 \\ 0.1 & 0.5 & 0.3 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Etape 2

Pour tout état «i» de l'espace $\Omega = \{A, B, C\}$, déterminons la valeur de $\mu(i)$.

$\mu(A) = R(A) / P(A, x^*) = 0.3$; $\mu(B) = R(B) / P(B, x^*) = 20$;

$\mu(C) = R(C) / P(C, x^*) = 10$. Remarquons que $\mu(A) = \max_{i \in \Omega} \mu(i) = 30$.

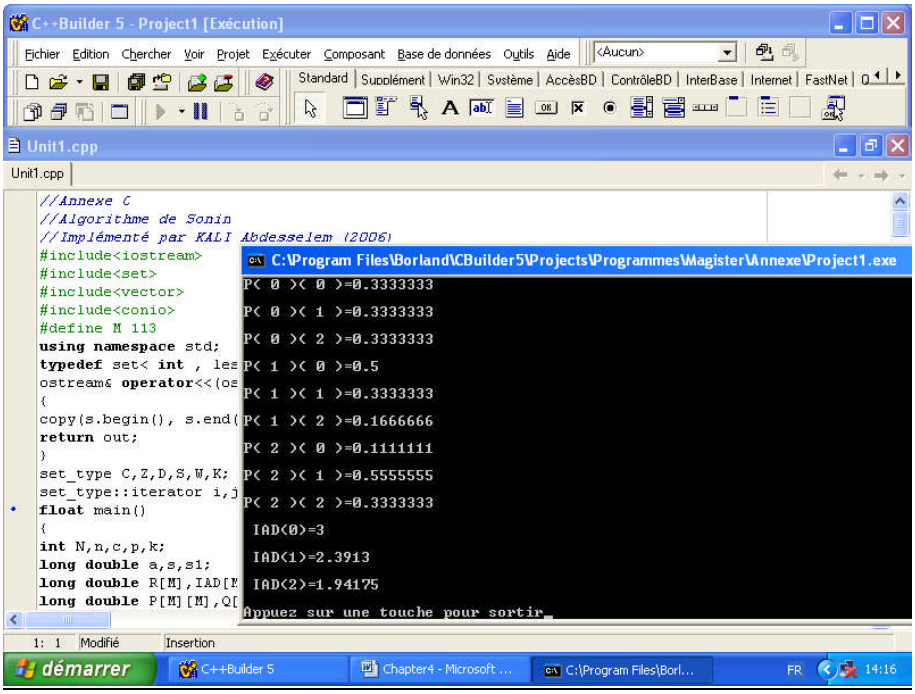
D'où $D = \{A\}$. Par le lemme 4.1, $\gamma(A) = \mu(A) = 30$ et l'I.A.D à l'état «A» est $\nu(A) = (1 - \alpha) \times \mu(A) = 3$.

D est réduit de l'ensemble de Ω. L'ensemble D contient un seul élément. Des formules de calculs (3) et (4) du théorème 4.1 de nouveaux gains et de nouvelles probabilités de transition sont recalculés. Recalculons le vecteur des gains R pour $\Omega = \{B, C\}$ et la matrice des probabilités de transition P pour $\Omega \cup \{x^*\} = \{B, C, x^*\}$, on obtient $R = \begin{bmatrix} 3.93 \\ 1.43 \end{bmatrix}$ et $P = \begin{bmatrix} 0.5 & 0.34 & 0.16 \\ 0.54 & 0.34 & 0.12 \\ 0 & 0 & 1 \end{bmatrix}$.

Après l'élimination de l'état «A» dans la première étape, Ω est non vide. La condition d'arrêt n'est pas vérifie. Dans la suite de l'exécution du programme pour cet exemple, le principe de la deuxième étape est répété encore deux fois.

Une première fois pour l'espace d'états $\Omega = \{B, C\}$. On a obtenu que l'état associe à μ maximum est B et l'I.A.D est $\nu(B) = 2.456$. Une deuxième fois, après l'élimination de B, pour $\Omega = \{ C \}$ l'I.A.D à l'état C est $\nu(C) = 1.941$.

Une image de l'écran du programme qu'on a implémenté et de son exécution est ci-dessous reproduite.



Ecran 4.5: Fenêtres du programme et de son exécution.

La figure suivante rapporte les I.A.D en tout état du processus et illustre leur comportement.

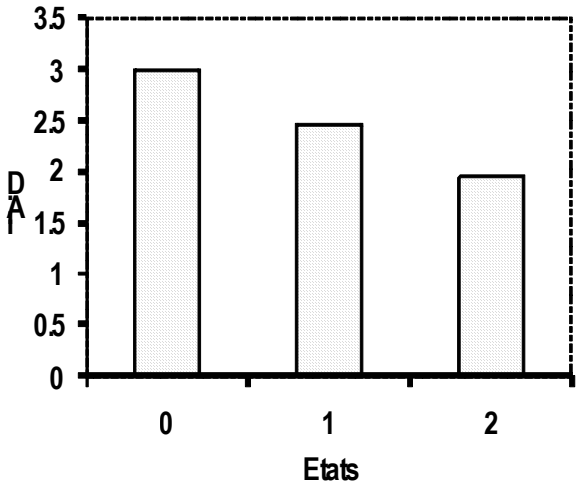


Figure 4.7: Représentation des I.A.D en tout état du processus.

4.3. Justification

L'indice d'allocation dynamique $\gamma(i)$ est défini dans le chapitre trois comme étant la plus petite valeur de $\mu(i)$ pour laquelle $V_\alpha(i, \mu(i)) = \mu(i)$.

$\gamma(i)$ est la valeur pour laquelle il est optimal de s'arrêter immédiatement. L'instant d'arrêt optimal $\tau(i) = 0$.

L'expression $V_\alpha(i, \mu(i)) = \mu(i)$ est équivalente à $R(i) + \alpha\, \mu(i) = \mu(i)$.

Ce qui donne que $\mu(i) = \dfrac{R(i)}{1-\alpha}$.

Posons maintenant $\mu = \max_{i\in\Omega} \mu(i)$ et définissons D l'ensemble des états «i» de Ω vérifiant $\mu(i) = \mu$. Le lemme suivant découle aussitôt.

Lemme 4.1

Pour tout état «i» de l'ensemble D on a $\gamma(i) = \mu$.

Et pour tout état «i» de l'ensemble $\Omega \setminus D$ on a $\gamma(i) < \mu$.

Soit X_1 un processus bandit d'espace d'état fini Ω_1, la matrice des probabilités de transition entre les états $P_1 = \{p_1(i, j)\}$ et le vecteur des gains en tout état $R_1 = \{R_1(i)\}$. Pour un sous-ensemble non vide D de Ω_1, posons $\Omega_2 = \Omega_1 \setminus D$.

Le processus bandit X_2 d'espace d'état Ω_2, la matrice des probabilités de transition $P_2 = \{p_2(i , j)\}$ et le vecteur des gains $R_2 = \{R_2(i)\}$ déterminés par le théorème 4.1 est appelé *processus bandit réduit* du processus X_1.

Théorème 4.1 (Kolmogorov, Doblin)

(1) $P_2 = P_1' + H_1 U_1 = P_1' + H_1 N_1 T_1$ avec $P_1 = \begin{bmatrix} Q_1 & T_1 \\ H_1 & P_1' \end{bmatrix}$.

Où $P_1' = \{P_1(i , j)\};\ i, j \in \Omega_2$.

$H_1 = \{P_1(i , j)\};\ i \in \Omega_2$ et $j \in D$.

$Q_1 = \{P_1(i , j)\};\ i, j \in D$.

$T_1 = \{P_1(i , j)\};\ i \in D$ et $j \in \Omega_2$.

N_1 est la matrice fondamentale, c'est-à-dire $N_1 = \sum_{n=0}^{\infty} Q_1^n = (I - Q_1)^{-1}$.

(2) $R_2 = R_{1,\Omega 2} + H_1 N_1 H_{1,D}$.

Où $R_{1,\Omega 2} = \{R_1(i)\};\ i \in \Omega_2$ et $R_{1,D} = \{R_1(i)\};\ i \in D$.

Quand D = {k} on a

(3) $P_2(i, j) = P_1(i, j) + P_1(i, k)(1- P_1(k, k))^{-1} P_1(k, j)$; $i, j \in \Omega_2$.

Et

(4) $R_2(i) = R_1(i) + P_1(i, k)(1- P_1(k, k))^{-1} R_1(k)$; $i \in \Omega_2$.

Soient X_1 et X_2 les deux processus bandits déjà définis.

Posons $\Omega_1 = \{x_n , n = 0, 1, \ldots\}$.

Et soient $\tau_0, \tau_1, \ldots, \tau_n, \ldots$ les instants de zéro, premier, …,n ième… visite de la chaîne (x_n) un état de l'ensemble $\Omega_2 = \Omega_1 \setminus D$. On suppose que τ_n est fini pour tout $n = 0,1,\ldots$ .Ils sont définis par $\tau_0 = 0, \ldots, \tau_n = \min \{ k > \tau_{n-1} , x_k \in \Omega_2 \}$.

$u_1(x, \Omega_2, .)$ et $u_2(x, \Omega_2, .)$ sont respectivement des distributions de la chaîne (x_n) à l'instant τ_1 et τ_2 de la première et deuxième visite un état de Ω_2 à partir de $x \in D$.

Le lemme 4.2 montre que ces deux distributions sont identique et en fait la distribution de (x_n) est gardée aux moments des visites à n'importe quel sous-ensemble d'états de Ω_1.

<u>Lemme 4.2</u>

Soient X_1 et X_2 les deux processus bandits déjà définis,

Alors $u_1(x, \Omega_2, y) = u_2(x, \Omega_2, y)$; $x \in D$ et $y \in \Omega_2$.

Rappelons que dans la première étape de l'algorithme de SONIN, un état absorbant noté x^* est ajouté à l'espace d'états Ω. Une façon de définir la matrice P des probabilités de transition entre les états de $\Omega \cup \{x^*\}$ est données par le lemme 4.3.

<u>Lemme 4.3</u>

$P(i, j) = \alpha P (i, j)$; $\forall i , j \in \Omega$.

$P(i, x^*) = 1 - \alpha$; $\forall i \in \Omega$.

$P(x^*, j) = 0$; $\forall j \in \Omega$.

$P(x^*,x^*) = 1$.

Nous remarquons qu'après chaque réduction d'état la valeur 1 - P(i, x*) sera le nouveau taux d'actualisation à l'état «i» où P est la probabilité définie dans le théorème 4.1.

4.4. Implémentation et expérimentation

L’idée principale de l’algorithme est de réduire l’ensemble des états en éliminant ceux qui ont l'indice le plus élevé, le second plus élevé etc. Nous rappelons que si le nombre d'états à éliminer est plus de un, cette réduction est faite par l'inversion d'une matrice. Qui se détermine par la résolution d’un système linéaire. Pour un problème donné, l’efficacité de l’algorithme en temps de calcul et en nombre d’état pris en charge dépend du choix de la méthode de résolution du système linéaire et reste primordiale pour les résultats espérés. Dans le traitement des problèmes d’ordonnancement stochastique, il s’est avéré que les matrices associées à ses systèmes ne sont pas du type unitaire, triangulaire, symétrique définie positive…etc. qui respectivement les systèmes se résolvent par les méthodes naïve, de remontée, du gradient conjugué…etc. Dans notre cas, tous les sous-matrices diagonales d'une matrice quelconque sont inversibles. L’utilisation de la méthode LU s’impose et sera adoptée dans la suite. Pour plus de détails et une bonne présentation de la méthode, voir [1].

Après plusieurs séries d’expérimentations numériques où les données sont générées aléatoirement, l’algorithme permet de résoudre des problèmes de taille au plus 113 états en un temps de calcul au plus 0.39 secondes. Ces résultats sont surprenants par rapport à ceux donnés par ROBINSON et VARAIYA et AL.

Ils sont reproduits dans le tableau ci-dessous.

Pour chaque nombre d'états fixé, le temps d’exécution indiqué correspond à une moyenne des temps d’exécution de cent problèmes.

Tableau 4.5: Temps d'exécutions d'un processus, fonction du nombre de ses états.

Nombre d'états	Temps d'exécution (seconde)
5	0.001
10	0.002
15	0.0035
20	0.0055
25	0.008
30	0.01
35	0.015
40	0.02
45	0.03
50	0.04
55	0.05
60	0.07
65	0.09
70	0.108
75	0.13
80	0.15
85	0.175
90	0.21
95	0.24
100	0.27
105	0.31
110	0.36
113	0.39

Le comportement des temps d'exécution en fonction du nombre des états du processus, une allure *branche parabolique*, est représenté graphiquement.

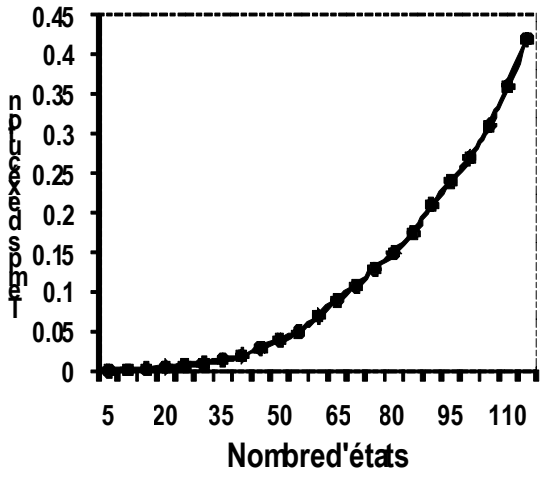


Figure 4.8: Comportement des temps d'exécution
en fonction du nombre des états du processus

Pour plus de compréhension, de simplicité et de clarté, l'algorithme de SONIN est appliqué à un exemple de trois taches A, B et C et une politique optimale est prescrite.

3.5. Exemple 6

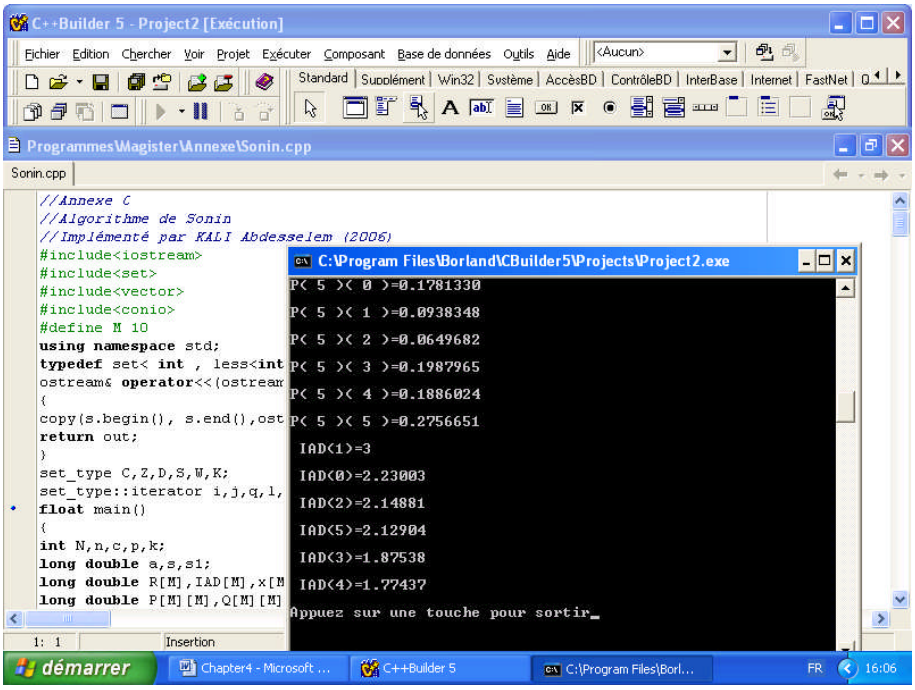
Soient trois tâches A, B, C à qui on associe des processus bandit à six états 0, 1, 2, 3, 4, 5 et un taux d'actualisation α égal à 0.95 sont à exécuter sur une machine non soumise à la panne.

Le vecteur des gains R en tout état du processus, la matrice P des probabilités de transition entre ces états et les I.A.D sont donnés dans le tableau ci-dessous. Les probabilités de transition entre les états sont générées par la procédure genmarkov du logiciel scilab-3.1.1.

Tableau 4.6: Récapitulatif des données des trois tâches A, B et C.

Tâche	R	P						I.A.D
A	2	0.1023311	0.2499700	0.2055333	0.2404499	0.0597980	0.1419177	2.23003
	3	0.0967033	0.2159291	0.0656922	0.2269214	0.2501892	0.1445648	**3**
	2	0.2207388	0.1198769	0.2121000	0.3422456	0.0000862	0.1049525	2.14881
	1	0.1801622	0.3482552	0.0866287	0.0255226	0.1233043	0.2361270	1.87538
	1	0.1378639	0.0890673	0.0959665	0.2327731	0.2761580	0.1681712	1.77437
	2	0.1781330	0.0938348	0.0649682	0.1987965	0.1886024	0.2756651	**2.12904**
B	1	0.1347674	0.1954713	0.2927729	0.2707723	0.0152147	0.0910014	1.77532
	3	0.2964406	0.1895138	0.1305608	0.0681095	0.1548756	0.1604997	**3**
	2	0.3535555	0.2553979	0.1525843	0.0417847	0.0983570	0.0983206	**2.22834**
	2	0.0918926	0.2382347	0.2349686	0.1834160	0.1109586	0.1405295	2.21922
	1	0.1913095	0.2565406	0.0579181	0.0779062	0.1427906	0.2735350	1.71777
	1	0.3294305	0.1567788	0.0896770	0.3128225	0.0574436	0.0538476	1.73497
C	3	0.2795262	0.2122751	0.0394545	0.1279682	0.0738157	0.2669603	**3**
	2	0.0224982	0.3473954	0.0999618	0.2905811	0.2196462	0.0199173	2.28122
	1	0.2270101	0.0137214	0.2360156	0.1611928	0.2086212	0.1534389	1.98412
	2	0.2912134	0.2670593	0.1518322	0.0152665	0.0392800	0.2353486	**2.32325**
	3	0.1812674	0.1312696	0.1681486	0.0714188	0.2150675	0.2328281	**3**
	1	0.1640183	0.1747262	0.2851895	0.2412439	0.0579261	0.0768960	1.95143

Pour la tâche A, l'image écran 4.6 ci-dessous défile l'exécution du programme de l'algorithme de SONIN qu'on a réalisé.



Ecran 4.6: Fenêtres du programme et de son exécution de la tâche A.

De la lecture de cette image écran, nous remarquons que les I.A.D en tout état de la tâche A sont fournit dans un ordre décroissant. Identiquement, des images écran pour les tâches B et C peuvent être associées et reproduits.Le coût maximal associé aux deux processus A et B à l'état «1» est identique et est égal à 3, il découle que l' I.A.D en ces états est égal à ce même nombre. Pour les états 0, 3 et 4 il est optimal d'exécuter la tâche C.

Nous reproduisons dans un même graphe les histogrammes des I.A.D en tout état pour ces tâches A, B et C et à partir duquel une politique optimale d'exécution est indiquée.

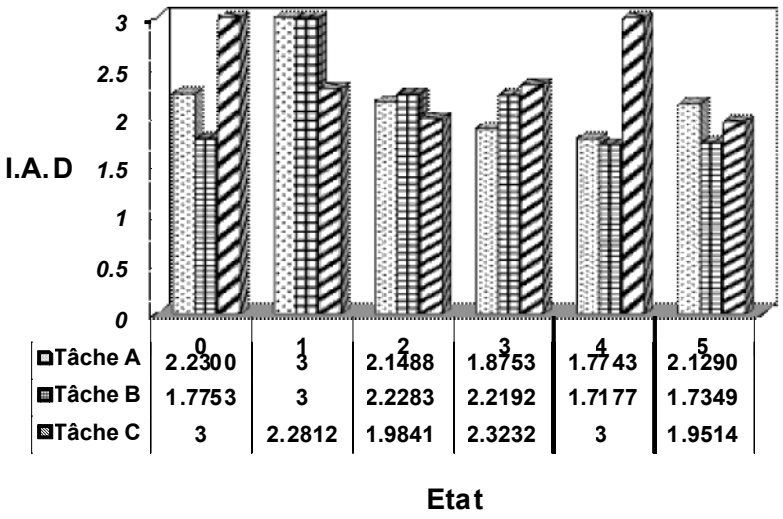


Figure 4.9: Détermination de la politique optimale.

Il s'agit d'exécuter dans cet ordre (C, B, B, C, C, A).

CHAPITRE 5
UN NOUVEL ALGORITHME DE DÉTERMINATION DES INDICES D'ALLOCATION DYNAMIQUES

1. Introduction

Le nombre maximum d'états que les algorithmes de ROBINSON [13], de VARAIYA et AL. [17] et de SONIN [15] peuvent prendre en charge ne dépassait pas respectivement 45, 143 et 113 états.
Dans [11], un nouvel algorithme de détermination des I.A.D est proposé. Il permet de résoudre des problèmes d'ordonnancement stochastiques. Il dérive essentiellement de l'algorithme général de BEALE exposé dans la partie discussion de [06] où des systèmes d'équations linéaires sont à résoudre.
Nous rappelons que si un processus suit une trajectoire {x(t), t = 0, 1,…}, pour toute politique π, on définit le critère de l'espérance du coût prévisionnel par $E_\pi\left\{\sum_{t=0}^{\infty}\alpha^t\ R(x(t))/x(0)=i\right\},\ i\geq 0.$

Dans la fonction objectif, si le taux d'actualisation α, ne dépasse pas la valeur 0.5, une méthode itérative est utilisée pour résoudre les systèmes linéaires. Il peut prendre en charge des problèmes allant jusqu'à cent soixante états. Le temps de calculs est négligeable. Il est le plus rapide parmi les trois autres.
Un cas particulier et d'intérêt est celui où la fonction objectif est $E_\pi\left\{\sum_{i=1}^{n} v_i e^{-\gamma C_i}\right\}$, v_i et C_i sont respectivement les gains et les dates de fin d'exécution du processus «i».
Si le facteur γ tend vers zéro, par un développement limité d'ordre deux au voisinage de 0 de la fonction objectif, nous obtenons que

$$E_\pi\left\{\sum_{i=1}^{n} v_i e^{-\gamma C_i}\right\} = \sum_{i=1}^{n} v_i - \gamma\ E\sum_{i=1}^{n} v_i\ C_i + O(\gamma^2).$$

Une politique d'indices qui maximise l'espérance de la somme des gains, est celle qui minimise le flowtime pondéré $E\sum_i v_i\ C_i$ qui s'interprête par un coût de stockage dans un atelier appelé « coût d'encours ».

Si le taux d'actualisation dépasse 0.5, l'algorithme consiste à utiliser une décomposition LU. Il peut prendre en charge des problèmes à 143 états. Il est présenté en séparant les deux cas de la valeur du taux d'actualisation. Des exemples d'application sont donnés. Sa justification, sa finitude et son implémentation sont élaborées.

2. Nouvel algorithme [11]

GITTINS et JONES [05] ont proposé le premier algorithme de détermination des I.A.D. Ils utilisent le principe de la programmation dynamique qui en général, requiert l'utilisation des méthodes itératives convergentes. Ils montrent que ces indices peuvent être obtenus par la résolution d'un ensemble d'équations fonctionnelles récursives. Si le taux d'actualisation est proche de un, leur ordre de convergence est faible. BEALE, dans la partie discussion de [06], indique un algorithme général où il faut résoudre à chaque itération du déroulement de l'algorithme un ensemble des équations linéaires. Aucune indication sur la résolution n'est donnée. Nous avons pu réduire la taille de ces systèmes et proposé deux méthodes efficaces de leur résolution, point fixe et décomposition LU.

2.1. Cas où le taux d'actualisation est inférieur à 0.5

La méthode utilisée pour résoudre les systèmes linéaires est itérative de type point fixe.

2.1.1. Algorithme_PF

- **Input** L'espace d'état Ω ;
  La valeur du taux d'actualisation α dans l'intervalle $]0, 0.5[$;
  Les valeurs des probabilités de transition $P_{ij} \ \forall\ i, j \in \Omega$;
  Les valeurs des gains $R_i \ \forall\ i \in \Omega$;
  Fixer l'erreur de troncature $\varepsilon > 0$;
  Fixer un état i_0 dans Ω;
- **Output** ν_{i0} L'indice d'allocation dynamique à l'état i_0.

1. $k := 1$; $E_1 := \Omega$;

2. $P_{ij}^k := P_{ij} - P_{i_0 j} \quad \forall i, j \in E_k$;

$R_i^k := R_i - R_{i_0} \quad \forall i \in E_k$;

$\left\| P^k \right\|_\infty := \max_{i \in E_k} \sum_{j \in E_k} \left| P_{ij}^k \right|$;

$\left\| R^k \right\|_\infty := \max_{i \in E_k} \left| R_i^k \right|$;

3. ***Si*** ($\left\| P^k \right\|_\infty = 0$ ou $\left\| R^k \right\|_\infty = 0$) ***faire*** $\nu_{i_0} := R_{i_0}$;

Sinon faire

3.1. t : = 1;

3.2. ***Pour*** tout i dans E_k ***faire*** $y_i^k(0) := 0$;

3.3. ***Si*** $t > \dfrac{\text{Log} \dfrac{\varepsilon}{\left\| R \right\|_\infty}}{\text{Log}(\alpha \left\| P \right\|_\infty)}$ ***aller à*** l'étape 3.5;

Sinon aller à l'étape 3.4;

3.4. ***Pour*** tout i dans E_k ***faire*** $y_i^k(t+1) := R_i^k + \alpha \sum_{j \in E_k} P_{ij}^k \, y_j^k(t)$;

t : = t + 1 ;

Retourner à l'étape 3.3;

3.5. ***Pour*** tout i dans E_k ***faire***

Si $y_i^k(t+1) > 0$ ***insérer*** i dans l'ensemble E_{k+1};

Si ($E_{k+1} = E_k$ ou bien $E_{k+1} = \emptyset$) ***faire*** $\nu_{i_0} := R_{i_0} + \alpha \sum_{j \in E_k} P_{i_0 j} \, y_j^k$;

Sinon k : = k + 1 ; ***allez*** à l'étape **2;**

A l'itération 1, l'ensemble E_1 appelé premier ensemble d'exécution est initialisé à tout l'ensemble des états du processus Ω. En chaque itération k, pour tout ensemble E_k, les indices des composantes strictement positives de la solution d'un système linéaire régissant les I.A.D constituent un nouvel ensemble d'exécution noté E_{k+1}. En cardinalité, il est ainsi plus réduit que E_k. Ce procédé est itératif. Une condition d'arrêt s'impose à savoir un ensemble d'exécution est invariant ou il est vide.

Si E_k représente le dernier ensemble d'exécution non vide, pour tout état fixé i_0, un I.A.D en cet état, est défini par la valeur du gain octroyé en cet état et la sommation prévisionnelle à une facteur α du produit d'une probabilité de transition à un état j de E_k par la valeur strictement positive de la solution du système auparavant résolu à l'itération k.

L’exemple ci-dessous illustre les différentes étapes de l’algorithme.

2.1.2. Exemple 1

Soit un processus bandit d'espace d’état $\Omega = \{0, 1, 2, 3\}$. Les gains en tout état sont donnés par R(0) = 3, R(1) = 2, R(2) = 1 et R(3) = 4. Les probabilités de transition P_{ij} sont tel que $P_{00} = P_{01} = P_{02} = 0.3$, $P_{03} = 0.1$, $P_{10} = 0.45$, $P_{11} = 0.3$, $P_{12} = 0.15$, $P_{13} = 0.1$, $P_{20} = 0.1$, $P_{21} = 0.5$, $P_{22} = 0.3$, $P_{23} = 0.1$, $P_{30} = P_{31} = P_{32} = 0$ et $P_{33} = 1$. L'erreur de troncature ε est fixée à 10^{-7} et le taux d’actualisation α étant de 0.45.

Les trois autres algorithmes fournissent des I.A.D dans un ordre décroissant. La détermination d’un I.A.D à un état peut ne pas être directe et nécessite la détermination des I.A.D en d’autres états de valeur plus grande. Pour cet algorithme, l’utilisateur peut choisir l’ordre des états à qui on associe son I.A.D.

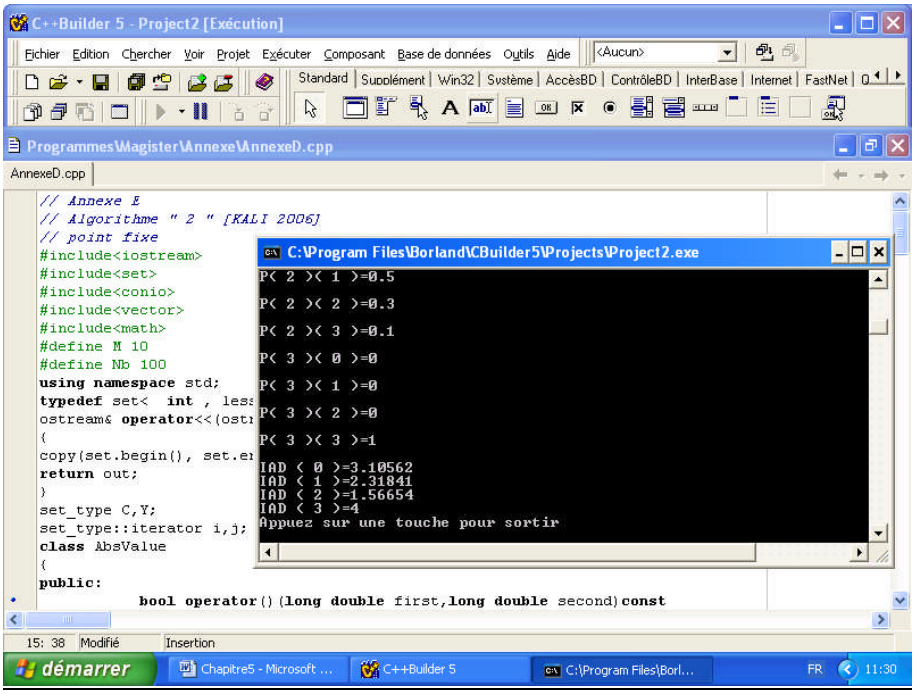
Sans confusion, l’indice de l’ensemble d’exécution constitue l’indice de l’itération de l’algorithme, le second indice est celui de la méthode itérative du point fixe.

Pour l’I.A.D à l’état 0, Le premier ensemble d’exécution est $E_1 = \Omega = \{0, 1, 2, 3\}$. 79 itérations de la méthode itérative du point fixe sont nécessaires pour déterminer le deuxième ensemble d’exécution E_2. A la dernière itération, la solution y du système linéaire Ay = B est constituée d'une seule composante strictement positive à savoir y_3 =2.34709. D'où $E_2 = \{3\}$. E_2 est non vide et il est différent de E_1. La condition d'arrêt n'est pas vérifiée. 17 itérations sont nécessaires pour déterminer la nouvelle valeur de y_3 qui est strictement positive de 1.68067. De ce fait, l'ensemble d'exécution est invariant: $E_3 = \{3\} = E_2$. L'I.A.D à l’état 0, "ν_0", est obtenu et est égal à 3.10562**.**

De la même manière qu'à l'état 0, on obtient l'I.A.D à l'état 1, $\nu_1 = 2.31841$ et l'I.A.D à l'état 2, $\nu_2 = 1.56654$.

Pour l'I.A.D à l'état 3, initialement $E_1 = \Omega = \{0, 1, 2, 3\}$. Le nombre d’itérations pour déterminer le deuxième ensemble d'exécution E_2 est de 81. A la fin d’exécution, toutes les composantes du vecteur y solution du système Ay = B sont négatives et par conséquent E_2 et vide. D'où l'I.A.D à l'état 3 est $\nu_3 = R(3) = 4$.

Une image de l'écran du programme qu'on a implémenté et de son exécution est ci-dessous exposée.



Ecran 5.1 : Fenêtres de programme_PF et de son exécution.

La figure suivante rapporte les I.A.D en tout état du processus et illustre leur comportement.

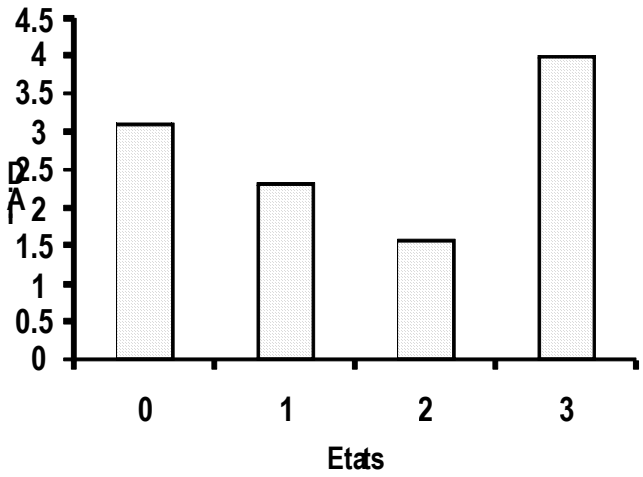


Figure 5.1: Représentation des I.A.D en tout état du processus.

2.1.3. Justification

Considérons un processus bandit "x" d'espace d'états Ω.

"f" une règle d'arrêt pour x si l'état initial est «i_0».

τ est l'instant d'arrêt défini par f supposé fini.

Si la règle f est optimale, le rapport $\frac{V_{i0}}{W_{i_0}} = \frac{E\left[\sum_{t=0}^{\tau-1} \alpha^t R(x(t)) / x(0) = i_0\right]}{E\left[\sum_{t=0}^{\tau-1} \alpha^t / x(0) = i\right]}$

sera le suprémum. Dans ce cas l'I.A.D "ν" à l'état i_0 est tel que $\nu = \frac{V_{i0}}{W_{i0}}$

et par suite $V_{i_0} - \nu W_{i_0} = 0$.

Pour une autre règle d'arrêt "g" qui n'est pas optimale notons par:

R_i le gain pour exécuter le processus quand on est à l'état «i».

P_{ij} la probabilité de transition de l'état «i» à l'état «j».

E_k l'ensemble des états desquels la règle g applique la décision «exécuter» pour la $k^{ième}$ fois.

V_i^k et W_i^k sont respectivement l'espérance du gain prévisionnel et temps prévisionnel quand l'état initial est «i» et que la règle d'arrêt g applique la décision «exécuter» à l'état «i» pour la $k^{ième}$ fois.

Si $i \notin E_k$ on a $V_i^k = W_i^k = 0$.

Sinon V_i^k et W_i^k sont respectivement solutions des équations

$$V_i^k = R_i + \alpha \sum_{j \in E_k} P_{ij} V_j^k \qquad (5.1)$$

et $$W_i^k = 1 + \alpha \sum_{j \in E_k} P_{ij} W_j^k \qquad (5.2) \quad [14]$$

Posons $$\nu_k = \frac{V_{i_0}^k}{W_{i_0}^k} = \frac{R_{i_0} + \alpha \sum_{j \in E_k} p_{i_0 j} V_j^k}{1 + \alpha \sum_{j \in E_k} p_{i_0 j} W_j^k} \qquad (5.3)$$

Un nouvel ensemble d'exécution E_{k+1} est défini par la condition

$$R_i + \alpha \sum_{j \in E_k} P_{ij} V_j^k > \nu_k (1 + \alpha \sum_{j \in E_k} P_{ij} W_j^k) \quad (5.4)$$

C'est-à-dire $V_i^k > \nu_k W_i^k$.

Avec cette algorithme on obtient $\nu_{k+1} \geq \nu_k$ et $\nu_k = \nu$ si $E_{k+1} = E_k$.

L'algorithme peut être énoncé en écrivant $y_i^k = V_i^k - \nu_k W_i^k$.

De (5.1) et (5.2) en déduit que $y_i^k = R_i - \nu_k + \alpha \sum_{j \in E_k} P_{ij}\, y_j^k$ si $i \in E_k$ (5.5).

De (5.3) que $R_{i_0} - \nu_k + \alpha \sum_{j \in E_k} P_{i_0 j}\, y_j^k = 0$ (5.6).

Les conditions (5.5) et (5.6) forment un système d'équations linéaires desquelles y_i^k et ν_k peuvent être calculées. La condition (5.4) peut être écrite $i \in E_{k+1}$ si et seulement si $y_i^k = R_i - \nu_k + \alpha \sum_{j \in E_k} P_{ij}\, y_j^k > 0$.

On peut écrire les équations définissant les valeurs finales de ν et y_i par

$$y_i = \max\left(0, R_i - \nu + \alpha \sum_j P_{ij}\, y_i\right) \quad (5.7)$$

$$R_{i_0} - \nu + \alpha \sum_{j \in E_k} P_{i_0 j}\, y_j = 0 \qquad (5.8)$$

Avec cet algorithme on peut calculer l'I.A.D à un état indépendamment des I.A.D de tous les autres états du processus. Pour le déterminer, on doit déterminer, à chaque itération «k», y_i^k et ν_k en résolvant le système linéaire donné par BEAL

$$\begin{cases} y_i^k = R_i - \nu_k + \alpha \sum_{j \in E_k} p_{ij}\, y_j^k \\ R_{i_0} - \nu_k + \alpha \sum_{j \in E_k} p_{i_0 j}\, y_j^k = 0 \quad (5.9). \\ i \in E_k \end{cases}$$

A chaque itération «k» et pour chaque état «i» dans l'ensemble E_k, nous pouvons remarquer que si l'ensemble d'exécution E_{k+1} est différent de E_k, la détermination de ν_k n'est pas indispensable.

D'où par une différence des équations une et deux du système (5.9).

On obtient le système réduit $\begin{cases} y_i^k = R_i - R_{i_0} + \alpha \sum_{j \in E_k} (P_{ij} - P_{i_0 j})\, y_j^k \\ i \in E_k \end{cases}$ (5.10).

(5.10) a moins d'une équation et d'une inconnue que (5.9).

On peut avoir à une itération «k», $E_{k+1} = \phi$, et par conséquent $\nu_i = R_i$.

Dans la résolution numérique d'un système linéaire, on sera confronté à des erreurs dues aux arrondis et à des erreurs de troncature. Le codage des données en virgule flottante est responsable des erreurs d'arrondis. Pour minimiser ces effets qui peuvent être désastreux, on essaiera d'étudier le *conditionnement numérique* défini par le produit des normes de la matrice, du problème à résoudre, et de son inverse.
Tant que le conditionnement de la matrice du système s'éloigne de «un» les erreurs d'arrondis on une grande influence sur la solution obtenue.
Le conditionnement numérique d'un système est d'autant meilleur que la matrice associée ressemble à la matrice identité ou à une matrice orthogonale.
Il existe néanmoins des méthodes d'équilibrage et / ou de pré conditionnement.
Le bon sens exige que l'on adapte les méthodes employées à la structure des problèmes que l'on résout. La structure et la densité de remplissage ont beaucoup d'importance.
Souvent, on essais, avant de résoudre le système, de réorganiser la matrice (permutation de lignes et de colonnes) pour faire apparaître la structure la plus simple possible.
La complexité d'une méthode numérique est directement liée à la quantité de données à stocker ou à manipuler et au nombre d'opérations élémentaires à effectuer pour sa mise en œuvre dans la résolution d'un problème d'ordre "n".
Parmi les méthodes numériques, on différencie les méthodes directes et les méthodes indirectes. Les premières consistent principalement à «appliquer une formule» nécessitant un nombre fini d'opérations qui dépendant de la dimensionnalité du système. On doit donc obtenir le résultat exact si l'on effectue toutes les opérations avec une précision infinie. Compte tenu des erreurs d'arrondis, il est illusoire de penser que l'on trouvera une solution exacte.
Les méthodes indirectes ou itératives consistent à appliquer, un nombre de fois qui peut être infini, une transformation sur les données formant une suite convergente vers la solution recherchée. En pratique, un test d'arrêt est à imposer. La difficulté consiste à déterminer l'écart entre le résultat obtenu et le résultat désiré. Il dépend des erreurs de troncatures et des erreurs d'arrondis. Pour plus de détails, voir [01].
Dans notre cas d'étude, le problème étudié est quelconque. Certes, à chaque itération de déroulement de l'algorithme, le système linéaire à résoudre $A^k y^k = B^k$ n'est pas prédéfini auparavant. Il est difficile donc de mettre en ouvre un procédure d'équilibrage et / ou de pré conditionnement de la matrice A^k. En outre, ce procédure est très coûteux en terme nombre d'itération à faire. Une autre remarque importante, si à chaque itération «k» la matrice A^k est de rang "p" (p est l'ordre du système $A^k y^k = B^k$), on peut

transformer le système $A^k y^k = B^k$ en un système $A_1{}^k y^k = B_1{}^k$ où $A_1{}^k$ est symétrique définie positive en multipliant les deux membres par ${}^tA^k$. En utilisant l'algorithme du gradient conjugué qui converge vers la solution au plus de "p" itérations, et si le conditionnement de la matrice $A_1{}^k$ est égal à «un» alors on converge en une seule itération. Si le nombre «p» est réduit, le produit des matrices ${}^tA^k\ A^k$ est déterminé rapidement et une solution du système est obtenue. Si «p» est « grand », la multiplication des deux matrices est démesurée. Aucune solution ne peut être obtenue rapidement. En pratique, le nombre d'itérations «k» a réalisé et le rang de la matrice A^k sont inconnus. On ne peut rien décider de l'utilisation de cette méthode. Les données tel le taux d'actualisation, les probabilités de transition et les gains ne fournissent pas des informations suffisantes.

Dans le cas où le taux d'actualisation est inférieur à 0.5 on adapte une méthode itérative de résolution du système linéaire (5.10).
Qui peut s'écrire sous la forme $y^k = F(y^k)$.

$y^k = (y_i^k)_{i\in E_k}$ vecteur colonne et $F(y^k) = R + \alpha\, P\, y^k$.

$R = (R_i - R_{i0})_{i\in E_k}$ vecteur colonne et $P = (P_{ij} - P_{i0j})_{i,j\in E_k}$ matrice d'ordre $|E_k|$.
Par ailleurs, la résolution du système (5.10) revient à la recherche de la racine vectorielle de la fonction F. Donc on est amené à mettre en ouvre un processus de point fixe $\begin{cases} y^k(0)\in R^n \\ y^k(t+1) = R + \alpha\, P\, y^k(t) \end{cases}$ pour déterminer la solution y^k.

Une condition suffisante pour que la suite $\left(y^k(t)\right)_{t\in N}$ converge est que $\alpha < 0.5$.

En effet, la suite $\left(y^k(t)\right)_{t\in N}$ converge si et seulement si le rayon spectral ρ de la matrice $\alpha\, P$ est inférieur à 1 (ie. $\rho(\alpha P) < 1$).
Pour évaluer le rayon spectral, on doit déterminer le polynôme caractéristique et les valeurs propres. On peut montrer l'implication que si $|||\ \alpha P\ ||| < 1$ alors $\rho(\alpha P) < 1$ pour toute norme matricielle subordonnée $|||\ .\ |||$. En dimension fini, toutes les normes vectorielles sont équivalentes et d'après la propriété de la matrice des probabilités de transition, $\sum_j P_{ij} = 1$, il est préférable de choisir d'utiliser la norme vectorielle $\|\ .\ \|_\infty$ et la norme matricielle subordonnée $|||\ .\ |||_\infty$.

On a $|||\ P\ |||_\infty \le 2$

puisque $\left\|\left|P\right|\right\|_\infty = \max_i \sum_j \left|p_{ij} - p_{i_0 j}\right| \le \max_i \sum_j p_{ij} + \max_i \sum_j p_{i_0 j} \le 1+1 = 2$.

De $\alpha < 0.5$ il découle que $\alpha|||\ P\ |||_\infty \le 1$.

Si ε est l'erreur de troncature, l'inégalité $\left(\alpha \left\|\left|P\right|\right\|_\infty\right)^t \left\|R\right\|_\infty \le \varepsilon$ est équivalente à l'inégalité $t > \dfrac{\text{Log}\dfrac{\varepsilon}{\left\|R\right\|_\infty}}{\text{Log}(\alpha \left\|\left|P\right|\right\|_\infty)}$ (si $R \neq 0$ et $P \neq 0$).

Sachant que la norme $\left\| y^k(t+1) - y^k(t) \right\|_\infty = \left(\alpha \left\|\left|P\right|\right\|_\infty\right)^t \left\|R\right\|_\infty$.

Si le taux d'actualisation α tend vers zéro, la convergence est rapide. Dans le cas où R = 0 ou P = 0 l'indice ν_{i0} sera R_{i0}.

2.1.4. <u>Implémentation et expérimentation</u>

L'algorithme présenté ci-dessus a été implémenté. Il a été testé sur des instances ou problèmes de tailles différentes et variées. Les processus considérés sont de «un» état à «cent soixante» états. Les données d'un problème sont sa taille ou la cardinalité de l'espace d'état, le taux d'actualisation de la fonction coût, les valeurs des gains octroyés en tout état et les probabilités de transition entre les états du processus. Elles sont générées à partir d'un générateur de données qu'on a réalisé. C'est un programme informatique qui permet d'obtenir des données dans un ordre aléatoire, de distribution uniforme. Toute méthode itérative engendre des erreurs d'arrêt ou de troncature. Avec une précision de calcul de l'ordre de 10^{-15}, l'algorithme a été appliqué sur un nombre maximum d'états égal à «cent soixante». Son temps de calcul est appréciable de l'ordre de 3.5 secondes.

Pour un nombre d'états fixé, nous considérons par «temps d'exécution», la moyenne des temps d'exécution de 100 instances. Par pas de cinq états, le tableau ci dessous récapitule les résultats obtenus.

Tableau 5.1 : Temps d'exécutions en fonction du nombre des états du processus.

Nombre d'état	Temps d'exécution en seconde
5	0.001
10	0.003
15	0.005
20	0.0075
25	0.01
30	0.015
35	0.02
40	0.04
45	0.05
50	0.07
55	0.1
60	0.13
65	0.18
70	0.22
75	0.3
80	0.35
85	0.44
90	0.55
95	0.66
100	0.80
105	0.97
110	1.12
115	1.3
120	1.44
125	1.65
130	1.89
135	2.09
140	2.4
145	2.6
150	2.89
155	3.2
160	3.5

Une représentation graphique à comportement de type *branche parabolique*, des temps d'exécution en fonction du nombre d'état d'un processus peut facilement s'obtenir.

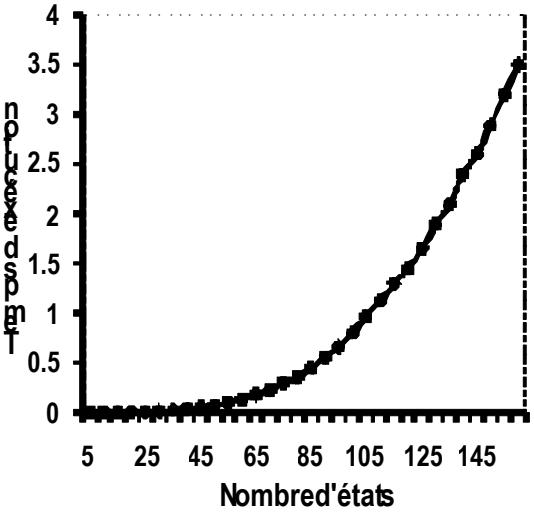


Figure 5.2: Comportement des temps d'exécution
en fonction du nombre des états du processus.

Dans l'exemple suivant, chaque tâche est modélisée par un processus bandit. Dans un ordonnancement discret, les unités du temps peuvent représenter les état du processus. En tout instant, on attribue à chaque tâche un I.A.D. On exécute la tâche qui a le plus grand indice. En cas d'égalité entre deux plus grands indices, on exécute l'une des tâches arbitrairement. Dans ce cas la politique d'exécution des tâches est dite à indices. Une politique optimale est à indices [06].

2.1.5. Exemple 2

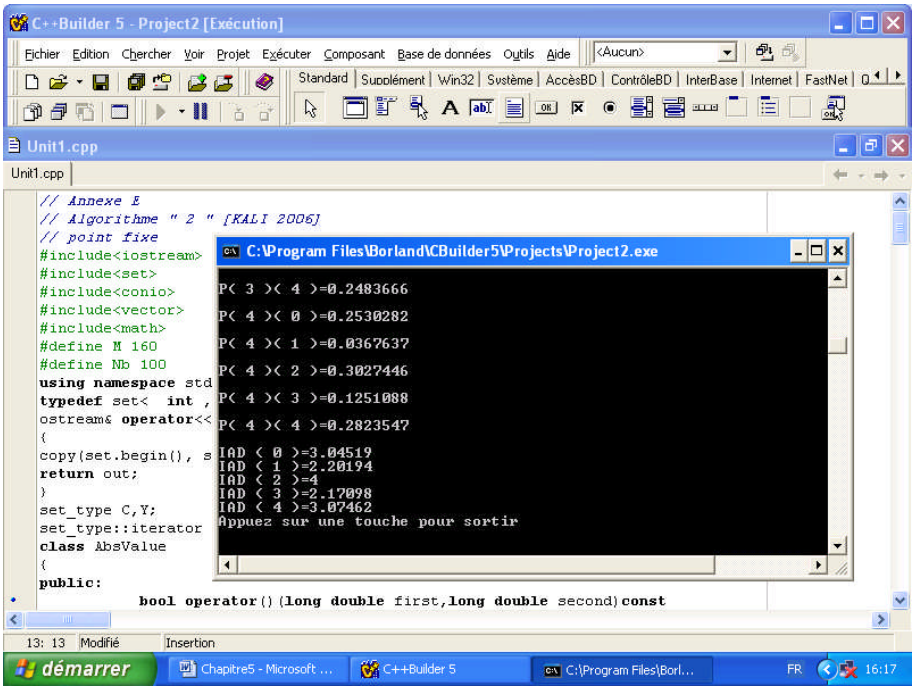
Soient trois tâches A, B et C à exécuter sur une machine non soumise à la panne. A chaque tâche, on associe un processus bandit à cinq états 0, 1, 2, 3, 4 et un taux d'actualisation égal à 0.23. Le vecteur des gains R en tout état, la matrice des probabilités de transition entre ces états et les I.A.D sont donnés dans le tableau ci-dessous. L'erreur de troncature est fixée à 10^{-7}.

La procédure genmarkov du logiciel scilab-3.1.1 de calcul numérique est utilisée pour générer les probabilités de transition entre les états d'un processus.

Tableau 5.2: Récapitulatif des données des trois tâches A, B et C.

Tâche	Etat	R	P					I.A.D
A	0	3	0.0222811	0.2697559	0.1310236	0.2977397	0.2791997	3.04519
	1	2	0.1740691	0.1250174	0.2228307	0.2568908	0.2211919	**2.20194**
	2	4	0.1452540	0.1607858	0.6116494	0.0360346	0.0462762	**4**
	3	2	0.1173445	0.2520207	0.1577082	0.2245600	0.2483666	2.17098
	4	3	0.2530282	0.0367637	0.3027446	0.1251088	0.2823547	**3.07462**
B	0	3	0.2721638	0.0599987	0.3551711	0.3002420	0.0124244	**3.07966**
	1	2	0.2950239	0.3759944	0.0019201	0.0597842	0.2672774	2.13762
	2	4	0.3047965	0.1000199	0.2203982	0.2284823	0.1463031	**4**
	3	2	0.0310411	0.2981923	0.1689453	0.3701371	0.1316842	2.12199
	4	3	0.2118222	0.3741910	0.0966083	0.1256751	0.1917034	3.02852
C	0	3	0.1404611	0.2342576	0.1491704	0.3208727	0.1552382	3.03592
	1	2	0.1026307	0.1848333	0.2563262	0.1797637	0.2764461	2.02759
	2	4	0.0343239	0.1109583	0.3470074	0.2022862	0.3054242	**4**
	3	2	0.1537023	0.0392081	0.1929727	0.2498978	0.3642191	**2.2026**
	4	3	0.2033744	0.2128645	0.0794731	0.1832122	0.3210758	3.01984

Pour la tâche A, l'image écran 5.2 défile l'exécution de l'algorithme _ PF.



Ecran 5.2: Fenêtre du programme et de son exécution de la tâche A.

De la lecture de cette image écran, nous remarquons que les I.A.D en tout état ne sont pas fournis dans aucun ordre prescrit.

Identiquement, des images écrans pour les tâches B et C peuvent être reproduits. A l'état 2, le coût maximum associé aux trois processus A, B et C est de quatre. L'I.A.D en cet état est aussi de quatre.

Pour la détermination d'une politique optimale d'exécution des trois tâches A, B et C nous reproduisons dans un même graphe les histogrammes I.A.D en tout état pour ces tâches.

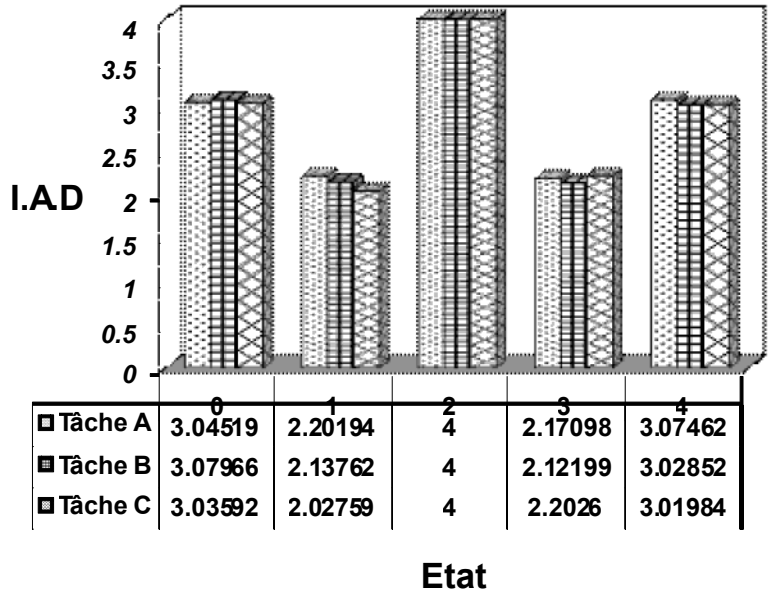


Figure 5.3: Détermination de la politique optimale.

Un graphe indiquant une politique optimale est ci-dessous reproduit.

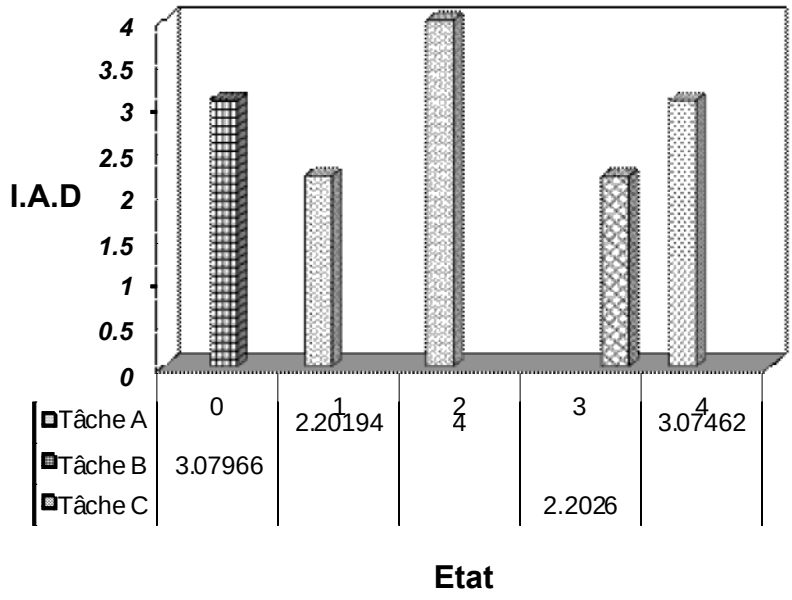


Figure 5.4: Une politique optimale.

De la lecture des deux figures ci-dessus, la politique optimale commence par exécuter respectivement les deux tâches B et A pendant une unité de temps. A l'instant 2, on continue l'exécution de la tâche A ou en fera son interruption et on basculera sur l'une des tâches B ou C. Dans un atelier, le dernier choix d'interrompre la tâche A entraîne une perte de temps et un coût de changement du contexte sera encourut. Il est préférable de continuer l'exécution de la tâche A.
Les politiques optimales sont (B, A, A, C, A), (B, A, B, C, A) et (B, A, C, C, A).

2.2. Cas où le taux d'actualisation est supérieur ou égal à 0.5

La méthode utilisée pour résoudre les systèmes linéaires (5.10) dans le développement de notre algorithme est la factorisation LU.

2.2.1. Algorithme LU

- **Input** L'espace d'état Ω ;

  La valeur du taux d'actualisation α dans l'intervalle [0.5 , 1[;

  Les valeurs des probabilités de transition $P_{ij} \ \forall \ i, j \in \Omega$;

  Les valeurs des gains $R_i \ \forall \ i \in \Omega$;

Fixer un état i_0 dans Ω;

- **Output** ν_{i0} l'indice d'allocation dynamique à l'état i_0.

1. k: = 1 ; E_1: = Ω;

2. ***Pour*** i et j = 0 à n -1 ***faire***

si i = j ***faire*** $a_{ii} := 1 - \alpha(P_{ii} - P_{i_0 i})$;

sinon faire $a_{ij} := -\alpha(P_{ij} - P_{i_0 j})$;

Pour i = 0 à n -1 ***faire*** B_i : = $R_i - R_{i\,0}$;

3.1. ***Pour*** tout i dans l'ensemble E_k (E_k est parcouru au sens croissant) ***faire***

Début

L_{ii}^{k} : = 1;

Pour tout j dans l'ensemble E_k tel que j < i ***faire***

$$L_{ij}^{k} := \left(a_{ij} - \sum_{k=0}^{j-1} L_{ik}^{k}\, U_{kj}^{k} \right) / U_{jj}^{k};$$

Pour tout j dans E_k tel que $j \geq i$ ***faire*** $$U_{ij}^{k} := a_{ij} - \sum_{k=0}^{j-1} L_{ik}^{k}\, U_{kj}^{k};$$

Fin

3.2. ***Pour*** tout i dans E_k (E_k est parcouru au sens croissant) ***faire***

$$X_i^{k} := b_i - \sum_{j<i} L_{ij}^{k}\, X_j^{k};$$

3.3. ***Pour*** tout i dans E_k (E_k est parcouru au sens décroissant) ***faire***

$$Y_i^{k} := (X_i^{k} - \sum_{j<i} U_{ij}^{k}\, Y_j^{k}) / U_{ii}^{k};$$

4. ***Pour*** tout i dans l'ensemble E_k ***faire si*** $y_i^{k} > 0$ ***insérer*** y_i^{k} dans l'ensemble E_{k+1} ;

5. ***Si*** $E_{k+1} = E_k$ ***faire*** $\nu_{i0} = R_{i0} + \alpha \sum_{j \in E_k} P_{i_0 j}\, y_i^{k}$;

Sinon si $E_{k+1} = \phi$ ***faire*** $\nu_{i0} = R_{i0}$;

Sinon faire k = k + 1 et ***retourner*** à l'étape **3.1 ;**

Au risque de se répéter, l'algorithme_LU est semblable à celui de l'algorithme_PF. La différence réside dans la méthode de résolution des systèmes linéaires, de type LU et la valeur du taux d'actualisation prise dans le domaine [0.5, 1[.

Pour une assimilation et une compréhension rapides, illustrons les différentes étapes de cet algorithme dans l'exemple suivant.

2.2.2. Exemple 3

$\Omega = \{0, 1, 2\}$ représente l'espace d'états d'un processus bandit. Les gains en tout état sont donnés par $R(0) = 2$, $R(1) = 1$ et $R(2) = 3$. Les probabilités de transition P_{ij} sont telles que $P_{00} = P_{02} = 0.25$, $P_{01} = 0.5$, $P_{10} = 0.3$, $P_{11} = 0.25$, $P_{12} = 0.45$, $P_{20} = P_{21} = 0$ et $P_{22} = 1$. Le taux d'actualisation α est de 0.9.

Identiquement que dans l'algorithme_PF, l'ordre de détermination des I.A.D avec cet algorithme n'est pas imposé.

Pour l'I.A.D à l'état 0, le premier ensemble d'exécution est $E_1 = \Omega = \{0, 1, 2\}$. La matrice A et le vecteur B du système linéaire Ay = B sont tel que

$$A = \begin{pmatrix} 1 & -0 & -0 \\ -0.045 & 1.225 & -0.18 \\ 0.225 & 0.45 & 0.325 \end{pmatrix} \text{ et } B = \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}.$$

En utilisant la factorisation LU de la matrice A, la solution y du système Ay = B est $y = \begin{pmatrix} 0 \\ -0.302635 \\ 3.49596 \end{pmatrix}$. $y_2 = 3.49596$ est la seule composante strictement positive.

Le second ensemble d'exécution est $E_2 = \{2\}$. Il est non vide et il est différent de E_1. D'où, la condition d'arrêt de l'exécution de l'algorithme n'est pas vérifie. On doit résoudre une seconde fois un système linéaire réduit à une seule équation du type $0.325 \times y_2 = 1$ ou $y_2 = 3.07692$.

L'ensemble d'exécution reste invariant. $E_3 = \{ 2 \} = E_2$.

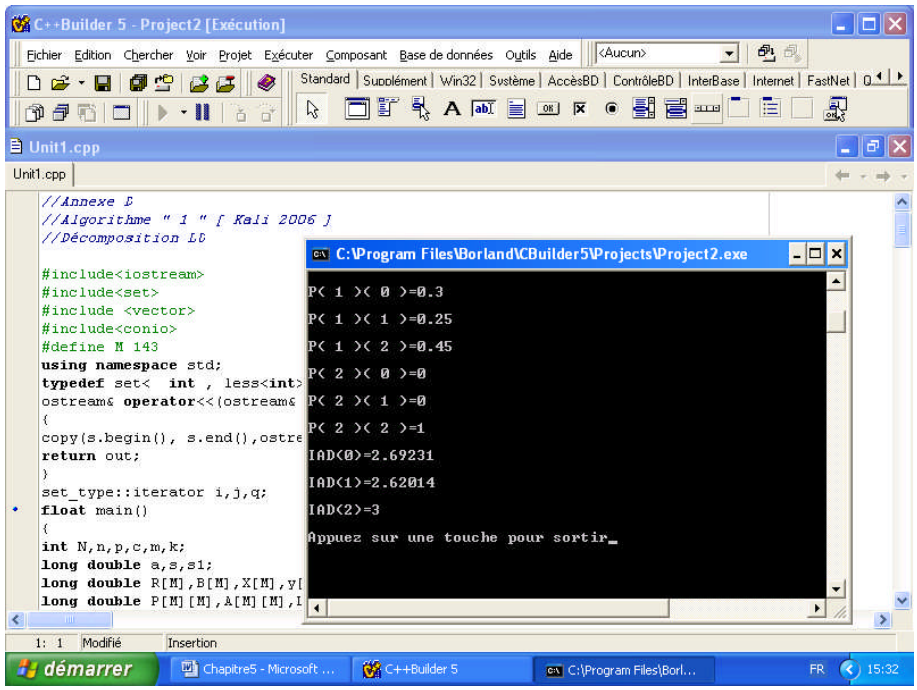
L'I.A.D à l'état zéro, ν_0, est ainsi obtenu et est égal à 2.69231. D'une manière analogue qu'à l'état 0, on obtient l'I.A.D à l'état 1, $\nu_1 = 2.62014$. Pour l'I.A.D à l'état 2, le premier ensemble d'exécution est $E_1 = \{0, 1, 2\}$.

La matrice A et le vecteur B du système Ay = B sont tels que

$$A = \begin{pmatrix} 0.775 & -0.45 & 0.675 \\ -0.27 & 0.775 & 0.495 \\ -0 & -0 & 1 \end{pmatrix} \quad \text{et} \quad B = \begin{pmatrix} -1 \\ -2 \\ 0 \end{pmatrix}.$$

En utilisant la factorisation LU de la matrice A, la solution du système Ay = B est $y = \begin{pmatrix} -3.49596 \\ -3.79859 \\ 0 \end{pmatrix}$. Dans ce cas, toutes les composantes du vecteur y sont négatives ou nulles et par conséquent le second ensemble d'exécution E_2 est vide. Il résulte que l'I.A.D à l'état 2 , $\nu_2 = R(2) = 3$.

Une image de l'écran du programme qu'on a implémenté et de son exécution est ci-dessous exposée.



Ecran 5.3: Fenêtre de programme_LU et de son exécution.

.

La figure suivante représente les I.A.D en tout état du processus et illustre leur comportement.

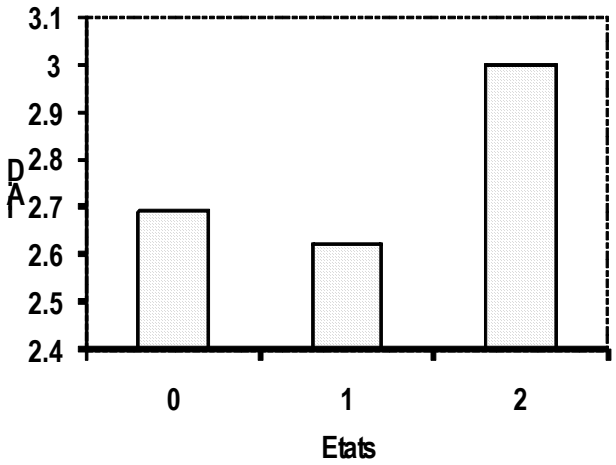


Figure 5.5: Représentation des I.A.D
en tout état du processus.

2.2.3. Justification

Dans le cas où le taux d'actualisation est supérieur ou égal à 0.5 la méthode de résolution du système linéaire (5.10) $\begin{cases} y_i^k = R_i - R_{i_0} + \alpha \sum_{j \in E_k} (P_{ij} - P_{i_0 j})\, y_j^k \\ i \in E_k \end{cases}$ adoptée est «*la factorisation* LU». Le système (5.10) peut s'écrire sous la forme Ay = B où les coefficients a_{ij} de la matrice A sont déterminés par les formules données dans l'étape 2 de l'algorithme.

A chaque itération «k», la matrice A^k du système est factorisée sous forme d'un produit L^k U^k, où L^k est triangulaire inférieure avec une diagonal formée de «un» et U^k est triangulaire supérieure. Pour résoudre le système A^k y^k = B^k, on résout successivement L^k x^k = B^k puis U^k y^k = x^k en utilisant la même place mémoire pour le stockage de L^k et U^k que celui de A^k.

Comme les déterminants des sous matrices diagonales de A^k sont vérifiés et montrés différents de zéro, les sous matrices diagonales de A^k sont inversibles et la méthode LU est applicable. Itérativement, pour calculer les coefficients de L^k et U^k, on procède par identification en utilisant les propriétés de L^k et de U^k.

Pour résoudre les systèmes qui déterminent l'indice d'un état, si "n" est la dimension d'espace d'état, il faut au plus « $n(3n^3+22n^2+69n+50)/12$ » opérations élémentaires.
Et pour "n" états, on aura au plus « n^2 $(3n^3+22n^2+69n+50)/12$ » opérations élémentaires. Ce qui est polynomial en $O(n^5)$. Il peut être considéré comme un élément de la frontière des problèmes faciles et les problèmes exponentiels.

2.2.4. Implémentation et expérimentation

Dans les mêmes conditions d'application que celles de l'algorithme_PF, nous constatons que l'algorithme_LU permet de résoudre des exemples de taille industrielle dans des temps raisonnables. Il a été appliquée sur un nombre maximum d'états égal à 143 avec un temps de calcul de 63.94 secondes. Pour un nombre d'états fixé, le résultat indiqué dans le tableau ci-dessous correspond à la moyenne des temps d'exécution observés sur 100 instances.

Tableau 5.3: Les temps d'exécutions en fonction du nombre des états du processus

Nombre d'état	Temps d'exécution en seconde
5	0.005
10	0.008
15	0.01
20	0.03
25	0.08
30	0.15
35	0.27
40	0.45
45	0.70
50	1.05
55	1.51
60	2.12
65	2.87
70	3.82
75	5.06
80	6.44
85	8.19
90	10.17
95	12.75
100	15.54
105	18.82
110	21.5
115	27
120	31.64
125	37.32
130	43.6
135	50
140	56
143	63.94

En transcrivant les valeurs des temps d'exécution de ces expérimentations par un graphe, il est possible d'obtenir une allure montrant leur comportement en fonction du nombre d'états. Il s'avère une branche *parabolique*.

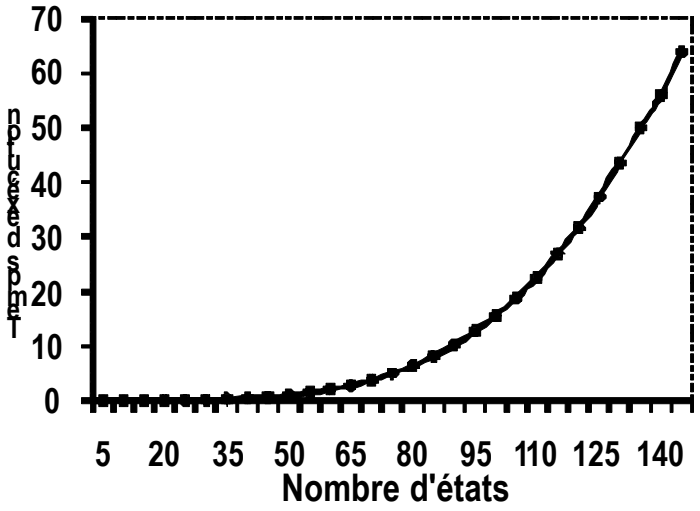


Figure 5.6: Comportement des temps d'exécution
en fonction du nombre des états du processus.

Dans l'exemple suivant, en tout état des trois processus A, B et C l'algorithme_LU est appliqué et une politique optimale est déterminée.

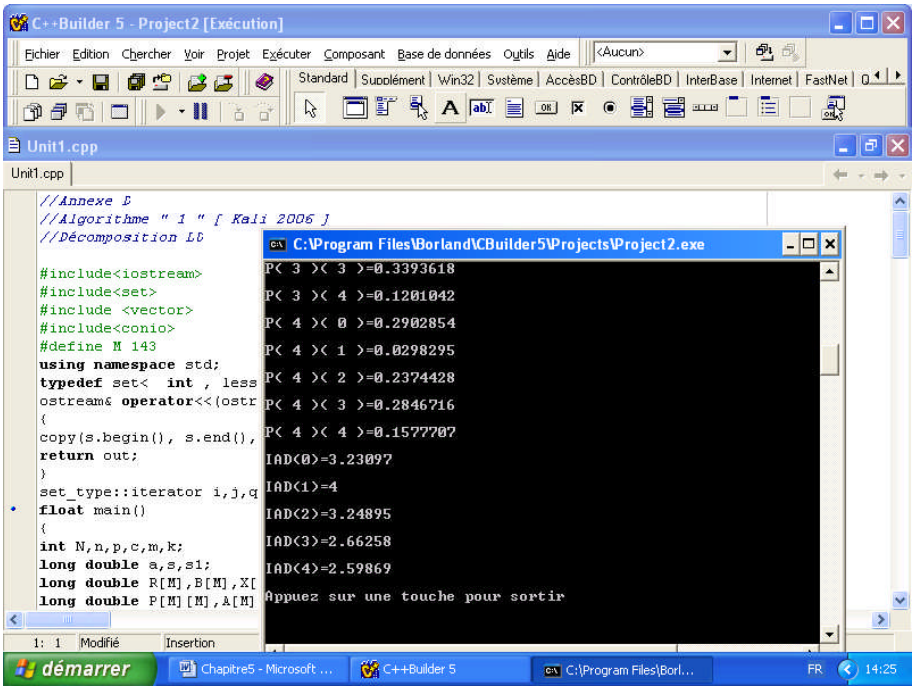
2.2.5. Exemple 2

Trois tâches A, B et C à qui on associe cinq états 0, 1, 2, 3, 4 et un taux d'actualisation α égal à 0.73 sont exécutées sur une machine non soumise à la panne. R, le vecteur des gains en tout état, la matrice des probabilités de transition entre ces états et les I.A.D déterminés par l'algorithme_LU sont reproduits ci-dessous. La même « *procédure genmarkov* » est utilisée pour générer les probabilités de transition entre les états d'un processus.

Tableau 5.4 : Récapitulatif des données des trois tâches A, B et C.

Tâche	Etat	R	P					I.A.D
A	0	3	0.1089164	0.3238717	0.2890601	0.1196108	0.1585410	**3.23097**
	1	4	0.2202712	0.2475709	0.1929758	0.0673664	0.2718157	**4**
	2	3	0.0001200	0.3719992	0.3940348	0.1174282	0.1164178	**3.24895**
	3	2	0.1268981	0.3373748	0.0762611	0.3393618	0.1201042	2.66258
	4	2	0.2902854	0.0298295	0.2374428	0.2846716	0.1577707	2.59869
B	0	1	0.1090980	0.1872394	0.3428958	0.1047759	0.2559908	1.97201
	1	4	0.4264783	0.3289241	0.0329282	0.0963795	0.1152899	**4**
	2	3	0.1781506	0.0994063	0.1778567	0.2872842	0.2573021	3.08717
	3	1	0.1455517	0.2771827	0.1156606	0.0928521	0.3687528	1.98467
	4	2	0.3072229	0.2097444	0.2147215	0.0580455	0.2102657	2.44451
C	0	3	0.1540453	0.2218113	0.1457297	0.2761626	0.2022511	3.18097
	1	4	0.3063159	0.2390087	0.3216166	0.0912156	0.0418432	**4**
	2	3	0.0425038	0.3325381	0.3542672	0.1864457	0.0842452	3.22724
	3	2	0.2183328	0.1357567	0.1461126	0.2041456	0.2956523	**2.79344**
	4	3	0.1030679	0.2600612	0.1771837	0.1360895	0.3235977	**3.19595**

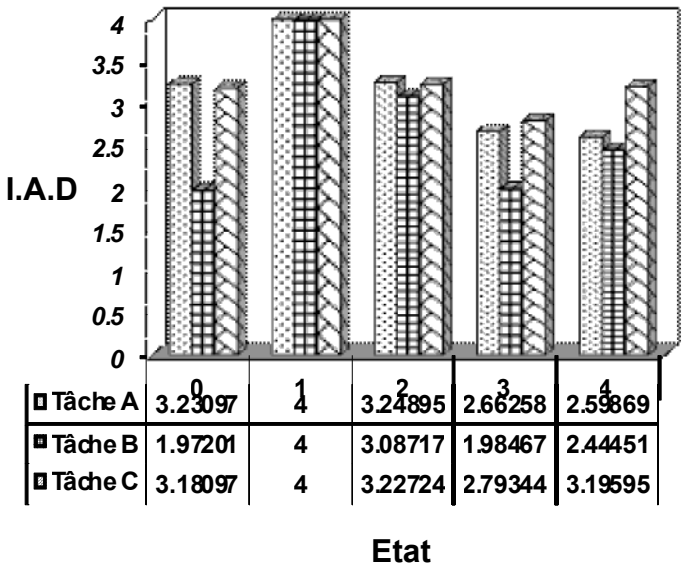
Pour la tâche A, une image écran 5.4 défile l'exécution du programme réalisé de l'algorithme_LU.



Ecran 5.4: Fenêtre du programme et de son exécution de la tâche A.

De la lecture de cette image écran, nous remarquons que les I.A.D en tout état de la tâche A ne sont pas fournis dans un ordre prédéfini.
De même pour les tâches B et C, des images écrans identiques peuvent s'obtenir et qu'on a omis de reproduire.
Pour la détermination d'une politique optimale d'exécution des trois tâches A, B et C nous reproduisons l'histogramme des I.A.D en tout état.

.



	0	1	2	3	4
Tâche A	3.23097	4	3.24895	2.66258	2.59869
Tâche B	1.97201	4	3.08717	1.98467	2.44451
Tâche C	3.18097	4	3.22724	2.79344	3.19595

Figure 5.7: Détermination de la politique optimale.

Un graphe indiquant une politique optimale est ci-dessous reproduit.

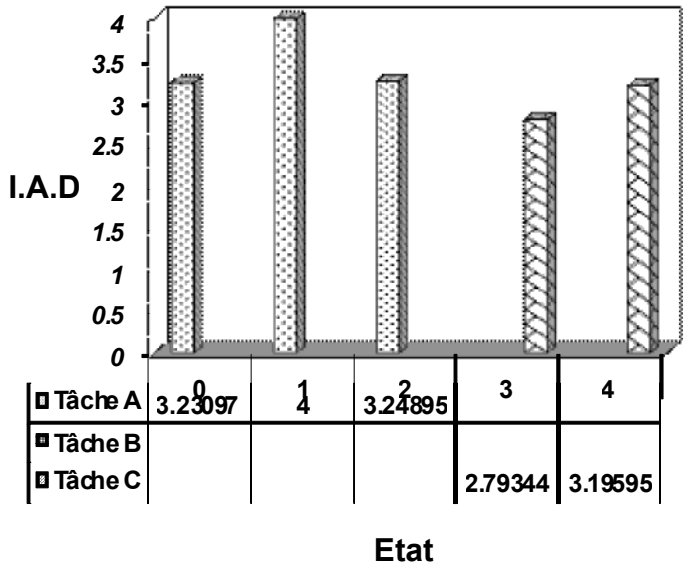


Figure 5.8: Une politique optimale.

La politique optimale (A, A, A, C, C) exécute la tâche A pendant trois unités de temps et la tâche C pendant deux unités de temps. La tâche B est gelée.

CHAPITRE 6
ÉTUDE COMPARATIVE DE QUATRE ALGORITHMES DE DÉTERMINATION DES I.A.D

1. Introduction

Nous avons implémenté les algorithmes de ROBINSON [13], de VAAIYA et AL. [17], de SONIN [15] et notre nouvel algorithme [11] en utilisant un langage évolué de programmation. Au risque de se répéter, les programmes ont été déroulés sur un processeur Intel Pentium III cadencé à 800 MHz et 176 Mo de RAM sous Windows XP. Ils sont annexés à la fin de ce mémoire.
Les critères de comparaison sont le temps de calcul fournit et le nombre maximum d'états que l'algorithme peut prendre en charge.
Une configuration d'un problème est appelée instance. Des expérimentations numériques sur un grand nombre d'exemples de problèmes d'ordonnancement, de l'ordre de mille, ont été effectuées. Un générateur de jeux d'essais est indispensable et est réalisé. Le compilateur du langage incorpore une procédure pour engendrer des nombres pseudo-aléatoires. Pour chaque problème, à partir de tels nombres, selon une loi uniforme, on associe un nombre d'états, un taux d'actualisation, les probabilités de transition et les gains. Pour évaluer les I.A.D, un outil logiciel a été réalisé et sera présenté dans le chapitre sept.

2. Comparaison entre les quatre algorithmes

Une étude comparative expérimentale et systématique de ces quatre algorithmes est élaborée. Nous mesurons leur performance en déterminant le nombre maximum d'états qu'ils peuvent prendre en charge et leur temps de calcul.

2.1. Nombre maximum d'états que chaque algorithme peut prendre en charge

Pour chaque algorithme et pour un nombre d'état fixé, on applique mille problèmes reproduits par notre générateur de jeux d'essais. Un problème se pose est celui de la gestion de la pile des données. Quand le système ne peut plus agrandir dynamiquement la pile. On réalise que ces algorithmes ne peuvent pas prendre

en charge l'exécution des problèmes au-delà d'un certain nombre d'états, qui est de cent soixante. On fait augmenter le nombre d'état par pas de «un» ou de plusieurs pas jusqu'à un blocage surviendra. Cela peut être provoqué par des variables locales très volumineuses, des appels récursifs de routines très profonds
Nos résultats obtenus sont ci-dessous récapitulés.

Tableau 6.1: Nombre maximum d'états que chaque algorithme peut prendre en charge.

Algorithme		**Nombre maximum d'état pris en charge**
Nouvel algorithme	Décomposition LU	143
	Point fixe PF	160
Algorithme de Robinson		45
Algorithme de Varaiya et al.		143
Algorithme de Sonin		113

Ils sont schématiquement présentés par l'histogramme suivant.

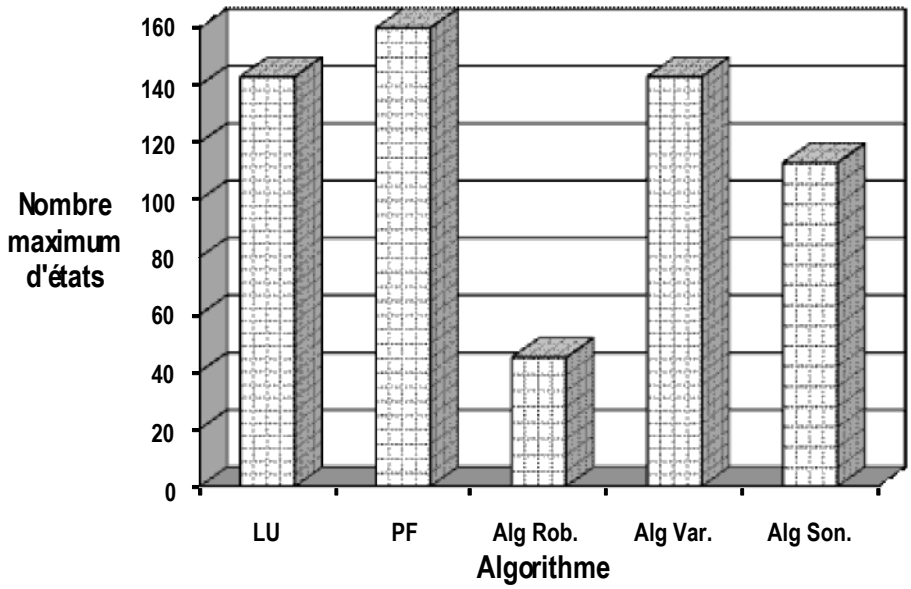


Figure 6.1: Histogramme donnant le nombre maximum d'états que chaque algorithme peut prendre en charge.

2.2. Temps de calculs

Pour chaque algorithme et pour un nombre d'état fixé, on exécute mille problèmes et on chronomètre par l'horloge du système le temps de calcul. La moyenne de ces temps d'exécution d'un problème en fonction du nombre d'états est récapitulée ci-dessous. Elle représente son temps de calcul donné en seconde. L'intervalle de confiance de la distribution de chaque échantillon est donné dans la dernière ligne. Il représente qu'il y a 99 % de chances pour que la moyenne de la population soit dans cet intervalle.

Tableau 6.2: Comparaison numérique entre les quatre algorithmes.
Nombre d'états en fonction du temps de calcul.

Nombre d'états	**Temps d'exécution en seconde**				
	Nouvel algo.		**Robinson**	**Var et al.**	**Sonin**
	LU	**PF**			
5	0.005	0.001	0.003	0.005	0.001
10	0.008	0.003	0.0038	0.007	0.002
15	0.01	0.005	0.005	0.0085	0.0035
20	0.03	0.0075	0.0065	0.01	0.0055
25	0.08	0.01	0.0098	0.02	0.008
30	0.15	0.015	0.018	0.04	0.01
35	0.27	0.02	0.03	0.07	0.015
40	0.45	0.04	0.05	0.11	0.02
45	0.7	0.05	0.075	0.17	0.03
50	1.05	0.07		0.25	0.04
55	1.51	0.1		0.35	0.05
60	2.12	0.13		0.49	0.07
70	3.82	0.22		0.87	0.108
80	6.44	0.35		1.45	0.15
90	10.17	0.55		2.28	0.21
100	15.54	0.8		3.43	0.27
110	21.5	1.12		5	0.36
113	27	1.18		5.9	0.39
120	31.64	1.44		7	
130	43.6	1.89		9.6	
140	56	2.41		12.9	
143	63.94	2.56		14.7	

150		2.89			
160		3.5			
Intervalle de confiance	[8.392,20.646]	[0.5286,1.2834]	[0.0059,0.0387]	[1.887,4.652]	[0.073,0.161]

Comme le nombre maximum d'états que chaque algorithme peut prendre en charge n'est pas identique et pour plus de clarté, dans deux figures consécutives, les figures 6.2.a et 6.2.b, on présente graphiquement les temps d'exécution en fonction du nombre d'états du problème.

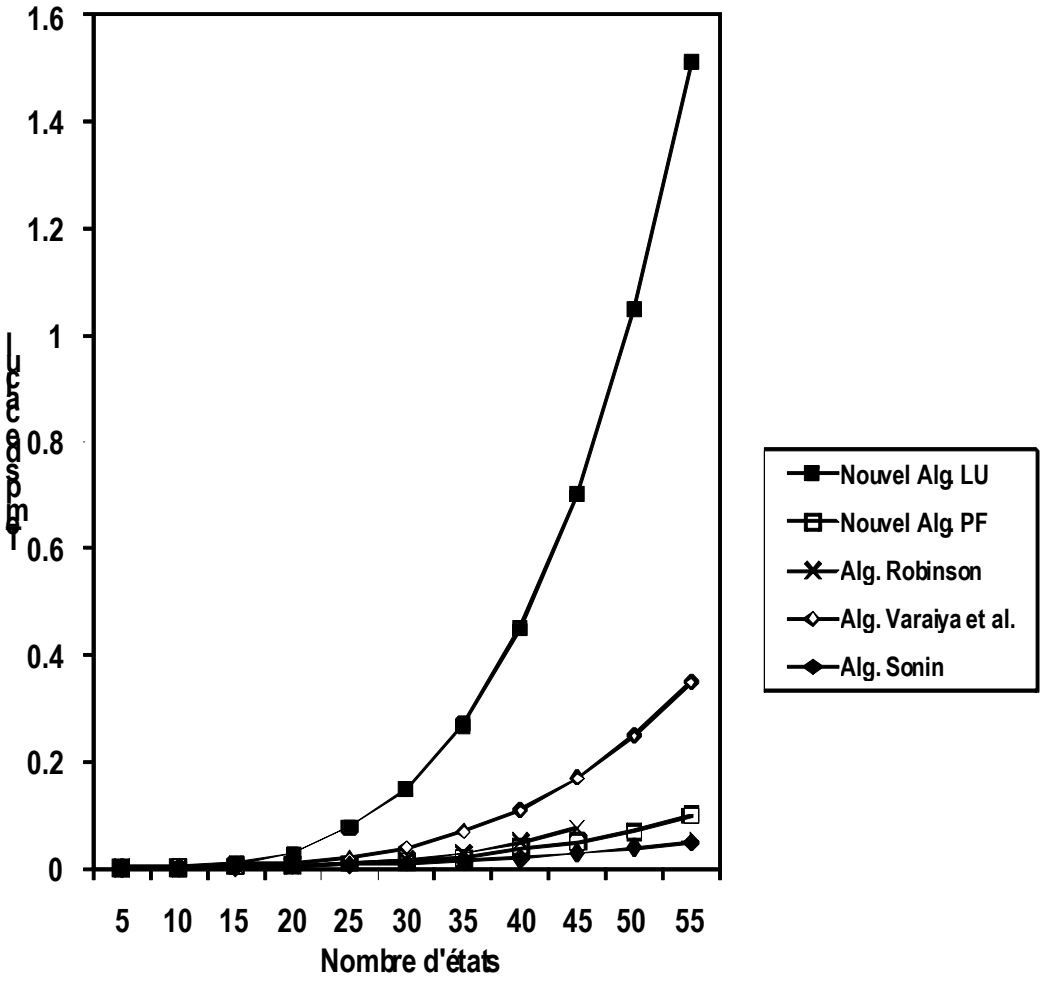


Figure 6.2.a : Comportement des quatre algorithmes.
Temps de calcul en fonction du nombre d'états.
Le nombre d'états ne dépassant pas les 55 états.

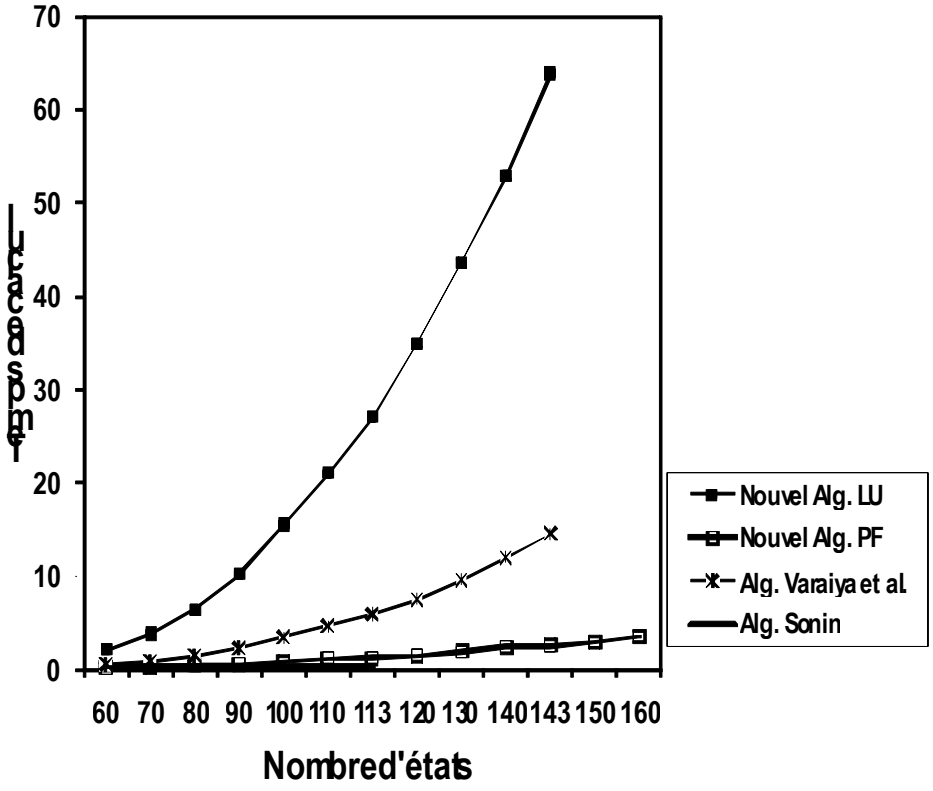


Figure 6.2.b: Comportement des quatre algorithmes.
Temps de calcul en fonction du nombre d'états.
Nombre d'états variant entre 60 et 160.

De la lecture des deux figures ci-dessus, et en tenant compte de la machine de calcul utilisée, nous remarquons que les quatre algorithmes fournissent des I.A.D en un temps raisonnable. La différence dans leurs temps d'exécution n'est pas trop significative.

2.3. La précision des résultats et l'efficacité de l'implémentation

Dans la suite, la précision des résultats obtenus par chaque algorithme et l'efficacité de notre implémentation est d'intérêt.

- Nous avons choisi cinquante problèmes de tailles différentes, au plus de 45 états, qui représentent de différentes configurations d'un problème d'ordonnancement stochastique, à qui on a appliqué ces quatre algorithmes. Les valeurs des I.A.D sont presque identiques. Les écarts dans les résultats fournis sont dus aux erreurs d'arrondis et à des erreurs de troncature dans la méthode du point fixe.

- Nous avons testé chaque algorithme sur 1000 problèmes de tailles différentes générés aléatoirement. Pour respectivement le nouvel algorithme, l'algorithme de ROBINSON, l'algorithme de VARAIYA et AL. et l'algorithme de SONIN la taille des problèmes varie respectivement de 1 à 160 états, de 1 à 45 états, de 1 à 143 états et de 1 à 113 états. En l'absence d'erreurs d'exécution ou de blocage, l'implémentation réalisée est qualifiée de stable et robuste.
  En utilisant l'horloge du système indiqué auparavant, nous avons calculé pour chaque algorithme la moyenne des temps d'exécution de ces 1000 problèmes. Le tableau suivant récapitule les résultats obtenus et observées. Toutes les mesures de temps sont données en secondes.

Tableau 6.3: Moyenne des temps de calcul que fournit l'application de chaque algorithme à 1000 problèmes.

Algorithme	**Temps de calcul : moyenne pour 1000 problèmes en seconde**
Nouvel Algorithme (De 1 à 160 états)	0.77951
Algorithme de Robinson (De 1 à 45 états)	0.0287
Algorithme de Varaiya et al. (De 1 à143 états)	1.85445
Algorithme de Sonin (De 1 à 113 états)	0.13009

Les résultats du tableau ci-dessus montre que l'implémentation réalisée est très efficace sur les différentes typologies de problèmes d'ordonnancement stochastiques.
Ils sont présentés par le histogramme suivant.

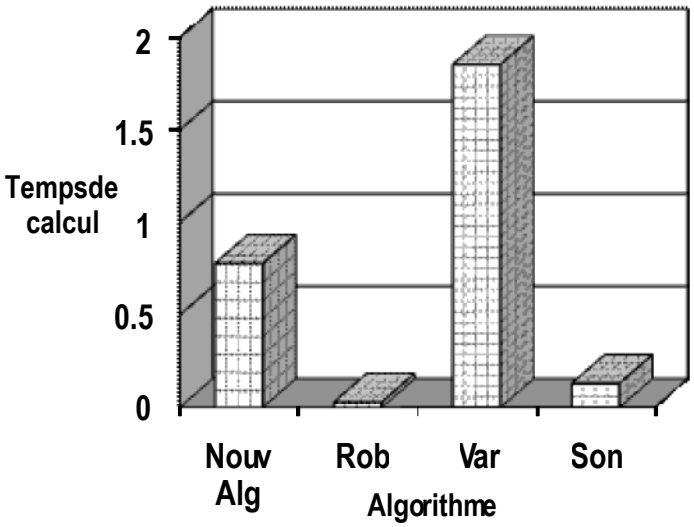


Figure 6.3. Histogramme donnant la moyenne des temps de calcul que fournit l'application de chaque algorithme à 1000 problèmes.

L'algorithme de ROBINSON n'exécute que des problèmes ne dépassant pas 45 états. Son temps d'exécution est appréciable et est considéré comme rapide.

Celui de SONIN peut aller au delà de 113 états. Il est le plus rapide des algorithmes disponibles cités dans la littérature.

Celui de VARAIYA et al. et notre nouvel algorithme_LU peuvent prendre en charge des problèmes à 143 états.

Notre nouvel algorithme_PF peut aller jusqu'à un nombre d'états de 160. Si le taux d'actualisation est proche de zéro, le temps de calcul est négligeable.

Pour des problèmes donnés à un ensemble d'états de cardinal inférieur à 113, l'algorithme de SONIN a donné des solutions le plus tôt possible, le second meilleur algorithme a été toujours notre nouvel algorithme_PF proposé dans cette étude.

CHAPITRE 7
PRÉSENTATION DU LOGICIEL I.A.D VERSION 1.0

Le logiciel I.A.D version 1.0. développé dans le cadre de ce mémoire de magister est un logiciel démonstratif purement pédagogique. C'est une étude expérimentale pour résoudre les problèmes d'ordonnancement stochastiques en utilisant l'approche indices d'allocation dynamiques. Ses intérêts sont multiples:

- Il permet de déterminer les I.A.D des processus en exécution par l'un des quatre algorithmes vus auparavant et de ce fait d'exhiber une politique d'indices.
- Il permet d'effectuer les calculs des exemples proposés dans ce mémoire et de vérifier les résultats obtenus.
- Il permet également de tester les différents modèles et de tracer des graphiques.

1. Démarrage de I.A.D. 1.0.

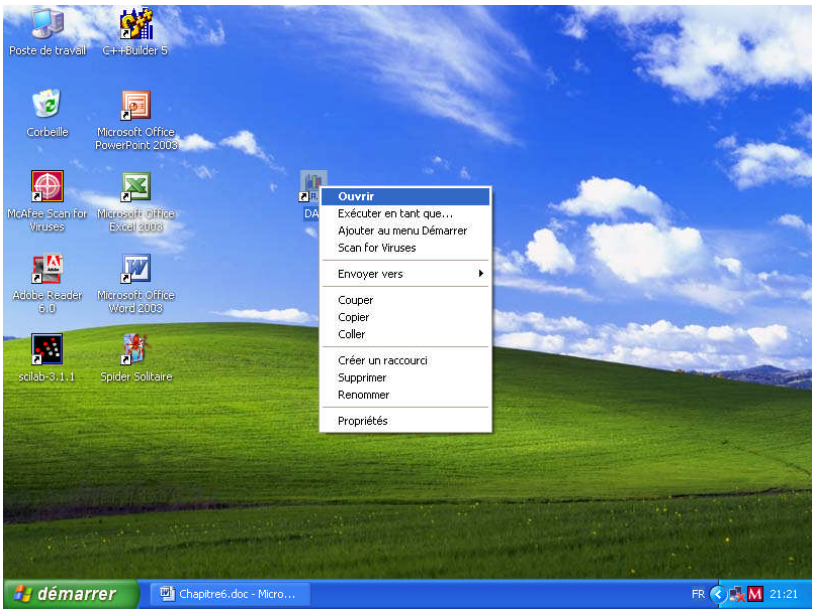
Le logiciel I.A.D version 1.0. est conçu en utilisant un langage de programmation évolué, Borland C++ Builder 5.

Borland C++ Builder 5 étant installé sur votre disque dur vous devez démarrer I.A.D. Pour cela suivez les instructions suivantes:

☞ Double-cliquez sur l'icône I.A.D (Ecran 7.1).

Ecran 7.1: Icône I.A.D.

☞ Eventuellement cliquez sur **Ouvrir** si vous obtenez un menu déroulant (Ecran 7.2).



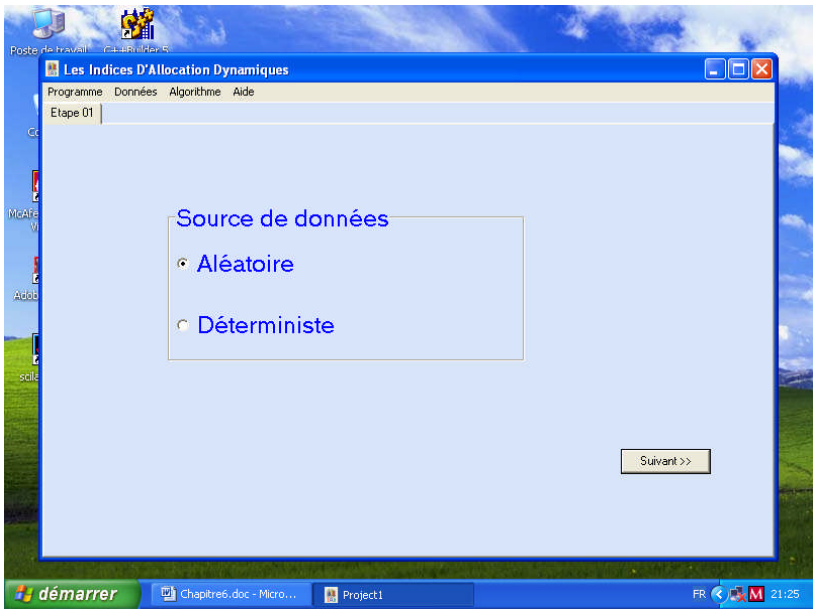
Ecran 7.2: Ouvrir dans le menu déroulant.

2. Fenêtres d'ouverture et d'accueil

Après le démarrage du logiciel I.A.D, on obtient la fenêtre d'ouverture de I.A.D (Ecran 7.3). Au bout de quelques secondes, on accède à la fenêtre d'accueil (Ecran 7.4).

Ecran 7.3: Fenêtre d'ouverture.

La fenêtre d'ouverture contient le nom, la version et la date de réalisation du logiciel. Les renseignements concernant l'auteur, l'université, la faculté et le département sont aussi fournis.



Ecran 7.4: Fenêtre d'accueil.

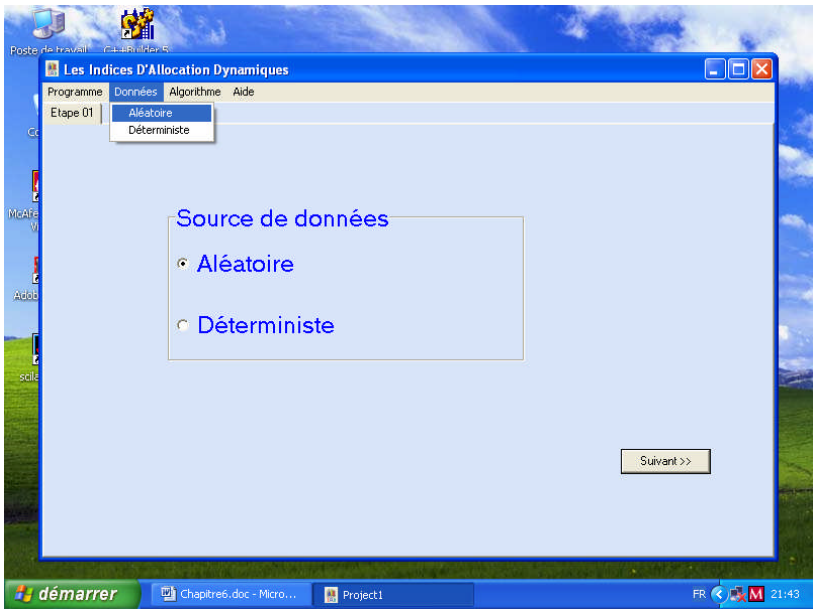
La fenêtre d'accueil est constituée de:

- La barre des menus. Il y a 4 titres de menus : **Programme, Données, Algorithme et Aide**.
- La barre des outils. Dans le premier temps une seule icône est affichée : **Etape1.**

3. Choix d'une source de données

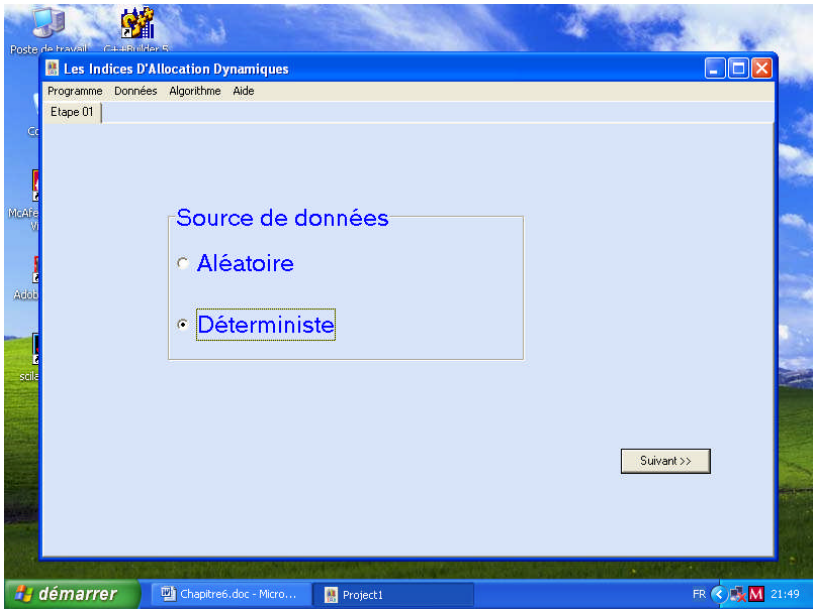
Dans la fenêtre **Etape 1,** vous devez choisir la source de données. Deux choix mutuellement exclusifs que l'utilisateur peut faire passer de Oui à Non sont possibles. **Aléatoire** ou **Déterministe**. Ils se font :

- dans la barre des menus, vous pouvez utiliser **Données** puis sélectionnez votre choix (Ecran 7.5).



Ecran 7.5: Choix d'une source de données
en utilisant **Données** dans la barre des menus**.**

☞ Ou vous utilisez le groupe de choix: **Source de données** puis sélectionner l'option désirée **(**Ecran 7.6).

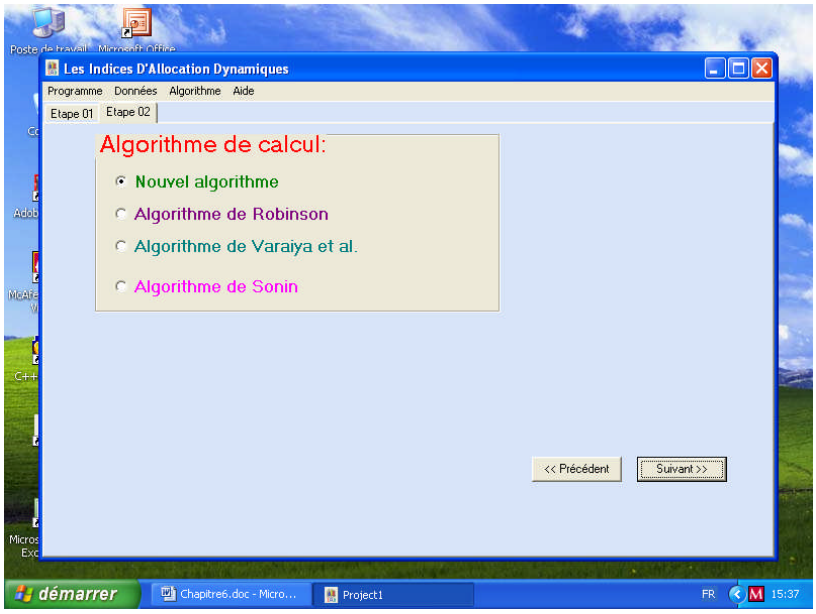


Ecran 7.6: Choix d'une source de données
en utilisant le groupe de choix.

☞ Après que le choix est fait, cliquez sur le bouton **Suivant** situé en bas de la fenêtre pour accéder à l'étape suivante.

4. Choix d'un algorithme

Si la source de données choisie à l'**Etape 01** est **Déterministe** alors dans l'**Etape 02** (Ecran 7.7) vous accédez à une nouvelle fenêtre qui vous propose de choisir un algorithme de détermination des I.A.D.

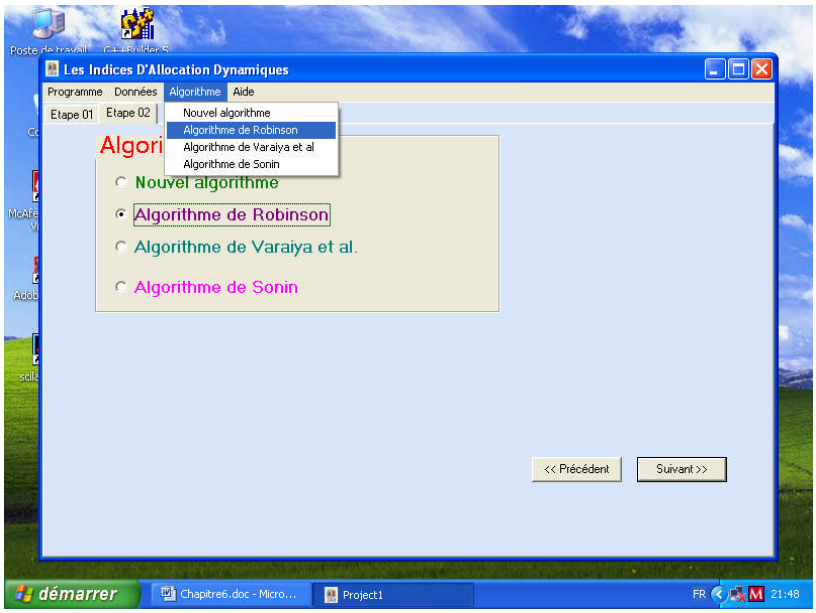


Ecran 7.7: Fenêtre permettrant de spécifie l'algorithme de détermination. Cas **Déterministe**.

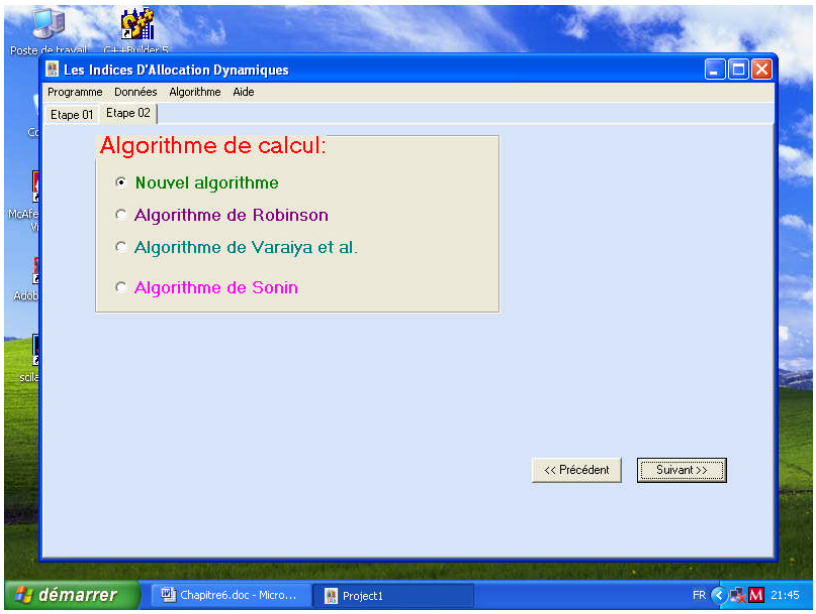
Quatre algorithmes sont disponibles: le **Nouvel algorithme**, **Algorithme de Robinson**, **Algorithme de Varaiya et al.** et **Algorithme de Sonin.**

☞ Deux façons de choisir sont offertes, barre des menus (Ecran 7.8) et le groupe de choix (Ecran 7.9).

.

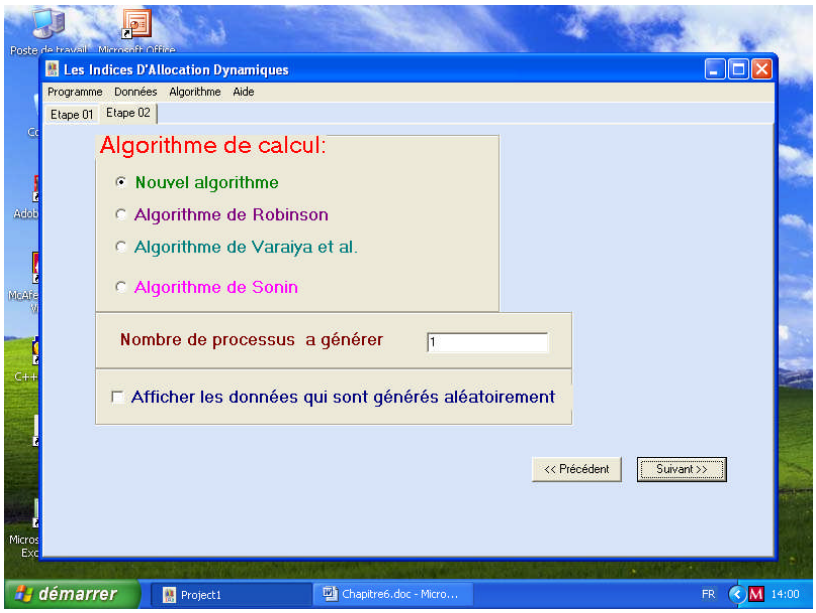


Ecran 7.8: Choix d'un algorithme en utilisant **Algorithme** dans la barre des menus**.** Cas **Déterministe**.



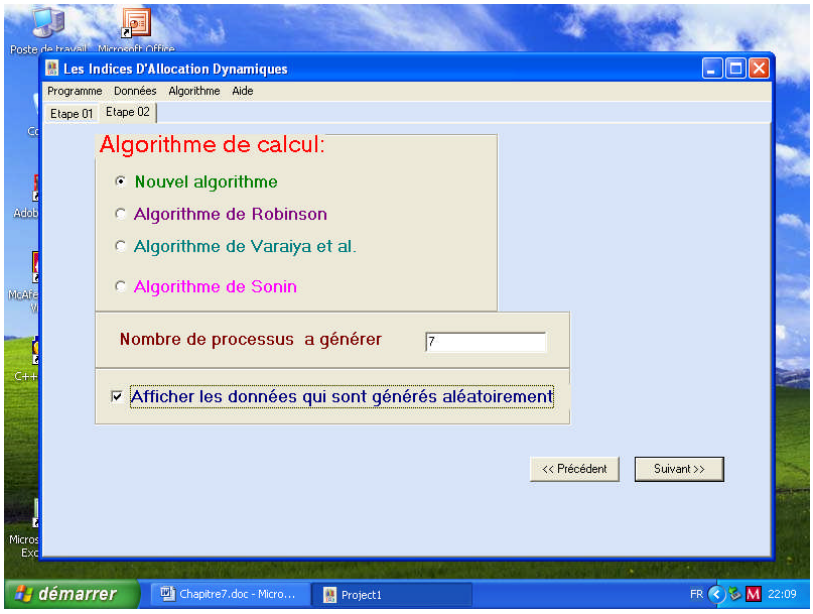
Ecran 7.9: Choix d'un algorithme en utilisant le groupe de choix. Cas **Déterministe**.

Si la source de données choisie à l'**Etape 01** est **Aléatoire,** vous devez, en outre que dans le cas déterministe, définir le nombre de processus bandits que l'ordinateur doit générer aléatoirement et uniformément (Ecran 7.10). Dans cette première version, nous limitons au cas uniforme. Des extensions aux cas exponentiel, normal et général peuvent s'obtenir. Par défaut le nombre de processus est de un.



Ecran 7.10: Fenêtre permettrant de spécifie
l'algorithme de détermination et le nombre de processus a générer.

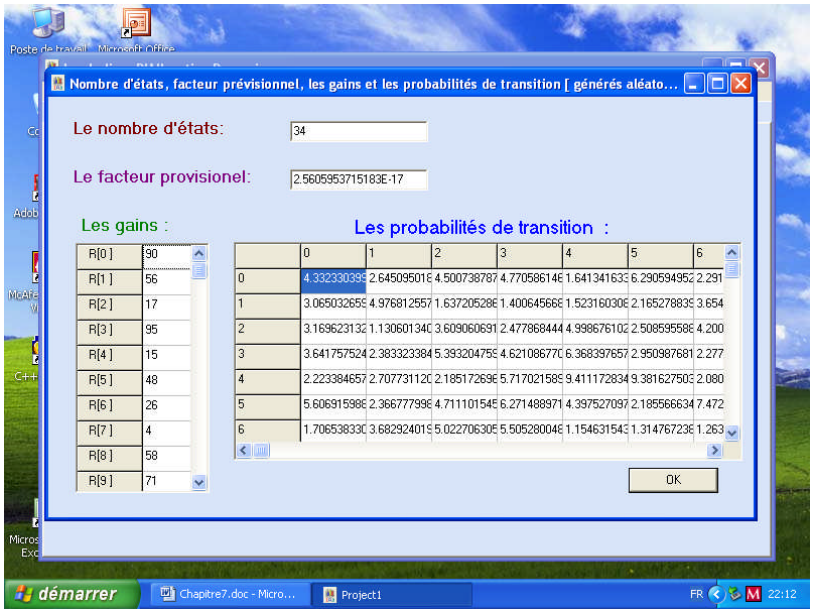
☞ En sélectionnant **Afficher les données qui sont générés aléatoirement** puis en appuyant sur le bouton **Suivant,** l'utilisateur a la possibilité de visualiser les valeurs des caractéristiques de ces processus produites par le générateur de jeux d'essais (Ecran 7.11).



Ecran 7.11: Le logiciel doit générer
7 processus et afficher leurs caractéristiques.

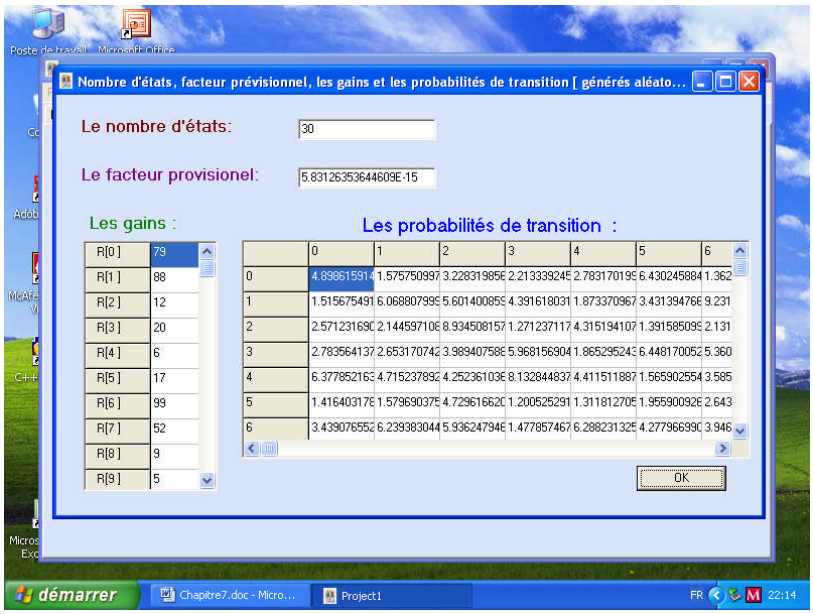
L'algorithme de calcul utilisé est notre nouvel algorithme. Les caractéristiques des processus sont le nombre d'état, taux d'actualisation, les gains octroyés en tout état

et les probabilités de transition entre ces états (Ecran 7.12). Les caractéristiques de chaque processus sont affichées dans une seule fenêtre.



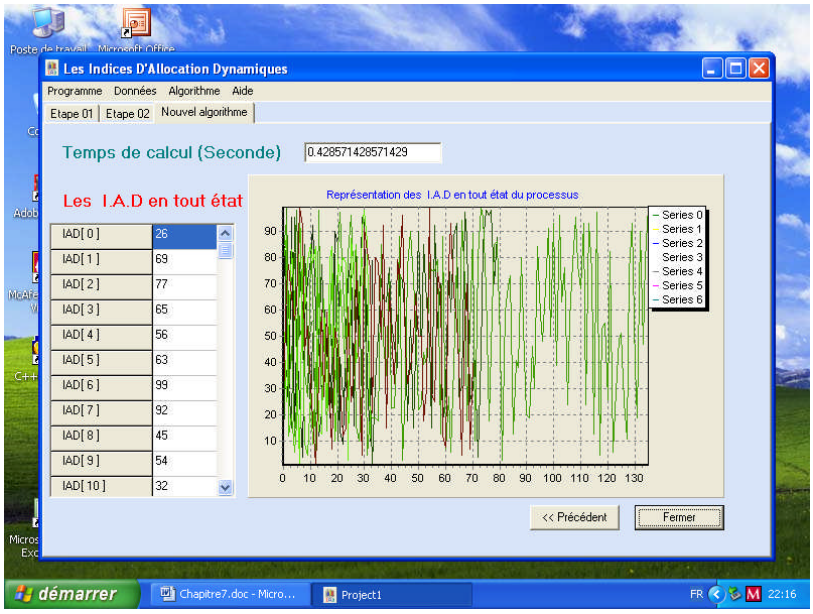
Ecran 7.12: Fenêtre donnant les caractéristiques
du premier processus produites par le générateur de jeux d'essais.

☞ Pour visualiser les caractéristiques des autres processus bandit, un déroulement d'images s'obtient en cliquant autant de fois possibles sur le bouton **OK** situé en bas de la fenêtre (Ecran 7.13).



Ecran 7.13: Fenêtre donnant les caractéristiques
du cinquième processus produite par le générateur de jeux d'essais.

☞ Si vous cliquez sur le bouton **OK** situé en bas de la fenêtre, donnant les caractéristiques du septième processus, une fenêtre porte le nom de l'algorithme utilisé apparaît. Elle donne l'accès aux valeurs des I.A.D du dernier processus, la moyenne des temps de calcul de ces sept processus, qui est de 0.43 seconde, et les graphiques « indice fonction de son état » du processus sont superposés dans un même repère. Les couleurs des graphiques sont choisies d'une manière aléatoire (Ecran 7.14).

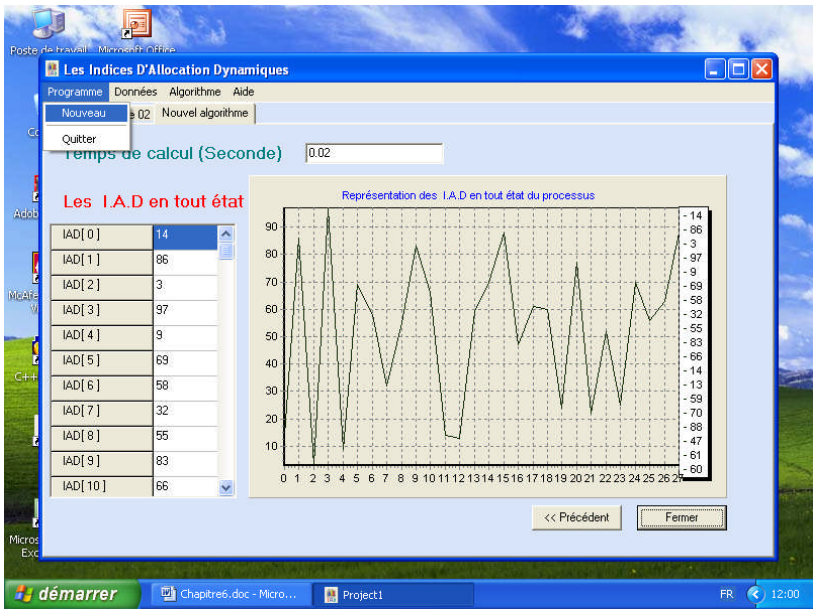


Ecran 7. 14: Valeurs de I.A.D, graphiques des processus
la moyenne des temps de calcul.

Cette fenêtre donne les valeurs des I.A.D du dernier processus déterminé par notre algorithme, les graphiques des 7 processus, I.A.D en fonction de son état et la moyenne des temps de calcul pour ces 7 processus.

Si dans l'**Etape 02,** cas **Aléatoire** (Voir Ecran 7.11) le choix d'**Afficher les données qui sont générés aléatoirement** n'est pas effectué, en cliquant sur le bouton **Suivant** on atteint directement la fenêtre **Nouvel algorithme** (Ecran 7.14).

☞ On peut toutefois revenir aux étapes précédentes en appuyant directement sur l'**Etape 01** ou l'**Etape 02** situées dans la barre de menus ou bien en cliquant sur le bouton **Précédent.** Il n'est pas possible de changer les données choisies auparavant. Ces fonctionnalités sont figées dans cette version. Par contre vous pouvez démarrer un nouvel exemple en utilisant dans la barre d'outils **Programme** puis **Nouveau** (Ecran 7.15).

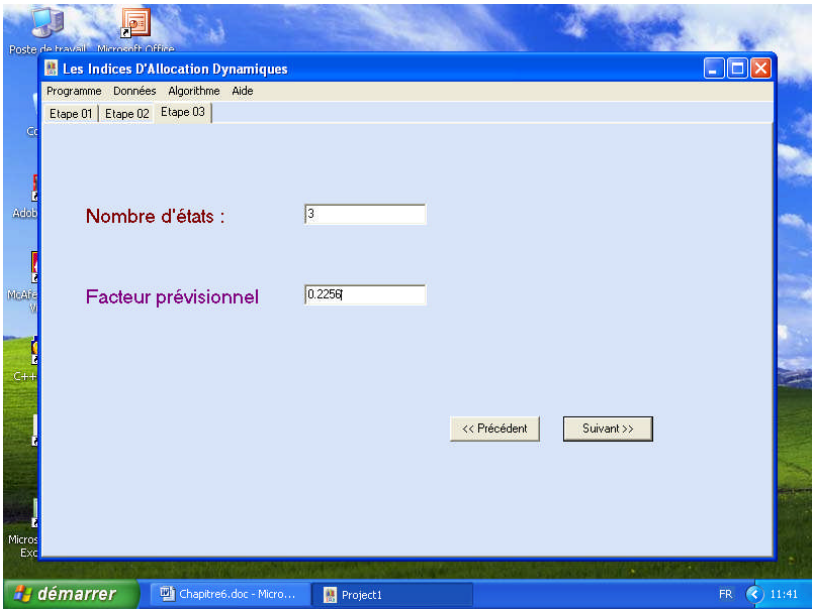


Ecran 7.15: Choix d'un nouvel exemple.

Si le choix d'un nouvel exemple est fait, nous accédons immédiatement à l'**Etape 01** (Ecran 7.4).

5. <u>Entrer les données</u>

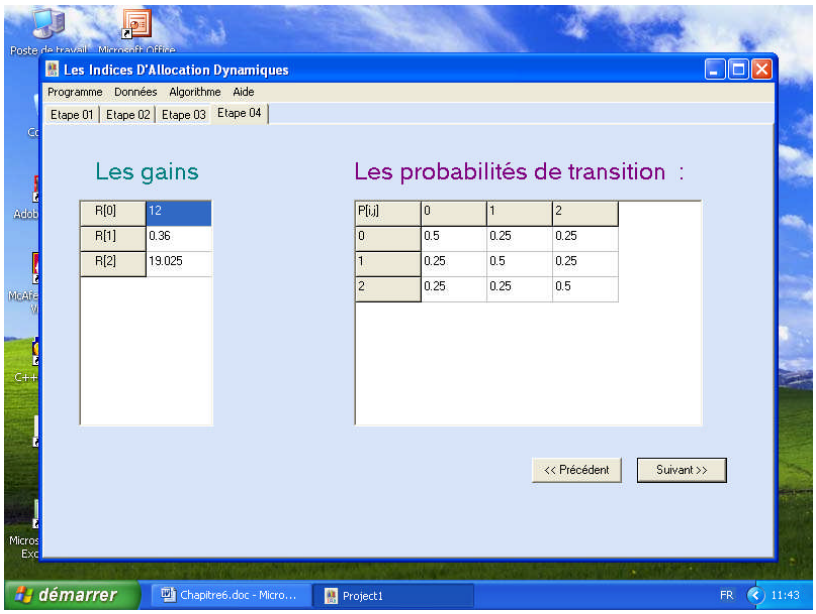
☞ Dans la fenêtre **Etape 02**, cas **Déterministe** (Ecran 7.9) cliquez sur **Suivant** pour accéder à la fenêtre **Etape 03** (Ecran 7.16)**.**



Ecran 7.16: Fenêtre permettent de spécifie le nombre d'états du processus et la valeur du taux d'actualisation.

La fenêtre **Etape 03** vous propose de définir les deux paramètres: le nombre d'états du processus et la valeur du taux d'actualisation.

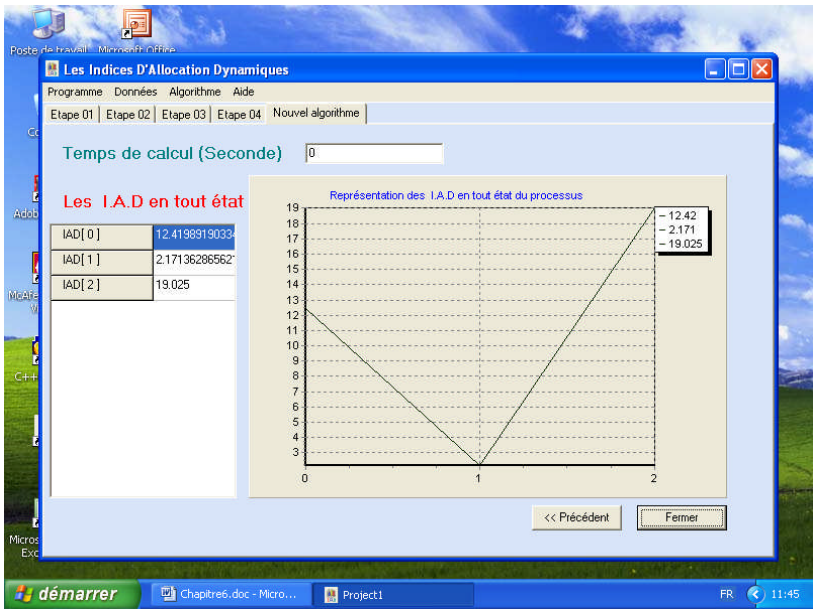
☞ Appuyez sur **Suivant** pour accéder à la fenêtre **Etape 04** (Ecran 7.17).



Ecran 7.17: Fenêtre permettent de spécifie les valeurs des gains en tout état du processus et les probabilités de transition entre ces états.

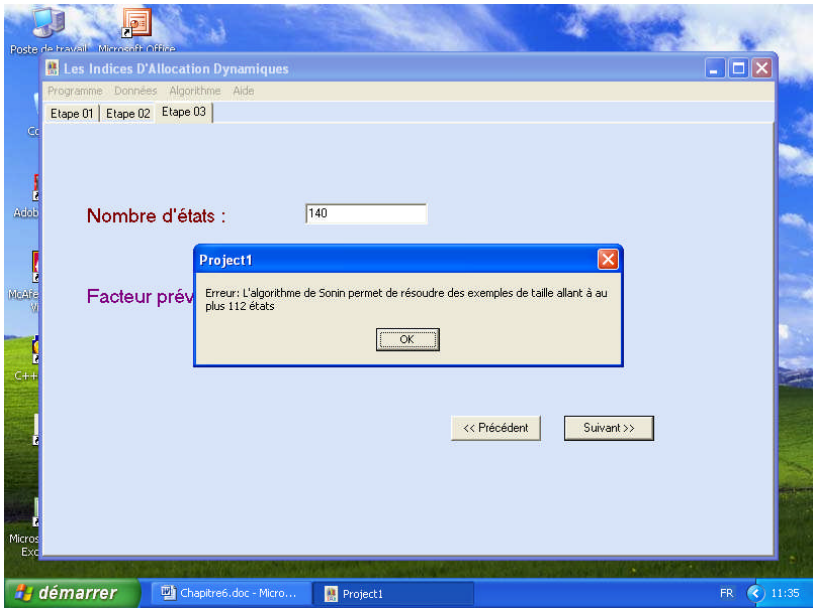
La fenêtre **Etape 04** vous propose deux tableaux où vous devez saisir les valeurs des gains en tout état du processus et les probabilités de transition entre ces états. Le nombre d'états choisi dans l'**Etape 3** définit la taille des deux tableaux.

☞ Si vous appuyez sur **Suivant,** la fenêtre qui porte le nom de l'algorithme utilisé apparaît (Ecran 7.18).

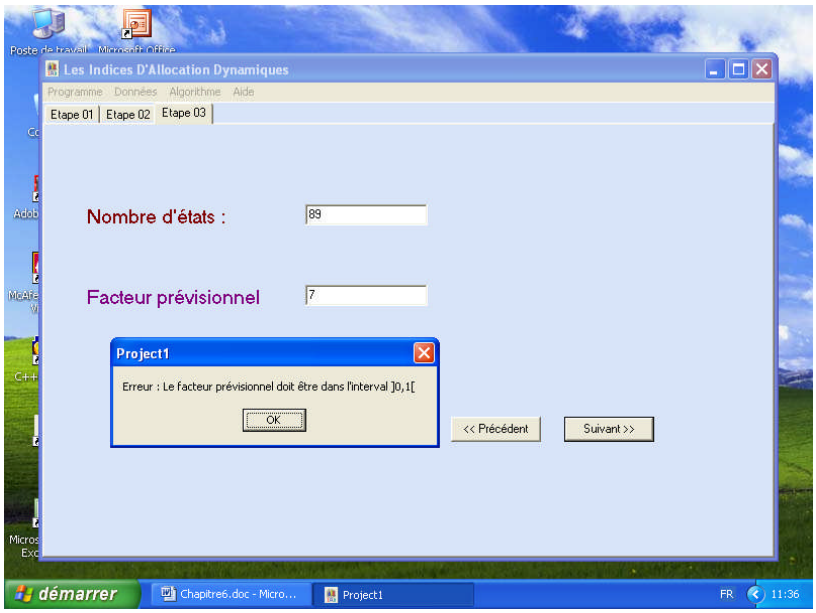


Ecran 7.18: Fenêtre donne l'accès aux valeurs des I.A.D du processus, le temps de calcul et le graphique état du processus fonction de son I.A.D.

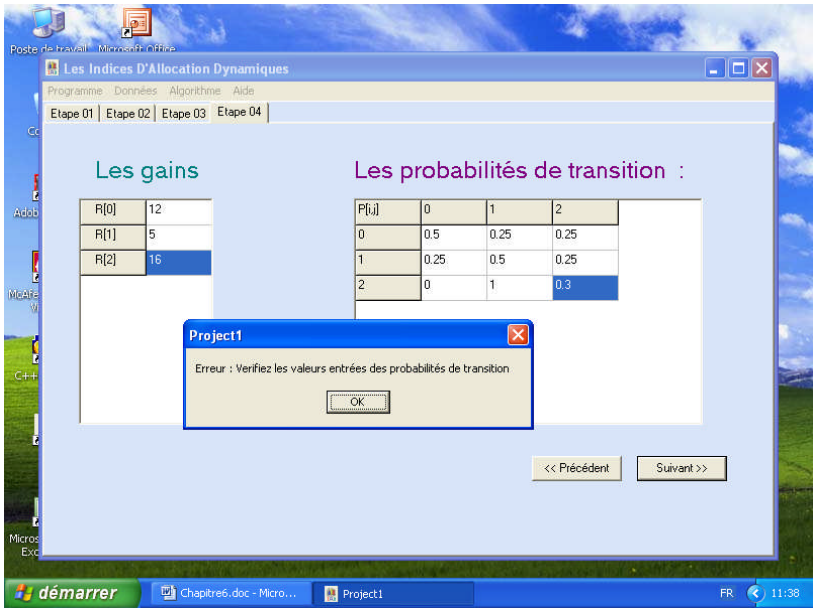
Toutefois, il faut signaler que le logiciel dans les **Etapes 03** et **04**, offre à l'utilisateur la potentialité de ne pas pouvoir saisir un caractère. Si le nombre saisi n'est pas dans les normes un message d'erreur vous dirige et une possibilité d'entrer une nouvelle valeur est offerte.



Ecran 7.19: Boite de dialogue précis l'erreur dans le choix de la taille du processus pour l'algorithme de Sonin.



Ecran 7.20: Boite de dialogue précis l'erreur
dans le choix de la valeur du taux d'actualisation.



Ecran 7.21: Boite de dialogue précis l'erreur
dans les valeurs des probabilités de transition.

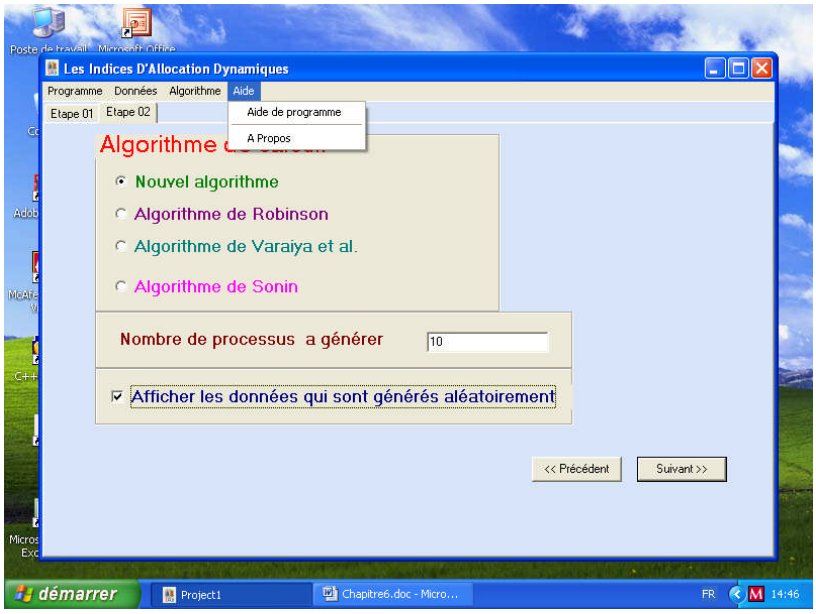
☞ D'une manière analogue que dans le cas **Aléatoire** vous pouvez atteindre les étapes précédentes en choisissant directement **Etape 01, 02, 03 ou 04** située dans la barre de menus ou bien en cliquant sur le bouton **Précédent** situé en bas de la fenêtre mais il n'est pas possible de changer les données. Ces fonctionnalités sont aussi figées dans cette version. Par contre vous pouvez démarrer un nouveau exemple en utilisant **Nouveau** dans le menu déroulant du titre **Programme**.

6. Enregistrement des résultats et appel d'un fichier enregistré

La possibilité d'enregistrer les résultats obtenus dans une base de donnée que nous pouvons les consulter dans d'autres applications n'est pas accessible dans cette première version. Elle est un des objectifs que nous désirerons réaliser prochainement dans la conception d'une version complète.

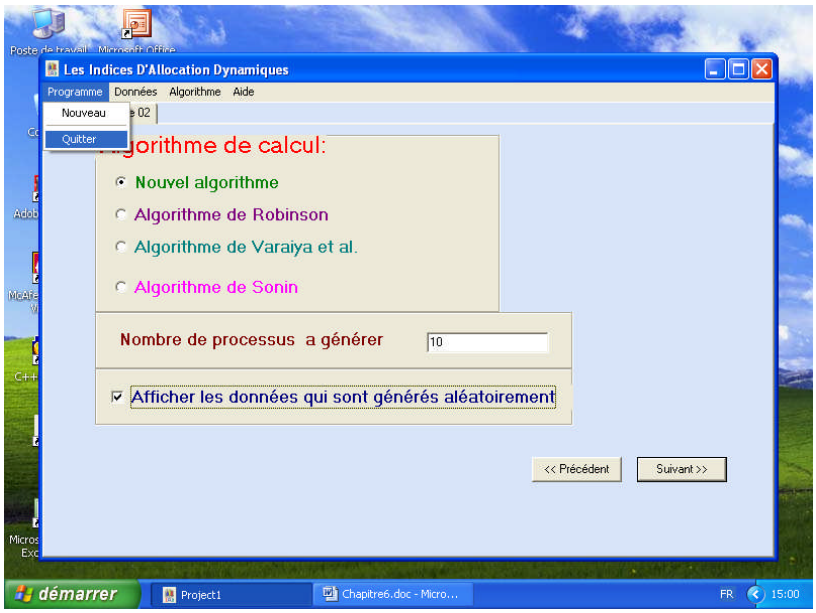
7. Le titre **Aide**

Dans le menu déroulant de **Aide (**Ecran 7.22), **Aide de programme** vous propose le contenu du paragraphe 3 de ce chapitre, notamment le contenu de notre *mémoire de magister* en format pdf et **A propos de** donne l'accès à la fenêtre d'ouverture.



Ecran 7.22: Les possibilités accessibles à partir du titre **Aide** dans la barre des menus.

☞ L'utilisateur peut quitter le programme en choisissant **Quitter** dans le menu déroulant du titre **Programme** dans la barre des menus (Ecran 7. 23) ou en utilisant fermer dans la page titre.



Ecran 7.23: Possibilité **Quitter**
dans le menu déroulant du titre **Programme**.

☞ Eventuellement, vous pouvez quitter le programme en appuyant sur **Fermer** dans les fenêtres des résultats (Ecrans 7.15 et 7.18).

CONCLUSION ET PERSPECTIVES

Nous avons introduit les processus bandits, des processus de décision semi-Markoviens, un outil de résolution des problèmes d'ordonnancement stochastiques. On associe, à chaque tâche et en tout instant, une priorité dynamique appelé indice d'allocation dynamique et est noté I.A.D. En tout instant, on exécute la tâche qui a le plus grand indice. En cas de conflit ou d'égalité on arbitrera entre les plus grands indices.

Si l'objectif du problème est une fonction à coûts séparables, cette politique d'indices est montrée optimale. L'existence, la caractérisation et la détermination de ses indices ont été étudiés. Un nouvel algorithme de détermination des indices est développé. Il permet de considérer et de prendre en charge des problèmes où les processus associés aux tâches ont des ensembles d'états allant jusqu'au nombre de cent soixante et en un temps appréciable.

Nous n'avons considéré que le cas idéal où les tâches sont indépendantes, la préemption est autorisée et la machine n'est pas soumise à la panne ou à la réparation. Aussi, un processus ne change d'état que s'il est exécuté. Dans le cas où il est gelé, il ne changera pas d'état.

Dans le cas où la machine est soumise à la panne, dans la bibliographie, des indices d'allocation dynamiques sont caractérisés pour les probabilités de panne de type Bernouilli, tâches simples avec des taux de fin d'exécution monotones. Il peut faire l'objet des études ultérieures.

Des problèmes sont encore ouverts dans le cas où la fonction objectif n'est pas à variables séparables. Les problèmes d'existence, de caractérisation et de détermination de la politique optimale persistent. Une règle d'indices peut ne pas être optimale si le processus change d'état au cas où il n'est pas exécuté. C'est la notion de processus « bandits actifs » (de l'Anglais restless bandits).

ANNEXE "A"

PROGRAMME DE L'ALGORITHME DE ROBINSON

```
#include<iostream>
#include<set>
#include<vector>
#include<conio>
#define M 45
using namespace std;
typedef set< int , less<int> > set_type;
ostream& operator<<(ostream& out, const set_type& set)
{
copy(set.begin(), set.end(),ostream_iterator<set_type::value_type,char>(cout,"  "));
return out;
 }
float main()
{
int N,n,k,t;
long double a,s,s1;
long double R[M],IAD[M],V[M];
long double P[M][M],S[M][M],T[M][M],z[M][M];
long double A[M][M][M];
cout<<"\nEntrez la valeur de N : le nombre d'état du processus bandit";
cout<<"\nN ="; cin>>N;
if(N<=0)
cout<<"\nErreur : le nombre d'état d'un processus bandit doit être un entier >=1";
else
{
cout<<"\nEntrez la valeur du taux d’actualisation a";
cout<<"\n a = "; cin>>a;
if(a<=0||a>=1)
cout<<"\nErreur : le taux d’actualisation doit être dans l'interval ]0,1[";
else
{
cout<<"\nEntrez les valeurs des gains";
t=0;
for(n=0;n<N;n++)
{
cout<<"\nR[ "<<n<<" ]="; cin>>R[n];
if(R[n]<0)
{
t++;
cout<<"\nErreur : le gain doit être >=0";
break;
}
}
if(t==0)
{
```

```
cout<<"\nEntrez les valeurs des probabilités de transition";
for(n=0;n<N;n++)
{
s=0;
for(k=0;k<N;k++)
{
cout<<"\nP[ "<<n<< " ][ "<<k<<" ]="; cin>>P[n][k];
s=s+P[n][k];
if(s>1)
{
cout<<"\nErreur : Vérifiez les valeurs entrées des probabilités de transition";
break;
}
}if(s>1)break;
}if(s<=1)
{
set_type I,Y;
set_type::iterator i,j,l,m,x;
for(n=0;n<N;n++)
I.insert(n);
vector<long double> L(R,R+N);
vector<long double>::iterator it;
it=max_element(L.begin(),L.end());
L.clear();
for(i=I.begin();i!=I.end();i++)
if (R[*i]==*it)
{
x=i;
IAD[*i]=*it;
cout<<"\n IAD["<<*i<<"]="<<*it<<endl;
break;
}
I.erase(*x);
for(k=2;k<=N;k++)
{
for(i=Y.begin();i!=Y.end();i++)
{
z[k-1][*i]=0;
for (l=Y.begin();l!=Y.end();l++)
z[k-1][*i]= z[k-1][*i]+A[k-1][*i][*l]*P[*l][*x];
}
s=0;
for(l=Y.begin();l!=Y.end();l++)
s=s+P[*x][*l]*z[k-1][*l];
A[k][*x][*x]= 1/(1-a*P[*x][*x]-pow(a,2)*s);
for(i=Y.begin();i!=Y.end();i++)
A[k][*i][*x]=a*A[k][*x][*x]*z[k-1][*i];
for(j=Y.begin();j!=Y.end();j++)
{
s=0;
for(l=Y.begin();l!=Y.end();l++)
s=s+P[*x][*l]*A[k-1][*l][*j];
A[k][*x][*j]=a*A[k][*x][*x]*s;
}
for(i=Y.begin();i!=Y.end();i++)
```

```
for(j=Y.begin();j!=Y.end();j++)
A[k][*i][*j]=A[k-1][*i][*j]+a*z[k-1][*i]*A[k][*x][*j];
Y.insert(*x);
for(l=Y.begin();l!=Y.end();l++)
{
S[k][*l]=0; T[k][*l]=0;
for(m=Y.begin();m!=Y.end();m++)
{
S[k][*l]=S[k][*l]+A[k][*l][*m]*R[*m];
T[k][*l]=T[k][*l]+A[k][*l][*m];
}
}
for(j=I.begin();j!=I.end();j++)
{
s=0;s1=0;
for(l=Y.begin();l!=Y.end();l++)
{
s=s+P[*j][*l]*S[k][*l];
s1=s1+P[*j][*l]*T[k][*l];
}
V[*j]=(R[*j]+a*s)/(1+a*s1);
L.push_back(V[*j]);
}
it=max_element(L.begin(),L.end());
L.clear();
for(i=I.begin();i!=I.end();i++)
if (V[*i]==*it)
{
x=i;
IAD[*i]=*it;
cout<<"\n IAD["<<*i<<"]="<<*it<<endl;
break;
}
I.erase(*x);
}
}
}
}
}
cout<<"\nAppuyer sur une touche pour sortir";
getch();
return 0;
}
```

ANNEXE "B"

PROGRAMME DE L'ALGORITHME DE VARAIYA ET AL.

```
#include<iostream>
#include<set>
#include<vector>
#include<conio>
#define M 143
using namespace std;
typedef set< int , less<int> > set_type;
ostream& operator<<(ostream& out, const set_type& s)
{
copy(s.begin(), s.end(),ostream_iterator<set_type::value_type,char>(cout,"  "));
return out;
}
set_type C,Y,Z,T;
set_type::iterator i,j,q;
float main()
{
int N,n,c,p,k;
long double a,s,s1;
long double R[M],IAD[M],X[M],v[M],nu[M],w[M];
long double P[M][M],A[M][M],L[M][M],U[M][M];
cout<<"\nEntrez la valeur de N : le nombre d'état du processus bandit";
cout<<"\nN ="; cin>>N;
if(N<=0)
cout<<"\nErreur : le nombre d'état d'un processus bandit doit être un entier >=1";
else
{
cout<<"\nEntrez la valeur du taux d’actualisation a";
cout<<"\n a = "; cin>>a;
if(a<=0||a>=1)
cout<<"\nErreur : le taux d’actualisation doit être dans l'interval ]0,1[";
else
{
cout<<"\nEntrez les valeurs des gains";
c=0;
for(n=0;n<N;n++)
{
cout<<"\nR( "<<n<<" )="; cin>>R[n];
if(R[n]<0)
{
c++;
cout<<"\nErreur : le gain doit être >=0";
break;
}
}
if(c==0)
{
```

```
cout<<"\nEntrez les valeurs des probabilités de transition";
for(n=0;n<N;n++)
{
s=0;
for(p=0;p<N;p++)
{
cout<<"\nP( "<<n<< " )( "<<p<<" )="; cin>>P[n][p];
s=s+P[n][p];
if(s>1)
{
cout<<"\nErreur : Vérifiez les valeurs entrées des probabilités de transition";
break;
}
}if(s>1)break;
}
if(s<=1)
{
for(n=0;n<N;n++)
C.insert(n);
cout<<"\nC = { "<<C<<"}"<<endl;
vector<long double> L1(R,R+N);
vector<long double>::iterator it;
it=max_element(L1.begin(),L1.end());
for(i=C.begin();i!=C.end();i++)
if (R[*i]==*it)
{
IAD[*i]=*it;
cout<<"\n IAD["<<*i<<"]="<<*it<<endl;
Y.insert(*i);
}
cout<<"\n Y = { "<<Y<<"}"<<endl;
k=1;
while(k==1)
{
insert_iterator<set<int, less<int> > >
        Z_ins(Z, Z.begin());
set_difference(C.begin(), C.end(),Y.begin(), Y.end(),Z_ins);
cout<<"\n Z = { "<<Z<<"}"<<endl<<endl;
for(i=Y.begin();i!=Y.end();i++)
for(j=Y.begin();j!=Y.end();j++)
{
if(*i==*j)
A[*i][*i]=1-a*P[*i][*i];
else
A[*i][*j]=-a*P[*i][*j];
}
for(i=Y.begin();i!=Y.end();i++)
{
L[*i][*i]=1;
s=0;s1=0;
for(j=Y.begin();*j<*i;j++)
{
for(q=Y.begin();*q<*j;q++)
s=s+L[*i][*q]*U[*q][*j];
L[*i][*j]=(A[*i][*j]-s)/U[*j][*j];
```

```
}
for(*j=*i;j!=Y.end();j++)
{
for(q=Y.begin();*q<*i;q++)
s1=s1+L[*i][*q]*U[*q][*j];
U[*i][*j]=A[*i][*j]-s1;
}
}
for(i=Y.begin();i!=Y.end();i++)
{
s=0;
for(j=Y.begin();*j<*i;j++)
s=s+L[*i][*j]*X[*j];
X[*i]=R[*i]-s;
}
set_type::reverse_iterator r(Y.end());
set_type::reverse_iterator r_end(Y.begin());
set_type::reverse_iterator t(Y.end());
for(; r != r_end; r++)
{
s=0;
for(;*t>*r;t++)
s=s+U[*r][*t]*v[*t];
v[*r]=(X[*r]-s)/U[*r][*r];
}
for(i=Y.begin();i!=Y.end();i++)
{
s=0;
for(j=Y.begin();*j<*i;j++)
s=s+L[*i][*j]*X[*j];
X[*i]=1-s;
}
set_type::reverse_iterator h(Y.end());
set_type::reverse_iterator h_end(Y.begin());
set_type::reverse_iterator d(Y.end());
for(; h != h_end; h++)
{
s=0;
for(;*d>*h;d++)
s=s+U[*h][*d]*w[*d];
w[*h]=(X[*h]-s)/U[*h][*h];
}
vector<long double>V;
for(i=Z.begin();i!=Z.end();i++)
{
s=0;s1=0;
for(j=Y.begin();j!=Y.end();j++)
{
s=s+P[*i][*j]*v[*j];
s1=s1+P[*i][*j]*w[*j];
}
v[*i]=R[*i]+a*s;
w[*i]=1+a*s1;
nu[*i]=v[*i]/w[*i];
V.push_back(nu[*i]);
```

```
}
it=max_element(V.begin(),V.end());
for(i=Z.begin();i!=Z.end();i++)
if (nu[*i]==*it)
{
IAD[*i]=*it;
cout<<"\nIAD["<<*i<<"]="<<IAD[*i]<<endl;
T.insert(*i);
}
insert_iterator<set<int, less<int> > >
            Y_ins(Y, Y.begin());

set_union(Y.begin(), Y.end(),T.begin(), T.end(),Y_ins);
cout<<"\n Y = { "<<Y<<"}"<<endl;
T.clear();
Z.clear();
cout<<"\nY==C "<<(Y==C?"true":"false")<<endl;
if(Y==C)break;
}
}
}
}
}
cout<<"\nAppuez sur une touche pour sortir";
getch();
return 0;
}
```

ANNEXE "C"

PROGRAMME DE L'ALGORITHME DE SONIN

```
#include<iostream>
#include<set>
#include<vector>
#include<conio>
#define M 113
using namespace std;
typedef set< int , less<int> > set_type;
ostream& operator<<(ostream& out, const set_type& s)
{
copy(s.begin(), s.end(),ostream_iterator<set_type::value_type,char>(cout,"  "));
return out;
}
set_type C,Z,D,S,W,K;
set_type::iterator i,j,q,l,m;
float main()
{
int N,n,c,p,k;
long double a,s,s1;
long double R[M],IAD[M],x[M],y[M],b[M],d[M],T[M];
long double P[M][M],Q[M][M],L[M][M],U[M][M],E[M][M],A[M][M],N1[M][M];
cout<<"\nEntrez la valeur de N : le nombre d'état du processus bandit";
cout<<"\nN ="; cin>>N;
if(N<=0)
cout<<"\nErreur : le nombre d'état d'un processus bandit doit être un entier >=1";
else
{
cout<<"\nEntrez la valeur du taux d’actualisation a";
cout<<"\n a = "; cin>>a;
if(a<=0||a>=1)
cout<<"\nErreur : le taux d’actualisation doit être dans l'interval ]0,1[";
else
{
cout<<"\nEntrez les valeurs des gains";
c=0;
for(n=0;n<N;n++)
{
cout<<"\nR( "<<n<<" )="; cin>>R[n];
if(R[n]<0)
{
c++;
cout<<"\nErreur : le gain doit être >=0";
break;
}
}
```

```
if(c==0)
{
cout<<"\nEntrez les valeurs des probabilités de transition";
for(n=0;n<N;n++)
{
s=0;
for(p=0;p<N;p++)
{
cout<<"\nP( "<<n<< " )( "<<p<<" )="; cin>>P[n][p];
s=s+P[n][p];
if(s>1)
{
cout<<"\nErreur : Vérifiez les valeurs entrées des probabilités de transition";
break;
}
}if(s>1)break;
}
if(s<=1)
{
P[N][N]=1;
for(n=0;n!=N;n++)
{
P[n][N]=1-a;
P[N][n]=0;
for(p=0;p<N;p++)
P[n][p]=a*P[n][p];
}
for(n=0;n<N;n++)
{
C.insert(n);
Z.insert(n);
}
vector<long double> V;
vector<long double>::iterator it;
k=1;
while(k==1)
{
for(i=Z.begin();i!=Z.end();i++)
{
d[*i]=R[*i]/P[*i][N];
V.push_back(d[*i]);
}
it=max_element(V.begin(),V.end());
V.clear();
for(i=Z.begin();i!=Z.end();i++)
if (d[*i]==*it)
D.insert(*i);
for(i=D.begin();i!=D.end();i++)
{
IAD[*i]=(1-a)**it;
cout<<"\n IAD("<<*i<<")="<<IAD[*i]<<endl;
}
insert_iterator<set<int, less<int> > >
            W_ins(W,W.begin());
set_union(S.begin(), S.end(),D.begin(), D.end(),W_ins);
```

```
S.clear();
for(i=W.begin();i!=W.end();i++)
S.insert(*i);
W.clear();
if(S==C)break;
insert_iterator<set<int, less<int> > >
            K_ins(K,K.begin());
set_difference(Z.begin(),Z.end(),D.begin(),D.end(),K_ins);
Z.clear();
for(i=K.begin();i!=K.end();i++)
Z.insert(*i);
K.clear();
if(D.size()==1)
{
m=D.begin();
for(i=Z.begin();i!=Z.end();i++)
R[*i]=R[*i]+(P[*i][*m]*R[*m])/(1-P[*m][*m]);
Z.insert(N);
for(i=Z.begin();i!=Z.end();i++)
for(j=Z.begin();j!=Z.end();j++)
P[*i][*j]=P[*i][*j]+(P[*i][*m]*P[*m][*j])/(1-P[*m][*m]);
Z.erase(N);
}
else
{
for(i=D.begin();i!=D.end();i++)
for(j=D.begin();j!=D.end();j++)
{
if(*i==*j)
Q[*i][*i]=1-P[*i][*i];
else
Q[*i][*j]=-P[*i][*j];
}
for(l=D.begin();l!=D.end();l++)
{
for(i=D.begin();i!=D.end();i++)
{
if(*i==*l)
b[*i]=1;
else
b[*i]=0;
}
for(i=D.begin();i!=D.end();i++)
{
L[*i][*i]=1;
s=0;s1=0;
for(j=D.begin();*j<*i;j++)
{
for(q=D.begin();*q<*j;q++)
s=s+L[*i][*q]*U[*q][*j];
L[*i][*j]=(Q[*i][*j]-s)/U[*j][*j];
}
for(*j=*i;j!=D.end();j++)
{
for(q=D.begin();*q<*i;q++)
```

```
s1=s1+L[*i][*q]*U[*q][*j];
U[*i][*j]=Q[*i][*j]-s1;
}
}
for(i=D.begin();i!=D.end();i++)
{
s=0;
for(j=D.begin();*j<*i;j++)
s=s+L[*i][*j]*y[*j];
y[*i]=b[*i]-s;
}
set_type::reverse_iterator r(D.end());
set_type::reverse_iterator r_end(D.begin());
set_type::reverse_iterator t(D.end());
for(; r != r_end; r++)
{
s=0;
for(;*t>*r;t++)
s=s+U[*r][*t]*x[*t];
x[*r]=(y[*r]-s)/U[*r][*r];
}
for(i=D.begin();i!=D.end();i++)
E[*i][*l]=x[*i];
}
Z.insert(N);
for(i=Z.begin();i!=Z.end();i++)
for(j=D.begin();j!=D.end();j++)
{
A[*i][*j]=0;
for(q=D.begin();q!=D.end();q++)
A[*i][*j]=A[*i][*j]+P[*i][*q]*E[*q][*j];
}
for(i=Z.begin();i!=Z.end();i++)
for(j=Z.begin();j!=Z.end();j++)
{
N1[*i][*j]=0;
for(q=D.begin();q!=D.end();q++)
N1[*i][*j]=N1[*i][*j]+A[*i][*q]*P[*q][*j];
}
for(i=Z.begin();i!=Z.end();i++)
for(j=Z.begin();j!=Z.end();j++)
P[*i][*j]=P[*i][*j]+N1[*i][*j];
Z.erase(N);
for(i=Z.begin();i!=Z.end();i++)
{
T[*i]=0;
for(j=D.begin();j!=D.end();j++)
T[*i]=T[*i]+A[*i][*j]*R[*j];
R[*i]=R[*i]+T[*i];
}
}
D.clear();
}
}
}
```

```
}
}
cout<<"\nAppuez sur une touche pour sortir";
getch();
return 0;
}
```

ANNEXE "D"

PROGRAMME DE L'ALGORITHME_PF

```
#include<iostream>
#include<set>
#include<conio>
#include<vector>
#include<math>
#define M 160
#define Nb 1000000
using namespace std;
typedef set<  int , less<int> > set_type;
ostream& operator<<(ostream& out, const set_type& set)
{
copy(set.begin(), set.end(),ostream_iterator<set_type::value_type,char>(cout,"  "));
return out;
}
set_type C,Y;
set_type::iterator i,j;
class AbsValue
{
public:
        bool operator()(long double first,long double second)const
         {
                return (fabsl(second)>fabsl(first));
         }
};
float main()
{
int N,n,m,k,t;
long double a,b,erreur,S,NR,NP;
long double R[M],IAD[M],z[M],K[M];
long double P[M][M],y[M][Nb],H[M][M];
vector<long double> V;
vector<long double>::iterator it;
cout<<"\nDonnez l'erreur";
cout<<"\nErreur = "; cin>>erreur;
if(erreur<=0||erreur>=1)
cout<<"\nErreur: l'erreur du troncature doit être positive proche de zero";
else
{
cout<<"\nEntrez la valeur de N : le nombre d'état du processus bandit";
cout<<"\nN ="; cin>>N;
if(N<=0)
cout<<"\nErreur : le nombre d'état d'un processus bandit doit être un entier >=1";
else
{
cout<<"\nEntrez la valeur du taux d’actualisation a";
cout<<"\n a = "; cin>>a;
```

```
if(a<=0||a>=0.5)
cout<<"\nErreur : le taux d’actualisation doit être dans l'interval ]0,0.5[";
else
{
cout<<"\nEntrez : les valeurs des gains";
k=0;
for(n=0;n<N;n++)
{
cout<<"\nR( "<<n<<" )="; cin>>R[n];
if(R[n]<0)
{
k++;
cout<<"\nErreur : Le gain doit être>=0";
break;
}
}
if(k==0)
{
cout<<"\nEntrez les valeurs des probabilités de transition";
for(n=0;n<N;n++)
{
S=0;
for(m=0;m<N;m++)
{
cout<<"\nP( "<<n<< " )( "<<m<<" )="; cin>>P[n][m];
S=S+P[n][m];
if(S>1)
{
cout<<"\nErreur : Vérifiez les valeurs entrées des probabilités de transition";
break;
}
}if(S>1) break;
}
if(S<=1)
{
for(n=0;n<N;n++)
y[n][0]=0;
for(m=0;m<N;m++)
{
for(n=0;n<N;n++)
C.insert(n);
cout<<"\nC= { "<<C<<"}"<<endl;
k=1;
while (k==1)
{
for(i=C.begin();i!=C.end();i++)
{
K[*i]=R[*i]-R[m];
V.push_back(K[*i]);
}
it=max_element(V.begin(),V.end(),AbsValue());
NR=fabsl(*it);
V.clear();
for(i=C.begin();i!=C.end();i++)
{
```

```
S=0;
for(j=C.begin();j!=C.end();j++)
{
H[*i][*j]=fabsl(P[*i][*j]-P[m][*j]);
S=S+H[*i][*j];
}
V.push_back(S);
}
it=max_element(V.begin(),V.end());
NP=*it;
V.clear();
if(NR==0 || NP==0)
{
IAD[m]=R[m];
cout<<"\nIAD ( "<<m<<" )="<<IAD[m];
break;
}
else
{
b=(logl(erreur/NR))/(logl(a*NP));
t=0;
while(k==1)
{
t++;
if(t>b||t>Nb)break;
for(i=C.begin();i!=C.end();i++)
{
S=0;
for(j=C.begin();j!=C.end();j++)
S=S+(P[*i][*j]-P[m][*j])*(y[*j][t-1]);
y[*i][t]=R[*i]-R[m]+a*S;
}
}
for(i=C.begin();i!=C.end();i++)
{
if(y[*i][t-1]>0)
{
cout<<"\n y("<<*i<<")("<<t-1<<")="<<y[*i][t-1];
Y.insert(*i);
}
}
cout<<"\nY = { "<<Y<<"}"<<endl;
if(Y.empty()==true)
{
IAD[m]=R[m];
cout<<"\nIAD ( "<<m<<" )="<<IAD[m];
C.clear();
break;
}
if(Y==C)
{
cout<<"\nC == Y  "<<(C==Y?"true":"false")<<endl;
S=0;
for(i=Y.begin();i!=Y.end();i++)
S=S+P[m][*i]*y[*i][t];
```

```
IAD[m]=R[m]+a*S;
cout<<"\nIAD ( "<<m<<" )="<<IAD[m];
C.clear();
Y.clear();
break;
}
C.clear();
for(i=Y.begin();i!=Y.end();i++)
C.insert(*i);
cout<<"\nC = { "<<C<<"}"<<endl;
Y.clear();
}
}
}
}
}
}
}
}
cout<<"\nAppuez sur une touche pour sortir";
getch();return 0;
}
```

ANNEXE "E"

PROGRAMME DE L'ALGORITHME_LU

```
#include<iostream>
#include<set>
#include <vector>
#include<conio>
#define M 143
using namespace std;
typedef set<  int , less<int> > set_type;
ostream& operator<<(ostream& out, const set_type& s)
{
copy(s.begin(), s.end(),ostream_iterator<set_type::value_type,char>(cout,"  "));
return out;
}
set_type::iterator i,j,q;
float main()
{
int N,n,p,c,m,k;
long double a,s,s1;
long double R[M],B[M],X[M],y[M],IAD[M];
long double P[M][M],A[M][M],L[M][M],U[M][M];
cout<<"\nEntrez la valeur de N : le nombre d'état du processus bandit";
cout<<"\nN ="; cin>>N;
if(N<=0)
cout<<"\nErreur : le nombre d'état d'un processus bandit doit être un entier >=1";
else
{
cout<<"\nEntrez la valeur du taux d’actualisation a";
cout<<"\n a = "; cin>>a;
if(a<=0||a>=1)
cout<<"\nErreur : le taux d’actualisation doit etre dans l'interval ]0,1[";
else
{
cout<<"\nEntrez les valeurs des gains";
c=0;
for(n=0;n<N;n++)
{
cout<<"\nR( "<<n<<" )="; cin>>R[n];
if(R[n]<0)
{
c++;
cout<<"\nErreur : le gain doit être >=0";
break;
}
}
if(c==0)
{
cout<<"\nEntrez les valeurs des probabilités de transition";
```

```
for(n=0;n<N;n++)
{
s=0;
for(p=0;p<N;p++)
{
cout<<"\nP( "<<n<< " )( "<<p<<" )="; cin>>P[n][p];
s=s+P[n][p];
if(s>1)
{
cout<<"\nErreur : Vérifiez les valeurs entrées des probabilités de transition";
break;
}
}if(s>1)break;
}
if(s<=1)
{
set_type C,Y;
for(m=0;m<N;m++)
{
for(n=0;n<N;n++)
C.insert(n);
cout<<"\nC = { "<<C<<"}"<<endl;
for(i=C.begin();i!=C.end();i++)
for(j=C.begin();j!=C.end();j++)
{
if(*i==*j)
A[*i][*j]=1-a*(P[*i][*i]-P[m][*j]);
else
A[*i][*j]=-a*(P[*i][*j]-P[m][*j]);
}
for(i=C.begin();i!=C.end();i++)
B[*i]=R[*i]-R[m];
k=1;
while(k == 1)
{
for(i=C.begin();i!=C.end();i++)
{
L[*i][*i]=1;
s=0;s1=0;
for(j=C.begin();*j<*i;j++)
{
for(q=C.begin();*q<*j;q++)
s=s+L[*i][*q]*U[*q][*j];
L[*i][*j]=(A[*i][*j]-s)/U[*j][*j];
}
for(*j=*i;j!=C.end();j++)
{
for(q=C.begin();*q<*i;q++)
s1=s1+L[*i][*q]*U[*q][*j];
U[*i][*j]=A[*i][*j]-s1;
}
}
for(i=C.begin();i!=C.end();i++)
{
s=0;
```

```
for(j=C.begin();*j<*i;j++)
s=s+L[*i][*j]*X[*j];
X[*i]=B[*i]-s;
}
set_type::reverse_iterator r(C.end());
set_type::reverse_iterator r_end(C.begin());
set_type::reverse_iterator t(C.end());
 for(; r != r_end; r++)
{
s=0;
for(;*t>*r;t++)
s=s+U[*r][*t]*y[*t];
y[*r]=(X[*r]-s)/U[*r][*r];
}
for(i=C.begin();i!=C.end();i++)
if(y[*i]>0)
Y.insert(*i);
cout<<"\nY = { "<<Y<<"}"<<endl;
if(Y.empty()==true)
break;
cout<<(Y==C?"true":"false")<<endl;
if(Y==C)break;
C.clear();
for(i=Y.begin();i!=Y.end();i++)
C.insert(*i);
cout<<"\nC = { "<<C<<"}"<<endl;
Y.clear();
}
s=0;
for(i=Y.begin();i!=Y.end();i++)
s=s+P[m][*i]*y[*i] ;
IAD[m]=R[m]+a*s;
cout<<"\nIAD("<<m<<")="<<IAD[m]<<endl;
C.clear();
Y.clear();
}
}
}
}
}
cout<<"\nAppuez sur une touche pour sortir";
getch();
return 0;
}
```

ANNEXE "F"

GÉNÉRATEUR DE JEUX D'ESSAIS
TIRAGE UNIFORME

```
#include<iostream>
#include<conio>
#define N 5
#define M 10000
#define H 10000

using namespace std;
float main()
{
int n,k,r;
long double a,b;
long double R[N],C[N];
long double P[N][N],B[N][N];
randomize();
for(r=0;r<H;r++)
{
b=rand()%M;
if(b!=0 && b!=M-1)
break;
}
a=b/(M-1);
cout<<"\na ="<<a<<endl;
for(n=0; n<N; n++)
{
for(r=0;r<H;r++)
{
R[n]=rand()%M;
if(R[n]!=0)break;
}
cout<<"\nR["<<n<<"]="<<R[n]<<endl;
}
for(n=0; n<N; n++)
{
B[n][0]=rand()%M;
P[n][0]=B[n][0]/(M-1);
cout<<"\nP("<<n<<")(0)="<<P[n][0]<<endl;
C[n]=1-P[n][0];
}
for(n=0; n<N; n++)
for(k=1; k<N-1; k++)
{
```

```
for(r=0;r<H;r++)
{
B[n][k]=rand()%M;
P[n][k]=B[n][k]/(M-1);
if(P[n][k]<=C[n])break;
}
if(P[n][k]<=C[n])
{
C[n]=C[n]-P[n][k];
cout<<"\nP("<<n<<")("<<k<<")="<<P[n][k]<<endl;
}
else
{
P[n][k]=0;
cout<<"\nP["<<n<<"]["<<k<<"]="<<P[n][k]<<endl;
}
}
for(n=0; n<N; n++)
{
P[n][N-1]=C[n];
cout<<"\nP("<<n<<")("<<N-1<<")="<<P[n][N-1]<<endl;
}
cout<<"\nAppuyer sur une touche pour sortir";
getch();
return 0;
}
```

ANNEXE "G"

PROGRAMME DONNANT LE TEMPS D'EXÉCUTION D'UN ALGORITHME

```
#include<iostream>
#include<stdlib>
#include <dos>
#include<conio>

using namespace std;

float main()
{
long double cr,di,df,ho,mi,se,fr;
struct  time t;
gettime(&t);
   fr=t.ti_hund;
   se=t.ti_sec;
   mi=t.ti_min;
   ho=t.ti_hour;
   di=fr+se*100+mi*6000+ho*360000;
```

ALGORITHME

```
gettime(&t);
   fr=t.ti_hund;
   se=t.ti_sec;
   mi=t.ti_min;
   ho=t.ti_hour;
   df=fr+se*100+mi*6000+ho*360000;
   cr=df-di;

cout<<"\nla durée d'exécution de l'algorithme est  "<<cr<<"  centiemes de seconde";
cout<<"\nAppuyer sur une touche pour sortir";
getch();
return 0;
}
```

RÉFÉRENCES

1. Culioli, J. C., (1980), "Introduction à l'optimisation", édition ellipses, 26-31.

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

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

4. Frostig, E. et Weiss, G., (1999), "Four proofs of Gittins multiarmed bandit theorem", *Applied. Proba. Trust*, 1-20.

5. Gittins, J.C. et Jones, D.M., (1972), "A dynamic index for the sequential Design of experiments, *Colloquia Mathematica Janes Bolai*, European meeting of statisticians, Budapest, Hungary, 241-266.

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

7. 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.

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

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

10. Glazebrook, K.D., (1983), "Optimal strategies for families of alternative bandit processes", *IEEE Transactions on automatic control*, vol AC-28, n°8, 858-861.

11. Kali, A. et Derbala, A. (2006),"Détermination des Indices d'allocation dynamiques", *MOSIM 2006, Actes de la 6e Conférence Francophone de MOdélisation et SIMulation – « Modélisation, Optimisation et Simulation des Systèmes : Défis et Opportunités »*. 3, 4 et 5 avril 2006, Rabat, Maroc, volume 2, session 34- Modèles stochastiques, 1742-1748.

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

13. Robinson, D.R. (1982),"Algorithms for evaluating the dynamic allocation index", *Operations Research Letters,* volume 1, n°2, 72-74.

14. Ross, M., (1970), "Applied probability models with optimization applications". Holden-Day.

15. 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.

16. Tanenbaum, A (1994). SYSTÈMES D'EXPLOITATION : Systèmes centralisés, systèmes distribués. Prentice Hall, InterEditions.

17. 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.

18. Whittle, P., (1980), "Multi-armed bandits and the Gittins index", *J. Roy. statist Soc. Ser. B* 42, 2, 143- 149.

19. Yih, Y. et Thesen, A. (1991), "Semi-Markov decision models for real-time scheduling", *Int. J. Prod.Res.*, vol. 29, N0. 11, 2331-2346.